UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

FIRST PRINCIPLES MACHINE LEARNING IN RADAR: AUGMENTING SIGNAL PROCESSING TECHNIQUES WITH MACHINE LEARNING FOR DETECTION, TRACKING, AND NAVIGATION

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

ALEXANDER J. STRINGER
Norman, Oklahoma
2024

FIRST PRINCIPLES MACHINE LEARNING IN RADAR: AUGMENTING SIGNAL PROCESSING TECHNIQUES WITH MACHINE LEARNING FOR DETECTION, TRACKING, AND NAVIGATION

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

To Eileen and Zoey

Acknowledgments

I would like to start by thanking my wonderful partner, Eileen. She has been my closest friend and confidant for 19 years, and her encouragement has helped me to stay grounded and upbeat throughout my graduate studies. I would also like to thank my advisor, Dr. Justin Metcalf, for guiding me through my doctoral journey. I find his dedication to his students and passion for this field inspiring, and hope to carry some of that forward throughout my carrier. I would like to thank my committee, Dr. Yeary, Dr. Goodman, and Dr. Hougen, for sharing some of their wisdom with me. Their insights were invaluable to the progression of this work. I would also like to thank the US Air Force's 76 SWEG and the Air Force Office of Scientific Research (AFOSR) for funding this research. I would especially like to thank my AFOSR sponsor, Dr. Erik Blasch. A huge thank you to my fellow researcher and partner in (figurative) crime, Geoff. He has worked closely with me on much of this research and made my time as a doctoral candidate fun. Another huge thank you to my research team, especially Joe and Timmy. We've accomplished a lot, and I'm excited to see where we go from here. I would like to thank my 76 SWEG research director, Adam Bowersox, for the leadership and support he provided to our team. He helped establish an open and collaborative environment for our team. Finally, I would like to thank my mother, Dr. Gillian Bond. She has always been my role model and one of my biggest supporters. Her intellectual curiosity, drive, and integrity are the standards I strive to uphold.

Table of Contents

List of Tables

List of Figures

Abstract

Machine learning (ML) provides a set of tools for learning approximate system models from data. It has the potential to improve classic radar signal processing (RSP) algorithms by allowing them to maintain performance when the environmental assumptions used to derive them are violated. This could mitigate performance degradation experienced in more challenging scenarios, like those commonly found in airborne and maritime radar. However, the integration of ML into RSP algorithms presents a unique challenge due to the strict performance requirements of radar systems and often unpredictable nature of ML.

This work examines an architectural approach to explainable ML that allows for the seamless integration of ML with more traditional algorithmic methods. This approach is then paired with causal ML concepts to develop a method for mitigating measurement drift in tracking and navigation. Next, an integrated system of low-cost ML systems are developed to enable adaptive detection algorithms to maintain CFAR-like performance across a range of interference distributions. Finally, generative ML techniques are used to reduce sample support requirements for adaptive detectors by directly constructing whitening filters from a small set of interference samples. This dissertation presents a framework for the successful integration of ML into RSP algorithms using a targeted approach based on a clear understanding of the first principles physics at play in a given application.

Chapter 1

Introduction

Since its inception in the early 1900s, radar has rapidly become one of primary sensing methods for observing our surroundings. It is broadly used for detecting and tracking objects moving through a space, mapping and imaging environments, and distance determinations in navigation. This is enabled by radar signal processing (RSP) techniques that are used to extract information from radar signal returns. Such methods are commonly used to detect targets of interest, determine range-Doppler information of entities, generate images from radar returns, and use multiple radar pulses to reduce power requirements and increase the observable range.

RSP algorithms are typically derived from the first principles equations used to model energy travelling through space and its interactions with objects in that space. Due to the highly complex nature of these interactions, many aspects must be modeled probabilisticly under a number of key assumptions. This is particularly true when modeling measurement noise and interference. Noise in radar systems is assumed to follow an uncorrelated Gaussian distribution and is caused by ambient energy in the environment and inside the system itself. Clutter is caused by reflections of the radar signal from the environment. Perfectly modeling clutter returns would be shockingly difficult, given that any environment will have vast numbers of scatterers and many environmental factors (wind, temperature, etc.) may cause significant variation in the reflection characteristics of any particular scatterer between measurements. However, by assuming that the environment contains an effectively infinite number of point scatterers whose reflections are independent and identically distributed, it is commonly approximated using a correlated Gaussian distribution.

The assumptions used to derive existing RSP algorithms hold true in many situations, and the algorithms themselves have proven to be highly effective under those conditions. However, modern advances in radar have led to its use in increasingly challenging scenarios such as air- and ship-borne sensing. This has led to many RSP algorithms being used under conditions that severely degrade their performance. Adaptive detection algorithms used in radar are prime examples of this. These algorithms are derived to maximize detection performance while maintaining a constant rate of false alarms under a clutter model that is assumed to be known and consistent. When the clutter changes rapidly or no longer fits the expected distribution, as happens regularly on airborne systems, either the radar's ability to detect targets is reduced or the rate of false alarms is increased. Either case can interfere with radar tracking and resource management. Considerable work has been done to improve RSP algorithms and reduce their constraints [1]. One of the key focus areas of this research in recent years has been the use of machine learning (ML) in RSP [2, 3, 4].

ML refers to a broad set of tools and techniques that allow models of complex systems to be approximated based on examples. This is accomplished by first defining a parameterized transformation structure to convert some input to a target output (such as a decision tree or neural network). The transformation structure is referred to as a model. A set of example inputs are then provided and the model produces outputs. For each output, some statistical measure of correctness is calculated and used to update the parameters of the model. That process is repeated until some stopping criterion is reached. These methods have been shown to be highly effective at providing functional models of systems for which closed form expressions don't exist. The use of ML in radar systems has seen mixed results. Techniques like reinforcement learning (RL) have been used to great effect in radar task management [5] and scheduling frequency hopping [6]. On the other hand, ML has seen limited adoption in radar detection algorithms [7, 8]. There are a number of factors that contribute to this, chief among them are computational overhead, data limitations, and the strict error tolerance required to achieve useful false alarm rates.

Much of the challenge associated with trying to broadly utilize ML-based algorithms in radar is rooted in the design philosophy that is applied. Most exploration of ML application to the radar domain has focused on replacing existing RSP methods with ML models. There are many issue with that approach. Since traditional RSP algorithms are derived from the first principles equations used to describe the radar scenario, their behaviors are consistent, provable, and repeatable under the appropriate conditions. Each of these properties is essential to the design process when building actual radar platforms. Conversely, the learned and more approximate nature of ML models makes them less predictable and precludes any sort of mathematically provable characteristic (like the false alarm rate in detectors). Worse, the more complex the task or system being learned, the worse the approximation errors become. Unexpected or undesirable behaviors in ML systems are also difficult to replicate or troubleshoot. In spite of this, ML has significant potential to improve the field of RSP. This is because ML is highly versatile and does not rely on the same sorts of assumptions that are used to derive traditional RSP algorithms. By changing our design approach, it is possible to achieve both the flexibility afforded by ML and the consistency of traditional RSP.

In this dissertation, I explore methods for effectively integrating ML techniques into adaptive radar detection, tracking, and navigation. This integration is achieved by using a three-stage design approach. I begin by conducting analysis on a radar problem space (such as detection in variable clutter backgrounds) using a first principles understanding to break it down into component pieces. As a part of this analysis, it is crucial to identify the assumptions and constraints of existing RSP methods and identify any that could be mitigated or removed through the use of ML. I then design and implement a combination of efficient, reliable ML models and well-known RSP algorithms to solve each component of the problem space. Finally, I assemble the ML and RSP elements into a fully orchestrated system. The effectiveness of the proposed design approach for ML-enhanced RSP is demonstrated through its application to create a drift correction method for tracking and navigation systems, the development of an adaptive detection algorithm, and the use of generative ML for adaptive filtering.

1.1 Summary of Contributions

In order to showcase the potential of RSP enhanced with first principles ML, a number of tools and techniques have been designed, implemented, and tested for a range of radar applications. The work presented in this dissertation can be broadly separated in four areas, each with its own set of contributions.

1.1.1 The Self-Explaining Decision Architecture

The first major research focus area explored here was ML explainability. ML systems approximate functions through iterative learning processes and often contain vast numbers of parameters. That combination makes it extremely difficult to identify why ML systems arrive at particular outputs and to troubleshoot issues that arise with model performance. To address these limitations, I developed an architecture for ML that combines ML attention mechanisms with hierarchical decision structures to allow ML systems to simultaneously make decisions and provide text-based explanations of those decisions. This framework, titled the self-explaining decision architecture (SEDA), broke down complex decisions into increasingly simpler components and generated explanations modeled after human descriptions containing both attention information and decision reasoning. It was designed in a modular fashion to allow for the easy integration of different ML and non-ML components. While the development of SEDA is very much a ML contribution, the methods defined within it for breaking down complex problems, orchestrating systems of ML models, and integrating ML and non-ML components have served as the basis for my subsequent work in detection, tracking, and navigation.

SEDA was initially published in the proceedings of the 2021 IEEE/AIAA 40th Digital Avionics Systems Conference [9] and a provisional patent application was subsequently submitted for the architecture in 2022. That provisional patent was awarded, and as of the time of writing it is now in the process of being converted to a non-provisional US patent. I then worked with several colleagues to successfully adapt SEDA for use in a ML-based data fusion and tracking algorithm developed under the Air Force Research Laboratory Contract FA8750-21-C-0105 and for use in computer vision applications in 2023 [10].

1.1.2 Counterfactual Drift Correction

Next, a method was developed to mitigate drift errors in navigation and tracking systems caused by compounding measurement errors. This was achieved using a combination of causality-aware ML techniques, a multi-objective genetic algorithm (MOGA), and extended Kalman filtering (EKF). This approach framed the issue of drift correction as a counterfactual evaluation problem where multiple alternative paths were proposed and evaluated by the MOGA, whose objective functions were designed around historic motion data and the first principles equations used to model object motion. Testing of the proposed method was conducted on data simulated in Matlab using a constant acceleration model with additive white Gaussian noise and drift. Its performance was evaluated and compared with the EKF and spline interpolation methods.

An early implementation of this method was published in the proceedings of the 2022 IEEE/AIAA 41st Digital Avionics Systems Conference [11]. A considerable expansion of that work was then presented at a special session of the 2023 IEEE/ION Position, Location and Navigation Symposium and published in it proceedings [12]. A continuation of this research areas has also been funded by the Air Force Office of Scientific Research (AFOSR) under award number F4FGA03158J001.

1.1.3 Meta-Cognitive Radar Detection

A meta-cognitive method was then developed for adaptive radar detection across different clutter distributions. It uses an established adaptive detection algorithm derived for Gaussian distributed clutter augmented with a ML system trained to identify the distribution of clutter in a sample under test (SUT) and dynamically adjust the detection threshold to maintain a constant rate of false alarms. That method begins by analyzing the target clutter distributions and identifying the statistically distinct regions within them. This novel framing of the problem space minimizes confusion in the ML learning process while still providing sufficient information for key features to be identified by the model. A multi-layer ML system is then designed and trained. It consists of multiple lower-level regression neural networks each trained to set the threshold for a Gaussian generalized likelihood ratio test (GLRT) detector [13] in a given clutter distribution region. They are orchestrated by a higher-level classification model trained to recognize the distribution of clutter from support samples and select the appropriate threshold setting model.

The proposed meta-cognitive architecture, coupled with the process of identifying distribution regions with unique behaviors simplifies the solution space that each ML model needs to learn. This minimizes the confusion in each individual model and ensures that the integrated system is still able to maintain state-of-the-art receiver operating characteristics. It also enables the use of smaller, faster ML models, minimizing the computational overhead associated with integrating ML and simplifying the troubleshooting and retraining processes.

This work was published in 2024 in the IEEE Transactions on Aerospace and Electronic Systems as part of the Special Section on Deep Learning for Radar Applications [14]. The continuation of this work has also been funded by AFOSR under award number F4FGA03158J001.

1.1.4 Covariance Estimation

Finally, an approach for using generative ML to approximate inverse clutter covariance matrices was developed for use in adaptive detection where limited sample support is available. Modern ML image-to-image generation concepts use an encoder-decoder structure. In this structure the encoder identifies underlying feature information contained within the input image and the decoder uses those features to construct an image related to those features. Drawing from this concept, K support samples of length N are simulated in a radar scenario and arranged into a $K \times N$ matrix. They are then provided as an input to a U-Net structure trained to extract covariance information as features and construct an inverse covariance matrix that could be used to decorrelate the data. The use of a symmetric U-Net structure allows for variable input data sizes. The model is trained both as part of a GAN system and in a supervised manner with two custom loss functions designed to teach the model to produce accurate and invertible inverse covariance matrices. The proposed method has been integrated with a Gaussian GLRT and demonstrated improved performance over traditional estimation methods while using half as much support data.

The work on the GAN structure and initial integration and testing with the GLRT were presented at the 2024 IEEE Radar Conference and published as part of its proceedings [15]. Another publication on the custom loss functions and testing on high-fidelity simulation data generated with RFView is currently in work.

Chapter 2

Machine Learning Concepts

2.1 Introduction

The focus of this dissertation is to explore ways of leveraging ML concepts designed with a first principles understanding of the problem space to enhance radar system capabilities. ML is a broad term that encompasses a wide range of tools. Ultimately, it refers to any computer system that can update its own performance by ingesting data, taking some action, and receiving feedback. While it has existed for roughly 70 years, the field of ML has seen a huge surge in both research investment and applications within the last decade. Its current popularity is largely due to advances in computing hardware that compensate for the high computational costs associated with most ML approaches. ML uses probability theory to enable the creation of approximated models for systems based on sets of input data paired with some method for evaluating the quality of the model's output. Given an infinite amount of training data and infinite training time, a perfect model would theoretically be achieved. Since neither is possible, ML models are approximations with imperfect behavior. However, these models can be trained to approximate the target system closely enough to predict the outcome for a given input with a high probability. As a result, ML can be extremely useful for modeling highly complex or poorly understood systems for which developing a closed form expression is impractical. This makes ML extremely useful for predicting or controlling the way that complex systems behave. That usefulness comes at the cost of high complexity, difficult design, and high costs for both computation and data. The potential ML provides, paired with the associated costs, makes it very important to understand when and where to apply ML as opposed to simply developing a mathematical expression.

ML takes many different forms, each of which may have drastically different levels of complexity. At a base level, ML systems can be grouped into four broad categories based on how they are trained: supervised, semi-supervised, unsupervised, and reinforcement. Supervised learning is the most basic form and uses only known input data paired with know output data for training the model. For the remainder of this dissertation, the known input data used for training models will be referred to as training data, and the paired output data used in training will be referred to as either label data or simply labels. During training, supervised algorithms take training data as an input, produce an output, compare that output with the label data, and update their parameters based on how close their output was to the label. This form of ML can arguably produce the most accurate or useful models given sufficient training and label data. However, model quality is highly dependent on the quantity and quality of available training data, making it infeasible for many applications.

Semi-supervised learners train using a combination of labeled training data and unlabeled training data. These models are first trained using a relatively small set of labeled training data. The trained models are then used to produce training labels for the unlabeled dataset. The labels produced by the model with high confidence are added into the labeled training data and used to retrain the initial model. This process is performed iteratively until a sufficiently large portion of the originally unlabeled training data has been labeled with high confidence. This form of ML essentially functions in much the same way as supervised learning but reduces the amount of training data required. The fidelity of the trained models will also be lower than those produced by supervised learning.

Unsupervised learning uses exclusively unlabeled data for training. These algorithms evaluate unlabeled datasets to identify underlying patterns or features within them. This process generally involves finding ways of grouping the unlabeled data and using some set of metrics to evaluate the quality of the groups that are formed. These metrics include things like the size of each grouping, the distance between group members within the solution space, the distance or overlap between groups within the solution space, etc. Unsupervised learning is generally used to identify patterns within data that are highly complex or not well understood. It is rarely applied on its own and is generally used to inform data analysis or the design of another model.

Similar to unsupervised learning, reinforcement learning (RL) generally doesn't rely on labeled training data. Rather, RL systems are designed to take direct actions in an environment and are trained iteratively within that environment. The training process consists of providing the RL agent with an observation of its environment, allowing it to perform an action, taking another observation of the environment, and rewarding or penalizing the agent based on how its action impacted the environment. This form of ML is effective for complex problem spaces where it may not be possible to develop specific labels for training data. However, it can be very difficult to design effective reward/penalty functions.

For the remainder of this chapter, I will provide an overview of the key ML techniques relevant to the work conducted in this dissertation. Only supervised learning is used in this work, so the discussion will be limited to that form of ML.

2.2 Supervised Learning

Supervised learning in ML refers to a form of learning in which models are trained using curated datasets in which each entry possesses a corresponding label containing the desired model output. Many forms of ML fall into the category of supervised learning models. Simpler forms of supervised ML, like the random forest, are easy to train and require less computational resources, but are unable to learn the kinds of challenging patterns that more complex and data hungry methods, like deep neural networks, can handle. Here I will provide an overview of supervised ML techniques.

2.2.1 Decision Trees and Random Forests

One of the oldest and most basic supervised learning algorithms is the decision tree. Decision trees are designed to produce a prediction for a variable (either classification or regression) based on a set of simple decision rules. They are hierarchically structured, often taking a from similar to the one depicted in Figure 2.1. This structure consists of tiered nodes that split the possible decision space into branches. A branch coming out of a node from one layer will feed into a node in the next layer of the tree. These nodes are typically divided into three categories: root, intermediate, and leaf. Root nodes divide the decision space but have no incoming branches. They can therefore be thought of as origin nodes. Intermediate nodes have incoming branches and split the decision space. Leaf nodes have incoming branches but do not split the decision space. They are terminal nodes that result in an output decision from the tree. Decision trees are formed during training by recursively splitting an input training dataset based on class attributes. Splitting continues until some predefined stopping criteria is met.

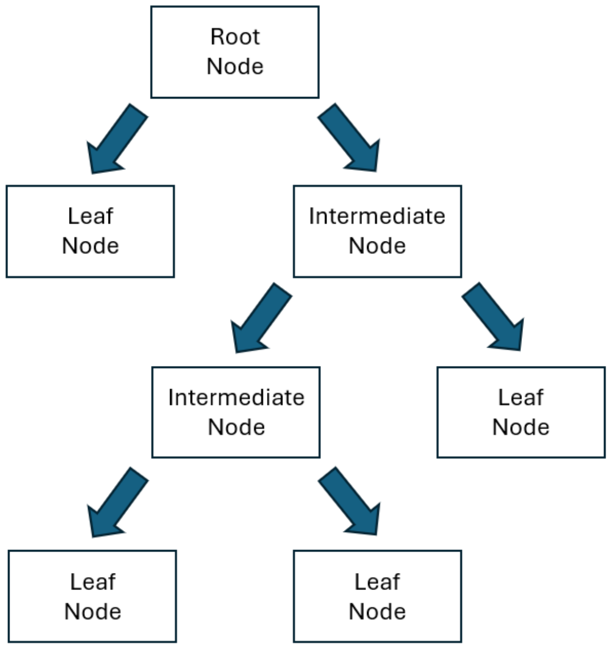


Figure 2.1: Depiction of a simple decision tree structure.

The primary method of determining the optimal split point for a decision tree is rooted in information theory, using the concept of information gain (or entropy reduction). In information theory, entropy is defined as the level of uncertainty or randomness in a system or set of data. It is quantified using the following equation:

$$H(S) = \sum_{c \in C} p(c) \log_2 \left(\frac{1}{p(c)} \right) \tag{2.1}$$

where H denotes the entropy, S is a set of data points, C denotes all classes in S, c denotes one class in C, and $p(c)$ is the probability of a data point in S belonging to class c. The values of entropy in Equation (2.1) can lie between 0 and 1. An entropy value of 1 occurs when every data point in S has an equal chance of belonging to each class in C, while an entropy of 0 occurs when each data point in S can fall into exactly 1 class in C.

When data is being classified, the best way to determine how to split the data is to try and minimize the entropy in the resulting system. A reduction of entropy like this is referred to as entropy gain and can be calculated as:

$$I(S, a) = H(S) - \sum_{v \in V(a)} \frac{|S_V|}{|S|} H(S_V) \tag{2.2}$$

where I is the level of information gain, a is the class attribute used to split S, v denotes a possible value in S satisfying a, V denotes all possible values in S that satisfy a, and S_V is the subset of S satisfying a. Splits in the decision tree occur where the class values maximize the information gain.

Alternatively, information gain in decision tree training algorithms may be formulated in terms of minimizing its Gini index. The Gini index refers to the level of impurity in a given node. In information theory, impurity refers to the likelihood that a randomly selected entry from the input dataset will result in an incorrect output at a given node. The Gini index is calculated as:

$$GI(S) = 1 - \sum_{i=1}^{k} p_i^2 \tag{2.3}$$

where GI denotes the Gini index of set S, i is an index referring to a particular possible node output, k denotes the total number of possible node outputs, and p_i is the probability of a random entry in S resulting in node output i. A perfectly “pure” node is one with a Gini index of 0, meaning that all data points in S belong to a single class, while an "impure" node has a uniformly distributed probability of each possible output.

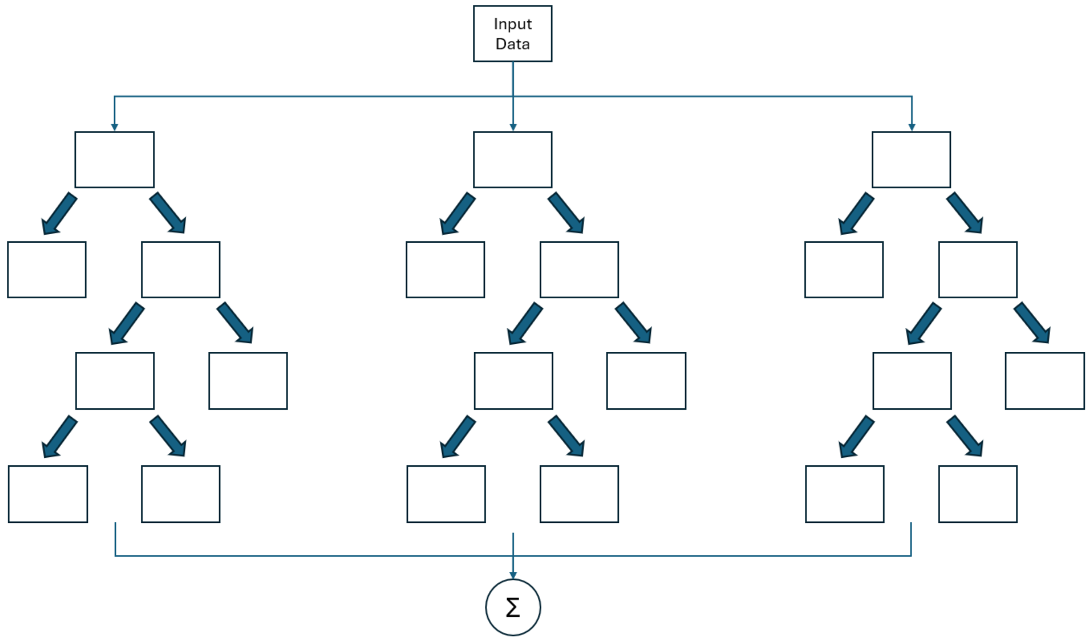


Figure 2.2: Depiction of a simple random forest.

Random forest algorithms consist of multiple decision trees, each of which is provided the same input data entry, and whose outputs are then aggregated. When forming the random forest, each decision tree is trained independently on a randomly selected subset of the training data. The trees are then combined, and their outputs are combined using averaging for regression or voting for classification. An example of this can be seen in Figure 2.2. Random forests are more computationally costly than decision trees, but they also tend to generalize better on complex datasets.

2.2.2 Neural Networks

In more recent years, neural networks have become one of the most widely used forms of ML algorithms. While they have existed in some form for nearly 70 years, they are one of the more complicated forms of learning algorithms and have recently seen some significant advancements. The basic building block of the earliest neural networks was the perceptron. Perceptrons are loosely inspired by neurons in the brain and are used to make predictions based on a weighted sum of a set of input features. They take as an input the set of features $\mathbf{x} = 1, x_1, x_2, ..., x_n$ of length n. The input features are then modified by a set of weights $\mathbf{w} = w_0, w_1, w_2, ..., w_n$ and summed. Finally, an activation function (which will be denoted $f_a(\bullet)$ is applied to the weighted sum of the input features to generate an output prediction:

$$y = f_a\left(\sum(w_0 + x_1w_1 + x_2w_2 + ... + +x_nw_n)\right) \tag{2.4}$$

Activation functions can take many forms, and in neural networks they are commonly used to introduce non-linearity. I will discuss different activation functions in more detail shortly, but for the original perceptron structure f_a was a simple step function. The structure of the perceptron is shown in Figure 2.3.

It is clear from Figure 2.3 and Equation (2.4) that perceptrons were structured as linear equations. The key element of the perceptron that made it revolutionary was that the values of $\mathbf{w}$ used to weight the inputs were trained, rather than being defined mathematically. While this may not initially sound impressive, it is important to note that this allowed the perceptron to approximate a multivariate mathematical model for systems that didn't necessarily have closed form expressions. This was an incredibly

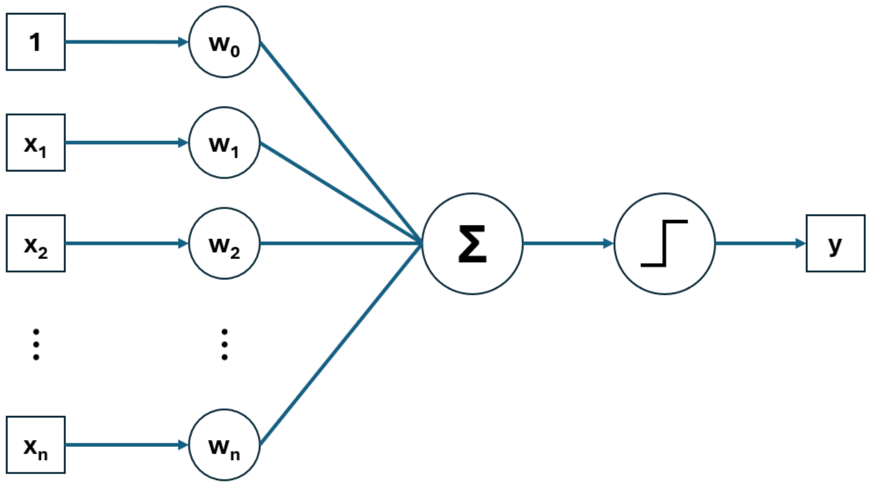


Figure 2.3: The basic structure of the perceptron.

powerful capability that was unique in the world of computing at the time. They were trained iteratively using training data and labels using what is known as a loss function. Loss functions in neural networks now take many forms and have become a major area of research. I will discuss them in more detail shortly, but I will note here that they are used to calculate some distance measure between the output of a learning system and some target output. That distance is then used to update the weights of the learning system.

In the early days of ML, the perceptron proved to be a powerful tool for autonomously splitting solution spaces. However, a single perceptron could not be applied to non-linear problems. This was found by Minsky and Papert in [16] when they found that the single-layer perceptron could not be used to implement the XOR gate. Fortunately, this limitation is easily overcome by stacking multiple layers of perceptrons together [17] to establish non-linear relationships between the input and output. This results in the multi-layer perceptron (MLP) network structure shown in Figure 2.4.

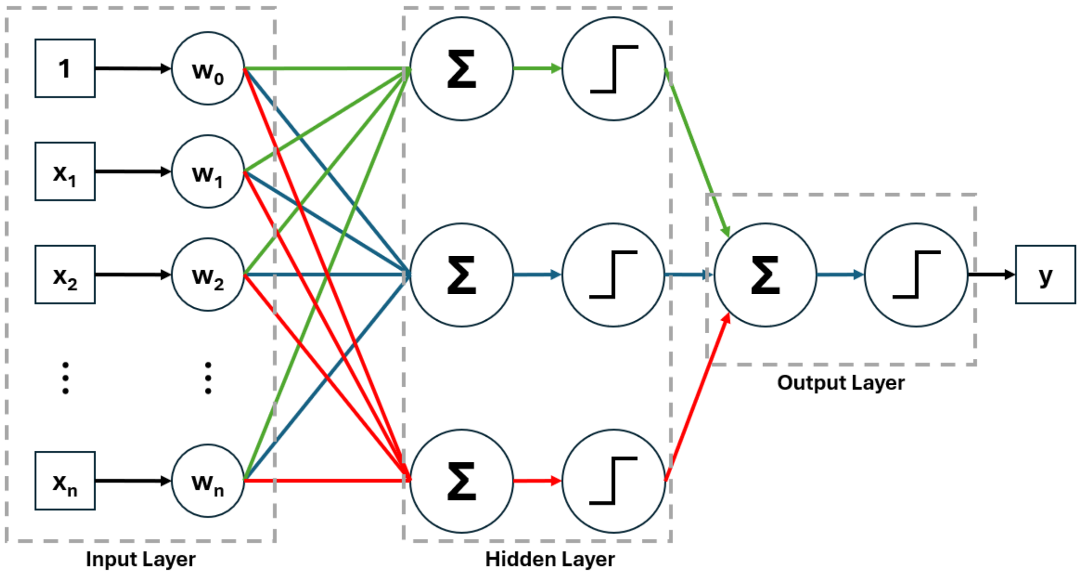


Figure 2.4: The basic structure of the multi-layer perceptron.

The MLP structure consists of an input layer, some number of intermediate "hidden" layers containing a number of stacked perceptrons (or neurons), and an output layer. Each hidden layer consists of a set of weights, a summation, and an activation function. The output of each neuron of a given hidden layer is passed as an input to each neuron of the next layer. Due to the fact that each neuron in one layer is connected to every neuron in the previous and subsequent layer, these hidden layers are commonly referred to as fully connected layers.

As with the single layer perceptron, the MLP learns iteratively by taking example inputs, producing outputs, using data labels to calculate the loss, and then updating the weights. However, due to the increased complexity weight updates in the MLP are made using a process known as backpropagation [17]. Back-propagation is a method of updating parameters in a network that tries to map the contribution of each parameter to the network's final prediction error and update them accordingly. This is accomplished by calculating the loss gradients at each neuron and then updating the weights to minimize that loss according to:

$$w_{i,j} = w_{i,j} - \epsilon \frac{\delta L_j}{\delta w_{i,j}} \tag{2.5}$$

where $w_{i,j}$ is the weight for input i to network layer j, ϵ is the learning rate, and L_j is the loss at layer j. That process starts at the output layer, and the portion of the loss gradient associated with each input to that layer is propagated back to the neuron from which it originated. That loss is then used in that neuron's loss gradient calculation, continuing until all weights have been updated (hence the name backpropagation). The MLP is arguably one of the most common neural network architectures used even today, and it serves as the basis for what are now known as the family of feed-forward neural networks (FFNN). They can have either singular outputs, as shown in Figure 2.4 or multiple outputs as in Figure 2.5.

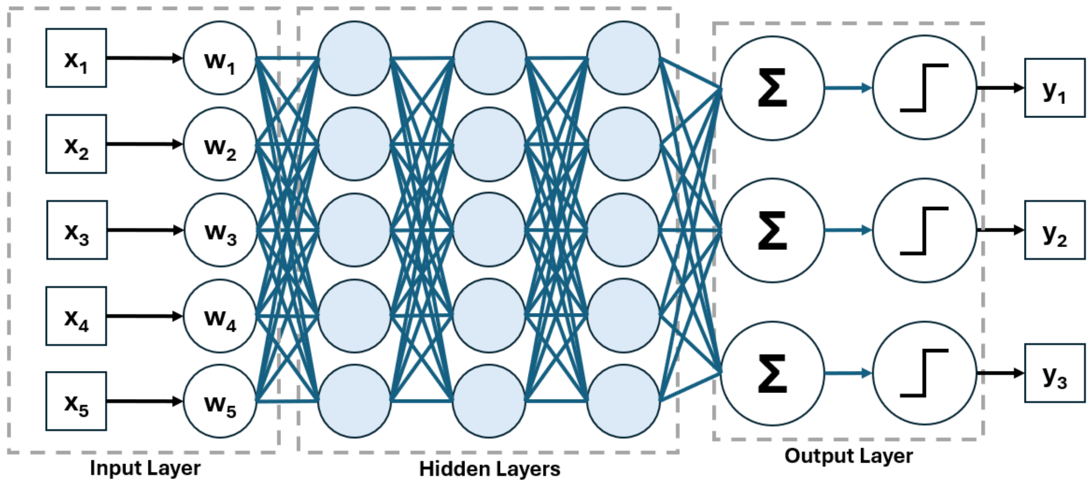


Figure 2.5: The basic structure of a multi-output fully connected ANN.

FFNNs and MLPs are extremely useful in many applications, as they are simple, fast, and allow for the approximation of complex relationships. However, there are a number of limitations with these types of neural networks, two that I will focus on now. The first is that they struggle to learn sequential or temporal patterns, and the second is that they struggle to learn spatial patterns in higher dimensional data. The source of the first limitation is simple, FFNNs have no memory or mechanism for carrying information between inputs. Since these models must evaluate each input independently of any other input, they cannot learn patterns that span them. However, there are many applications for which such patterns are essential. A good example of this is modeling motion patterns of an object based on its position when trying to track that object and predict where it will move next. This has led to the development of another category of neural network architectures referred to as recurrent neural networks (RNNs).

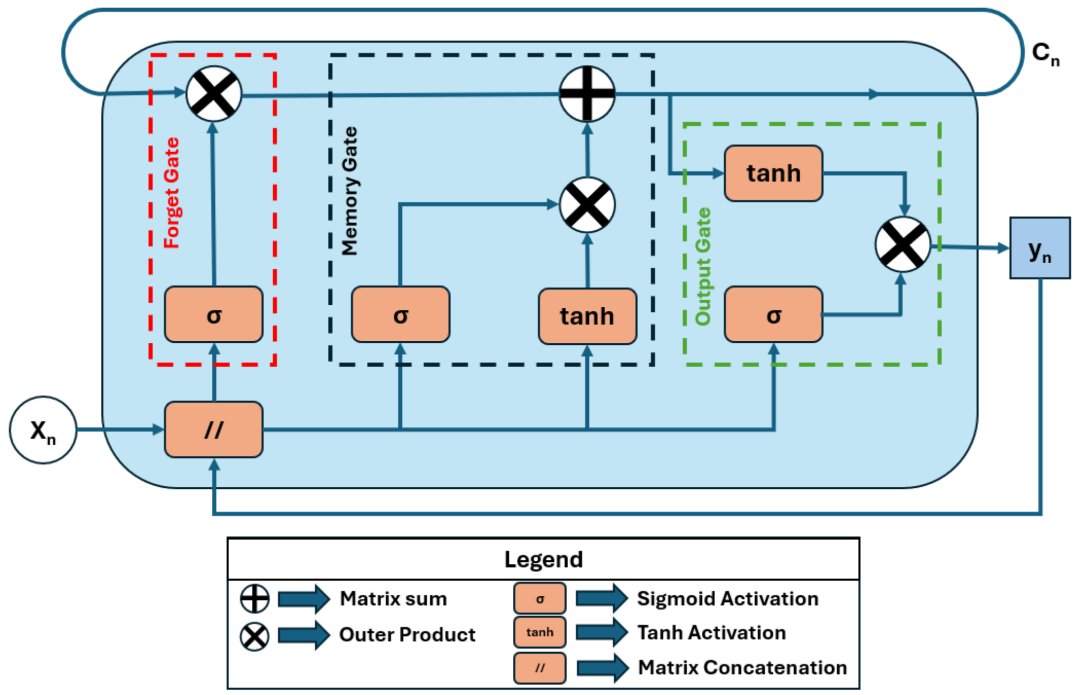


Figure 2.6: The basic structure of the LSTM node.

RNNs are so named because they have a feedback connection from the output back to the input, enabling them to retain information between inputs. This, of course, allows them to learn sequential patterns. Today, the most widely used RNN structure is called the long short-term memory (LSTM) network. The LSTM structure, which is shown in Figure 2.6, was designed to allow it to learn both long and short term patterns in data [18]. This is achieved through the use of two feedback connections; the hidden state and the cell state. The purpose of the hidden state is to maintain short term dependencies and ultimately produces the layer output. The cell state maintains long term dependencies. For each input to the LSTM, it identifies patterns in the data and then uses two specialized structures called gates to modify the cell state by adding and removing information. This allows the LSTM to "remember" important new information and "forget" information that may no longer be relevant. As shown in Figure 2.6, the Forget Gate is responsible for removing information from the cell state:

$$\mathbf{f}_n = \sigma(\mathbf{W}_f[\mathbf{y}_{n-1}, \mathbf{x}_n] + \mathbf{b}_f) \tag{2.6}$$

where $\sigma(\bullet)$ is the sigmoid function, $\mathbf{W}$ is a learnable weight matrix, $\mathbf{b}$ is a learnable bias vector, $\mathbf{y}$ is the hidden state, and $\mathbf{x}$ is the input. The Memory Gate is responsible for adding new information to the cell state:

$$\mathbf{m}_n = \mathbf{i}_n \otimes \tilde{\mathbf{c}}_n \tag{2.7}$$

where:

$$\mathbf{i}_n = \sigma(\mathbf{W}_i[\mathbf{y}_{n-1}, \mathbf{x}_n] + \mathbf{b}_i) \tag{2.8}$$

$$\tilde{\mathbf{c}}_n = tanh(\mathbf{W}_c[\mathbf{y}_{n-1}, \mathbf{x}_n] + \mathbf{b}_c) \tag{2.9}$$

where $tanh(\bullet)$ is the hyperbolic tangent function. Combining Equations (2.6) and

(2.7), the cell state is updated according to:

$$\mathbf{c}_n = \mathbf{f}_n \otimes \mathbf{c}_{n-1} + \mathbf{m}_n \tag{2.10}$$

A final gate is used to produce the output, which consists of a modified version of the cell state, filtered by the current input and previous hidden state:

$$\mathbf{y}_n = \sigma(\mathbf{W}_o[\mathbf{y}_{n-1}, \mathbf{x}_n] + \mathbf{b}_o) \otimes tanh(\mathbf{c}_t) \tag{2.11}$$

The LSTM features prominently in the work presented in Chapters 4 and 5.

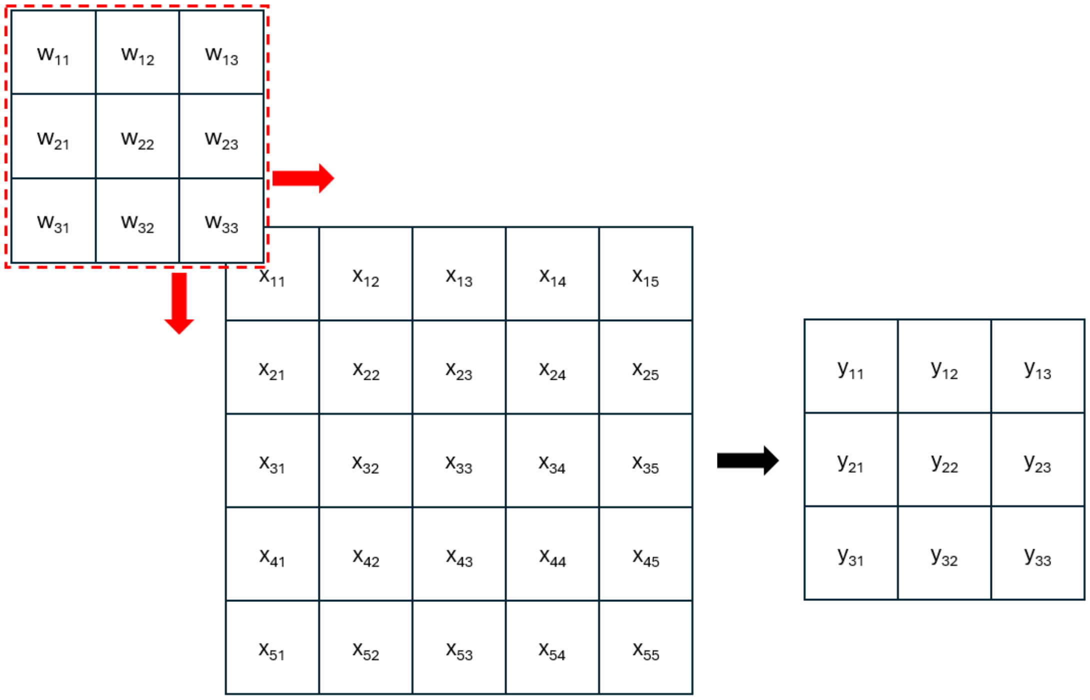
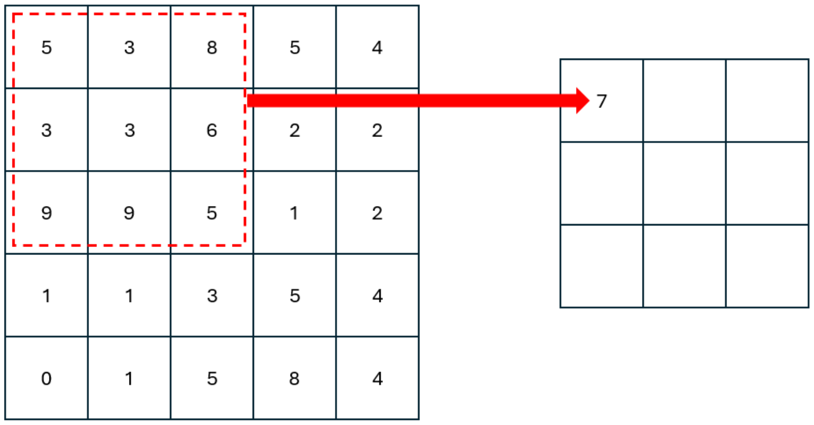
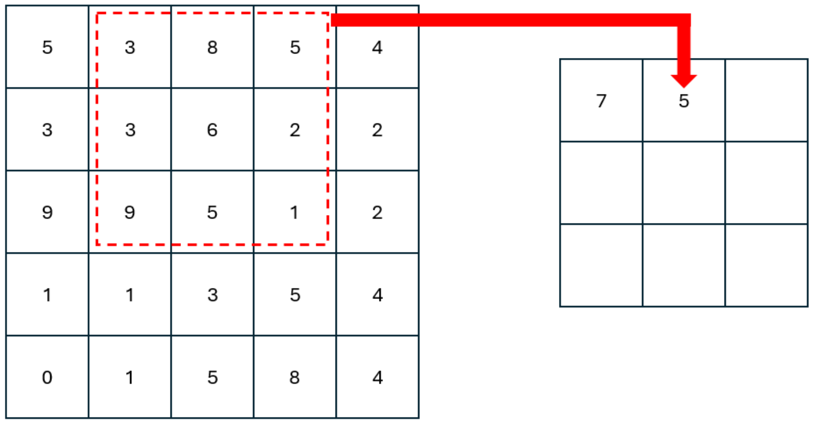


Figure 2.7: The basic structure of the Convolutional layer.

Both MLPs and RNNs struggle to learn spatial patterns in higher dimensional data. This is due in part to the fact that neither has a structure for learning such patters. Additionally, the fully connected nature of the MLP generally interferes with learning spatial patterns. However, there are many applications for ML that rely on learning spatial patterns in higher-dimensional data, like recognizing or predicting actions from image and video data [19]. This has led to the creation of the convolutional neural network (CNN). The CNN structure generally consists of some combination



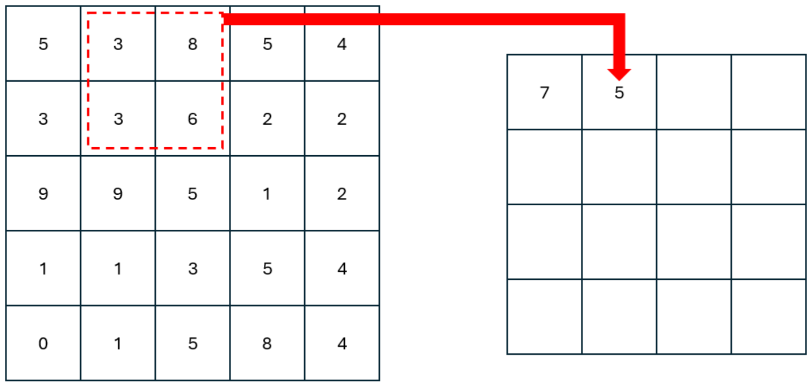
(a) Initial position of the convolutional kernel.



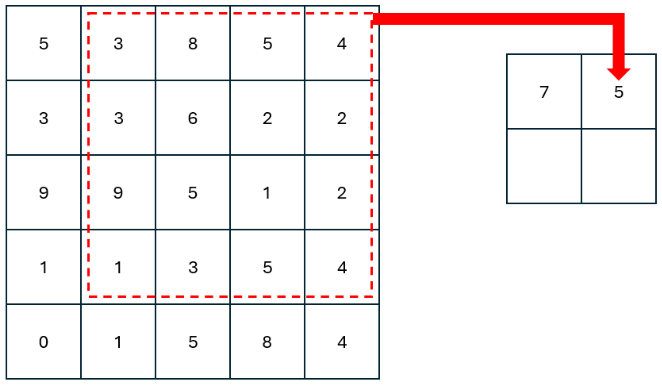
(b) Second position of the convolutional kernel.

Figure 2.8: Function of the convolutional layer.

of convolutional layers, pooling layers, and fully connected layers. The convolutional layer is based on classical image processing techniques that involved convolving a filter across an image to suppress or accentuate spatial features of the image. Whereas the filters in classical image processing are fixed, the filters (or kernels) in a convolutional layer are learnable weights. These layers are designed to identify important spatial features in the data. The structure of the convolutional layer is shown in Figure 2.7.



(a) Convolutional layer with kernel size of $2x2$.



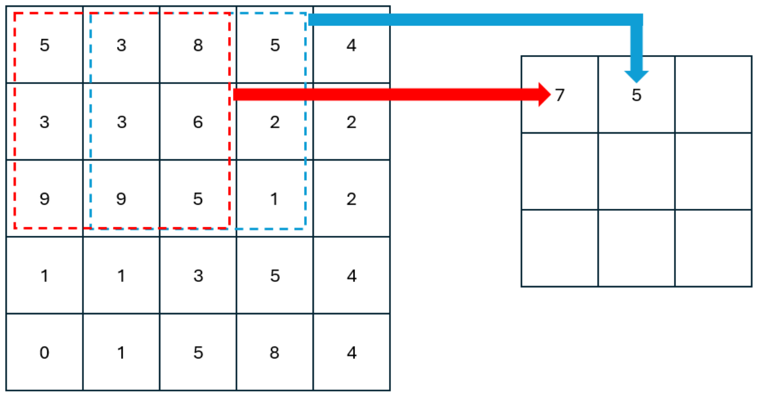
(b) Convolutional layer with kernel size of $4x4$.

Figure 2.9: Impact of kernel size on convolutional layer.

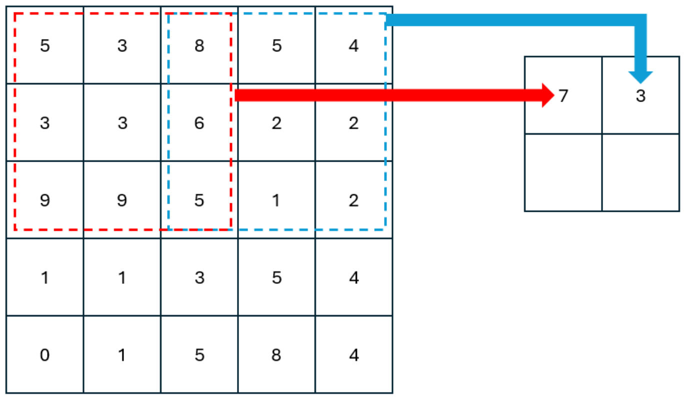
The convolutional layer of a CNN takes as an input an array of some dimensionality. For the purposes of this discussion, I will focus on two dimensional arrays. The output of these layers would then be another two dimensional array whose size is dependent on three parameters: 1) kernel size, 2) kernel stride, and 3) padding. Each value in the output matrix is defined as the weighted sum of a corresponding sub-region of the input. Using Figure 2.7 as an example, the kernel $\mathbf{W}$ would first be applied to the top left sub-region of the input matrix $\mathbf{X}$ to produce the first value of the output matrix $\mathbf{Y}$:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + ... + w_{32}x_{32} + w_{33}x_{33} \tag{2.12}$$

The kernel is then shifted to a new sub-region to calculate the next value of $\mathbf{Y}$, as shown in Figure 2.8.



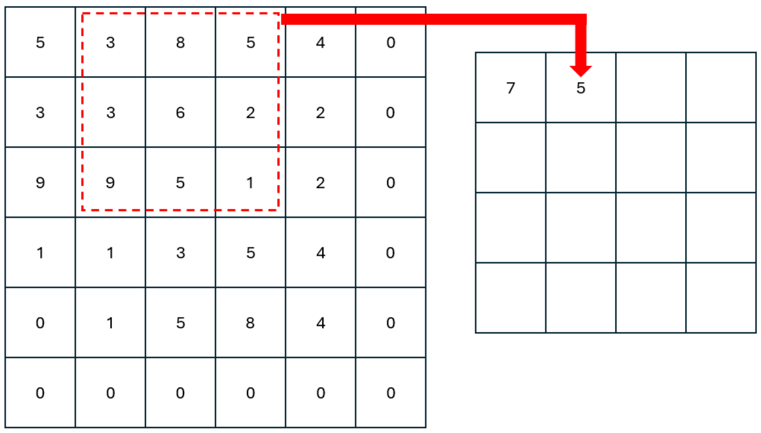
(a) Kernel movement with stride of 1.



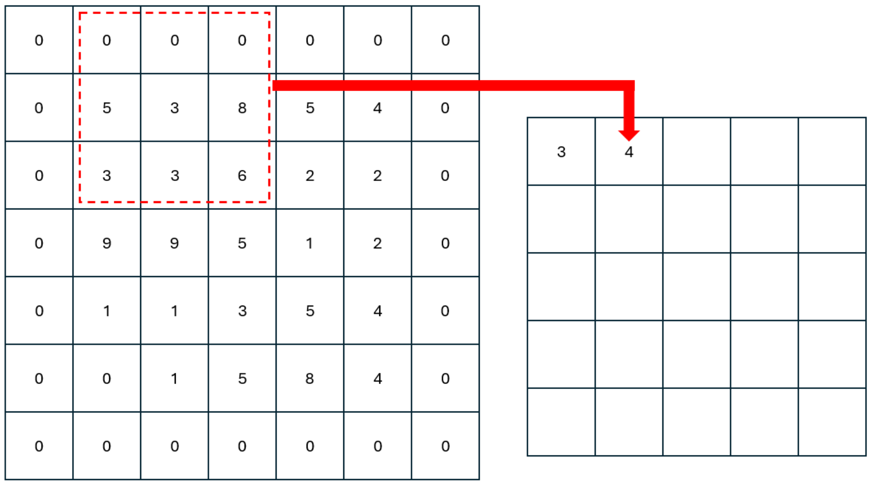
(b) Kernel movement with stride of 2.

Figure 2.10: Impact of kernel stride on convolutional layer.

The kernel size, stride, and padding can each be tuned to alter the behaviors of a CNN. Using a larger kernel causes the layer to learn broader patterns, while smaller kernels learn more localized patterns. Additionally, increasing the kernel size reduces the size of the layer output and can reduce the likelihood of the model over-fitting on particular patterns. The effect of kernel size is depicted in Figure 2.9.



(a) Kernel movement with one-sided padding of 1.



(b) Kernel movement with full padding of 1.

Figure 2.11: Impact of padding on convolutional layer.

Kernel stride refers to the number of positions the kernel is shifted across the input matrix for computing the next output matrix value. Increasing the kernel size reduces the overlap between features in the output and reduces the size of the output array. This reduces the computation cost of the network and can help reduce over-fitting, but it also throws away considerable amounts of data and may interfere with the learning process. The effects of kernel stride are depicted in Figure 2.10.

Padding refers to increasing the size of the input matrix prior to convolving the kernel across it. It is used to increase the size of the output matrix and reduce the loss of information that occurs during convolution. This can help to improve the accuracy of a trained CNN, but it also increases the computational cost of the network and can increase the odds of the model over-fitting. Several examples of padding are shown in Figure 2.11.

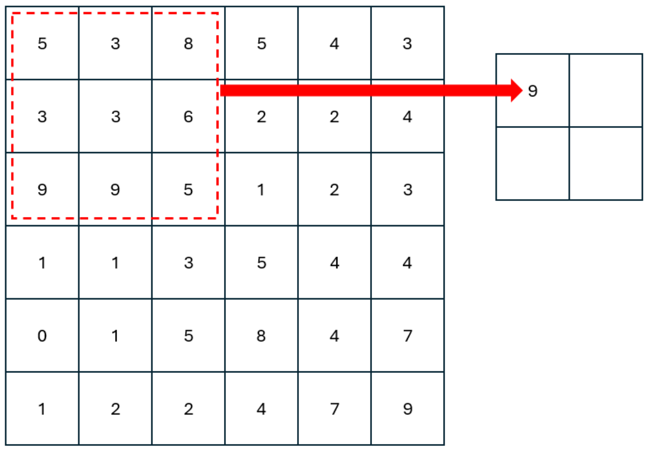
Convolutional layers excel at identifying spatial features in data. However, they are known to be very sensitive to rotational and translational variations in those spatial patterns, which can severely limit their ability to generalize. Pooling layers were developed to address that limitation. They consist of a $N \times N$ kernel with a stride of N. Unlike convolutional layers, pooling layer kernels do not have learnable weights, but rather apply a constant function to extract some piece of information from a sub-region of the input array and discard all the rest. Intentionally discarding information in this way allows CNNs to generalize and reduces the computational cost of the model, but can risk discarding critical information. There are three commonly used types of pooling layers: 1) maximum (max) pooling, 2) minimum (min) pooling, and 3) average pooling.

$$MaxPool(\bar{\mathbf{X}}) = Max(\bar{\mathbf{X}}) \tag{2.13}$$

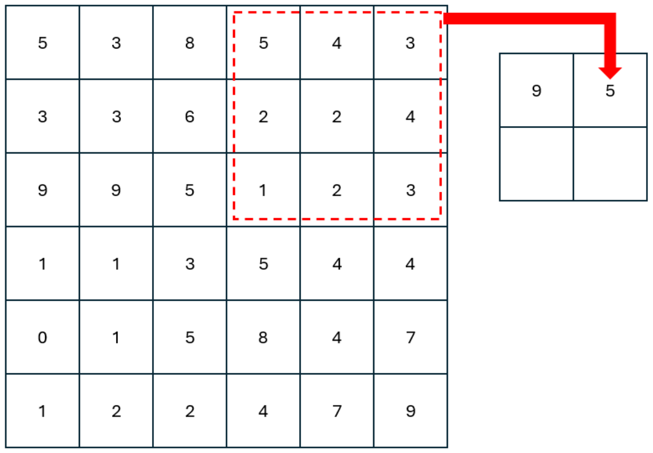
$$MinPool(\bar{\mathbf{X}}) = Min(\bar{\mathbf{X}}) \tag{2.14}$$

$$AvePool(\bar{\mathbf{X}}) = Mean(\bar{\mathbf{X}}) \tag{2.15}$$

where $Max(\bullet)$ is a function returning the maximum value of an array, $Min(\bullet)$ is a function returning the minimum value of an array, $Mean(\bullet)$ is a function returning the mean value of an array, and $\bar{\mathbf{X}}$ is the sub-array of $\mathbf{X}$ under evaluation. An example of the pooling process is depicted in Figure 2.12.



(a) Initial position of a maxpooling layer.



(b) Second position of a maxpooling layer.

Figure 2.12: Basic function of the maxpooling layer.

As noted before, CNNs generally consist of a combination of convolutional layers used to extract features from data, pooling layers to improve generalizability and speed, and fully-connected layers to produce a useful output from the features identified by the convolutional layers. An example of the full CNN structure is shown in

Figure 2.13. Considerable design work must be done when developing CNNs architectures to balance the layer structure and the various parameters in each layer. CNNs are used considerably in the work presented in Chapters 4 and 7.

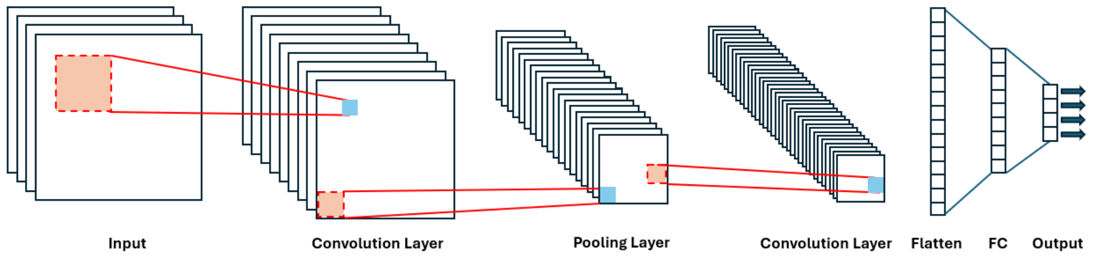


Figure 2.13: Structure of a simple CNN.

The neural network structures that have been discussed up to this point can be used to process a variety of different input data types. However, they each effectively perform the same operations; extract features from the input data and produce an output of fixed size (either classification or regression) from those features. This has historically proved to be both useful and powerful. However, as people began trying to apply neural networks to increasingly complex problems, it quickly became apparent that it was too restrictive for many applications. One of the key areas where this proved to be the case was natural language processing (NLP) [20]. In order to train a network to translate or produce human speech, a new structure needed to be developed that allowed for the generation of variable length output sequences from variable length input sequences.

In 2014, Sutskever *et al.* developed the first sequence-to-sequence learning structure to support NLP. The structure that they introduced consisted of two models working in tandem. The first model is referred to as the encoder model. It processes an input sequence to extract latent features and encode them into a context vector. The second model, called the decoder, is trained to interpret the context vectors produced by the encoder and produce an output sequence based on them. Since the context vectors are encoded with latent features which are shared between the encoder and decoder, it effectively defines a shared latent space between the encoder and decoder models. This structure is depicted in Figure 2.14.

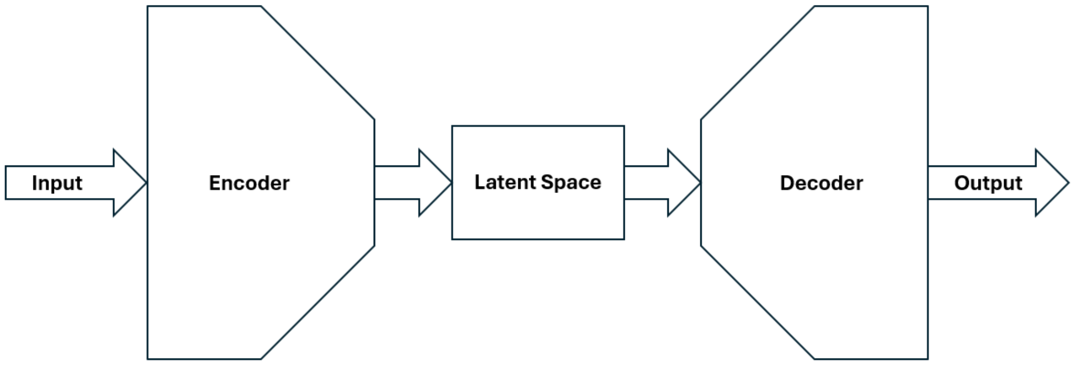


Figure 2.14: Structure of a simple Encoder-Decoder network.

The encoder-decoder structure and the concept of the latent space has served as the basis for a major advancement in ML: generative models. While this structure was originally developed for NLP [20], it has now been broadly applied to other types of input-output pairings. Notably, this structure is used for image and audio signal generation [21, 22]. I will discuss generative ML in more detail later in this chapter and use these concepts for generating adaptive filters in Chapter 7.

While this discussion has only touched on a portion of the available types of neural networks structures, it is clear that they can be highly complex and that developing a functional architecture requires significant design work. Considerations must be made for the types of input data to the model, the desired output, as well as the number, type, and configuration of the layers used. When attempting to integrate neural networks into complex systems or use them to augment existing algorithms, it is crucial to understand your problem space and allow that understanding to drive your design decisions. Well designed neural networks can add considerable flexibility to systems, while poorly designed ones will destroy performance and drive up computational costs. I will now focus on two additional design elements of neural networks: activation functions and loss functions. Bear in mind, however, that there are a large number of additional neural network design and training parameters that will not be discussed in this dissertation.

2.2.3 Activation Functions

Activation functions, sometimes referred to as transfer functions, are a key component of neural networks that drive behavior and performance. Table 2.1 lists some of the most commonly used activation functions. The activation function produces the output signal for a given neuron based on its input. They also allow for the modeling of non-linear relationships. There are two types of activation functions in most neural networks: hidden layer activation and final activation. The final activation function produces the output for the network itself, and it must be selected based on the kind of output we expect from the model (binary vs. continuous, single- vs multi-output, etc.). Hidden layer activation functions drive the types of relationships each hidden layer can learn.

Activation functions can vary considerably, and custom functions can also be designed and used. However, there are generally two practical restrictions placed on activation functions. The first is that they are monotonic to allow for disambiguous mappings between inputs and outputs. The second is that they are differentiable to allow for the calculation of error gradients during backpropagation. The nature of the mapping between input and output at each neuron/layer of a network will drive the kinds of relationships that it can learn to model, making the selection of activation

Activation Function	**Equation**	**Conditions**
Linear	$f_a(x) = x$	N/A
Sigmoid	$f_a(x) = \frac{1}{1+e^{-x}}$	N/A
Hyperbolic Tangent (tanh)	$f_a(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	N/A
Softplus	$f_a(x) = ln(1 + e^x)$	N/A
Softmax	$f_a(x) = \frac{e^x}{\sum_{i=1}^{N} e^{x_i}}$	N/A
Binary Step	$f_a(x) = \begin{cases} 1, & x \geq 0 \\ 0, & otherwise \end{cases}$	N/A
Rectified Linear Unit (ReLU)	$f_a(x) = \begin{cases} x, & x \geq 0 \\ 0, & otherwise \end{cases}$	N/A
Leaky ReLU (LeLU)	$f_a(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & otherwise \end{cases}$	$0 < \alpha < 1$
Exp. Linear Unit (ELU)	$f_a(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & otherwise \end{cases}$	$0 < \alpha < 1$

Table 2.1: Commonly Used Neural Network Activation Functions

functions a critical design decision.

2.2.4 Loss Functions

Loss functions, sometimes referred to as error functions, are used during model training to calculate the distance between the outputs from the model and the label set. Put another way, they are used to determine the quality of guesses being produced by the model. The loss value produced by this function is then used to update each of the trainable parameters. While they are only used during the training process, these functions drive the quality and speed of model training, making their selection one of the most critical design decisions for any neural network. Table 2.2 lists some of the most widely used loss functions. Two notes on the variety of loss functions should be made, however. The first is that there are many additional commonly used loss functions. The second is that it is extremely common to design custom loss functions.

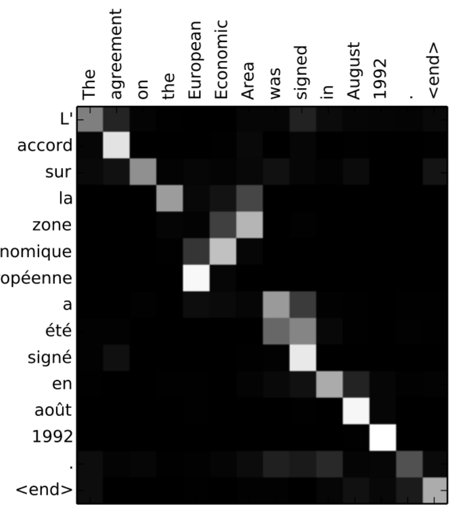
Loss Function	**Equation**
Mean Absolute	$L(x) = \frac{1}{N}\sum_{i=1}^{N} \lvert x_i - \hat{x}_i \rvert$
Mean Bias	$L(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)$
Mean Squared	$L(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$
Relative Absolute	$L(x) = \frac{1}{N}\sum_{i=1}^{N} \frac{\lvert x_i - \hat{x}_i \rvert}{\lvert x_i - \bar{x} \rvert}, \qquad \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
Relative Squared	$L(x) = \frac{1}{N}\sum_{i=1}^{N} \frac{(x_i - \hat{x}_i)^2}{(x_i - \bar{x})^2}, \quad \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
Root Mean Squared	$L(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$
Mean Squared Log	$L(x) = \frac{1}{N}\sum_{i=1}^{N}(log(x_i + 1) - log(\hat{x}_i + 1))^2$
Root MSLE	$L(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(log(x_i + 1) - log(\hat{x}_i + 1))^2}$
Log Cosh	$L(x) = \sum_{i=1}^{N} log(cosh(\hat{x}_i - x_i))$
Huber	$L(x) = \begin{cases} \alpha(x - \hat{x}) - \frac{\alpha^2}{2}, & \lvert x - \hat{x} \rvert \geq \alpha \\ \frac{1}{2}(x - \hat{x})^2, & otherwise \end{cases}$
Quantile	$L(x) = \begin{cases} \alpha(x - \hat{x}), & \lvert x - \hat{x} \rvert \leq \alpha \\ (1 - \alpha)(\hat{x} - x), & otherwise \end{cases}$

Table 2.2: Commonly Used Neural Network Loss Functions

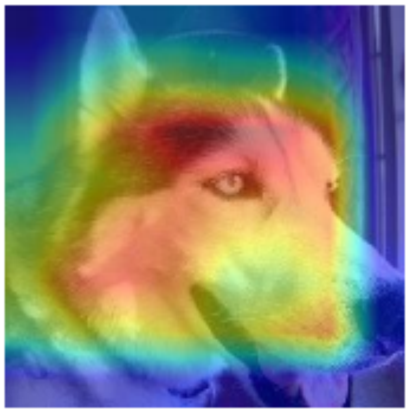
The purpose of a loss function seems simple on the surface, which may lead one to wonder why there would be such a variety of them. After all, the loss function is designed to measure the error between an estimation and a known value. However, neural networks are regularly trained to approximate complex systems, and the accuracy with which they do so determines their usability. These systems can often have highly variable behaviors within different regions of their solution space, and an ideal model or system should learn to approximate all of it. Poorly designed or selected loss functions can cause sub-regions of the solution space to dominate training, preventing models from generalizing. They can also cause large error values to occur that result in updates to the learnable parameters that throw the model into an unstable training region, preventing the model convergence. Additionally, knowledge of the problem space for which a model is being trained can be integrated into the loss function to encourage different behaviors for different sub-regions. Ultimately, the loss function is one of the primary drivers behind both what and how well a model learns to perform. I use custom loss functions in Chapter 7 to train a model to generate adaptive filters.

2.3 Machine Learning Attention and the Transformer

The ability of large scale neural networks to learn highly complex patterns makes them powerful tools in automation. However, it has also given rise to a limitation colloquially known as the "black box" problem. This term refers to the fact that traditional neural networks receive an input and produce an output without the model's users or designers having a full understanding of how it arrives at said output. While it is technically possible to fully map out the behaviors of a deep neural network, the fact that model behavior is learned instead of designed, the high number of trainable parameters in large models, and the abstractness of the patterns that can be learned make this infeasible. For most uses of ML, the black box nature of the model is irrelevant as long as it has high accuracy. However, it restricts the ability for a system or user to identify why models behave the way they do. It also makes it extremely difficult to troubleshoot abnormal behaviors or encourage a model to engage in particular behaviors. In the early 2000's to mid 2010's, a growing number of ML applications required additional insights into model behaviors, and researchers began attempting to develop mechanisms for providing contextual information to adjust model behaviors for specific scenarios. In particular, researchers were making strides in natural language processing (NLP) for language translation. Additionally, the computer vision community was beginning to explore using deep convolutional models in critical systems and needed to provide visual visual explanations for model behaviors. This gave rise to the notion of attention in ML systems.



(a) Attention applied to natural language processing [23].



(b) Attention applied to computer vision explainability [24].

Figure 2.15: Examples of attention mechanism outputs.

The concept of modern ML attention is often attributed to Bahdanau, Cho, and Bengio in [24]. This work provides a method for quantifying the impact of particular portions of input information, and/or hidden state information in the case of RNNs, on the output of a model or layer. In [24], the authors developed an attention mechanism for use in an RNN-based Encoder-Decoder framework for sequence to sequence translation in NLP. One of the key limitations in NLP at the time was the inability for ML models to understand non-sequential contextual variations in the meaning of words in language. In this basic RNN Encoder-Decoder framework, an input sequence $\mathbf{x}$ is provided to an encoder to produce an encoder hidden state, h_t and an encoded vector, c:

$$h_t = f(x_t, h_{t-1}) \tag{2.16}$$

$$c = q(\mathbf{h}) \tag{2.17}$$

where $f(\bullet)$ and $q(\bullet)$ are nonlinear functions. The context vector is then passed to a decoder that uses this context vector and its previous predictions to predict the next value in the output sequence:

$$p(\mathbf{y}) = \prod_{t=1}^{T} g(y_(t-1), s_t, c) \tag{2.18}$$

where $g(\bullet)$ is a nonlinear function and s_t is the encoder hidden state.

The authors of [24] proposed changing Equation (2.18) so that the probability of each output is set by a unique context vector:

$$p(\mathbf{y}) = \prod_{t=1}^{T} g(y_(t-1), s_t, c_t) \tag{2.19}$$

where

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{2.20}$$

and each c_t is calculated as a weighted sum of the encoder hidden states:

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \tag{2.21}$$

Note that t is used to denote the output sequence position, and j is used to denote the input sequence position. The value of α_{tj} then reflects the impact of the encoder input at position j on the decoder output at position t. Put another way, it quantifies how much attention the model pays to input x_j when producing output y_t. In [24], this attention value was computed as:

$$\alpha_{tj} = softmax(e_{tj}) = \frac{\exp{(e_{tj})}}{\sum_{k=1}^{T} \exp{(e_{tk})}} \tag{2.22}$$

The value e_{tj} is referred to as the alignment score between input x_j and output y_t, and it is computed based on the encoder hidden state h_j and the decoder hidden state s_{t-1}. An alignment score can be calculated in various ways, but the two most common are the additive score (proposed by Bahdanau, Cho, and Bengio in [24]) and the dot product score (proposed by Luong, Pham, and Manning in [25]). The additive scoring method is calculated as:

$$e_{tj} = \mathbf{V}_a^T tanh(\mathbf{W}_a s_{t-1} + \mathbf{U}_a h_j) \tag{2.23}$$

where V_a, W_a, and U_a are weight matrices. The dot product scoring is calculated as:

$$e_{tj} = s_{t-1}^T h_j \tag{2.24}$$

These concepts were then adapted in the field of NLP to map the interactions between words on an input sequence within a single RNN structure [26, 27] (rather than mapping between an encoder and decoder structure). This is now commonly referred to as self-attention (or intra-attention) and is accomplished by replacing s_{t-1}

with h_i in Equations (2.23) and (2.24):

$$e_{tj} = \mathbf{V}_a^T tanh(\mathbf{W}_a h_i + \mathbf{U}_a h_j) \tag{2.25}$$

$$e_{tj} = h_i^T h_j \tag{2.26}$$

where h_j now represents the hidden state of the current input position, and h_i represents the hidden state(s) of some subset of previous input positions.

In 2017, Vaswani *et al.* proposed two advances to self-attention in [28], which have ultimately shaped the entire field of generative ML and driven much of the ML research conducted since. The first update they proposed was a generalized form of self attention that is computationally efficient and readily adaptable to new application spaces. This approach consists of using three linear layers with learnable parameters to transform inputs to the attention block into a query vector, Q, a key vector, K, and a value vector, V. These vectors are then used to calculate a scaled dot-product attention similar to the one obtained by plugging Equation (2.26) into Equation (2.22):

$$Attention(Q, K, V) = softmax\left(\frac{qK^T}{\sqrt{d_k}}\right)V \tag{2.27}$$

where d_k is the length of Q and V. Figure 2.16 depicts the structure of this generalized self-attention module. The query and key vectors are used to compute attention scores which are then weighted by the value vector before being summed. The structure is easily parallelizable and allows the linear layers used to generate Q, K, and V to learn complex and nuanced relationships between elements of the input sequence that are not dependent on the application space. The trainable parameters in those layers can also be readily extended to increase the module's degrees of freedom.

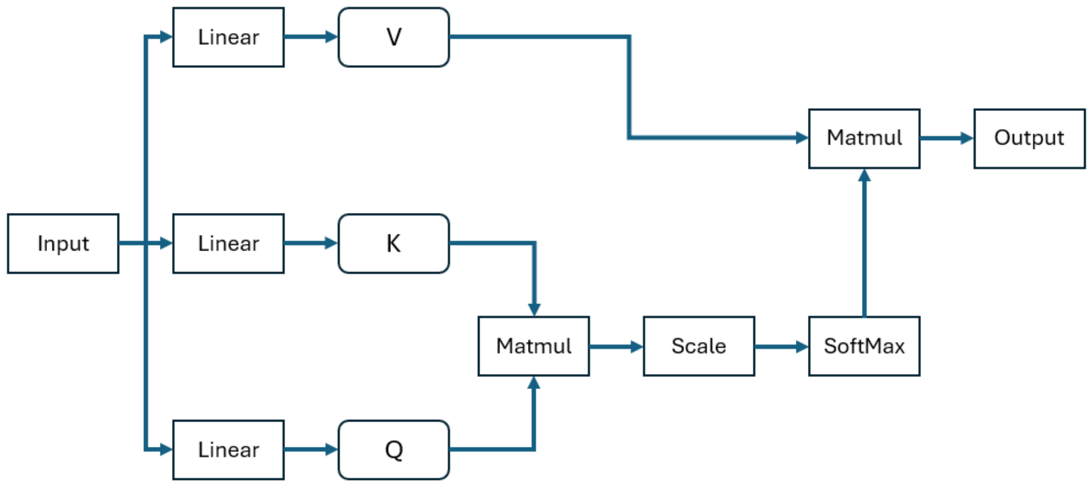


Figure 2.16: Structure of generalized self-attention.

The other key update to attention proposed in [28] was the idea of multi-head attention. This concept builds on the attention function defined in Equation (2.27) by setting up multiple parallel attention structures to simultaneously process multiple sets of queries, keys, and values. First, it is important to define what constitutes an attention head:

$$head = Attention(QW^Q, KW^K, VW^V) \tag{2.28}$$

where W^Q, W^K, W^V are learned weight matrices that project Q, K, and V into a lower dimensional space. The structure of a single attention head is depicted in Figure 2.17. We can then say that for some number of parallel attention heads, h, the multi-head attention is defined as:

$$MultiheadHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \tag{2.29}$$

where $Concat(\bullet)$ refers to matrix concatenation, W^O is an output transformation weight matrix, and:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.30}$$

The full structure of multi-head attention is shown in Figure 2.18.

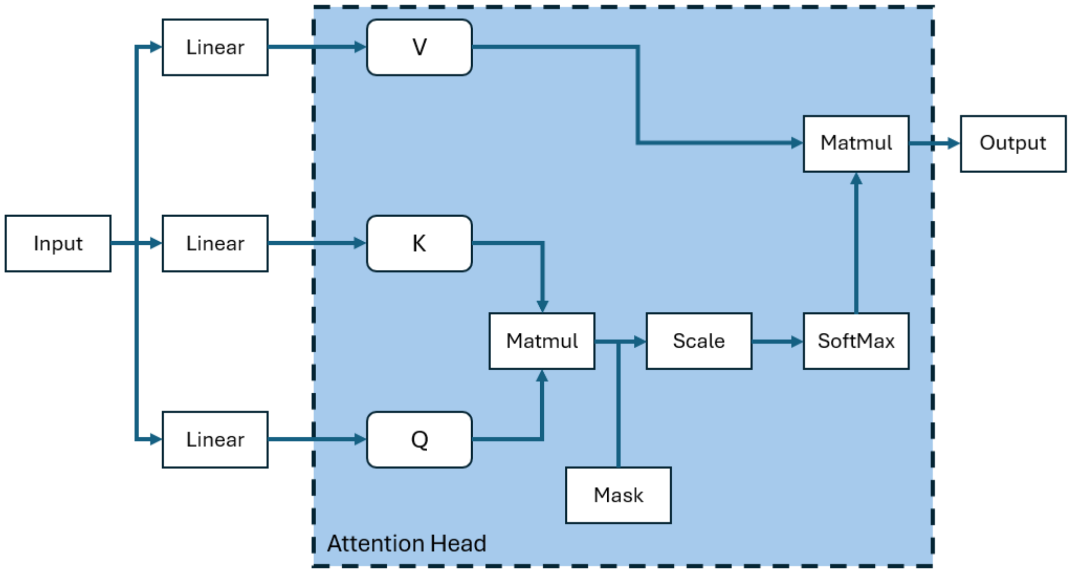


Figure 2.17: Structure of the self-attention head.

Multi-head attention has been shown to learn highly nuanced relationships between portions of an input sequence. Subdividing the input into regions and computing attention scores for each region gives the multi-head attention block greater capacity to identify a broader range of contextual relationships within input data. The parallelized structure of multi-head attention also means that the computational cost of doing so is minimal. It should also be noted that self-attention and multi-head attention mechanisms can learn non-sequential relationships, which standard RNN structures can't.

This concept of multi-head attention has led to a major revolution within the AI/ML community: the transformer model. This architecture, which was also pro-

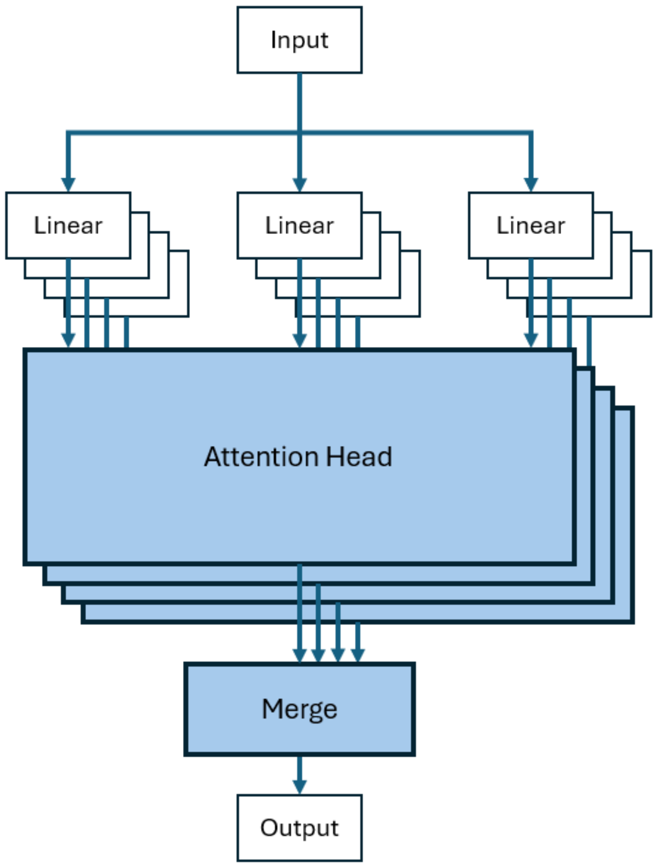


Figure 2.18: Structure of multi-head attention.

posed by Vaswani *et al.* in [28], uses an encoder-decoder structure with three sets of multi-head attention: 1) encoder self-attention, 2) decoder self-attention, and 3) encoder-decoder attention. It was originally designed for sequence to sequence translation. Here, the encoder self-attention is used to learn positional inter-dependencies between elements on the input sequence, the decoder self-attention is used to learn the inter-dependencies between the current output sequence position and all previous output positions, and the encoder-decoder attention is used to create a mapping between the two. This application of stacked, multi-head attention not only allows the transformer to learn non-sequential patterns, but also long-term dependencies in data that both RNNs and CNNs are typically unable to learn. The structure also allows a latent space mapping between concepts that can be easily scaled to provide a large number of degrees of freedom. The original structure of the transformer proposed in [28] is shown in Figure 2.19.

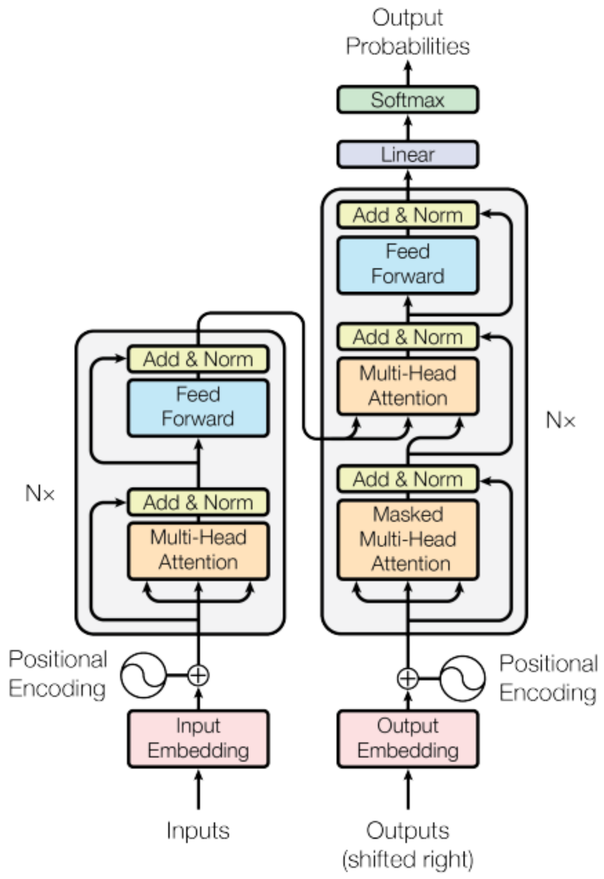


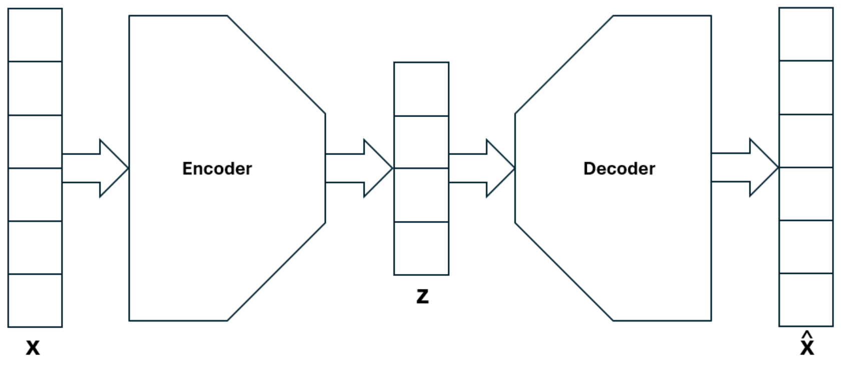
Figure 2.19: Structure of the transformer model [28].

2.4 Generative Machine Learning

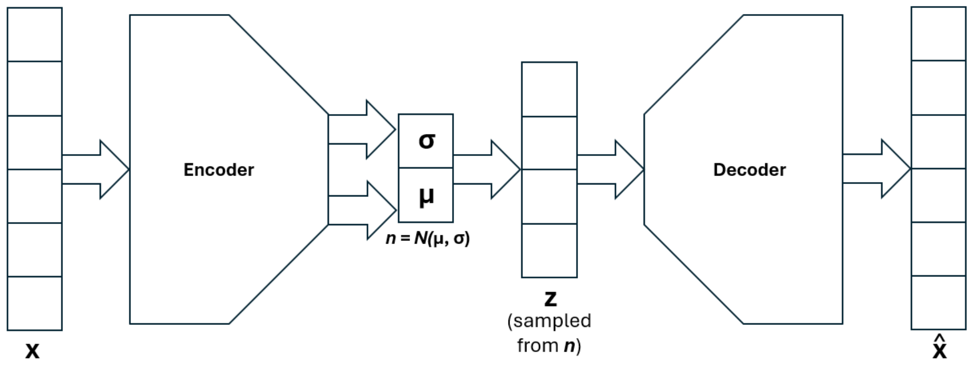
Generative ML is a relatively new sub-field of ML that has grown rapidly in popularity over the last few years. This sudden rise in prominence can be largely attributed to the explosion of AI generated art (like Stable Diffusion), text generators (like Chat-GPT), and video generators (like SORA). These tools, all of which are built on generative models, have provided a set of powerful tools that are easily accessible and have served to capture the imagination of the public. In the scientific community, generative AI models have also been used for audio and electromagnetic signal generation and modification. The broad enthusiasm for generative models, paired with huge leaps in computing hardware, has resulted in a massive research push and rapid advances in the domain.

There are numerous varieties of generative models, but they are all based on a similar approach that uses the encoder-decoder architecture. The encoder is trained to map input data or prompts into a probabilistic latent space representation and the decoder is trained to generate an output based on the distance between an input's latent space representation and the features contained within the latent space. When any point in the latent space is sampled (either randomly or through an input/prompt), it will then generate an output corresponding to that point in the latent space. For generative models to be effective, it is critical that their latent spaces contain two properties: 1) any point in the latent space should produce a meaningful output from the decoder and 2) points with low distance values between them in the latent space should produce similar outputs from the decoder. Otherwise the generated output may be completely nonsensical or not related to the input/prompt provided to the model.

Among the earliest generative neural network architectures was the variational auto-encoder (VAE). Traditional auto-encoders used a standard encoder-decoder structure and were trained to reproduce the input data provided to it. For example, such an auto-encoder might take in pictures as inputs. Its encoder would then translate those images to a latent space representation, and its decoder would use that latent space representation to reproduce the original input as closely as possible. They use a relatively simple loss function (called reconstruction loss) to calculate the distance between the



(a) Basic auto-encoder architecture.



(b) Basic VAE architecture.

Figure 2.20: Comparison of the structures for Auto-Encoder and VAE.

model output and the target label. Auto-encoders provided an interesting tool for creating lower-dimensional latent space representations and identifying critical features in data. However, the latent spaces developed in training were generally observed to be inconsistent and irregular, limiting their utility to generative processes.

The VAE, proposed in [29], uses the concept of stochastic variational inference to instead map inputs to latent distributions (rather than discrete latent variables). That distribution is then sampled to provide a noise vector to the decoder, which is trained to find the output that most closely maps to the distribution produced by the encoder. VAE loss is calculated in two parts. The first is the standard reconstruction loss used for auto-encoders and the second is a statistical distance metric (usually Kullback-

Leibler divergence). This provides a much more consistent and efficient mapping for a latent space and allows the decoder to be used independently of the encoder. It also provides a solid statistical foundation for the configuration of the model's latent space. VAEs have been commonly used for usable signal generation and modification. This was somewhat unique within the generative ML field and established a precedent and methods for applying such models to signal processing problems.

Another form of generative ML that is particularly applicable to signal processing is the generative adversarial network (GAN). GANs, which were first proposed in [30], use a similar encoder-decoder structure to the auto-encoder. However, the encoder-decoder is trained using an adversarial framework. These models operate by training two models in tandem; a generator model and a discriminator model [31]. The generator is trained to create images (or some equivalent output), and the discriminator is trained to distinguish between real images and generated ones. These models are trained together, with the discriminator weights updating after each iteration and the generator only updating when its generated output is successfully identified by the discriminator. When properly designed and tuned, each pushes the other to improve.

In this dissertation, an application of convolutional GANs [32] is explored for filter approximation. Originally, convolutional GANs for image generation were trained to take purely random noise inputs and generate "realistic" images, while the discriminator was simply trained to distinguish between "realistic" or "unrealistic" images. More modern GANs encode specific information into the noise input to the GAN that restrict the types of outputs it will generate. This is equivalent to providing a prompt to a transformer or VAE. These are referred to as conditional GANs [33, 34], and allow the GAN to learn more targeted input-output relationships.

2.5 Machine Learning Interpretability and Explainability

One of the key limiting factors to the broad incorporation of ML in avionics is interpretability, which is the degree to which the workings of the system are inherently understandable by people [35]. ML is a powerful tool, but the sheer number of calculations performed by ML systems and the often non-intuitive nature of those calculations can make their outputs difficult to understand or verify in real time. Although a well-trained ML system can often perform an analysis much more quickly and accurately than a person, the lack of transparency into how the solution was derived often limits the trust placed in these systems. In addition, the inability to readily identify or mitigate issues within ML systems in real time limits their use in many areas.

The need for ethical decision-making that follows legal and safety requirements in applications such as these, where a machine may be making decisions that affect the health or well-being of humans, has begun to give rise to government regulation. Many of these require that explanations of the machine's decisions be made available upon request to those that are substantially affected by them [36, 37, 38]. This leads to a number of challenges to be addressed in the coming years. In particular, what constitutes a sufficient explanation and on what time frame those explanations need to be provided must be determined.

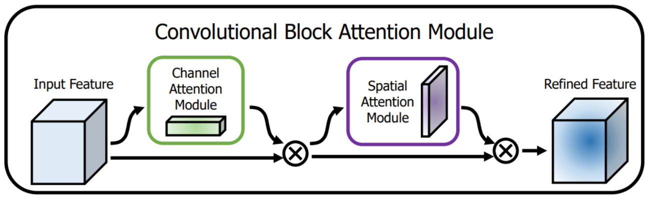
In an effort to address these issues, interpretability in autonomous systems has become a major focus area for ML researchers in recent years. Research has generally focused on either using simpler ML techniques that are inherently interpretable or generating explanations for the determinations made by more complex systems [39]. The former approach is the most straightforward and has the least overhead. However, ML techniques that are inherently interpretable may not be appropriate for applications that require complex analysis of large feature sets, requiring the use of complex, opaque ML approaches like deep neural networks (DNNs). To address the issue of explaining DNN determinations, significant research has been done into the use of attention mechanisms for providing explanations [40]. These mechanisms have proven to be highly effective at increasing the level of transparency in DNNs. However, attention may not provide sufficient information for explaining complex decisions in critical systems (like an autonomous driving system). In recent years, there has been a drive to expand the scope of explanations in decision-making DNNs to include a reasoning component [41, 42]. The most promising avenue of achieving this is integrating hierarchical decision elements, like decision trees with other ML and algorithmic elements, and tracking the progression through these hierarchical structures.

The concepts of explainability and interpretability require an ML designer to consider core elements of their target application space. They must then design their systems so that they are either more structured and consistent or so that critical decision-making elements of their models can be accessed and interpreted. In Chapter 4, I present my work developing a structured architecture to enable ML systems to simultaneously make decisions and generate text-based explanations for them. I then use these explainability concepts and design principles extensively throughout the rest of my work to intelligently break down problem spaces and orchestrate systems with ML components.

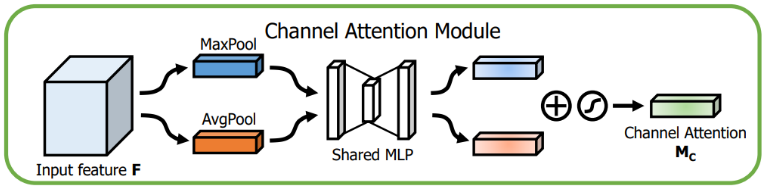
2.5.1 Convolutional Batch Attention Module

The Convolutional Batch Attention Module (CBAM) proposed in [23] is an attention mechanism developed by Woo *et al.* to provide feature importance maps (see

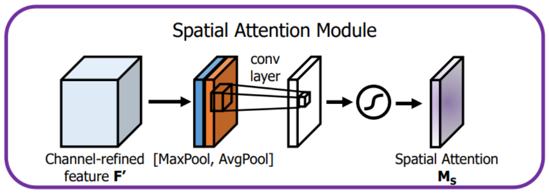
Figure 2.15b) for convolutional layers in neural networks. Its architecture is shown in Figure 2.21. The CBAM was developed to both improve the training process and provide explainability for CNNs. They can be placed immediately after any convolutional layer of a network and take a feature map $\mathbf{F}$ as an input.



(a) Architecture of the CBAM.



(b) Channel attention block.



(c) Spatial attention block.

Figure 2.21: The CBAM architecture from [23].

In the CBAM, attention is generated by calculating and applying two masks to the input feature map. The first mask is called the channel attention mask, $\mathbf{M}_c$, and identifies which channels of $\mathbf{F}$ were more heavily activated:

$$\mathbf{M}_c(\mathbf{F}) = \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F}))) \qquad (2.31)$$

where $MLP(\bullet)$ denotes the use of a MLP, $AvgPool(\bullet)$ is the average pooling function, and $MaxPool(\bullet)$ is the maximum pooling function. The second is the spatial attention mask, $\mathbf{M}_s$, which identifies the pixel regions of $\mathbf{F}$ were more heavily activated:

$$\mathbf{M}_s(\mathbf{F}) = \sigma(Conv^{N,M}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) \tag{2.32}$$

where $Conv^{N,M}(\bullet)$ denotes convolution with a filter of size $N \times M$. The final output of the CBAM then takes the form:

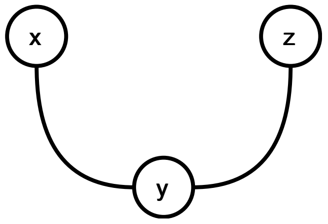
$$\mathbf{F}_{int} = \mathbf{M}_c(\mathbf{F}) \otimes \mathbf{F} \tag{2.33}$$

$$\mathbf{F}_{out} = \mathbf{M}_s(\mathbf{F}_{int}) \otimes \mathbf{F}_{int} \tag{2.34}$$

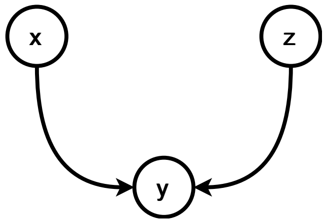
I use this mechanism extensively in Chapter 4 to provide insight into why a ML system was arriving at a given output.

2.6 Causality and ML

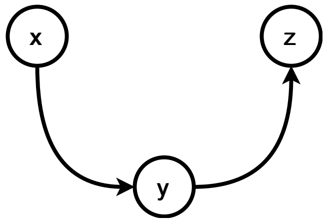
Another important concept used in ML and explainability is causality. In general, causality awareness refers to an understanding of the relationships between causal events and their resultant effects [43]. Knowledge of these cause-effect relationships plays an important role in many areas, particularly those in which analysis or optimization are required. It is also necessary to effectively evaluate alternative approaches to achieve a different outcome in a given scenario. Today, people perform most non-trivial tasks that require an understanding of causality. These problems are generally too complex or not well enough defined to be performed using traditional automation methods. Historically, machine learning systems have learned to perform tasks based on correlations between variables (a directionless relationship), and they tend to struggle with the inherent directionality of causality [44]. Figure 2.22 shows the difference between correlation and causal relationships.



(a) x and z correlate to y.



(b) x and z cause y.



(c) x causes y causes z.

Figure 2.22: Representing Causal vs Correlation Relationships.

Statisticians have spent several decades developing mathematical representations for causality. At a base level, cause-effect relationships are represented graphically (as in Figure 2.22) and determined through the evaluation of interventions [45]. Here, an *intervention* refers to fixing some given input(s) and evaluating the probability of an outcome when varying each other input. Interventions are evaluated with *do-calculus*, an approach to determining causality in non-parametric models, using [45]

$$P(y|\operatorname{do}(X=x)) = \sum_{z} P(y|x,z)P(z) \tag{2.35}$$

where $\operatorname{do}(.)$ is the *do*-operator that specifies the intervention being taken, y is the output of a system, x is the input being intervened upon, and z is another input not being intervened upon. Intervention allows one to quantify the effect one variable has on other elements within a system. That concept has been taken a step further to evaluate how altering certain aspects of an observed scenario might impact its outcome. By observing the output of a system and then intervening on some aspect(s) of the system and observing how the outcome changes, it is possible to identify the causal impact different elements of the system have on its output. This process is referred to as *counterfactual evaluation*, and is represented mathematically as [45]

$$E(Y_{X=1}|X=0, Y=Y_0) \tag{2.36}$$

where $E(\cdot)$ is the expectation, $Y_{X=1}$ is the hypothetical condition being evaluated which is predicated on an event that did not happen $X = 1$ that is counter to the observed event $X = 0$ as indicated by the condition realized in actuality $Y = Y_0$.

In recent years, researchers have begun exploring how to use the concepts of intervention and counterfactual evaluation to enable ML agents to discover and understand causal relationships [44]. While causal discovery has received more focus from the ML research community, causality awareness is rapidly gaining attention due to its potential to improve performance and flexibility in autonomous agents. It has also become a major consideration in ML explainability [46]. In Chapter 5, I use counterfactual evaluation in the context of target tracking and navigation to correct measurement drift errors by examining alternative paths an entity could have followed that are more consistent with high fidelity observations.

2.7 Genetic Algorithms

Genetic algorithms (GAs) are a family of optimization techniques inspired by biological evolution that fall within the umbrella of ML. They function by first generating a set of potential solutions to a problem space, referred to as the population. Each member of the population is then evaluated according to an objective function to determine its “fitness” as a solution to the target problem space. Some portion of the lower scoring population is removed, and a new population is created using the surviving members of the previous generation. This population consists of a combination of those surviving members, and others are generated using processes called cross-over, in which the properties of some number of population members are combined to create a new member, and mutation, in which some or all parameters of one member are altered. This new population is then evaluated and the process repeats until some stopping criteria is reached. When applied to a problem, GAs are able to rapidly explore a potential input space and identify the regions within it that produce results which are near optimal for some defined objective(s). They excel in scenarios in which the mapping between the input and output spaces are highly complex but there are clear ways to define the objective(s) by which a solution’s fitness is measured.

2.7.1 Multi-Objective Genetic Algorithms

There are many applications that appear to be ideal to approach using GAs but in which the complexity of the problem makes it unlikely that a singular objective function would be sufficient to determine the fitness of a potential solution. This has led to the formulation of multi-objective genetic algorithms (MOGAs). Similar to genetic algorithms, MOGAs function by producing a population of potential solutions and then evaluating their fitness. However, as their name suggests, this family of GAs uses multiple objective functions. That approach drastically increases the complexity of the algorithm, as the need to balance multiple objective functions to arrive at a singular fitness rating is challenging. The non-dominated sorting genetic algorithm II (NSGA-II) [47] is a well-known and widely-used MOGA.

NSGA-II is an elitist algorithm that consists primarily of two sorting algorithms: fast non-dominated sorting and crowding distance sorting. First, a population P is generated and each member p_i is evaluated based on each of the defined objective functions. For each p_i, the number of other solutions ($p_{j \neq i}$) that dominate it are computed to find its domination count, and the solutions are sorted from lowest to highest domination count. The population member $p_{j \neq i}$ dominates p_i if it is strictly better according to all objective functions used to evaluate the solutions, and a non-dominated solution is one whose domination count is zero. Next, the solutions are grouped according to their non-dominated front (domination count), and each front is sorted from highest to lowest according to crowding distance. The crowding distance for each p_i is computed as the average distance to the nearest solution on either side of it (p_{i-1} and p_{i+1}) within the same non-dominated front for each objective function. In effect, this is a measure of how similarly p_i performs compared to other solutions currently under evaluation. Once the solutions are sorted, all those falling below some threshold are rejected and a new population is generated from those that remain. This process is depicted in Figure 2.23 [47].

In Chapter 5, I use the NSGA-II as a component of a counterfactual system for correcting measurement drift errors. It serves to identify and evaluate alternative paths for an entity based on a set of objective functions derived from motion characteristics. NSGA-II is a well documented and effective method for performing optimization

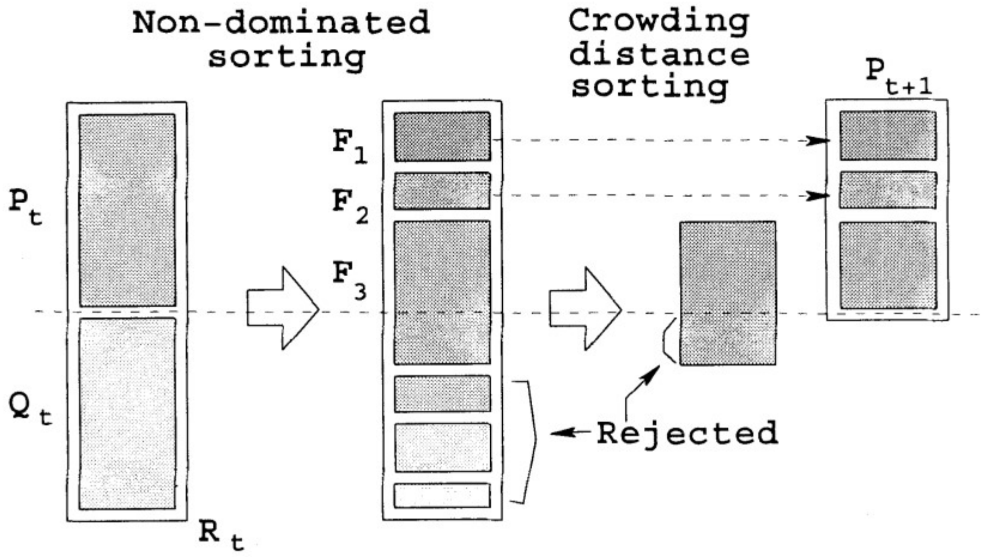


Figure 2.23: Depiction of the NSGA-II from Deb et al. [47]

based on multiple objectives simultaneously, but it does have some limitations [48]. As with all GAs, the identification and formulation of the objective functions is critical to the performance of the algorithm. Significant work was done on this effort to identify and optimize the objective functions used for track repair. Additionally, NSGA-II has a practical limitation on the number of objective functions for which it can optimize, as its performance degrades drastically for large numbers of objective functions. Alternatives have been developed (notably the NSGA-III algorithm [49]) that allow for more objectives to be used, but NSGA-II is still widely used.

Chapter 3

Radar Background

This dissertation focuses on the development of ML-enhanced algorithms for use in common radar applications that include detection, tracking, and navigation. These algorithms are designed based on a first principles understanding of the radar domain and seek to intelligently integrate ML with existing signal processing techniques in a way that takes advantage of the strengths of each field while mitigating their weaknesses. Before diving into a discussion of these algorithms, it is important to first discuss the key radar concepts that are used in each. In this chapter, I will discuss the basics of radar detection theory, adaptive filtering, and the Kalman filter.

3.1 Hypothesis Testing and Neyman-Pearson Lemma

Detection theory is rooted in a concept called hypothesis testing [50], and most modern detectors are derived from some formulation of a hypothesis test. Hypothesis testing for any scenario requires one to establish two or more possible cases, or hypotheses, for that scenario and then define a statistical representation of the likelihood of each hypothesis. As a rule, that process involves assigning a likelihood function to each hypothesis under evaluation and then deriving a test statistic to select between the hypotheses.

The simplest form of hypothesis testing is known as the binary hypothesis test. As the name suggests, it is formulated as a simple choice between a null hypothesis and a non-null hypothesis, referred to as H_0 and H_1, respectively. For the binary hypothesis test, there are four possible configurations, or modes, of the test output:

1. Type I Success: H_0 is selected when H_0 is true.
2. Type I Failure: H_0 is selected when H_1 is true.
3. Type II Success: H_1 is selected when H_1 is true.
4. Type II Failure: H_1 is selected when H_0 is true.

The binary hypothesis test is the most widely used form of hypothesis testing in radar detection problems, where a given radar return signal, z, is the subject of the test. In this domain, H_0 is defined as the case where the signal contains only interference signals and H_1 is defined as the case where the signal contains a combination of interference returns and a reflection from a target of interest. The probability of each hypothesis given the observed signal return are then defined as $P(y|H_0)$ and $P(y|H_1)$. How those probabilities are modeled is a major focus of research in radar detection and will be discussed in more detail later in this dissertation. In a given detection algorithm, they are used to calculate the test statistic which is compared against some threshold to determine whether a detection is recorded.

With this formulation of the binary hypothesis test, there are three metrics used to define the performance of the hypothesis test: 1) The probability of detection (P_d), 2) the probability of false alarm (P_{fa}), and 3) the probability of a missed detection (P_m). The P_{fa} and P_m are the two failure modes for this hypothesis test. P_{fa} is defined as the Type I failure mode and the P_m is defined as the Type II failure mode. Maximizing the P_d is the ultimate goal of any detection algorithm. When we consider the way in which a hypothesis test is conducted (i.e. comparing a test statistic against a threshold), it is clear that the P_d is closely tied to the threshold value. Lowering the threshold necessarily increases a detector's ability to detect targets. However, this will also naturally increase the rate of false alarms. A high P_{fa} can be extremely problematic, as it can impede effective radar resource allocation and drive up tracking times. As a result, P_d and P_{fa} must be carefully balanced. This has led to the formulation of the Neyman-Pearson criterion as a special case of the Bayes criterion. It states that a detector is considered optimal if it maximizes the P_d without raising the P_{fa} above some predefined threshold.

3.2 The Radar Detection Problem Space

In order to frame the rest of the discussion on detection, it is important to first establish the problem space used to derive and test detectors. Consider the scenario in which a pulsed-Doppler radar transmits m pulses over the course of a coherent processing interval. The return signal is received, processed, and sampled appropriately. For a given range bin, this results in the complex signal $\mathbf{z}$, where:

$$\mathbf{z} = [z_1, z_2, ..., z_m]^T \tag{3.1}$$

The function of a detection system is to determine whether or not $\mathbf{z}$ contains a signal return from a target of interest. In the event that it does, then $\mathbf{z}$ is comprised of an additive combination of the signal of interest and interference (here we will limit this to noise and clutter). Otherwise, $\mathbf{z}$ consists entirely of interference. What has just been described is a binary hypothesis test. The two hypotheses can be represented as:

$$\begin{cases} \mathbf{H_0}: & \mathbf{z} = \mathbf{n} \\ \mathbf{H_1}: & \mathbf{z} = \mathbf{s} + \mathbf{n} \end{cases} \tag{3.2}$$

In (3.2), $\mathbf{n}$ is the m-dimensional complex interference vector and $\mathbf{s}$ is the m-dimensional target return vector. Throughout the literature, the interference is modeled as $\mathbf{n} = \sqrt{\tau}\mathbf{x}$, where τ is referred to as the texture and $\mathbf{x}$ is the speckle. $\mathbf{x}$ is typically modeled as an m-dimensional correlated Gaussian random vector with zero mean and normalized Hermitian covariance matrix $E[\mathbf{n}\mathbf{n}^H] = \mathbf{M}$.

The target return signal is modeled as $\mathbf{s} = \alpha\mathbf{p}$. α is an unknown positive random value representing the amplitude of the signal return. This is meant to capture the effects of channel propagation. $\mathbf{p} = [e^{-j2\pi f_D T_s}, ..., e^{-j2\pi m f_D T_s}]$ is the m-dimensional, known temporal steering vector, where f_D is the Doppler frequency and T_s is the pulse repetition interval. We can now rewrite the previous hypotheses as:

$$\begin{cases} \mathbf{H_0}: & \mathbf{z} = \sqrt{\tau}\mathbf{x} \\ \mathbf{H_1}: & \mathbf{z} = \alpha\mathbf{p} + \sqrt{\tau}\mathbf{x} \end{cases} \tag{3.3}$$

There are a number of approaches to formulating detection algorithms, each of which make different assumptions about the nature of the interference present in the operational scenario for which they are designed. The most fundamental is the matched filter, which is known to produce optimal detection performance when the only interference present is white, Gaussian noise [1]. However, when interference is correlated, the matched filter ceases to maintain optimal performance, and some form of additional processing is required to distinguish between interference and target returns [51]. For the work presented in this dissertation, I focus on a family of detection algorithms known as adaptive detectors. These detection algorithms apply a whitening filter to decorrelate interference. They are referred to as adaptive because they approximate the covariance structure of the interference in order to define that whitening filter. This approximation is typically accomplished using samples from the area surrounding the sample under test. As a result, it is assumed that there are a number, K, of additional samples containing only interference in addition to the sample under test. These are typically referred to as training or secondary data in the literature, and here they are represented as $\mathbf{z}_k = \mathbf{n}_k$ where $E[\mathbf{n}_k\mathbf{n}_k^H] = E[\mathbf{n}\mathbf{n}^H] = \mathbf{M}$.

3.3 Interference Models for Adaptive Detection

Detection relies on the ability to distinguish between signal returns with targets present from those with targets absent. When the target signal and interference signal models are perfectly known, then the Neyman-Pearson Criterion can be used to maximize the probability of detection (P_d) while maintaining a constant false alarm rate (CFAR). In practice, the exact nature of these models is not known and must either be assumed or inferred. The most basic detectors assume that only uncorrelated Gaussian noise is present in the target absent hypothesis. Under this assumption, the matched filter is the optimal detector, since it maximizes the signal to noise ratio [1]. However, real world interference often consists of a combination of uncorrelated Gaussian noise and correlated clutter [51]. When correlated interference is present, the matched filter may no longer be optimal unless the signal can be effectively whitened. In order to do this, the covariance matrix of the interference model (M) must be known [52].

In practice, $\mathbf{M}$ will never be perfectly known. To mitigate this limitation, adaptive detectors have been developed as a family of algorithms that attempt to dynamically infer the correlation characteristics of interference and whiten the signal under test [13]. Test statistics and thresholds are then derived according to the assumed interference model in order to perform the binary hypothesis test described in 3.2. As discussed in the previous section, interference is generally modeled as a combination of texture (τ) and speckle (x), where x is a correlated Gaussian vector and τ modulates x. This allows clutter to be modeled as a set of compound Gaussian distributions based on the texture distribution:

$$f_{\mathbf{n}}(\mathbf{n}) = \frac{1}{(\pi)^N ||\mathbf{M}||} \int \frac{1}{(\tau)^N} e^{-\frac{\mathbf{n}^H \mathbf{M}^{-1} \mathbf{n}}{\tau}} f_\tau(\tau) d\tau \tag{3.4}$$

The well understood nature of Gaussian and compound Gaussian distributions makes such a formulation of the interference model desirable because it simplifies the derivation of detector test statistics, thresholds, and P_{fa} characteristics.

In much of the literature, $f_\tau(\tau)$ is assumed to be the Dirac delta function, $\delta(\tau - 1)$, resulting in $f_{\mathbf{n}}(\mathbf{n})$ being a purely homogeneous, correlated complex Gaussian interference signal:

$$f_{\mathbf{n}}(\mathbf{n}) = \frac{1}{(\pi)^N ||\mathbf{M}||} e^{-\mathbf{n}^H \mathbf{M}^{-1} \mathbf{n}} \tag{3.5}$$

The assumption that clutter can be modeled as a complex Gaussian distribution holds in many situations due to the Central Limit Theorem [1], as environments often approximate an infinitely large number of uniformly distributed point scatterers. However, this assumption does not hold in more complex scenarios, such as in urban and aquatic environments, on fast moving aerial platforms, or in the presence of active interference [53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64]. Instead, the interference signals may have heavy spikes that more closely resemble target returns. Various works have been published demonstrating that non-Gausian, heavy tailed clutter in various complex scenarios can be modeled with distributions that fall into the category of Spherically Invariant Random Processes (SIRP), including the K, Pareto, and Weibull distributions [65]. The ability to model clutter as a SIRP is desirable for a key practical reason: it is relatively easy to simulate a broad range of SIRP distributions with known correlation characteristics. The reasons for this are due to several key properties of SIRPs, which are discussed in [66]. In that work, Rangaswamy *et al.* present four key theorems regarding SIRPs and spherically invariant random vectors (SIRV).

The first of these theorems is [66]:

Theorem 1. *For each complex SIRV,* $\mathbf{Y} = [y_1\ y_2\ ...\ y_N]$*, there exists a non-negative random variable S with pdf* $f_S(s)$ *that can be used to condition* $\mathbf{Y}$ *to produce a complex Gaussian random vector,* $\mathbf{X} = [x_1 x_2 ... x_N]$ *with covariance* $\mathbf{M}$ *and zero mean.*

From Theorem 1 we get that $\mathbf{Y} = S\mathbf{X}$ and can represent the pdf of $\mathbf{Y}$ as:

$$f_{\mathbf{Y}}(\mathbf{y}) = \frac{1}{(\pi)^N ||\mathbf{M}||} \int \frac{1}{(s)^{2N}} \exp\left(-\frac{\mathbf{y}^H \mathbf{M}^{-1} \mathbf{y}}{s^2}\right) f_S(s) dS \tag{3.6}$$

where $f_S(s)$ is known as the characteristic pdf of the SIRV. This allows a SIRV to be entirely defined by its covariance, mean, and characteristic function. It is important to note that by setting $S = \sqrt{\tau}$, (3.6) becomes (3.4).

The second theorem allows us to extend this further:

Theorem 2. *When a SIRV,* $\mathbf{Y} = [y_1\ y_2\ ...\ y_N]$*, with characteristic function* $f_S(s)$ *undergoes a linear transform of the form:*

$$\mathbf{Z} = \mathbf{YA} + \mu \tag{3.7}$$

$\mathbf{Z}$ *is also a SIRV with characteristic function* $f_S(s)$, *mean* μ, *and covariance* $\mathbf{M} = \mathbf{A}\mathbf{A}^{\mathbf{T}}$.

The combination of Theorems 1 and 2 allows one to generate a complex white Gaussian random variable, use a characteristic pdf to generate a new SIRV, and apply a simple linear transform to set the mean and covariance characteristics. Given that most simulation software has built in functions for generating Gaussian distributed data, this is a powerful combination which enables the straightforward generation of any correlated SIRV whose characteristic pdf can be simulated.

The final two Theorems presented in [66] are related to the spherical and quadratic forms of SIRVs, respectively.

Theorem 3. *A vector,* $\mathbf{Y} = [y_1\ y_2\ ...\ y_N]$, *is a SIRV if and only if when its components are expressed in generalized spherical coordinates:*

$$\begin{aligned} y_1 &= r \times \cos\phi_1 \\ y_k &= r \times \cos\phi_k \prod_{i=1}^{k-1} \sin\phi \quad (1 < k \leq N-2) \\ y_{N-1} &= r \times \cos\theta \prod_{i=1}^{N-2} \sin\phi \\ y_N &= r \times \cos\theta \prod_{i=1}^{N-2} \sin\phi \end{aligned} \tag{3.8}$$

then r, ϕ, and θ are all mutually and statistically independent with pdfs of the form:

$$\begin{aligned} f_R(r) &= \frac{r^{N-1}}{2^{\frac{N}{2}-1}\Gamma(\frac{N}{2})}\int \frac{1}{(s)^{2N}}\exp{(-\frac{r^2}{2s^2})}f_S(s)dS \\ f_{\Phi_k}(\phi_k) &= \frac{\Gamma\left(\frac{N-k+1}{2}\right)}{\sqrt{\pi}\Gamma\left(\frac{N-k}{2}\right)}\sin^{N-k-1}(\phi_k)\times[(u(\phi_k)-u(\phi_k-\pi)] \\ f_\Theta(\theta) &= \frac{1}{2\pi}[(u(\theta)-u(\theta-2\pi)] \end{aligned} \tag{3.9}$$

where $\Gamma(\bullet)$ is the Eulero Gamma function and $u(\bullet)$ is the unit step function.

Theorem 3 allows SIRVs to be represented in a generalized spherical form that allows SIRVs with zero mean and identity covariance matrix to be fully characterized even without a closed form expression for its characteristic function.

Theorem 4. *The quadratic form of the characteristic pdf remains unchanged whether or not the SIRV is white and takes the form:*

$$f_P(p) = \frac{1}{2^{\frac{N}{2}}\Gamma\left(\frac{N}{2}\right)}p^{\left(\frac{N}{2}\right)-1}\int \frac{1}{(s)^{2N}}\exp{(-\frac{p}{2s^2})}f_S(s)dS \tag{3.10}$$

These theorems provide a powerful set of tools for generating a broad range of interference distribution models. Additionally, these distributions can be readily simulated with known mean and correlation properties by transforming a Gaussian vector with zero mean and unit covariance. This is incredibly important for evaluating the performance of adaptive detectors in a meaningful way across distributions and with different correlation characteristics. In [66], Rangaswamy *et al.* detail two methods for simulating SIRVs. The first assumes a known, closed form characteristic function, while the second does not. They also provide a list of known characteristic functions for generating common SIRVs used to model clutter with the first simulation method.

The process for generating SIRV data from a known characteristic function is relatively straightforward. First, a Gaussian random vector, $\mathbf{X}$ with zero mean and identity covariance matrix is generated. Next, a random variable, S, defined by the characteristic function is generated and normalized so that its mean square value is 1. A SIRV with zero mean and identity covariance matrix can then be obtained from the product $\mathbf{Y} = \mathbf{X}S$. Finally, Equation (3.7) can be used to transform $\mathbf{Y}$ into a SIRV with some target mean and covariance structure.

This simulation method was used for my work on ML-enhanced adaptive detection presented in Chapters 6 and 7. Three SIRV distributions were used for designing and characterizing those detection algorithms: 1) the Gaussian distribution, 2) the K-distribution, and 3) the Pareto distribution. The model for the Gaussian distribution was given by Equation (3.5).

In order to generate the K-distributed clutter model, τ from Equation (3.4) is sampled from a Gamma distribution with shape parameter a and scale parameter b:

$$f_\tau(\tau) = \frac{1}{(b)^a \Gamma(a)} (\tau)^{a-1} e^{\frac{-\tau}{b}} \tag{3.11}$$

Where $\Gamma(\bullet)$ is the Gamma function. The amplitude PDF of the clutter can then be represented as a K-distribution [65]:

$$f_{|\mathbf{n}|}(|\mathbf{n}|) = \frac{2|\mathbf{n}|}{b\Gamma(a)} \left(|\mathbf{n}| \sqrt{\frac{1}{2b}} \right)^{a-1} K_{a-1} \left(|\mathbf{n}| \sqrt{\frac{2}{b}} \right) \tag{3.12}$$

Where $K(\bullet)$ is the modified Bessel function of the second kind.

Pareto distributed clutter can be modeled by using an inverse gamma texture with shape parameter a and scale parameter b: [67],[60]:

$$f_\tau(\tau) = \frac{1}{(b)^a \Gamma(a)} (\tau)^{-(a+1)} e^{\frac{1}{-b\tau}} \tag{3.13}$$

3.4 Adaptive Detectors

In this section, a survey of some common adaptive detection algorithms will be discussed. This will include the GLRT, adaptive matched filter (AMF), the adaptive Rao detector, adaptive covariance estimator (ACE), adaptive beamformer orthogonal rejection test (ABORT) [68], and a number of two-stage algorithms. For this discussion, they will be evaluated on the following performance metrics ([69], [70], [71], [72], [73]): matched target return signal detection, clutter cancellation properties and sensitivity to model/data mismatches, sample support requirements, computational complexity, and constant false alarm rate (CFAR) properties.

3.4.1 Generalized Likelihood Ratio Test

In what would prove to be a pivotal work [13], Kelly proposed the GLRT as a new adaptive radar detector for use in the presence of unknown homogeneous Gaussian noise. This detector has been thoroughly investigated and expanded upon in subsequent works: [74], [75], [76], [77], [78], [79]. As its name suggests, it is based on the development of a likelihood ratio test. In order to set up this likelihood test, two joint probability density functions (PDFs) are first defined for the input signals, one for $\mathbf{H}_0$ and one for $\mathbf{H}_1$ [13]:

$$f_i[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = f[\mathbf{z}|_{\mathbf{H}_i}] \prod_{k=1}^{K} f[\mathbf{z}_k], \; where \; i = 0, 1 \tag{3.14}$$

For the null hypothesis, the signal consists only of interference that is assumed to be zero mean Gaussian and identically distributed to the training signals, so the joint PDF becomes:

$$f_0[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{\pi^N ||\mathbf{M}||} e^{-trace(\mathbf{M}^{-1}\mathbf{T}_0)} \right)^{K+1}, \tag{3.15}$$

$$\mathbf{T}_0 = \left(\frac{1}{K+1} \right) \left(\mathbf{z}\mathbf{z}^H + \sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H \right) \tag{3.16}$$

In this case, only the interference covariance matrix $\mathbf{M}$ is unknown.

For $\mathbf{H_1}$, where z contains both a target return and interference, the PDF becomes:

$$f_1[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{\pi^N ||\mathbf{M}||} e^{-trace(\mathbf{M}^{-1}\mathbf{T}_1)} \right)^{K+1}, \tag{3.17}$$

$$\mathbf{T}_1 = \left(\frac{1}{K+1} \right) \left((\mathbf{z} - \alpha\mathbf{p})(\mathbf{z} - \alpha\mathbf{p})^H + \sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H \right) \tag{3.18}$$

Here, both the interference covariance matrix $\mathbf{M}$ and the target return signal amplitude α are unknown.

Maximum likelihood estimation is used to independently maximise each of these PDFs over all unknown parameters. The ratio of the two, referred to as the test statistic, is then compared with a threshold to determine target detection. This takes the form:

$$\lambda = \frac{\max_{\mathbf{M},\alpha} f_{\mathbf{z}|\mathbf{H}_1}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]}{\max_{\mathbf{M}} f_{\mathbf{z}|\mathbf{H}_0}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]} \gtrless^{\mathbf{H_1}}_{\mathbf{H_0}} \lambda_0 \tag{3.19}$$

Plugging (3.15) and (3.17) into (3.19) and performing the maximization yields the test statistic:

$$\lambda = \frac{1 + \left(\mathbf{z}^H (\sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H)^{-1} \mathbf{z} \right)}{1 + \left(\mathbf{z}^H (\sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H)^{-1} \mathbf{z} \right) - \frac{|\mathbf{p}^H (\sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H)^{-1} \mathbf{z}|^2}{\mathbf{p}^H (\sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H)^{-1} \mathbf{p}}} \tag{3.20}$$

The true interference covariance matrix is generally not known and is estimated as:

$$\mathbf{S} = \sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H \tag{3.21}$$

Substituting (3.21) and $\eta = \frac{\lambda - 1}{\lambda}$ into (3.20) yields the more well known form of the GLRT test statistic:

$$\eta = \frac{|\mathbf{p}^H \mathbf{S}^{-1} \mathbf{z}|^2}{(\mathbf{p}^H \mathbf{S}^{-1} \mathbf{p})[1 + (\mathbf{z}^H \mathbf{S}^{-1} \mathbf{z})]} \tag{3.22}$$

Thresholds are selected in order to maintain some desired probability of false alarm (P_{fa}), which is defined as:

$$P_{fa} = P(\lambda > \lambda_0 | \mathbf{H_0}) \tag{3.23}$$

In [74], the (P_{fa}) for the GLRT detector is derived as:

$$P_{fa} = \frac{1}{\lambda_0^{K-N+1}} \tag{3.24}$$

Note that this is only a function of K, N, and λ_0, which proves that the GLRT detector is CFAR with respect to the clutter covariance.

The GLRT detector broadly serves as the benchmark against which other adaptive detectors are judged. For the case of matched target signal returns in homogeneous interference, it typically displays the best probability of detection with a relatively low false alarm rate. On the other hand, the GLRT is more complex and computationally costly than the AMF algorithm that will be discussed shortly. It also has relatively poor rejection of mismatched target signal returns. The GLRT detector is designed assuming that the interference is zero mean Gaussian with an unknown covariance matrix. This results in CFAR behavior with respect to the interference covariance but not to the interference power level [80]. So long as the noise is homogeneous and Gaussian, this isn't an issue. However, it degrades clutter rejection in the presence of partially homogeneous or non-Gaussian interference. Variations of the GLRT have been derived for alternative distributions, but same problems hold for those (i.e. performance degrades when not faced with the expected interference distribution). Identifying the distribution of interference present in data is a non-trivial problem that will be explored in more detail later.

3.4.2 Adaptive Matched Filter

The AMF detector ([81], [82], [83], [84]) is based on a similar ratio test to that of Kelly's GLRT with one key difference: the interference covariance matrix is assumed to be known. What this means is that when we set up the ratio test and take the MLE of each input PDF, one need only maximize for α [81]:

$$\lambda = \frac{\max_{\alpha} f_{\mathbf{z}|\mathbf{H}_1}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]}{f_{\mathbf{z}|\mathbf{H}_0}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]} \gtrless^{\mathbf{H_1}}_{\mathbf{H_0}} \lambda_0 \tag{3.25}$$

Plugging in (3.15) and (3.17) and performing the likelihood estimation results in the ideal test statistic based on the true interference covariance matrix:

$$\lambda = \frac{|\mathbf{p}^H(\mathbf{M})^{-1}\mathbf{z}|^2}{(\mathbf{p}^H(\mathbf{M})^{-1}\mathbf{p})} \tag{3.26}$$

In practice, the true covariance matrix is unknown, so the interference covariance matrix S is derived from the training samples as in (3.21) and substituted for $\mathbf{M}$ to arrive at the final test statistic:

$$\lambda = \frac{|\mathbf{p}^H(\mathbf{S})^{-1}\mathbf{z}|^2}{(\mathbf{p}^H(\mathbf{S})^{-1}\mathbf{p})} \tag{3.27}$$

The P_{fa} for the AMF detector can be expressed as [81],[84]:

$$P_{fa} = \int_0^1 (1 + r\lambda_0)^{-(K-N+1)} f(r)dr \tag{3.28}$$

Where r is a Beta-distributed loss term caused by using an estimated covariance matrix to calculate the test statistic:

$$f(r) = \frac{K!}{(N-2)!(K-N+1)!}(1-r)^{N-2}(r)^{K-N+1} \tag{3.29}$$

This P_{fa} is independent of the true clutter covariance, proving that the AMF detector is a CFAR detector in the presence of homogeneous Gaussian interference.

It is apparent that (3.27) is simply computing the squared magnitude of the quasi-whitened signal $\mathbf{z}_w = \mathbf{w}^H\mathbf{z}$. Since the steering vector is known and the interference is assumed to be homogeneous, zero-mean, and Gaussian, the quasi-whitening filter $\mathbf{w}$ can be computed in advance. For each new signal under test, one then only needs to multiply $\mathbf{w}$ and $\mathbf{z}$ and find the squared magnitude of the result. This drastically simplifies that calculation and reduces the computational cost of the adaptive detector when compared with the GLRT. However, this comes at a performance cost. In the case of perfectly matched target signal returns in homogeneous interference, the AFM has roughly equivalent detection performance to the GLRT, but it has poor rejection of sidelobe target returns and interference that is not perfectly homogeneous. It is also highly sensitive to model mismatches, meaning that it will experience more severe performance degradation than the GLRT as the actual interference data stops resembling a perfectly homogeneous Gaussian distribution.

3.4.3 Rao

Most of the detectors evaluated here make the assumption that the interference covariance matrix can be estimated solely using a set of training samples that are assumed to contain only interference. The adaptive Rao detector presented in [85], on the other hand, is developed by making the assumption that both the signal under test and the training samples should be used to estimate the interference covariance matrix. It is so named due to its original derivation using the Rao test ([86], [87]). However, for this discussion, we will focus on the alternative derivation of the test statistic [85] which follows closely the derivation of the AMF. For this detector, the interference covariance matrix is assumed to be known, and the PDFs and likelihood ratio test are each set up to match those of the AMF exactly (i.e. (3.15), (3.17), and (3.25), respectively). Unsurprisingly, after maximizing for the unknown signal amplitude α, the ideal test statistic derived with perfect a priori knowledge of $\mathbf{M}$ is identical to that of the AMF, (3.22).

At this stage, the AMF substitutes (3.21) into (3.26) to arrive at the final AMF test statistic (3.27). For the Rao detector, the test sample is used along with the training samples to estimate the interference covariance:

$$\mathbf{S}_{rao} = \mathbf{z}\mathbf{z}^H + \sum_{k=1}^{K} \mathbf{z}_k\mathbf{z}_k^H = \mathbf{z}\mathbf{z}^H + \mathbf{S} \tag{3.30}$$

Substituting (3.30) into (3.26) results in:

$$\lambda = \frac{|\mathbf{p}^H(\mathbf{z}\mathbf{z}^H + \mathbf{S})^{-1}\mathbf{z}|^2}{(\mathbf{p}^H(\mathbf{z}\mathbf{z}^H + \mathbf{S})^{-1}\mathbf{p})} \gtrless^{\mathbf{H_1}}_{\mathbf{H_0}} \lambda_0 \tag{3.31}$$

The P_{fa} for the adaptive Rao detector is derived in [85] and can be expressed as:

$$P_{fa} = (1 - \lambda_0)^K \tag{3.32}$$

It is a function of only the detector threshold and the training sample support, which both proves that the Rao detector is a CFAR detector with respect to the interference covariance and demonstrates its dependence on using a large number of training samples.

The inclusion of test signal data along with the training samples to approximate the interference covariance matrix has an interesting effect. It gives the Rao detector substantially improved mismatched target rejection capabilities compared to the GLRT and AMF detectors. In addition, this makes the detector less sensitive to model mismatches than the AMF. It also has similar detection performance characteristics to the aforementioned detectors in the matched case when a large number of training samples is available, and it displays the GLRT's and AMF's property of being CFAR with respect to the interference covariance. However, it lacks the computational efficiency of the AFM. Also, the Rao detector has a heavy reliance on large training sample support, and its detection performance degrades substantially if a large number of the samples is not available. This is due to the impact of the test signal on the estimation of M, as it will dominate the estimation if a sufficiently large number of training samples is not used to offset it.

3.4.4 Adaptive Coherence Estimator

The ACE detection algorithm (discussed in [53], [80], [88], [89], [90], [91], and [92]) makes a fundamentally different assumption about the nature of the interference in the signal model: that it is not necessarily homogeneous. More specifically, it maintains the assumption of the GLRT and AMF detectors that a set of training samples contains only interference $\mathbf{x}_k \sim \mathbf{CN}[0, \mathrm{M}]$. However, it assumes that the interference present in the signal under test has covariance that matches that of the training samples up to an unknown scaling factor σ^2 (i.e. $\mathbf{x} \sim \mathbf{CN}[0, \sigma^2\mathrm{M}]$). The PDFs for each hypothesis now become:

$$f_0[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{(\sigma^2)^{\frac{N}{K+1}}\pi^N||\mathbf{M}||}e^{-trace(\mathbf{M}^{-1}\mathbf{T}_0)}\right)^{K+1}, \tag{3.33}$$

$$\mathbf{T}_0 = \left(\frac{1}{K+1}\right)\left(\frac{\mathbf{z}\mathbf{z}^H}{\sigma^2} + \sum_{k=1}^{K}\mathbf{z}_k\mathbf{z}_k^H\right) \tag{3.34}$$

$$f_1[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{(\sigma^2)^{\frac{N}{K+1}}\pi^N||\mathbf{M}||}e^{-trace(\mathbf{M}^{-1}\mathbf{T}_1)}\right)^{K+1}, \tag{3.35}$$

$$\mathbf{T}_1 = \left(\frac{1}{K+1}\right)\left(\frac{(\mathbf{z}-\alpha\mathbf{p})(\mathbf{z}-\alpha\mathbf{p})^H}{\sigma^2} + \sum_{k=1}^{K}\mathbf{z}_k\mathbf{z}_k^H\right) \tag{3.36}$$

As with the AMF described above, the form of the covariance matrix (M) is assumed to be known when setting up the likelihood ratio test for the ACE detector. This leaves the target return signal amplitude α and the covariance matrix scaling factor σ^2 to be maximized for the MLE of the PDF for $\mathbf{H_1}$. Only the covariance matrix scaling factor σ^2 needs to be maximized for the MLE of the null hypothesis:

$$\lambda = \frac{\max_{\sigma^2,\alpha} f_{\mathbf{z}|\mathbf{H}_1}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]}{\max_{\sigma^2} f_{\mathbf{z}|\mathbf{H}_0}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]} \gtrless^{\mathbf{H_1}}_{\mathbf{H_0}} \lambda_0 \tag{3.37}$$

Evaluating each MLE and replacing the true covariance matrix $\mathbf{M}$ with (3.21) results in the ACE test statistic:

$$\lambda = \frac{|\mathbf{p}^H\mathbf{S}^{-1}\mathbf{z}|^2}{(\mathbf{p}^H\mathbf{S}^{-1}\mathbf{p})(\mathbf{z}^H\mathbf{S}^{-1}\mathbf{z})} \tag{3.38}$$

Interestingly, this test statistic has an important geometric property. It makes use of the inner product of the quasi-whitened signal and the quasi-whitened steering vector, meaning that it is a representation of the cosine squared of the angle between the return signal and the steering vector [88]:

$$\cos^2\phi = \frac{|\mathbf{p}^H\mathbf{S}^{-1}\mathbf{z}|^2}{(\mathbf{p}^H\mathbf{S}^{-1}\mathbf{p})\,(\mathbf{z}^H\mathbf{S}^{-1}\mathbf{z})} \tag{3.39}$$

The P_{fa} for the ACE detector can be expressed as [91], [84]:

$$P_{fa} = \int_0^1 (\frac{1-\lambda_0}{1-r\lambda_0})^{(K-N+1)} f(r)dr \tag{3.40}$$

Where r is a Beta-distributed loss term and $f(r)$ is given by (3.29). Clearly, the P_{fa} is independent of both $\mathbf{M}$ and σ^2, proving that the ACE detector is a CFAR detector with respect to both the interference covariance and the interference power level.

The ACE detector generally has poorer detection performance for matched targets in homogeneous interference environments than either the GLRT or the AMF, and it lacks the computational simplicity of the AMF algorithm. However, it has much better interference rejection capabilities, and its geometric properties mean that it excels at rejecting sidelobe target returns. In addition, the ACE test statistic has the property of being CFAR with respect to the interference power level, making it far less sensitive to heterogeneity in the interference patterns.

3.4.5 Adaptive Beamformer Orthogonal Rejection Test

The ABORT algorithm proposed in [68] was created to attempt to solve the issue of unmatched target signal returns [93]. This issue arises commonly in radar target detection, typically as the result of sidelobe returns. As with the other adaptive detection algorithms discussed here, ABORT sets up the detection problem as a binary hypothesis test based on likelihood ratios. However, the authors redefined the hypotheses for this algorithm:

$$\begin{cases} \mathbf{H_0}: & \mathbf{z} = \alpha\mathbf{p}_\perp + \sqrt{\tau}\mathbf{x} \\ \mathbf{H_1}: & \mathbf{z} = \alpha\mathbf{p} + \sqrt{\tau}\mathbf{x} \end{cases} \tag{3.41}$$

where $\mathbf{p}_\perp$ is an unknown steering vector orthogonal to $\mathbf{p}$. The decision now becomes whether a signal under test is comprised of interference plus target signal returns parallel to the steering vector or if it consists of interference plus target signal returns orthogonal to the steering vector.

As with the GLRT algorithm, the PDFs of each of these hypotheses are defined and used to set up the likelihood ratio test:

$$f_0[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{\pi^N ||\mathbf{M}||} e^{-trace(\mathbf{M}^{-1}\mathbf{T}_0)}\right)^{K+1}, \tag{3.42}$$

$$\mathbf{T}_0 = \left(\frac{1}{K+1}\right)\left((\mathbf{z} - \alpha\mathbf{p}_\perp)(\mathbf{z} - \alpha\mathbf{p}_\perp)^H + \sum_{k=1}^{K} \mathbf{z}_k\mathbf{z}_k^H\right) \tag{3.43}$$

$$f_1[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K] = \left(\frac{1}{\pi^N ||\mathbf{M}||} e^{-trace(\mathbf{M}^{-1}\mathbf{T}_1)} \right)^{K+1}, \tag{3.44}$$

$$\mathbf{T}_1 = \left(\frac{1}{K+1} \right) \left((\mathbf{z} - \alpha\mathbf{p})(\mathbf{z} - \alpha\mathbf{p})^H + \sum_{k=1}^{K} \mathbf{z}_k \mathbf{z}_k^H \right) \tag{3.45}$$

After setting up this test and maximizing the likelihood of each PDF for the unknown parameters ($\mathbf{M}$ and α for $\mathbf{H_1}$ and $\mathbf{M}$, α, and $\mathbf{p}_\perp$ for $\mathbf{H_0}$), the following test statistic is derived [68]:

$$\lambda = \frac{\max_{\mathbf{M},\alpha,\mathbf{p}_\perp} f_{\mathbf{z}|\mathbf{H}_1}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]}{\max_{\mathbf{M},\alpha} f_{\mathbf{z}|\mathbf{H}_0}[\mathbf{z}, \mathbf{z}_1, ..., \mathbf{z}_K]} \gtrless_{\mathbf{H_0}}^{\mathbf{H_1}} \lambda_0 \tag{3.46}$$

$$\lambda = \frac{1 + \frac{|\mathbf{p}^H \mathbf{S}^{-1} \mathbf{z}|^2}{\mathbf{p}^H \mathbf{S}^{-1} \mathbf{p}}}{1 + \mathbf{z}^H \mathbf{S}^{-1} \mathbf{z} - \frac{|\mathbf{p}^H \mathbf{S}^{-1} \mathbf{z}|^2}{\mathbf{p}^H \mathbf{S}^{-1} \mathbf{p}}} \tag{3.47}$$

Pulsone and Rader then use the relationship in (3.39) to write the test statistic in terms of ϕ, the angle between the target return signal and steering vector:

$$\lambda = \frac{1 + \mathbf{z}^H \mathbf{S}^{-1} \mathbf{z} \cos^2 \phi}{1 + \mathbf{z}^H \mathbf{S}^{-1} \mathbf{z} \sin^2 \phi} \tag{3.48}$$

Equation 3.48 intuitively demonstrates the goal of the ABORT test statistic: to compare the portion of the test signal energy aligned with the steering vector to that which is orthogonal to the steering vector.

The P_{fa} for the ABORT detector is derived in [68] and can be expressed as:

$$P_{fa} = \int_0^1 (1 + \lambda_0 - r)^{-(K-N+1)} f(r) dr \tag{3.49}$$

Where r is a Beta-distributed loss term and $f(r)$ is given by (3.29). The ABORT detector's P_{fa} is independent of the interference covariance, proving that it is a CFAR detector with respect to the interference covariance.

By assuming the possible presence of a misaligned target return in the adaptive detection test, ABORT manages to have substantially improved rejection characteristics when compared to simpler adaptive detectors like the GLRT and AMF. It does so while maintaining equivalent matched target detection performance. It does so at the cost of substantially increased computational complexity. To offset this cost, the authors of [68] proposed using ABORT as the second stage of a two-stage adaptive detector, with the far simpler AMF as the first stage. This effectively limits the use of ABORT to only those scenarios where the lower fidelity AMF threshold is passed.

3.4.6 Two-Stage Detectors

In addition to the common detectors described above, a number of two stage detectors have also been proposed ([94], [95], [68], [85], and [96]). These detectors all follow the same general process:

1. Apply one of the adaptive detectors listed above with a given threshold.
2. If the first test statistic falls below the threshold, the signal is deemed to not contain a target return.
3. If the first test statistic surpasses the threshold for the first detector, apply a second detector with another threshold.
4. If the second test statistic falls below the threshold, the signal is deemed to not contain a matched target return.
5. If the second test statistic surpasses the threshold for the second detector, a target detection is registered.

Additionally, these detectors typically follow a common trend: the first stage detector is a computationally efficient, less performant detector while the second stage detector is a more computationally expensive algorithm with better performance in terms of interference rejection. In all of those mentioned here, the AMF serves as the the first stage detector.

In [95], a two stage AFM-GLRT detector is proposed, with the goal of reducing the computational burden of Kelly's GLRT detector while also offsetting the poor interference rejection properties of the AMF. [94] describes the Adaptive Sidelobe Blanker (ASB), which is a two stage AMF-ACE detector. The ASB maintains the rejection benefits of the ACE algorithm while mitigating its high computational complexity. Additionally, the two detector thresholds can be tailored to help offset the relatively poor matched target detection performance of the ACE detector. In [85], De Maio describes a two stage approach to improve the efficiency of the adaptive Rao detector (although this does not effectively mitigate its reliance of sample support for detection). Finally, the authors who proposed ABORT in [68] also proposed using it as the second stage of a two stage detector with the AMF, using AMF as the primary detector and the ABORT algorithm to reject mismatched target returns.

3.5 Covariance Modeling and Estimation

Adaptive detection algorithms are designed to maintain CFAR behavior regardless of the clutter covariance. When deriving any such algorithm, it is generally required to prove that CFAR behavior mathematically, as was done with each algorithm presented in Chapter 3.4. However, demonstrating the CFAR nature of a detector experimentally and mapping out its detection probability over a range of parameters are essential before using such algorithms in any real-world systems. Doing so requires conducting extensive testing. It is generally preferable (and necessary) to conduct most of the detector characterization on simulated data. This is due in part to the expense and difficulty of obtaining the massive quantities of data needed to verify performance across the broad range of environments in which detectors may be used. Additionally, simulation allows the statistical properties of the signal to be directly generated, allowing for much more robust characterization.

Part of that characterization needs to be evaluation of the detector's performance for different clutter covariance matrix (CCM) structures. This will give key insights into the detector's clutter rejection characteristics. Additionally, since adaptive detection algorithms rely on covariance matrix estimation to decorrelate interference data, simulated CCM structures are needed to determine the effectiveness of the selected estimation method. I will now take some time to describe several common CCM models and estimation methods. In Chapters 6 and 7, I use these methods to characterize both a ML-enhanced adaptive detection algorithm and a technique for using generative ML to produce whitening filters with limited sample support.

Before discussing specific CCM models, I will note several general properties of covariance matrices [97, 66]. First, covariance matrices are always positive and semi-definite. The diagonal is always equal to the variance of the data. Additionally, they are always either symmetric (for a real matrix) or Hermitian (for a complex matrix).

This results in structures that follow the form:

$$C = \begin{bmatrix} \sigma & w_{1,2} - jc_{1,2} & w_{1,3} - jc_{1,3} & . \ . \ . & w_{1,N} - jc_{1,N} \\ w_{1,2} + jc_{1,2} & \sigma & w_{2,3} - jc_{2,3} & . \ . \ . & w_{2,N} - jc_{2,N} \\ w_{1,3} + jc_{1,3} & w_{2,3} + jc_{2,3} & \sigma & . \ . \ . & w_{3,N} - jc_{3,N} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ w_{1,N} + jc_{1,N} & w_{2,N} + jc_{2,N} & w_{3,N} + jc_{3,N} & . \ . \ . & \sigma \end{bmatrix} \tag{3.50}$$

During initial derivation and characterization, many of the adaptive detection algorithms presented in the literature are tested on relatively simple covariance models [13, 81]. This is done to simplify the simulation process and allow for rapid testing of the proposed algorithms. Specifically, they assume that the CCM is Toeplitz in addition to its other properties. Teoplitz matrices have constant values along each descending diagonal, and the combination of Hermetian and Toeplitz results in highly structured CCMs of the form:

$$C = \begin{bmatrix} \sigma & w_{1,2} - jc_{1,2} & w_{1,3} - jc_{1,3} & w_{1,4} - jc_{1,4} & w_{1,5} - jc_{1,5} & \cdot \;\cdot\;\cdot \\ w_{1,2} + jc_{1,2} & \sigma & w_{1,2} - jc_{1,2} & w_{1,3} - jc_{1,3} & w_{1,4} - jc_{1,4} & \\ w_{1,3} + jc_{1,3} & w_{1,2} + jc_{1,2} & \sigma & w_{1,2} - jc_{1,2} & w_{1,3} - jc_{1,3} & \\ w_{1,4} + jc_{1,4} & w_{1,3} + jc_{1,3} & w_{1,2} + jc_{1,2} & \sigma & w_{1,2} - jc_{1,2} & \\ w_{1,5} + jc_{1,5} & w_{1,4} + jc_{1,4} & w_{1,3} + jc_{1,3} & w_{1,2} + jc_{1,2} & \sigma & \\ \cdot & & & & & \\ \cdot & & & & & \\ \cdot & & & & & \end{bmatrix} \tag{3.51}$$

Using this structure makes sense when one considers the nature of the CCM in radar systems. For a given clutter patch, the covariance of the clutter returns reflect the level of joint variability between discrete points sampled from a continuous return signal at different times. For simple correlated clutter, it is reasonable to assume that any given sample will be perfectly correlated with itself, and it will then have reduced correlation with other samples more removed in time. As a result, the family of Toeplitz CCM models are generally defined by a correlation coefficient and a decay rate. The simplest Toeplitz model uses a linear decay:

$$\mathbf{M}_{ij} = 1 - \frac{|i-j|}{N} \tag{3.52}$$

where i is the row index, j is the column index, M_{ij} refers to the value of covariance matrix entry at position (i, j), and $N = max(i) = max(j)$ is the length of the radar return signal. A more common model assumes an exponential decay in correlation

[98]:

$$\mathbf{M}_{ij} = \rho^{|i-j|} \tag{3.53}$$

where ρ is the one-lag correlation coefficient. Note that both models in Equations (3.52) and (3.53) produce real-valued CCMs.

These CCM structures are effective for modeling simple clutter scenarios, but fail to capture the complex structures and phase information caused by Doppler components in the clutter frequently encountered in STAP radar. Various more realistic models have been developed, with the Ward model proposed in [51] being the most broadly used. When developing this model, Ward assumes that one is looking at a set of clutter rings, each at a constant distance from the radar platform, which are then subdivided into discrete clutter patches. Each clutter patch can then be represented by an elevation and azimuth, denoted θ_k and ϕ_l, relative to the platform. Looking at a particular clutter patch, which will be referred to as the kl-th clutter patch, Ward then defines a representation of the spatial frequency. The spatial frequency of the kl-th clutter patch is:

$$\nu_{kl} = \frac{d}{\lambda} \cos\theta_k \sin\phi_l \tag{3.54}$$

where d is the spacing between antenna elements in the array and λ is the signal wavelength. Assuming then that the clutter is a combination of range ambiguous returns and a large number of evenly distributed clutter sources, the clutter component of a space-time snapshot can then be defined as:

$$\chi_c = \sum_{k=1}^{N_r} \sum_{l=1}^{N_c} \alpha_{kl} \mathbf{v}(\nu_{kl}, \bar{\omega}_{kl}) = \sum_{k=1}^{N_r} \sum_{l=1}^{N_c} \alpha_{kl} \mathbf{v}_{kl} \tag{3.55}$$

where α_{kl} is the amplitude of the returns from the kl-th clutter patch (defined as a random variable), $\mathbf{v}_{kl} = \mathbf{v}(\nu_{kl}, \bar{\omega}_{kl})$ is the space-time steering vector of the kl-th clutter patch, $\bar{\omega}_{kl}$ is the normalized Doppler frequency of the kl-th clutter patch, N_r is the number of range ambiguities, and N_c is the number of clutter sources.

Ward assumes a constant Gamma model for clutter and defines the clutter-to-noise ration, ξ_{kl} such that:

$$E[|\alpha_{kl}|^2] = \sigma^2 \xi_{kl} \tag{3.56}$$

where σ^2 is the noise power. Finally, this allows the clutter covariance to be modeled in terms of the noise power, clutter power, and clutter space-time steering vector as:

$$\mathbf{M} = E[\chi_c \chi_c^H] = \sigma^2 \sum_{k=1}^{N_r} \sum_{l=1}^{N_c} \xi_{kl} \mathbf{v}_{kl} \mathbf{v}_{kl}^H \tag{3.57}$$

For testing an adaptive detection algorithm, once the CCM and clutter models have been chosen and implemented, a CCM estimation method must be selected. As previously noted, K secondary samples (denoted $\mathbf{z}_k$) containing only interference are used in these approximation techniques. The most straightforward and widely used method is derived from the definition of the second moment and assumes that all support samples are independent and identically distributed (i.i.d) [99]. Recall that the second moment of a random variable $\mathbf{z}$ is defined as:

$$\mathbf{\Sigma} = E[(\mathbf{z} - \mu)(\mathbf{z} - \mu)^H] \tag{3.58}$$

and that the definition of the expected value is:

$$E[\mathbf{z}] = \sum_{i=1}^{\infty} \mathbf{z}_i p_i \tag{3.59}$$

where $\mathbf{z}_i$ represents a possible value of $\mathbf{z}$ and p_i is the probability of $\mathbf{z}_i$. If the distribution of $\mathbf{z}$ is not known, but a number, K, of i.i.d. samples of $\mathbf{z}$ are available, then the probability of each $\mathbf{z}_i$ can be assumed to be approximately $p_i = \frac{1}{K}$. The expected value of $\mathbf{z}$ can then be approximated as:

$$E[\mathbf{z}] = \mu \approx \frac{1}{K}\sum_{i=1}^{K} \mathbf{z}_i \tag{3.60}$$

Extending this approximation to the second moment of $\mathbf{z}$ we get:

$$\mathbf{M} \approx \frac{1}{K}\sum_{k=1}^{K} (\mathbf{z}_k - \mu)(\mathbf{z}_k - \mu)^H \tag{3.61}$$

This approach will perfectly reconstruct the CCM as $K \to \infty$. This is not practical, however, as the more support samples are required the more time and computation are required for processing each radar return. Minimizing the amount of support data required for adaptive detection is a major research focus area. The approach in Equation (3.61) degrades quickly as K becomes small and requires at least $K = 2N$ to achieve a useful reconstruction [51].

Significant work has been done in the field to develop alternative estimation methods that either improve the fidelity of the estimation or that reduce the sample support requirements. Many attempt to do so by exploiting structural properties of covariance matrices. The Toeplitz and Hermetian assumptions have been used extensively in the literature with low-rank matrix approximations [100, 101, 102] and iterative methods [103]. A number of adaptive detection algorithms have also been derived under the assumption that CCMs will be persymmetric [104, 105, 106]. The low-rank and sparse properties of clutter covariance have also been exploited to solve this problem

[107, 108, 109, 110].

Recursive CCM estimation methods are also used extensively in the literature. In 2002, Conte *et al.* proposed a recursion method that alternatively locked the texture values and the CCM estimate and solved for the other. [111]. The fixed-point estimator is used to iteratively estimate the CCM in [112, 113]. There are multi-step estimators like the work of Ali *et al.* [114], which estimates the number of clusters, parameter values (angle of arrival, angle spread, and power contribution of clusters), and then uses this information to estimate a CCM for other frequency ranges. Finally, a cognitive radar system in [115] uses cosine similarity of CCMs as part of an adaptive detection system.

3.6 The Kalman Filter

I will close out this chapter with a brief discussion of the Kalman filter. This signal processing technique was first proposed by Kalman in [116], and variations of it have become some of the most widely used state estimation tools for tracking and navigation around the world. The original derivation of the KF assumed the system it was modeling was linear. This was a powerful tool, but it doesn't accurately reflect many real world systems [117, 118]. Various modifications have been proposed to allow the KF to extend to nonlinear systems, most notably the EKF and the Unscented Kalman Filter (UKF) [119, 120]. I specifically used the EKF in Chapter 5 when developing and testing an ML-based tool for correcting drift errors in tracking and navigation systems.

In its original formulation, the KF attempts to approximate a simple state model and use that model to predict the next state of the system under observation. This is accomplished iteratively through a cycle of predicting the next state, observing that state, and then updating the KF based on the error between its prediction and observation. Two models are initially defined: 1) a state model and 2) a measurement model [119]. As its name suggests, the state model is meant to approximate the state of the system being observed. It is assumed that this model can be represented as a probability distribution (usually Gaussian) that can be fully characterized by a mean μ and covariance R. This model can be represented as:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), \mathbf{b}(k), k) \tag{3.62}$$

In Equation (3.62), $\mathbf{x}(k)$ is the state at timestep k, $\mathbf{u}(k)$ is an input vector, and $\mathbf{b}(k)$ is the modelling error. The prediction step of the KF simply consists of propagating this model to the next timestep and calculating the mean and covariance as:

$$\mu(k+1|k) = E[f(\mathbf{x}(k), \mathbf{u}(k), \mathbf{b}(k), k)] \tag{3.63}$$

$$\mathbf{R}(k+1|k) = E[\{\mathbf{x}(k+1) - \mu(k+1|k)\} \times \{\mathbf{x}(k+1) - \mu(k+1|k)\}] \tag{3.64}$$

where $\mu(k+1|k)$ is the state mean at timestep $k+1$ given the state model at time k, $R(k+1|k)$ is the state covariance at timestep $k+1$ given the state model at time k, and $E(\bullet)$ is the expected value function.

Next and observation of the system is made:

$$\mathbf{z}(k+1) = g(\mathbf{x}(k), \mathbf{u}(k), k) + \mathbf{w}(k) \tag{3.65}$$

where $\mathbf{z}(k)$ is the observation at timestep k and $\mathbf{w}(k)$ is the observation error. In detection and tracking, this would take the form of receiving an update from a sensor system like a radar or GPS unit.

This observation is then used to update the state model using a linear, minimum, mean-squared error estimator:

$$\hat{\mathbf{z}}(k+1) = E[g(\mu(k+1|k), \mathbf{u}(k+1), k+1)] \tag{3.66}$$

$$\mathbf{v}(k+1) = \mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k) \tag{3.67}$$

$$\mu(k+1|k+1) = \mu(k+1|k) + \mathbf{G}_K(k+1)\mathbf{v}(k+1) \tag{3.68}$$

$$\mathbf{R}(k+1|k+1) = \mathbf{R}(k+1|k) - \mathbf{G}_K(k+1)\mathbf{P}(k+1)\mathbf{G}_K^T(k+1) \tag{3.69}$$

In these equations, $\mathbf{P}(k)$ refers to the covariance of $\mathbf{v}k$ at timestep k and $\mathbf{G}_K$ is known as the Kalman gain. The Kalman gain weights the model update and is designed to quantify how closely the state model matches the real system based on its prediction and observations. High Kalman gains reflect poor matching between the model and the system, and they result in larger updates to the state model. This gain is calculated at each update step as:

$$\mathbf{G}_K(k+1) = \mathbf{R}_{\mu\hat{z}}(k+1)\mathbf{P}^{-1}(k+1) \tag{3.70}$$

where $\mathbf{R}_{\mu\hat{z}}$ is the cross covariance between μ and $\hat{\mathbf{z}}$.

The EKF is a simple extension of the KF that allows it to be applied to non-linear systems. It applies a common technique known as linearization. If we assume that f is a non-linear function, then we can expand the error in the state model prediction using a Taylor series expansion. It is then assumed that all second order and higher terms in the expansion have a negligible contribution to the error, resulting in an approximation of the state prediction error:

$$\mathbf{x}(k+1) - \mu(k+1|k) \approx \mathbf{F}_x[\mathbf{x}(k) - \mu(k|k)] + \mathbf{F}_b\mathbf{b}(k) \tag{3.71}$$

where $\mathbf{F}_x$ is the Jacobian of $f(\bullet)$ with respect to $\mathbf{x}(k)$ and F_b is the Jacobian of $f(\bullet)$ with respect to $\mathbf{b}(k)$. That approximation allows us to reformulate the state predictions as:

$$\mu(k+1|k) = f(\mu(k|k), \mathbf{u}(k), 0, k) \tag{3.72}$$

$$\mathbf{R}(k+1|k) = \mathbf{F}_x\mathbf{R}(k|k)\mathbf{F}_x^T + \mathbf{F}_bQ(k+1)\mathbf{F}_b^T \tag{3.73}$$

where $\mathbf{Q}(k)$ is the covariance of $\mathbf{b}(k)$.

The use of linearization has proven to be highly effective while not adding significant computational overhead to the EKF. This has led the EKF to be the gold standard for state estimation in many fields like tracking and navigation. However, it has significant limitations due to the truncation of higher order terms in the Taylor series expansion. That truncation leads to substantially degraded performance of the EKF when the system being modeled is significantly non-linear.

Chapter 4

The Self-Explaining Decision Architecture

4.1 Introduction

In this chapter, I will describe my ideation, development, and proof of concept testing for a novel decision-making architecture designed to allow autonomous systems to provide human-readable explanations for their outputs in real time. This architecture was built to use the concepts of ML interpretability, explainability, attention, and hierarchical decision-making to try and create an extensible system capable of breaking down complex actions into sets of decisions and replicate human explanations for those decisions. I will first describe the architecture itself. Then I will detail its initial implementation before discussing how this design was tested. This work was published in [9] and a provisional patent was filed for the architecture in 2022 (the process to convert to non-provisional US patent is currently underway at the time of writing). Additionally, I have worked with several colleagues on two separate efforts to implement this architecture for applications to data fusion in computer vision [10] and Command and Control (C2) as part of the Air Force Research Laboratory Contract FA8750-21-C-0105.

My earliest forays into the realm of ML had me developing basic neural network architectures and employing supervised training to teach them to solve various classification and regression problems. After that, I looked for ways to apply these concepts and models to my work with the Air Force's C2 community. In particular, my team was exploring how to use ML to augment sensing, data fusion, and decision-making processes. However, I kept getting the same two concerns from each operational group I talked to about using ML in the C2 domain: 1) ML is not consistent enough to make decisions in mission or safety critical systems, and 2) decisions made by black box methods like neural networks can't be trusted. This led me to begin investigating ways to overcome these two challenges.

The concerns expressed by operators towards consistency of ML behavior were all rooted in the probabilistic nature of ML systems. More traditional algorithms either have a set of strictly defined states established by their developers or are defined by a set of deterministic equations that allow for easy mapping between inputs and outputs. ML algorithms, on the other hand, are trained using input-label pairs and their behavior cannot be readily mapped unless the model is trivially small. This can result in unexpected behaviors manifesting that can be difficult to troubleshoot. Even if a ML system has a much higher performance record than a deterministic algorithm, the inability to point to a simple calculation to explain why seemingly anomalous behaviors occur cause them to be perceived as a liability in many applications.

Similarly, concerns over the trustworthiness of ML models and their black box nature are rooted in their probabilistic behavior and complex structure. Rather than defining a full mathematical model for complex systems, ML systems attempt to learn models that approximate those systems as closely as possible. Again, this can result in behaviors that are anomalous, or at least seem anomalous to people. Human decision makers and deterministic algorithms often behave erroneously, but they can be questioned or evaluated in real time to mitigate the effect of such errors. However, the massive size of many ML systems, along with the abstract nature of the relationships they can learn, often make it infeasible to diagnose why ML systems arrive at the outputs they do in real time. In fact, it can even make it extremely difficult to diagnose model issues post hoc. The inability to question model outputs or quickly and consistently diagnose issues limits user trust in the ability of ML system to make critical decisions.

In exploring how to overcome these issues, I discovered the notions of ML interpretability and explainability. Each concept seeks to mitigate the issues associated with the ML black box. Interpretability does this by providing insights into the inner workings of the model itself so that users can understand mathematically why systems arrive at their outputs. This is commonly done by either intensive mapping of the internal structures of the model or by simplifying the model to make it intrinsically easier to comprehend. Simplifying models can also result in more consistent model behavior. Explainability, on the other hand, seeks to shed light on a model's behavior by translating its abstract decision-making process into a format that is readily understood by human observers. This can take the form of things like attention masks for images or text explanations output by a model. This is a more abstract concept than interpretability, but also has considerably more potential for improving operator trust.

This was a critical point in my early dissertation research that has helped to shape my work since. The idea of simplifying models to improve consistency led me to develop a structured approach to break down data processing and decision-making into component elements that can be performed by a combination of specialized ML models or deterministic systems. This set me on the path of designing ML systems based on first principles: pairing ML with traditional signal processing, augmenting existing techniques wherever possible, and replacing proven methods with ML only when it makes sense to do so.

The notion of translating ML decision processes into human terms caused me to research how humans make decisions and to find a combination of ML attention and architectural methods to replicate those. I found that human explanations can effectively be reduced to two elements: 1) attention (what someone was paying attention to) and 2) reasoning (how someone's observation translated into a given decision). The idea of ML attention has been well defined through attention mechanisms (as discussed in Chapter 2.3). Additional work has been done to integrate reasoning components into decision-making neural networks for generating post-hoc explanations [41, 42]. Ultimately, I was able to build on these concepts and design an architecture to enable ML systems to explain their own decisions in real time: the Self-Explaining Decision Architecture (SEDA). SEDA provides a framework for ML explainability and seeks to emulate the way people explain their decisions, enabling users to rapidly understand the system's outputs.

SEDA is designed to be a general architecture, capable of supporting a wide range of applications. Each element of it can be implemented with different ML or deterministic structures. In this chapter I will describe both the general SEDA framework and an initial implementation of it that makes use of a combined CNN and LSTM deep neural network (DNN) with attention mechanisms to fuse both spatial and temporal data. The DNN is followed by a decision hierarchy that recommends actions based on the output from the CNN-LSTM DNN. Finally, both the system attention and the path through the decision hierarchy are used to output a text-based explanation of the system's decisions. Using these two elements to create decision explanations is intended to provide information on the data elements the system focused on when making a decision as well as the reasoning that was used to select the recommended course of action. While I personally designed SEDA and developed its initial implementation, I worked closely with a colleague, Brian Sun, to test the system's performance. I will take care to note which portions of the testing were designed and run by each of us.

4.2 Self-Explaining Decision Architecture

The goal of my work on SEDA was not to create a specific tool. Rather, as its name might suggest, I sought to establish an architectural approach to streamline ML integration into critical decision-making systems by allowing them to be questioned by users in real time. This approach required me to break down the elements of sensing, decision-making, and explanations into component elements and then orchestrate them to form a cohesive system. I will now describe the SEDA framework, depicted in Figure 4.1. Note that the architecture is designed to be generalized, with multiple ways to implement each component. This was done to allow it to be as broadly and easily applicable as possible. SEDA consists of five primary subsystems: data encoding, feature extraction, sequence interpretation, decision generation, and explanation generation.

The feature extraction and sequence interpretation subsystems together form a compound module for detecting patterns in spatiotemporal data. SEDA was originally designed for processing spatiotemporal data due to that data format's broad applicability in the C2 domain. However, SEDA is readily adaptable to other application spaces and data formats. For instance, the sequence interpretation component could be readily implemented with a transformer to accommodate non-sequential patterns.

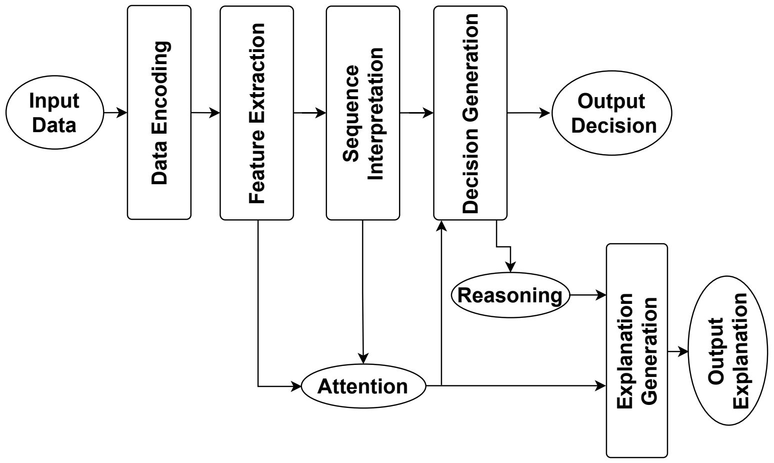


Figure 4.1: High-level diagram of the Self-Explaining Decision Architecture (SEDA).

Additionally, the attention mechanisms used to generate explanations can be readily substituted or adapted to work with most ML models.

4.2.1 Data-Encoding Subsystem

The data-encoding subsystem performs any preprocessing needed to ensure that the data is ready to be passed into the feature extraction component of SEDA. It includes operations like data cleaning and encoding non-numeric information. The exact preprocessing performed by the data-encoding subsystem will vary significantly depending on the application.

4.2.2 Feature Extraction Subsystem

The feature extraction subsystem serves to identify important features in the data that should be used in the decision-making process. These features are then passed on to the sequence interpretation subsystem. It is also responsible for keeping track of and storing information about what the subsystem was attending to when it was extracting features (via the use of attention mechanisms). This attention information is used to help generate decision explanations, and may also be used directly by the decision-generation subsystem in the decision-making process.

4.2.3 Sequence Interpretation Subsystem

The sequence interpretation subsystem serves to allow SEDA to identify patterns in sequential data, such as time-series data. It enables more complex datasets to be used and more complex decisions to be made. The observed patterns are provided to the decision-generation subsystem to initiate the decision-making process. This subsystem is also responsible for tracking and storing the impact of each sequence entry on the patterns it identifies. As with the feature extraction subsystem, this is accomplished using attention mechanisms. That information is used by the explanation-generation subsystem to help explain the decisions that are made. It is also made available to the decision-generation subsystem for possible use in the decision-making process. Note that SEDA is still applicable when sequential data is not used by simply assuming a sequence length of one.

4.2.4 Decision-Generation Subsystem

Together, the data encoding, feature extraction, and sequence interpretation subsystems seek to emulate the way in which people make observations by taking in information, processing it, and detecting patterns. They are followed by the decision-generation subsystem, which seeks to emulate the way in which people make decisions based on their observations. It achieves this by breaking down complex decisions into a collection of simpler ones, which it selects between based on the specific patterns that have been observed in the data as well as contextual information garnered from what the previous subsystems deemed to be important. This subsystem is designed to have some underlying reasoning behind its decision-making process. It does this using the output from the sequence interpretation subsystem along with the combined system attention collected by the feature extraction and sequence interpretation subsystems. The decision-generation subsystem must also record the reasoning behind its decisions and provide that information to the explanation-generation subsystem. The subsystem's decision can then either be output as a recommendation or directly executed, depending on the application and level of autonomy desired.

4.2.5 Explanation-Generation Subsystem

Finally, the combination of system attention (aggregated from the attention stored by the feature extraction and sequence interpretation subsystems) and system reasoning (from the decision-generation subsystem) are provided to the explanation-generation subsystem. The structure and complexity of this subsystem will vary significantly depending on the application and implementation. However, a design might involve generating decision reasoning to explain why a particular decision was reached. For example, an autonomous vehicle might explain avoiding an obstacle: "HAZARD DETECTED IN PATH → ALTERING PATH TO AVOID → CHANGING LANES AND REDUCING SPEED." If the user required additional information, attention may be used to provide details of what the system was focusing on when it made a given decision. Building on the previous example: "A TIRE HAS BEEN IDENTIFIED IN THE ROAD 200 FEET AHEAD OF US; GIVEN OUR CURRENT TRAJECTORY AND SPEED, ALONG WITH THE SIZE OF THE OBSTACLE, IT IS EXPECTED THAT SEVERE DAMAGE WILL BE DONE TO THIS VEHICLE UNLESS WE ALTER COURSE TO DRIVE AROUND THE OBSTACLE.".

4.3 Preliminary SEDA Implementation

I will now describe a preliminary implementation of SEDA intended to serve as a proof of concept. This preliminary implementation makes use of all components of SEDA from end to end, but was designed and implemented for simple, representative test cases. The prototype system was developed using Python and Tensorflow. It has been tested both with the openly available and widely used Modified National Institute of Standards and Technology (MNIST) image dataset [121] and with a simple simulation time-series image dataset developed in Matlab by Brian Sun. The MNIST dataset contains images of hand written digits and is widely used in proof of concept testing for CNN-based architectures. Each of these datasets uses only a single channel (grayscale). Due to the nature of the architecture and the simplicity of the data used for proof of concept, little to no modification of the prototype system was required for each set of tests.

4.3.1 Data Encoding Implementation

The prototype data-encoding subsystem performs minimal preprocessing. In the case of the MNIST images, this subsystem divides each image into four sub-images then arranges the sub-images into a sequence because images inherently contain spatial data but not sequential data. In the case of the simulated time-series data, no encoding is necessary as this data was custom generated for testing this prototype.

4.3.2 Feature Extraction Implementation

The prototype feature extraction subsystem is implemented as an attention-based CNN. This CNN consisted of three convolutional layers, with attention modules after the first and third, a maxpooling layer after the second convolutional layer, and a flattenting layer. The specific model parameters (such as the number of nodes in each layer) varies somewhat between the MNIST and custom time-series datasets, so details for the model configuration will be provided shortly in Section 4.3.6 as part of the experimental setup. In this implementation, the CNN is used to learn relevant spatial relationships in the data, which are then passed to the sequence interpretation subsystem as features. It utilizes multi-layer attention in the CNN, as depicted in Figure 4.2.

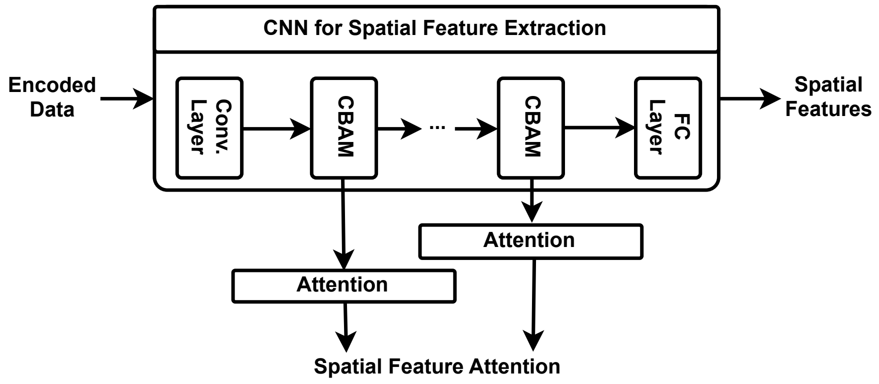


Figure 4.2: Prototype SEDA feature extraction subsystem.

Convolutional block attention modules (CBAMs) [23] are placed after various convolutional layers in the CNN to gather attention information from the network. Each CBAM layer generates two attention masks, one that reflects the overall importance of each kernel in the preceding convolutional layer (channel attention) and one that reflects the importance of each pixel (spatial attention). These masks are multiplied together to generate a $D \times W \times H$ attention matrix, where D is equal to the number of kernels in the preceding convolutional layer, and W and H are, respectively, the number of pixels in the height and width dimensions of the image at that point in the network. The CBAM was described in more detail in Chapter 2.5.1.

Each time a sequence is passed through the system the spatial attention matrix is saved as part of the system attention. Early layers in a convolutional network learn to recognize straightforward spatial relationships in images. The kernels in these layers will often learn to perform standard image processing like edge detection or color identification. Deeper convolutional layers impact wider regions of an image and will tend to learn more abstract patterns. While only one attention layer is necessary to satisfy the requirements of SEDA, utilizing attention layers at multiple points throughout the network enables more thorough explanations to be generated by analyzing some combination of simpler and more complex forms of attention.

4.3.3 Sequence Interpretation Implementation

The prototype sequence interpretation subsystem consists of an LSTM network with Luong attention [25], as shown in Figure 4.3. It is comprised of a single LSTM layer with attention followed by two fully connected layers. As with the feature extraction subsystem, the specific model parameters vary for the two test cases that were used for proof of concept, so details of the model configuration are provided in Chapter 4.3.6 as part of the experimental setup. Note that Luong attention is commonly referred to as dot product attention and was described in Chapter 2.3. The LSTM ingests feature maps, learns temporal (or sequential) relationships, and makes classifications based on those relationships. The attention mechanism enables the LSTM to learn sequential relationships that may include gaps and serves as another element in the generation of explanations.

4.3.4 Decision Generation Implementation

The prototype decision-generation subsystem uses a decision hierarchy. Decision hierarchies allow systems a certain degree of freedom to decide between multiple pos-

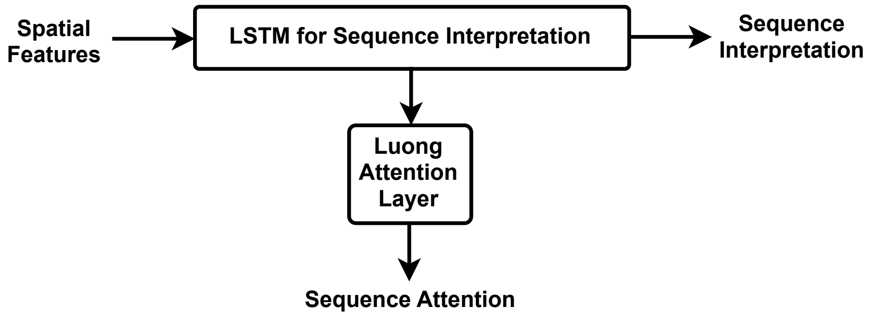


Figure 4.3: Prototype SEDA sequence interpretation subsystem.

sible actions in a given situation, rather than enforcing a strict decision path. Their hierarchical nature also makes the underlying reasoning behind decisions straightforward to develop and verify by referring to the path(s) traversed through the decision hierarchy. These structure have historically been used to to break down complex behaviors into simpler, directly executable ones, as described in [122]. An example of this is depicted in Figure 4.4. I believe this structure lends itself well to modeling human decision-making and served as the inspiration for the decision generation subsystem developed for this work. The implementation used in this work is depicted in Figure 4.5.

The prototype consists of a three-layer hierarchy. There are three decision options in the first decision layer, five in the second, and seven in the third. Each layer utilizes a simple feedforward, fully connected DNN with two hidden layers to arrive at its decisions. The DNNs are trained to interpret attention information to recognize key contextual details in the data, produce a human-readable message explaining its interpretation of the provided attention information, and arrive at a decision based on the combination of those contextual details and the output of the previous layer of the hierarchy. This emulates the way people make decisions based on observations of the world around them by breaking down complex or abstract decisions into a se-

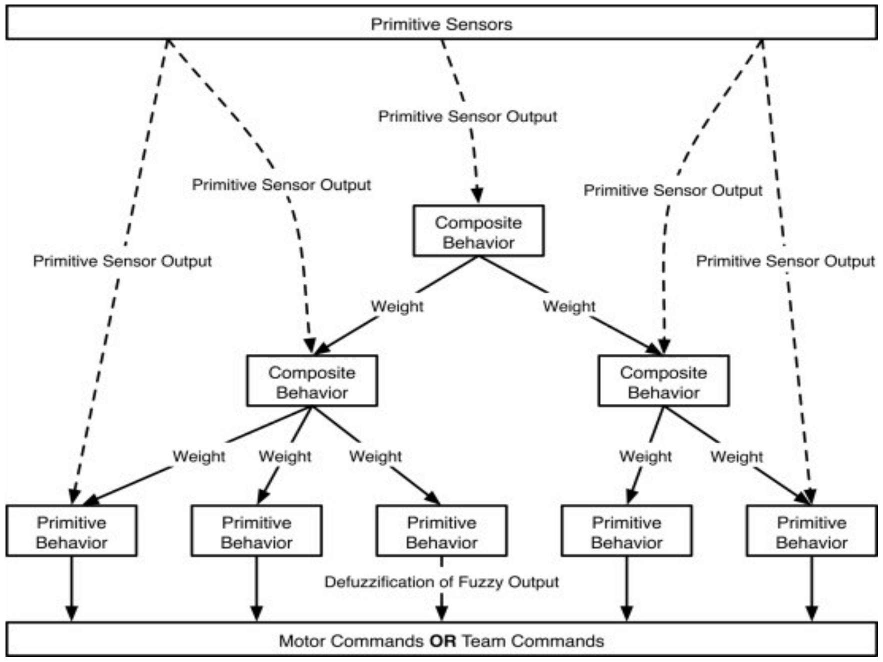


Figure 4.4: Example decision hierarchy from [122].

ries of increasingly simple ones and selecting a course of action from amongst their perceived options based on the current context. In this prototype, each decision layer was provided with the output from the previous layer (or the output of the sequence interpretation LSTM in the case of the first layer) and one set of attention information: CBAM attention from the first feature extraction convolutional layer was used by the top layer of the decision hierarchy, CBAM attention from the last feature extraction convolutional layer was used by the second layer of the decision hierarchy, and Luong attention from the sequence interpretation LSTM was used by the final layer of the decision hierarchy.

4.3.5 Explanation Generation Implementation

This subsystem uses the combination of attention and reasoning to provide real-time, human-readable explanations of system decisions. A high level view is shown in

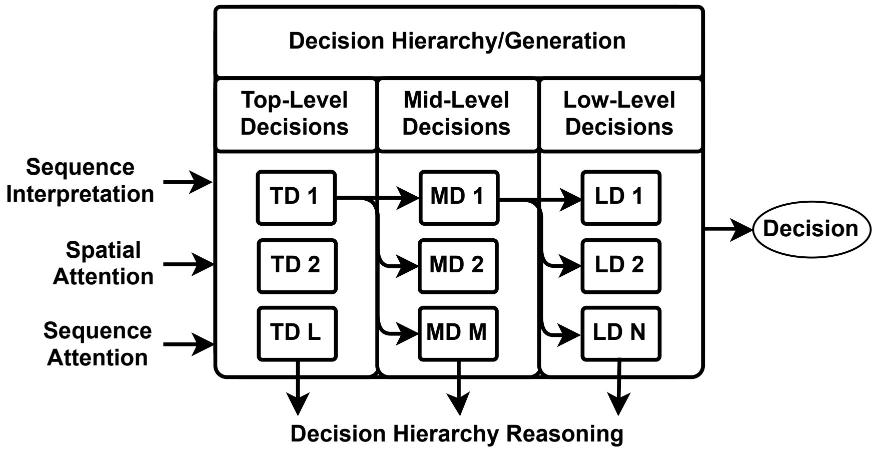


Figure 4.5: Prototype SEDA decision-hierarchy subsystem.

Figure 4.6. The decision reasoning, based on the path traversed through the decision hierarchy, is used to explain why a particular decision was reached. In this prototype explanation generation subsystem, decisions are text-based and written to a .txt file. Each decision level in the hierarchy is explained with one sentence. A secondary statement for the second and third decision layers are used to provide attention information. The format of the explanations is:

1. Decision number:
   (a) Top-level decision
      i. Contextualized spatial attention
   (b) Mid-level decision
      i. Contextualized sequence attention
   (c) Low-level decision

The goal for this explanation structure was to provide an easily readable structure and provide enough information to the user without overwhelming them. This may not have been necessary for the simple problems used to evaluate the prototype, but will be essential to any fully operational version of SEDA, particularly if it were used in the C2 domain as intended.

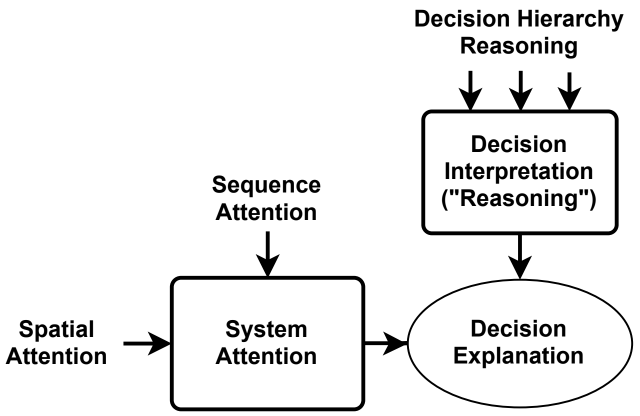
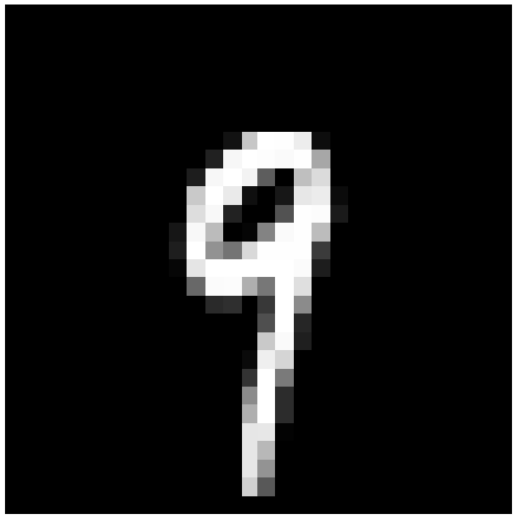
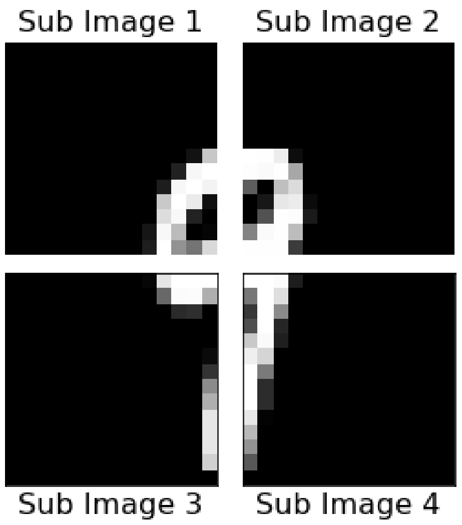


Figure 4.6: Prototype SEDA explanation-generation subsystem.

4.3.6 Experimental Setup

As previously mentioned, two datasets were used to the evaluate the prototype implementation of SEDA: 1) the MNIST dataset and 2) a custom time-series simulation developed by Brian Sun. For both datasets, the inputs to the system are structured as sequences of images. This was done to ensure that all elements of the SEDA framework could be tested. In the case of MNIST, the data-encoding subsystem divides each $1\times28\times28$ grayscale image of a handwritten numeral into four $1\times14\times14$ sub-images as shown in Figure 4.7 for a numeral '9', and each group of four sub-images is treated as a single sequence. Note that this provides a sequential component to the data, but not a temporal one. MNIST was selected at the time because it was a large, well-groomed, and thoroughly analyzed dataset that was widely used in the literature to characterize state-of-the-art convolutional networks, making it ideal for testing the feature extraction subsystem. Choosing such a dataset allowed us to isolate variables for more targeted testing of the architecture since MNIST does not have many common data issues (corruption/missing data, unbalanced classes, etc.) that can interfere with training ML systems.

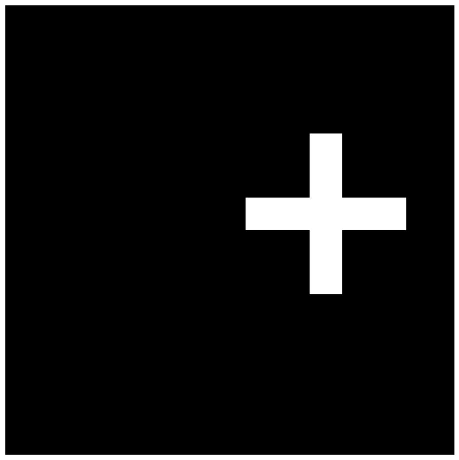

(a) Original MNIST Image

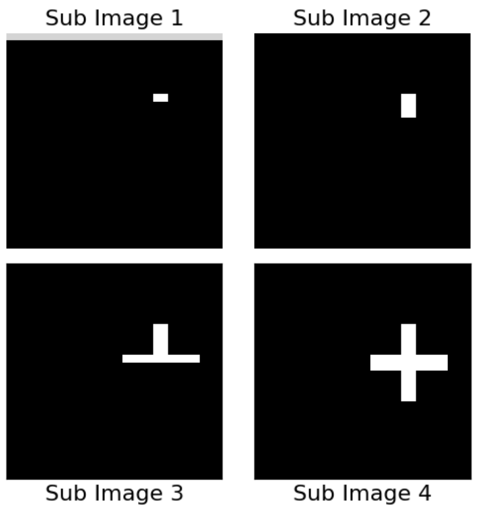


(b) Processed Image Sequence

Figure 4.7: Example MNIST input data item showing original (a), and divided into sequence of sub-images (b).

The custom time-series dataset was generated as a sequence of images in which a target gradually appears over time. Each image is 1×28×28 and somewhere within that image a 10×10 sub-image for one of two target patterns appears: 1) a cross pattern or 2) a chevron pattern. At each time step, a 1×28×28 snapshot of the scene is taken and a target progressively appears from top to bottom, with more target information being provided to the prototype SEDA system over time. Note that this provides both a sequential and temporal component to the data. Using a simple temporal dataset allowed for more robust testing of the SEDA prototype's sequence interpretation sub-system. An example of the custom dataset is depicted in Figure 4.8. Note that the custom time series dataset was not used on the full SEDA system. It was purely used to train the CNN-LSTM network and analyze the attention information produced by that network. The simulator for this dataset was developed by my colleague Brian Sun, who also used it to generate the full set of custom time-series data.


(a) Complete Image

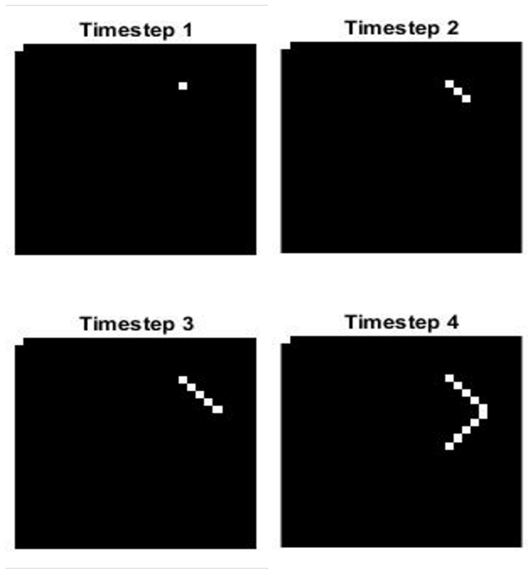


(b) Input Image Sequence

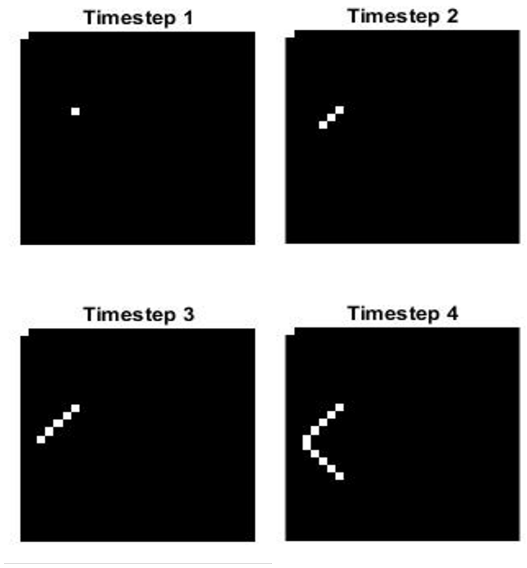
Figure 4.8: Example custom input data item showing complete (a), and sequence (b) images.

For each dataset, 60,000 images were used for training and 10,000 images were used for validation. Five sets of labels were generated for each MNIST image sequence to be used in training. Only one label set was generated for the custom time series dataset. The first set of labels were classification labels for the original images used to train the neural network that comprises the combined feature extraction and sequence interpretation subsystems. Note that these two subsystems were implemented as an integrated CNN-LSTM network and were trained together. The first set of training labels were pulled directly from the datasets. In the case of the MNIST dataset, these were just the numbers 0–9.

Four additional sets of labels were created to train the layers of the decision hierarchy for the MNIST dataset. As previously discussed, each layer of the decision hierarchy was implemented with a fully connected neural network, each of which needed to be trained. I defined and generated these labels based on a combination of



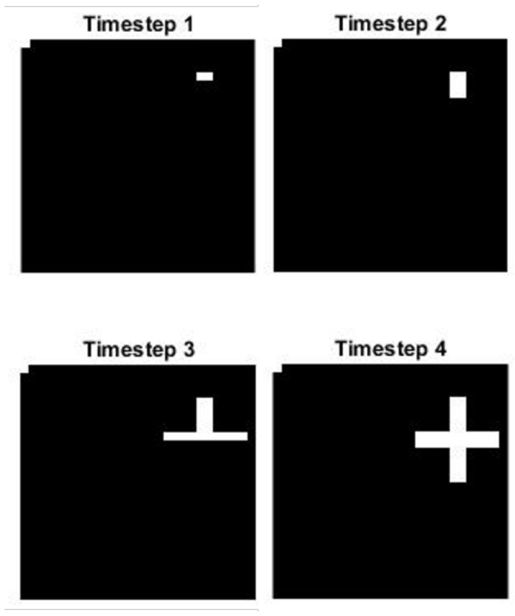
(a) Chevron Away Image



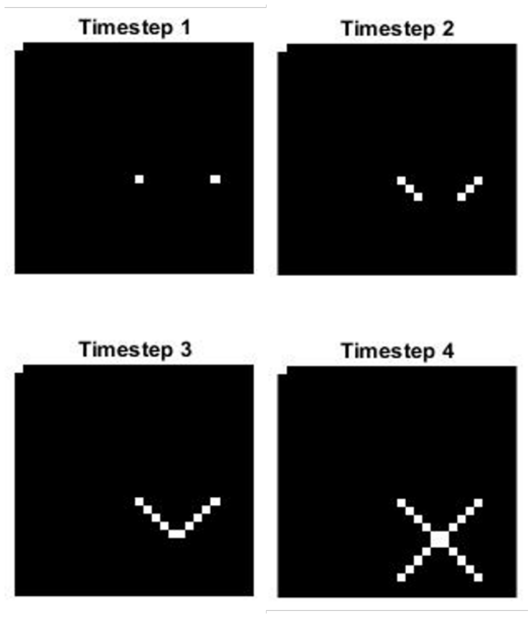
(b) Chevron Toward Image

Figure 4.9: Time series of chevron "away" (a) and "toward" (b).

properties in each image or sequence and the desired output of the previous layer of the decision hierarchy (or the output of the LSTM in the case of the first layer of the decision hierarchy). These four label sets are used to both train the system how to navigate the decision hierarchy and to interpret specific contextual information from the image sequence. In essence, these labels are used to teach the system to interpret its attention information and to establish its method of reasoning through decisions. That makes the selection of these labels and subsequent testing with them crucial to verifying the ability of the proposed architecture to provide explanations of its decisions in an intuitively understandable way. As a result, the properties selected for the generation of each label were chosen because they were readily observable by people, easily tested, statistically relevant, and represent either unique spatial or sequence properties of each grouping. Note that these labels were generated programatically using sets of rules that assumed ideal outputs from the LSTM and each layer of the decision hierarchy.



(a) Cross Away Image



(b) Cross Toward Image

Figure 4.10: Time series of cross "away" (a) and "toward" (b).

The label sets used to train the decision hierarchy layers for the MNIST dataset were generated based on specific properties of the images that were readily measurable. The label set for the top layer of the decision hierarchy was defined solely based on the output of the sequence interpretation LSTM. Two sets of labels were generated and tested for the middle decision layer. The results of each label set on the full system performance were then compared. The first was based on the presence or absence of open and closed loops in the input image. For instance, a '0' contains one closed loop, a '3' contains two open loops, and a '7' contains no loops. This label set was defined to test the ability of the prototype SEDA implementation to detect and interpret simpler positional structures and relationships within an image or higher dimensional dataset. The second label set for the middle layer was generated based on the average pixel intensity of the input images. This label set was designed to examine the prototype system's ability to identify and interpret more abstract spacial properties and relationships. The label set used to train the lowest level of the decision hierarchy was generated based on which of the sub-images in the sequence (corresponding to the quadrants of the image) had the highest average pixel intensity. This was designed to test whether the prototype system could be taught to interpret sequence-specific properties of the data. These labels are detailed in Table 4.1.

For the custom time-series data, the labels were generated directly by the simulator based on the type of target and its orientation. Four classes were defined in the label set: 1) chevron toward, 2) chevron away, 3) cross toward, 4) and cross away. The system was trained to identify chevron target patters as hostile and cross target patterns as friendly. The notion of "toward" versus "away" in this case refers to whether the observed target is determined to be moving toward or away from a location of interest (in this case the upper left corner of the simulated scene). It was arbitrarily defined for the simulation based on the orientation of the target sub-image. Specifically, a chevron target with the central point facing up or to the left is identified as moving toward the location of interest and a chevron target pointed down or to the right is identified as moving away. Equivalently, a cross target aligned with the x-y axes is determined to be traveling toward the region of interest and a cross target offset from the x-y axes by 45^o is determined to be traveling away from it. These labels for the custom dataset are shown in Table 4.2. Figure 4.9 and Figure 4.10 depict examples of the chevron and cross target data sequences, respectively.

After the data and labels were generated and the input data was preprocessed into image sequences, it was passed into the feature extraction subsystem. For the prototype SEDA implementation, the feature extraction subsystem consisted of a CNN containing three convolutional layers, with a CBAM attention layer using rectified linear (ReLU) activation after each convolutional layer, and a maxpooling layer after

Table 4.1: Decision Hierarchy Label Conditions for MNIST

Condition	**Decision**	**Value**
LSTM out = 0,1,2	Friendly	L1_label = 0
LSTM out = 3,6,7	Hostile	L1_label = 1
LSTM out = 4,5,8,9	Neutral	L1_label = 2
L1_label = 0 and closed loop	Assist	L2_label = 0
L1_label = 0 and no loop	Contact	L2_label = 1
L1_label = 0 and open loop	Assist	L2_label = 0
L1_label = 1 and closed loop	Evade	L2_label = 3
L1_label = 1 and no loop	Engage	L2_label = 2
L1_label = 1 and open loop	Engage	L2_label = 2
L1_label = 2	Ignore	L2_label = 4
L1_label = 0 and higher average	Assist	L2_label_alt = 0
L1_label = 0 and lower average	Contact	L2_label_alt = 1
L1_label = 1 and higher average	Evade	L2_label_alt = 3
L1_label = 1 and lower average	Engage	L2_label_alt = 2
L1_label = 2	Ignore	L2_label_alt = 4
L2_label = 0 and top left	Defend	L3_label = 1
L2_label = 0 and bottom left	Defend	L3_label = 1
L2_label = 0 and top right	Escort	L3_label = 0
L2_label = 0 and top right	Escort	L3_label = 0
L2_label = 1 and top left	CloseDistance	L3_label = 3
L2_label = 1 and bottom left	CloseDistance	L3_label = 3
L2_label = 1 and top right	Transmit	L3_label = 2
L2_label = 1 and bottom right	Transmit	L3_label = 2
L2_label = 2 and top left	CloseDistance	L3_label = 3
L2_label = 2 and bottom left	Strike	L3_label = 5
L2_label = 2 and top right	CloseDistance	L3_label = 3
L2_label = 2 and bottom right	Strike	L3_label = 5
L2_label = 3	IncreaseDistance	L3_label = 4
L2_label = 4	Ignore	L3_label = 5

the second CBAM layer. After the last CBAM layer, the feature data is flattened and a 0.25 dropout is applied.

The flattened feature map output by the CNN is fed into the sequence interpretation subsystem, which consists of a LSTM with four cells, and uses the sigmoid activation function. The LSTM is followed by a Luong attention layer that calculates the correlation between the output and each step in the input sequence. After the sequence attention values are determined, the output of the final LSTM layer is

Table 4.2: Custom Data Set Labels

Shape	**Orientation**	**Label**
Chevron	away	0 (Hostile-away)
Chevron	toward	1 (Hostile-toward)
Cross	away	2 (Friendly-away)
Cross	toward	3 (Friendly-toward)

Table 4.3: Implemented CNN-LSTM Feature Extraction and Sequence Interpretation Subsystems by Layer *L*

L	**Type**	**Input Shape**		**Kernel Shape**		**Output Shape**	
		MNIST	Custom	MNIST	Custom	MNIST	Custom
1	Convol.	14×14×1	28×28×1	2×2×8	3×3×8	12×12×8	26×26×8
2	CBAM	12×12×8	26×26×8	N/A		12×12×8	26×26×8
3	Convol.	12×12×8	26×26×8	2×2×8	3×3×8	10×10×8	24×24×8
4	CBAM	10×10×8	24×24×8	N/A		10×10×8	24×24×8
5	Maxpool	10×10×8	24×24×8	2×2×8	3×3×8	5×5×8	12×12×8
6	Convol.	5×5×8	12×12×8	2×2×32	3×3×32	3×3×32	10×10×32
7	CBAM	3×3×32	10×10×32	N/A		3×3×32	10×10×32
8	Flatten	3×3×32	10×10×32	N/A		288	3200
9	LSTM	4×288	4×3200	N/A		10	
10	Dense	10		N/A		50	
11	Dense	50		N/A		10	4

passed through two dense layers and a final softmax activation function to produce a single classification for the input sequence. Once each sequence is passed through sequence interpretation, the sequence attention matrix is saved as part of the full system attention. As previously noted, the feature extraction and sequence interpretation subsystems are implemented and trained as a single CNN-LSTM network. The general structure for the CNN-LSTM network remained constant for each dataset, but the size of the convolution kernels and layers differed. The architecture of this CNN-LSTM are given in Table 4.3. The network was trained over ten epochs using categorical cross entropy as the loss function with an Adam optimizer and learning rate of 0.01.

Both the output of the sequence interpretation subsystem and the spatial and sequence attention values are passed to the decision-generation subsystem. The output

Table 4.4: Decision Hierarchy Decisions and Messages

Layer	Decision	Message
1	Friendly	Entity is friendly.
	Hostile	Entity is hostile.
	Neutral/Unknown	Entity is neutral or unknown.
2	Assist	Assist entity.
	Contact	Contact entity.
	Engage	Engage entity.
	Evade	Evade entity.
	Ignore	Ignore entity.
3	Escort	Escort entity.
	Defend	Defend entity.
	Transmit	Transmit information to entity.
	Close Distance	Close distance to entity.
	Increase Distance	Increase distance to entity.
	Strike	Strike entity.
	Ignore	Ignore entity.

of the LSTM is fed into the top layer of the decision hierarchy, which then reaches one of three high-level decisions. The determination from the top-level is concatenated with the spatial attention matrix and then passed into the second layer to make one of five mid-level decisions. Finally, the sequence attention matrix and output of the second layer are concatenated and passed into the final layer of the decision hierarchy to reach one of seven low-level decisions. The final decision is output as the recommended course of action for the system. The potential decisions that can be made at each level of the prototype were designed to simulate a subset of those that might be encountered in real command and control applications. The decision path followed by the hierarchy is stored as the system's decision reasoning. Each possible decision at each layer was associated with a human readable text-based message. These messages were used by the explanation generation subsystem to explain the prototype system's reasoning. The list of possible decisions and associated messages are shown in Table 4.4.

Finally, the SEDA prototype generates a decision explanation based on the combination of system attention and decision reasoning. The explanation-generation subsystem read in the spatial and sequence attention matrices and the messages generated by each layer of the decision hierarchy to produce its explanations. These explanations consisted of both the messages generated by the decision hierarchy and a set of messages interpreting the system's attention. The interpretation of the system's attention was generated using the outputs of the decision hierarchy's fully connected networks. Each network was trained to prioritize decisions based on key features in the input sequence. The system determines the presence of these features by analyzing the outputs of the network at each layer and generates interpretation messages based on their perceived presence or absence. An example of these human readable explanations can be found in Figure 4.11. To aid with analysis, the prototype explanation generation subsystem also output the raw attention matrices from the feature extraction and sequence interpretation subsystems.

```
Decision 1:
Entity is neutral or unknown.
Identified closed loop on entity.
→ Ignore entity.
Entity features concentrated in southwest quadrant.
→ Ignore entity.

Decision 2:
Entity is hostile.
Identified no loop on entity.
→ Engage entity.
Entity features concentrated in northeast quadrant.
→ Close distance to entity.
```

Figure 4.11: Example explanation generated by SEDA prototype.

4.3.7 Results and Discussion

As I have noted before, relatively simple datasets were used to test and evaluate the prototype implementation of SEDA. There were a number of reasons why the prototype was characterized in this way. While the use of simple, curated datasets leaves many open questions with respect to particular applications of the architecture, it allows for more fundamental and robust testing of the architecture itself. It removes the need to struggle with common data issues while examining how the subsystems interact to output decisions and explanations. Such isolation of variables was essential for the proposed architecture, which was quite a novel approach to self-explaining ML at the time of its development. It also allowed for tight control of the expected outputs of each component, enabling me to perform higher fidelity analysis of both component and system level performance. I will now present the results of testing and my analysis of those results. Note that as much quantitative analysis was performed as possible, but certain elements of the explanation generation still do not have well-defined metrics today and required more qualitative evaluation.

4.3.7.1 MNIST-Based Dataset

The first round of testing conducted on the prototype SEDA implementation was done with the MNIST dataset. This testing was used to evaluate the performance of a full SEDA implementation and examine the behavior of the attention mechanisms on data with both spatial and sequential (though non-temporal) elements. The CNN-LSTM network was trained first for ten epochs and achieved a validation accuracy of 94.5%. This process was repeated an additional ten times to ensure that the model trained consistently. In each case, the validation accuracy was between 93%

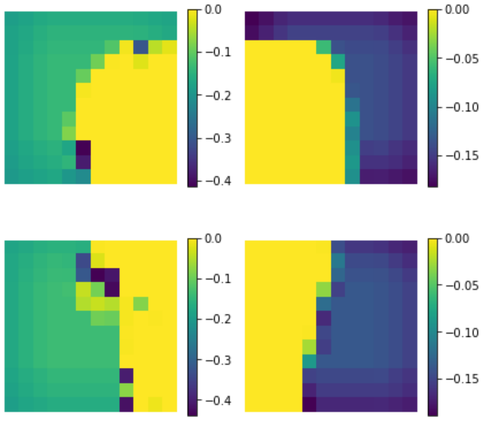
and 95%. For reference, the MNIST dataset has ten possible classes, so a system guessing randomly would achieve 10% accuracy. The consistently high classification accuracy demonstrated by the joint feature extraction and sequence interpretation network demonstrated that it was able to successfully identify consistent patterns within the data and use them to classify the observed digit. It also proves that separating each image into quadrants and forming an image sequence did not significantly impede the learning process.

Once the CNN-LSTM network was trained, attention information was collected and analyzed at each layer of the system. This was accomplished by running the full MNIST dataset through the trained CNN-LSTM model and saving the output of each attention layer, along with the input sequence used to generate it, into a text file. Full system attention consisted of three sets of spatial attention, each generated by one of the CBAMs in the CNN, and the sequence attention generated by the Luong attention module from the LSTM. Each set of spatial attention takes the form of a three dimensional matrix $\mathbf{A_{spatial}} = \mathbb{R}^{[D \times H \times W]}$, where D is the number of kernels in the convolutional layer that immediately precedes the CBAM and H and W are the dimensions of the input images to that same layer. What each CBAM outputs is essentially a set of D masks that show the degree to which each input pixel contributed to the corresponding kernel output. Put another way, it shows the level of attention each kernel paid to each input pixel. The sequence attention produced by the Luong attention module was a vector, $\mathbf{a_{sequential}} = \mathbb{R}^{[1 \times N]}$, where N is the sequence length. It maps the correlation between the final output of the network and each input sequence step, indicating the extent to which the network attends to each sequence step.

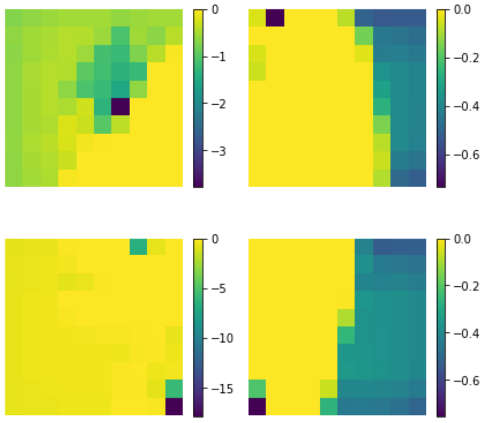
Examples of the attention information collected at each point in the systems is shown in Figure 4.12. Note that in this example, each set of spatial attention is summed along the channel dimension to produce a single $H \times W$ image for the sake of readability. Additionally, for this implementation of SEDA the attention from the first CBAM had dimensions $8 \times 12 \times 12$, the second $8 \times 10 \times 10$, and the third $32 \times 3 \times 3$. The dimensions of the sequence attention were 1×4. The attention information was used in two critical ways: 1) it was used as input information to the decision hierarchy and 2) it was output as part of the system explanation. Considerable time was spent reviewing and analyzing the attention information. The content of the system attention had a considerable impact on the structure of the decision hierarchy and the labels used to train it.

At this stage, testing was performed on the collected attention data to verify that it could be used in a MLP to classify data according to only the human observable features described in Section 4.3. Spatial attention was used to train the MLP to detect the presence of loops in the image and the average pixel intensity of each sub-image, and sequence attention to train it to identify the image quadrant with the highest average pixel intensity.

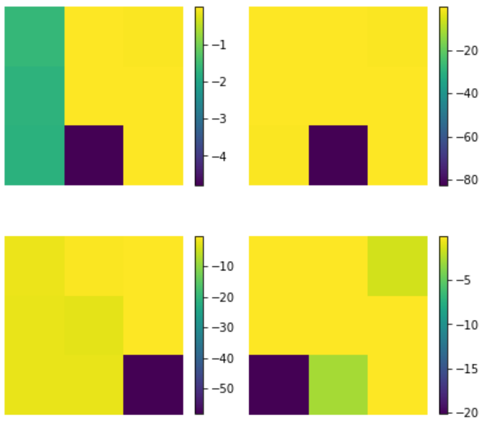
Additionally, tests were conducted to see if a MLP using only the spatial attention data could be trained to correctly classify the numeral present on the original image. For tests using spatial attention, the performance of the MLP was examined using attention from each CBAM individually, as well as the combination of attention from all three CBAMs in the system. This allowed for the examination of how well the system could be taught to interpret each level of attention to identify different types of features. In all cases, the system could be trained to recognize the desired features and correctly classify the image sequences based on those observations. For each test case, a two-layer MLP was used with Adam optimization and softmax final activation. These were trained for ten epochs with a learning rate of 0.01.



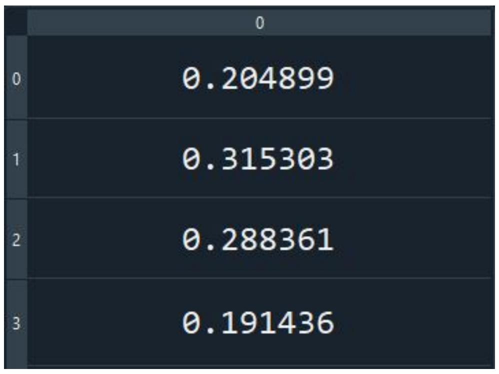
(a) First CBAM Attention



(b) Second CBAM Attention



(c) Third CBAM Attention



	0
0	0.204899
1	0.315303
2	0.288361
3	0.191436

(d) LSTM Luong Attention

Figure 4.12: Example MNIST-based image sequence attention showing heat map of CBAM attention from first convolutional layer (a), second convolutional layer (b), and third convolutional layer (c); and Luong attention for LSTM (d).

The representative set of simple test cases allowed for the verification that the system could be trained to interpret both spatial and sequence attention to detect readily observable (and verifiable) features in the original images. It also demonstrated that it could perform reliable classification based solely on those features. The success of this early implementation of SEDA on well-defined data provides a strong indication of its potential for use in more complex systems. In particular, it should be explored for object detection in image and sensor data, data fusion, patterns of behavior analysis, and control system for autonomous platforms.

Through inspection of the spatial attention from each CBAM, various observations can be made about the object properties the feature extraction subsystem focused on at each convolutional layer. It is clear from Figures 4.12a and 4.12b that earlier layers focus on simpler, more immediately observable properties, such as detecting object edges and distinguishing entities of interest from the background. This makes these layers well suited for training the decision-generation subsystem to identify the presence of objects in a scene and identifying their characteristics. This observation is consistent with the literature and supported by testing, which indicates that the attention from the first CBAM is highly effective at distinguishing between digits based on loop features.

Deeper layers of spatial attention appear to focus on specific points of interest with more abstract object properties. Figure 4.12c shows that the third convolutional layer of the CNN focused on the end of the tail of digit 9, as well as the two points at which a sharp curve appears in the loop of the 9. This kind of attention is well suited for teaching the system to interpret the presence of specific object characteristics. This is again supported by both the literature and experiments, which show that the MLP trained with only attention information from the third CBAM performs better than the others at identifying the average pixel intensity in the image and in determining which of the ten digits is present in the image.

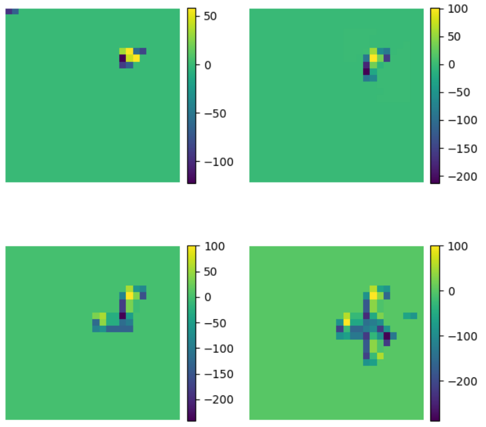
These results verified that the system can be trained to interpret various forms of attention and map that data to specific, human-interpretable concepts. It also demonstrates that the proposed architecture generates both consistent predictions and human readable explanations of attention features. However, it also highlighted a key challenge: useful implementations of self-explaining AI require significant domain expertise and design work. While some work should be done to explore ways to reduce the level of expertise needed, it should be noted that such design is essential to making ML systems that can be used in most real-world applications.

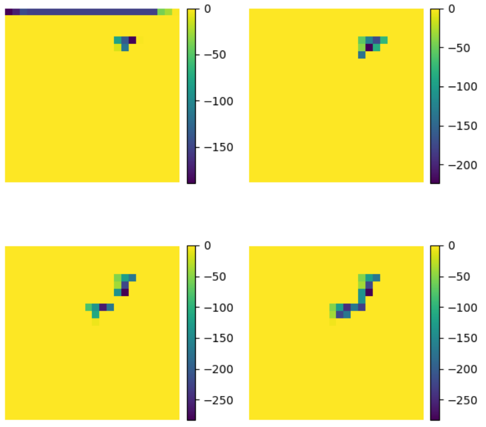
Finally, the full system described in Chapter 4.3 was tested. For that testing two configurations of the decision hierarchy were examined. They differed only in the labels and spatial attention values used to train the second layer of the hierarchy. In the first, the hierarchy was trained using the attention values from the first CBAM and the L2 labels. For the other configuration the hierarchy was trained using the attention values from the third CBAM and the L2 alternate labels (see Table 4.1). In both cases, the final decision accuracy was greater than 90%, and the network accurately generated explanations like those shown in Figure 4.11.

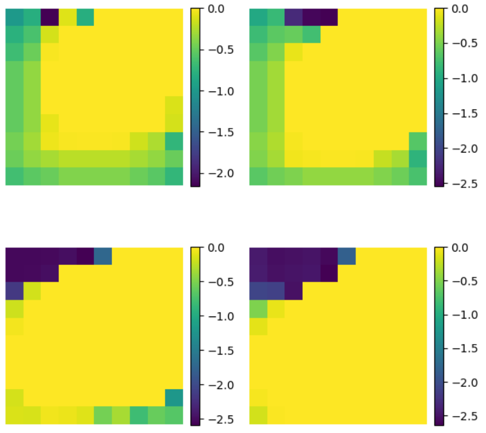
4.3.7.2 Custom Time-Series Dataset

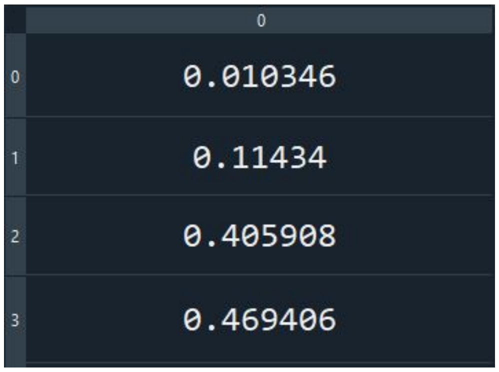
After evaluating the SEDA implementation's performance on the MNIST dataset, a second round of testing was conducted with Brian Sun's custom time series dataset. This second round of testing was only used for targeted analysis of the CNN-LSTM's performance when presented with temporal data and additional examination of the attention information produced by the model. This implementation had to classify inputs into one of four categories based on the appearance of a target shape over time and the orientation of that shape. This simple, but highly representative test case shows the ability of the attention-based CNN-LSTM network to solve common computer vision problems (differentiation of objects based on shape/orientation). It also showcased the behavior of sequence attention when presented with an initially limited information scenario in which additional information is discovered over time. The similarity in spatial features between the target shapes (cross versus chevron) encouraged the system to attend to specific physical features that were needed to differentiate between them, which provided very useful and intuitive attention masks for evaluation. To accommodate the new data, the size of many layers in the network had to be modified (see Table 5.1). The network was again trained for ten epochs eleven times to ensure that it converged consistently. Each time the model was able to achieve a classification accuracy in excess of 96%. Given the consistency of the features and the strong temporal component, it is unsurprising that the model trained to a higher classification accuracy on this dataset. Unlike MNIST, which contains handwritten digits whose features may vary due to handwriting styles, this dataset was generated by Matlab with no noise added to the images. As a result, the features for each class were always rendered identically with only their location and orientation changing. Additionally, the temporal pattern had little variation with the first two timesteps containing relatively little information and the last two adding in all features necessary to distinguish between classes.

The full training dataset was then run through the best performing trained model and each set of attention information was collected for evaluation. An example of the system attention for this dataset is presented in Figure 4.13. From this set of example attention data, a few observations can be highlighted. First of all, as with the MNIST


(a) First CBAM Attention


(b) Second CBAM Attention


(c) Third CBAM Attention



	0
0	0.010346
1	0.11434
2	0.405908
3	0.469406

(d) LSTM Luong Attention

Figure 4.13: Example time-series image sequence attention showing heat map of CBAM attention from first convolutional layer (a), second convolutional layer (b), and third convolutional layer (c), and Luong attention for LSTM (d).

testing, the deeper layers of the model clearly learned more abstract features. However, unlike the implementation trained on MNIST, very consistent patterns can be identified in the spatial attention maps. In particular, the model consistently attended to features in the upper left quadrant of the observed target and largely ignored the rest. This is almost certainly due to the aggregation of three factors: 1) the targets are rendered over time starting at the top, 2) the kernels at each layer are applied from left to right, and 3) there is generally enough information in any one quadrant of the target to successfully classify it. Put another way, only one quarter of the target is needed to successfully distinguish between most classes and the top left quadrant of the target is observed by the system first. So, it unsurprisingly learned to weight features in that region more heavily. The sequential attention similarly showed very consistent behavior that aligned with the strong temporal patterns in the data. Of the four time steps, the first contains almost no information, the second adds a small amount of information (but not enough to make reliable classifications), the third adds a great deal (and in fact adds enough for the system to make a successful classification) and after the fourth time step the SEDA prototype has access to full information about the scenario. This is reflected in the sequence attention collected, which consistently showed low correlation values between the output and the first two time steps and very high correlation between the output the last two time steps. This clearly demonstrated that the model was able to learn targeted spatial and temporal relationships. It also clearly demonstrated the validity of my hypothesis that these attention mechanism could be used to identify the features used by a SEDA implementation and could be translated into human readable context.

4.4 Conclusion

In this chapter, I presented my work developing the SEDA framework for self-explaining autonomous systems. The goal of this work was to develop a method for streamlining the deployment of ML into critical systems by integrating autonomous decision-making and explainability to create systems whose decision can be questioned in real time. This was accomplished by modeling human explanations as a combination attention (what the person was paying attention to) and reasoning (why they selected a given course of action). I then proposed using a combination of ML attention and hierarchical decision structures to replicate these in an autonomous system. This information would be contextualized and used to generate text-based statements explaining system decisions. The architecture was designed to be highly flexible and broadly adaptable. I presented the full architecture along with a prototype implementation that I developed and tested to prove out the concept. This prototype was evaluated on two simple datasets that allowed me to directly characterize it without having to consider the impacts of imperfect data.

While significant additional testing needs to be conducted to determine how readily SEDA can be adapted to highly complex systems, the targeted testing presented here was ideal for demonstrating the validity of the approach and developing methods for translating highly abstract ML attention data into human language. The prototype implementation proved to be well designed for testing each component of the architecture. However, the attention mechanisms used in it were configured so that they only provided observational data and did not contribute to the overall performance of the network. Having learned more about attention structures in subsequent years, for future implementations I would integrate attention more fully into the system in order to better utilize the strong contextualization and non-sequential pattern recognition they provide. I would also explore using multi-head attention to replace the fully connected layers used in the decision hierarchy.

One publication was generated directly [9] and a provisional patent was submitted for the framework. Additionally, I worked with colleagues to adapt the framework for use in computer vision and in data fusion for target tracking. Working on adapting the framework for use in data fusion and tracking was particularly interesting. Many of the issues that we ran into during that adaptation had to do with how to effectively mitigate the impacts of noise and error on long term tracking. Common tracking algorithms at the time either ignored historic error or developed filters based on assumptions about the object motion to mitigate error. This led me to investigate how to develop a ML system capable of capturing the causal relationships existing in object motion through physical space and proposing alternative paths to remove historic error. That, in turn, led me to research the topic of causal ML and counterfactual evaluation. I will describe this research in detail in Chapter 5.

The work presented in this chapter was more abstract and theoretical than much of the rest of the work presented in this dissertation. However, I believe it is important to begin the discussion of my research here. Not only because it was it my earliest work in ML, but also because it proved to be pivotal to the overall direction of my research since. In creating SEDA, I was introduced to two of the most critical issues in ML: 1) a lack of general trust in ML decisions and 2) a lack of first principles design in ML model development. The first issue limits the deployment of ML in most safety or mission critical systems. The second limits the effectiveness and consistency of ML models, as well as drastically driving up the data requirements for training those models. Much of the ML community has focused on the issue of trust over the last five years. SEDA has shown itself to be a highly effective architectural approach to solving both of these issues.

The lack of trust in ML has been the subject of considerable research mostly centering on methods for providing model interpretability and/or explainability, as well as developing metrics to quantify user trust. Conversely, the lack of system design in ML has largely been dismissed by the ML community. Most ML researchers and professionals view design and architecture as at best secondary elements of ML model development. It has broadly been believed that larger models and more data are all we need to solve any problem with ML. While I understand where this view comes from, I would argue that it is a short-sighted and unsustainable view that has limited the development of ML algorithms in most applications and contributed to the broad mistrust of ML. This has been showcased in recent years by high-profile failures of many state-of-the-art models trained on vast quantities of data [123, 124].

This view has been popularized in more straightforward ML domains like computer vision or language-to-language translation, where the data is highly structured, easily obtainable, and readily understood by human observers. Applications in these domains also have relatively high error tolerance, with 99% model accuracy typically considered acceptable, and low error impact. More complex applications of ML, like strategic decision-making, data fusion, resource management, and radar have seen little success due to the lack of representative data, more complex or probabilistic nature of the application, dependency on multi-modal data, and tight error tolerances. These domains typically rely on human input or on algorithmic solutions that make certain assumptions about the underlying nature of the problem space that often do not hold true. ML models, on the other hand, do not make those assumptions, and instead learn an approximate representation from the data presented to it. Understanding these challenges and working on system level ML design with SEDA led me to conclude that ML had considerable potential to augment, rather than replace, existing algorithmic solutions. By using a first principles understanding of complex problem spaces, assumptions and edge cases that interfere with existing algorithms can be identified and ML models trained using targeted (preferably synthetic) datasets to remove assumptions and mitigate challenging edge cases. This intelligent integration of ML and first principles algorithms creates more generalized and robust tools, provides theoretical basis for explaining system behaviors, and reduces data requirements.

In the field of radar in particular, ML is rarely used in practice and is viewed with extreme skepticism by the research community. This is unfortunate, as ML has a great deal of potential to improve radar systems. The field has a robust set of signal processing methods based on the first principles transmission and reflection characteristics of energy. These have a strong theoretical background and have been thoroughly tested and characterized. However, due to the extremely complex interactions between energy traveling through free space and the environment, certain assumptions must generally be made in order to derive radar algorithms. In most scenarios, the assumptions hold and the derived algorithms achieved the desired performance. However, modern radar systems are increasingly pushing the limitations of existing algorithms and the assumptions they are founded on. ML models designed with an understanding of these assumptions and the underlying properties of the data in a given radar application can be used to create a general model that helps radar systems maintain performance in traditionally complex or difficult scenarios. Integration of traditional radar signal processing techniques with ML can be used to develop systems with the benefits of both. The subsequent chapters of this dissertation will describe my research towards integrating ML into radar applications related to navigation/tracking and detection.

Chapter 5

Applying Causal ML to Predictive Navigation

5.1 Introduction

In this chapter, I will describe the creation and characterization of a method for removing sensor drift errors in navigation and tracking. This method was based on causality-aware ML and uses the concept of counterfactual evaluation to examine multiple possible paths an entity could have traversed and selecting the most probable one. I will first described the problem space and how data was simulated for training and testing. I will then discuss the ideation and initial proof of concept design for an implementation of the proposed method. Finally, I will detail expansions made to the system based on the results of early testing and the full characterization testing of the final system. Initial proof of concept work was published in [11] and the final design was published in [12].

Single target tracking and navigation both involve using sensor information to identify the historic path and expected trajectory of a mobile entity traversing a space. For navigation, we are generally interested in our own path and trajectory, while tracking focuses on those of an external entity. In simulating these types of scenarios, one can generally use similar motion models. The primary differences to consider are the types of sensors used, methods for associating data with tracks, and the types of error/interference to be injected. In this work, I proposed a new method for correcting historic path/track errors after data has been collected and associated. As such, I do not focus on modeling sensor systems or data association algorithms.

Drift is the primary form of error injected into both simulations. This is a common form of error in both navigation and tracking problems caused by compounding measurement or prediction errors. Drift can be particularly problematic for existing correction methods as it results in incorrect but realistic paths and trajectories. In tracking applications, drift might be the result of prediction errors after losing sight of a target for some period of time or incorrectly associated tracks after multiple targets path near to one another. For navigation, drift most commonly occurs when entities lose communication with the Global Navigation Satellite System (GNSS) and are forced to rely on localized inertial navigation system (INS) sensors.

Today, when a platform receives some indication that drift has occurred, typically in the form of a higher fidelity or correction measurement, the path/track is updated and historic measurements containing drift are either ignored or corrected using an approach like the Kalman Filter (KF). Ignoring historic error is not viable in most modern systems, as doing so creates unstable trajectories for the predictive models that are broadly used to augment tracking and navigation capabilities today. The KF has seen widespread use in correcting historic drift errors and is considered the state-of-the-art in this field. However, it can struggle to correctly model more complex motion patterns containing drift [125].

5.2 Related Work

In recent years, the use of ML to improve and automate navigation tasks has become an active field of research. One of the key focus areas within that field is the use of ML methods to mitigate errors in INS measurements [126, 127, 125, 128, 129, 130]. Navigation relying on INS is known to struggle with drift errors due to the localized frame of reference [125]. Drift error can be difficult for traditional signal processing methods like the Kalman and partical filters to model, leading to the exploration of ML as an avenue for mitigating drift. Al Bitar *et al.* [127] presented a method using a non-linear autoregressive neural network with external inputs (NARX) paired with the UKF to improve position and velocity INS measurements. In [128], Semeniu *et al.* used DNNs to augment the KF. Others have explored using fuzzy logic structures with supervised DNNs [126] and unsupervised C-means clustering [130].

The use of deep neural networks paired with GNSS positioning for adaptive tracking is another major focus area [131, 132, 133, 134, 135, 136]. DNNs have proven to be a powerful tool for GNSS position correction. In [136], Siemuri *et al.* present a survey exploring the various uses of ML techniques to improve the integration of GNSS and INU measurements. Exploration into the use of RNNs to compensate for INS error have been conducted in [135] and [133]. In [133], the authors combine technique for empirical mode decomposition threshold filtering (EMDTF) with an LSTM to reduce the noise of an INS and the predict track locations in a GNSS denied region. DNNs have also been used to mitigate multi-path and non-line-of-sight errors in measurements [131]. ML methods have also been used for localization and positioning based on the relationships between the line-of-sight and multiplath signal conditions of an environment [132, 134, 131].

The mitigation of drift error is also used heavily in SAR imaging and SAR target tracking [137, 138, 139, 140, 141]. SAR data processing can be extremely sensitive to phase instabilities caused by positioning errors in the measurement platform [139, 140]. To mitigate this, parametric techniques can compensate for slow phase errors though the use of low-order polynomial modeling. Fast phase errors are corrected using non-parametric techniques such as the Phase Gradient Auto-focus (PGA) algorithm [137, 138, 141].

Additionally, the use of GAs for optimizing parameters within the KF and its variants has also been explored [142, 132]. This notion of pairing GAs with existing signal processing methods is central to the work presented in this chapter. However, the approach presented here frames the problem very differently, using a GA to mitigate drift with an EKF to mitigate noise.

5.3 Problem Space and Motion Models

To prove out the concept of my path correction system, an initial implementation was developed and tested with a relatively simple model that could be readily adapted to either tracking or navigation. Considerable work was then done to improve the system and develop a more realistic navigation simulation. Here I will discuss the relevant tracking and navigation problem spaces and describe the development of each simulation. Both data models were developed in Matlab using the Phased Array Toolkit.

When both the objective functions and hyper-parameters are designed effectively, the genetic algorithm is able to narrow down large and highly-complex solution spaces to a near-optimal solution. The problem presented here is a similar solution space. There are effectively an infinite number of paths that can be used to travel from one point to another. However, many of these potential paths are either highly unlikely or impossible given an entity's current state and the constraints of travelling through a physical space. The authors here propose that by designing objective functions based on a combination of past observations and first principles physics, it is possible to have the NSGA-II identify position sequences that are consistent with the entity's real motion. This approaches the path correction problem in a manner similar to a human mind. The proposed tool uses past observations with suspect elements to evaluate each possible alternative path. This intuition is based on past motion characteristics and an understanding of the causal relationships between past motion, environmental characteristics, and future positions. It then selects the path most likely taken. This process falls squarely into the domain of counterfactual evaluation.

5.3.1 Initial Simulation Model

The initial simulation used to test the proposed path correction model consisted of a single entity travelling through a two dimensional space. The platform object from the Phased Array Toolkit was used to generate entities with positions in meters, velocities in meters per second, and accelerations in meters per second squared and to update their motion through an x-y plane over time. A constant acceleration model was used with the initial position, velocity, and acceleration being randomly set at the start of each scenario. The initial position consisted of x and y coordinates ranging from ± 13, the initial x and y velocity values each ranged from ± 5, and the initial x and y acceleration values ranged from ± 1. The scenario would then advance through 40 time steps (2 seconds each) and a track of 15 sequential positions was pulled from the generated data beginning at a random sample number between 20 and 25. Then each track was shifted so that it began at the origin. These 15 ideal measurements were recorded at this stage as the "true path".

The true path was then used to generate an "erroneous track" by injecting drift. Here the erroneous track also consisted of 15 positions, with the first 7 positions being treated as high-fidelity measurements containing no error, followed by 3 positions deemed to be suspect that may or may not contain drift, and finally 5 future positions generated by some prediction model using the first 10 positions. The drift injected at time steps 8, 9, and 10 was modeled as constant velocity drift and was randomly set to between ± 5 meters per second. Figure 5.1 depicts this early simulation, with the true path plotted in green and the erroneous track in red.

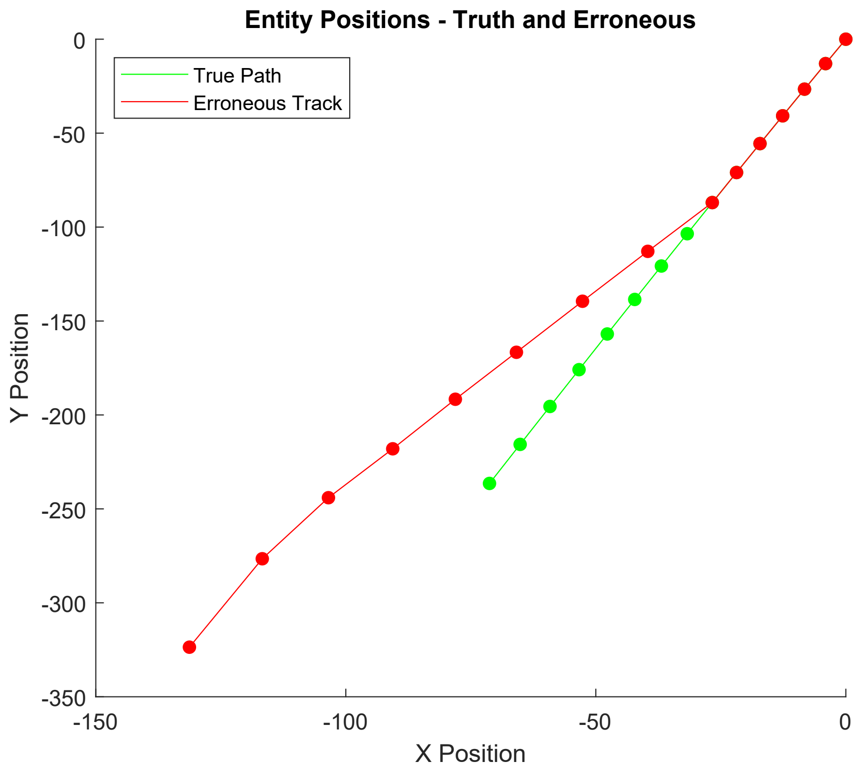


Figure 5.1: Plot of true entity position and erroneous track.

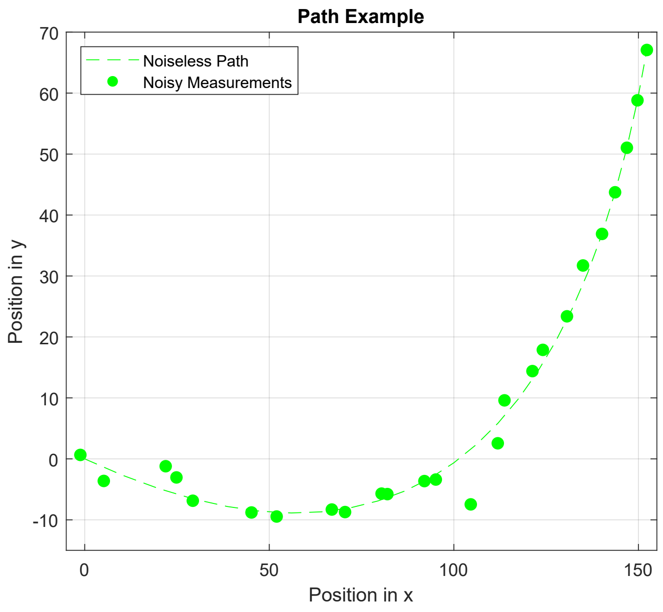
The inputs to the initial implementation of the path correction tool were each sequences of $11 (x, y)$ coordinates constructed from the first 10 positions in the erroneous track and position 11 of the ideal track to serve as a correction measurement (similar to receiving GPS updates to correct inertial navigation). I refer to these inputs as suspect tracks, and they take the following form. A suspect track of length k consists of a sequence of locations some of which have been deemed to be suspect. We will refer to this suspect track segment as T, where

$$T = [(x_1, y_1), ..., (x_k, y_k)] = [H_{1:i}, S_{i+1:j}, L_{j+1:k}] \tag{5.1}$$

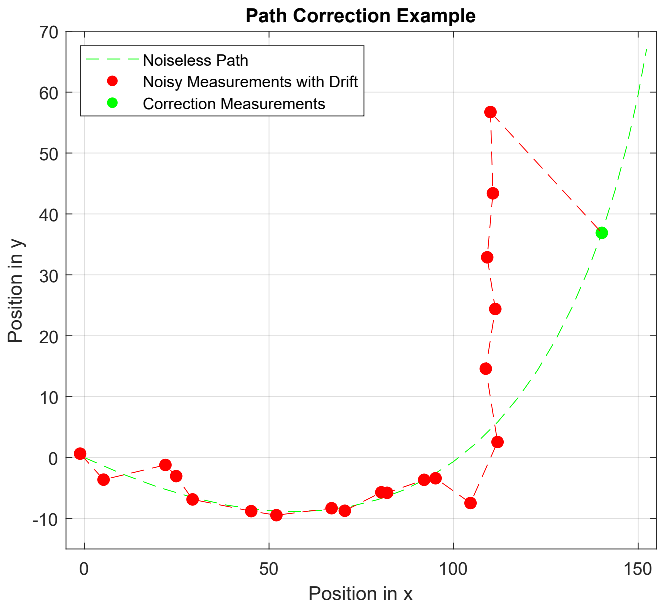
From the notation introduced in (5.1), track T consists of three segments: 1) an initial segment of high-fidelity locations that the system assumes to be true (H); 2) a segment of suspect locations that may or may not be erroneous (S); and 3) a final segment of one or more high fidelity measurements (L). Note that no measurement noise was injected into the simulation (aside from drift) for this early simulation used in initial development and proof of concept testing.

5.3.2 Expanded Simulation Model

After demonstrating the initial viability of the proposed counterfactual approach to path correction, the simulation was expanded to generate far more complex and realistic scenarios targeted towards navigation. These scenarios were defined as follows: an entity traveling through a 2-D space receives regular GPS updates for 15 time steps before entering a GPS-denied environment. It then traverses the environment for 5 time steps relying solely on INS updates for navigation before exiting and receiving a final GPS update that is used to correct the path. This results in a sequence of posi-



(a) Plot of entity positions with and without noise.



(b) Plot of entity positions with noise and sensor drift.

Figure 5.2: Plots demonstrating one scenario used to evaluate the proposed system.

tions, P, used as the input to the path correction scenario defined as:

$$P = [p_{gps,1}, ..., p_{gps,15}, p_{inu,1}, ..., p_{inu,5}, p_{cor,1}] \tag{5.2}$$

In (5.2), $p_{i,j}$ are each positions in a Cartesian space such that:

$$p_{i,j} = [x_{i,j}, y_{i,j}] \tag{5.3}$$

where i refers to the type of sensor update, j refers to the time step, $x_{i,j}$ is the x-position produced by update i after j time steps and $y_{i,j}$ is the y-position produced by update i after j time steps.

Entity motion was simulated using a constant acceleration model. For each experiment, entity position, velocity, and acceleration were randomly initialized. The initialization range was between ± 13 meters for position, ± 25 meters per second for velocity, and ± 4 meters per second squared for acceleration. Position information was updated every 0.5 seconds. A total of 25 positions were recorded for each test. Noise was then added to the first 20 simulated positions. This noise was i.i.d. Gaussian with 0 mean and a standard deviation of 3 meters and was used to model GPS sensor noise. Finally, a constant velocity offset was added to positions $16 - 20$, resulting in a noisy drift representative of compounding INS errors. The last 5 simulated positions are left error free and used for performance analysis of the system. Figure 5.2 shows an example of these simulated paths.

5.4 Initial Approach and Testing

As with the simulations models, implementation of the proposed counterfactual evaluation method path correction was performed in two stages. I will now describe the initial implementation, reasoning used for design decisions, testing, and analysis. Expansions to the design and testing are described in Section 5.5. The first stage of development was used to define the overarching structure of ingesting positional information, establishing a method for generating counterfactual paths, and creating tools for evaluating the quality of those paths. This initial system broadly consisted of preprocessing to prepare data for the system, a multi-objective GA to generate alternative paths, and a RNN used both in path generation and in performance evaluation. Moving into the second stage of implementation, the high-level structure of the system remained unchanged but each element was expanded and changed to improve system performance and overcome challenges observed in the first stage.

5.4.1 Initial Approach

The proof of concept system architecture for the proposed path correction method is depicted in Fig. 5.3. It is comprised of a combination of a RNN and an implementation of the NSGA-II with four objective functions. It outputs a new track of the same size and structure as the input track, with updated locations in the suspect segment of the input. Data for testing and training were generated in Matlab and the path correction tool was implemented in Python.

In developing the proposed track repair tool, an RNN was first implemented and trained to predict a target's next position based on a list of its M previous locations. RNNs are relatively lightweight neural networks that are used to learn sequential pat-

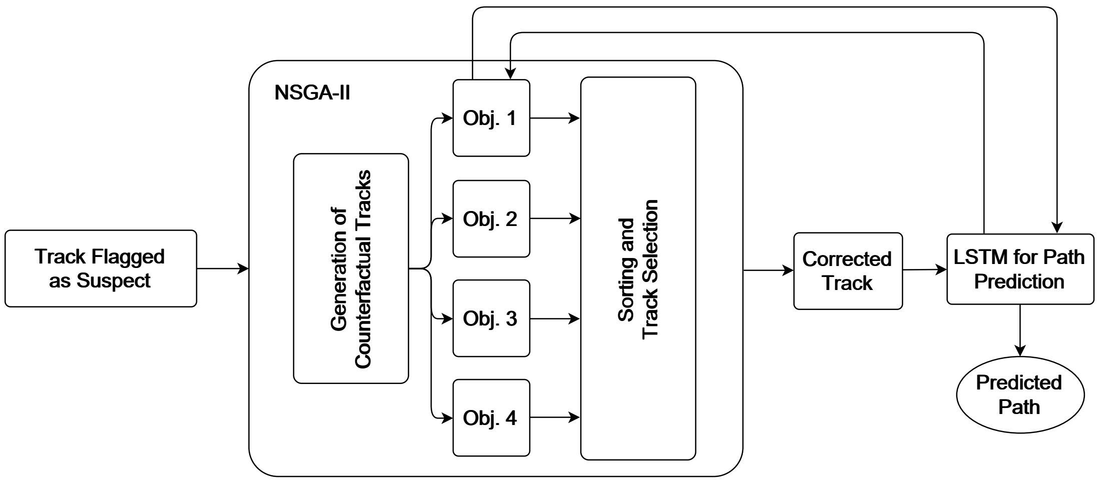


Figure 5.3: System architecture for the proposed path correction tool.

terns and predict future states of the sequence. Motion patterns in both tracking and navigation are inherently sequential, making RNNs ideal for predicting future position states in these domains. Given a sequence of past locations for an entity at known time intervals, it is possible to infer that entity's motion characteristics and thereby make an accurate prediction of its current or future location. Here, I used the RNN in two ways. First, it was integrated directly into one of the objective functions for the NSGA-II which compared its output to the correction samples. RNN outputs are highly dependent on the input data provided to them, which helped force the system to consider the stability of trajectories produced by the generated paths. Second, I used the RNN as a predictive model when evaluating the performance of the path repair method. At this stage, the RNN was implemented using a LSTM network, whose structure is detailed in Table 5.1. This network used linear activation for the final activation function and was trained on simulated track data using mean squared error (MSE) as the loss function for 100 epochs. It should be noted that this LSTM was not trained on noisy path data, so its predictions were heavily impacted by relatively minor path errors. This will be discussed further in Section 5.4.2.

Table 5.1: Implemented LSTM for Target Location Prediction

Layer	**Type**	**Input Shape**	**Output Shape**
1	Input	2×10	2×10
2	LSTM	2×10	2×100
3	Dense	2×10	2×100
4	Dense	2×100	2×1
5	Linear Activation	2×1	2×1

Next, NSGA-II and its objective functions were implemented. NSGA-II is a widely used multi-objective GA. GAs excel at exploring complex solution spaces and identifying solutions that optimize a defined objective (or fitness) function. The power of multi-objective GAs is that they allow us to define the fitness of a solution by balancing multiple objectives. In the context of tracking or navigation, this can be framed as establishing a set of objectives based on well-known motion characteristics for objects travelling through a space and defining the most likely path taken as the one that minimizes a weighted average of this set. It also makes it so that the system can be readily extended to incorporate additional types of data without major alteration. For this work, the GA was used to propose and evaluate counterfactual scenarios. We define our counterfactual scenarios as alternative paths that an observed target might have followed to move from one position to another. The observed (suspect) track is used to generate the first population of possible solutions, then each solution in the population is evaluated using the objective functions, and the population is sorted and pruned. The surviving members of the population are used to produce the next generation through crossover and mutation. For this work, the population size was set to 20 and the maximum number of generations was set to 100. While there are technically an infinite number of possible paths that could be traversed to move between positions, careful design of the objective functions was used to rapidly narrow down the solution space.

Four objective functions were used to evaluate the fitness of the proposed paths:

$$o_1 = \sum_n |f(x'_n) - y_n| \tag{5.4}$$

$$o_2 = \frac{1}{N} \sum_n ||x'_n - x_n|| \tag{5.5}$$

$$o_3 = \sum_n g(x', x),\ g(x') = \begin{cases} 1, \text{ if } x'_n \neq x_n \\ 0, \text{ otherwise} \end{cases} \tag{5.6}$$

$$o_4 = \sum_{n=2}^{N} \Big| ||x'_n - x'_{n-1}|| - \max(||\Delta x_H||) \Big| \tag{5.7}$$

In Equations (5.4)-(5.7) above, n is the track index, N is the length of the track under evaluation, x is the observed track, x' is the proposed counterfactual track, Δx_H is the magnitude of the distance traveled between time steps (i.e., $|x_i - x_{i-1}|$) in the high-fidelity portion of the input track, $f(x')$ is the LSTM's output given the proposed counterfactual, and y_n is the user's desired output.

The first objective function (Equation (5.4)) was used to evaluate both the stability and effectiveness of the proposed counterfactual tracks. This was determined by computing the Manhattan distance between the desired output vector and the output vector generated by the LSTM network when the counterfactual under evaluation is used as the input. It encouraged the algorithm to select tracks that cause the LSTM to arrive as close as possible to the expected destination.

The second and third objective functions (Equations (5.5) and (5.6)) were used to restrict how much the proposed counterfactual differed from the original observation. Equation (5.5) accomplished this by computing the Gower distance between original path and the counterfactual one. Equation (5.6) instead calculates the discrete number of vector elements that were altered. The combination of these two objectives encourages the algorithm to focus on results that are as similar as possible to the original observation. This both served to immediately encourage the GA to focus its exploration of the solution space to the region near the original observation and allowed it to effectively consider suspect track segments that may have been partially or entirely correct without introducing massive error to them.

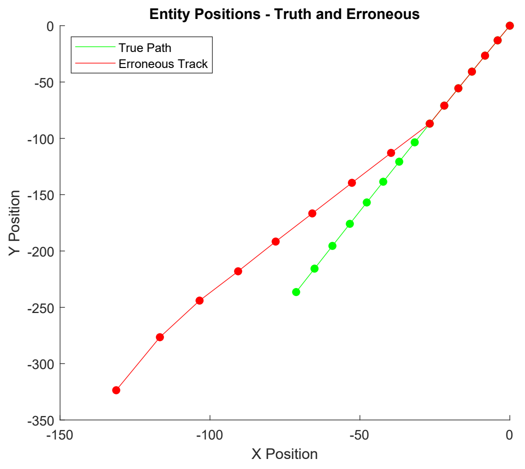
The final objective function (Equation (5.7)) was based on motion limitations of entities travelling through a physical environment. Object motion is constrained by its current location and motion characteristics (like velocity, acceleration, heading, etc.). For instance, an aircraft cannot instantaneously jump 200 miles to another location, but must instead follow its current trajectory or apply forces to gradually alter course. An understanding of these motion constraints can be used to develop objective functions that effectively limit the solution space explored by a GA and encourage it to produce solutions consistent with realistic trajectories. With that in mind, the final objective function measured the difference between the maximum change in position observed in the most recent high-fidelity track measurements and the change in position between each subsequent point in the proposed counterfactual track. In effect, it identifies the highest observed velocity when it was last confident in the track and seeks to minimize the degree to which the velocity at each point of the proposed track exceeds it. This objective served a critical role in generating realistic counterfactual paths by discouraging large instantaneous jumps in position.

By seeking to minimize these four objective functions, this early design successfully demonstrated the ability of the proposed approach to generate consistent, realistic path corrections for an entity travelling through space.

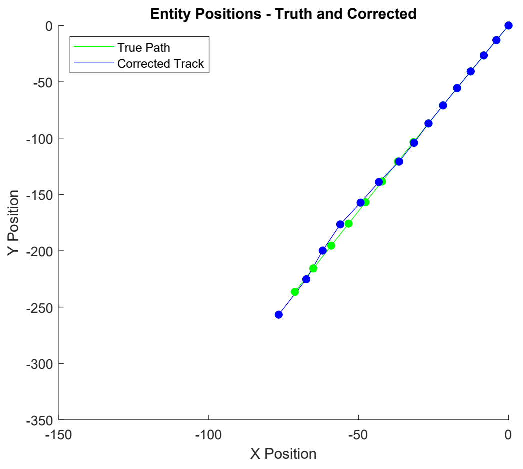
5.4.2 Initial System Testing

Proof of concepts testing for the proposed track repair system was conducted using data simulated in Matlab. To demonstrate both its performance and utility, I evaluated both how closely corrected tracks matched the entity's true path and how closely future track predictions made by the LSTM using corrected tracks at the input matched the true future path of the entity. Note that the simulation model described in Section 5.3.1 was used to generate all data for this testing. Track positions $1-11$ were used by the track repair tool to generate a "corrected" track. This corrected track was then used by the LSTM to predict the next 5 locations of the entity, which were compared with track positions $11-15$.

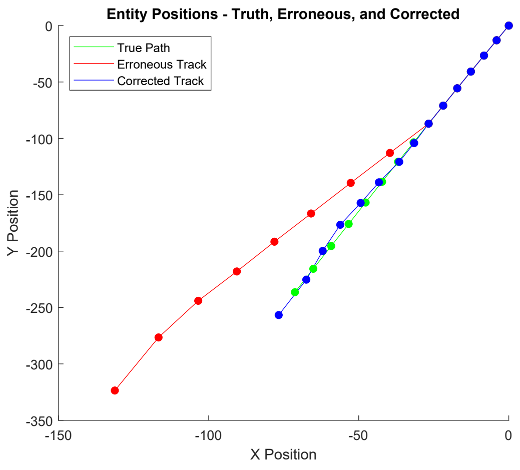
The simulator was run $11,000$ times to generate pairs of true paths and erroneous tracks. $7,000$ of these were used to train the path prediction LSTM, $3,000$ were used to validate it, and the last $1,000$ were used to test the path correction system and perform the analysis presented here. Once the LSTM was trained, the erroneous tracks used for testing of the system were also passed to it to predict the next 5 locations of the entity, providing a full set of erroneous tracks with the same number of locations (15) as the true paths. Figure 5.4a shows an example scenario with overlaid plots of the true path and the erroneous track. As discussed previously, when one of these erroneous tracks is passed to the track repair system, it generates a corrected track of the same length as the true path (7 true locations, 3 counterfactual locations, and 5 predicted locations). Figure 5.4b shows an example scenario with overlaid plots of the true path and the corrected track, and Figure 5.4c shows an example scenario with overlaid plots of the true path, the erroneous track, and the corrected track. It is clear from the plots in Fig.5.4 that the track repair tool substantially reduces the track error in the chosen



(a) Plot of true entity position and erroneous track.



(b) Plot of true entity position and corrected track.

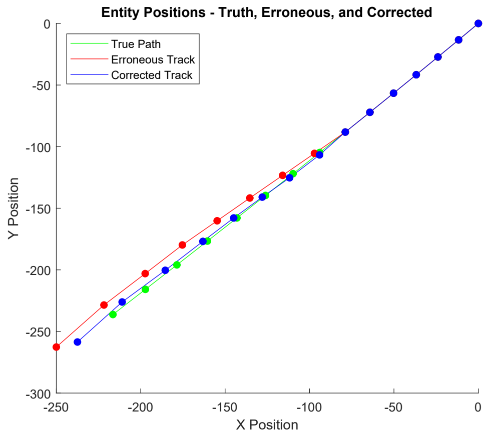


(c) Plot of true entity position, erroneous track, and corrected track.

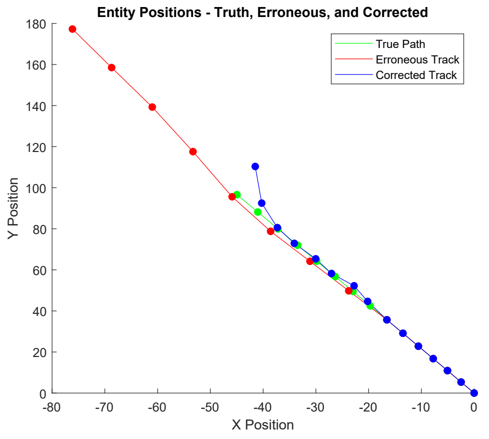
Figure 5.4: Plots demonstrating one scenario used to evaluate the proposed system.

example.

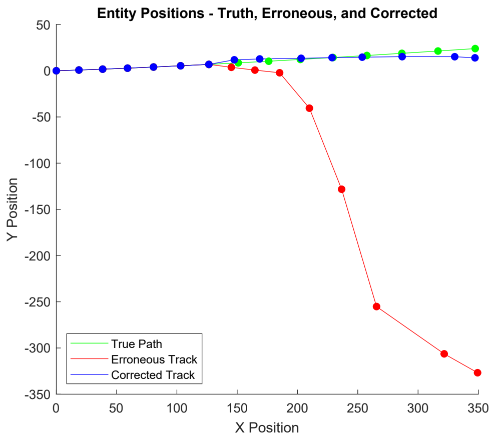
To demonstrate that the proposed track repair system is robust enough to handle the broad range of entity behaviors and drift that may be encountered in real-world applications, the scenarios used to test the track repair tool were designed to be diverse, with very different motion and error profiles. Figure 5.5 depicts several examples of



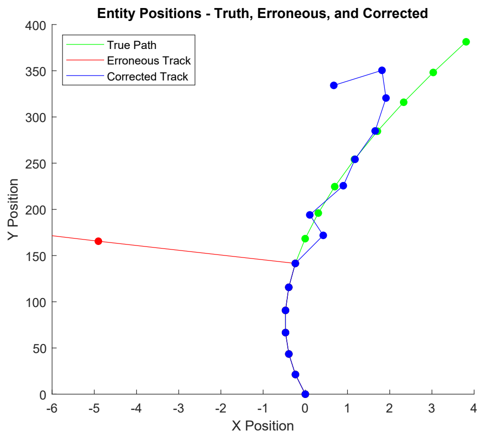
(a) Scenario with low error in erroneous track.



(b) Scenario with moderate overshoot in erroneous track.



(c) Scenario with extremely high error in erroneous track.



(d) Scenario with turning entity and high error.

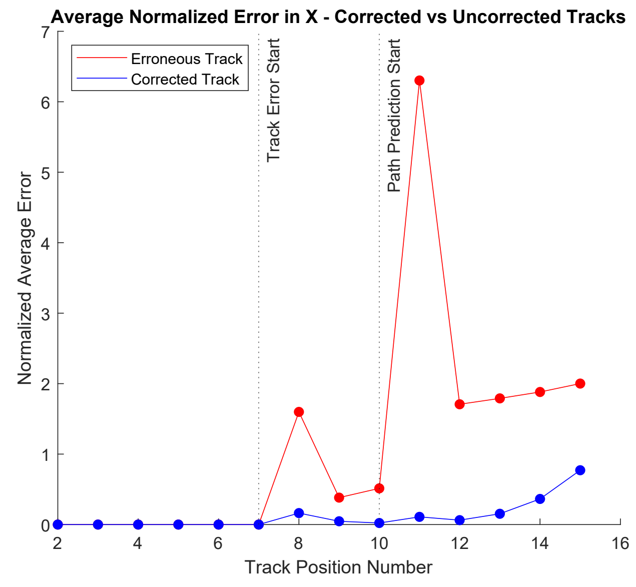
Figure 5.5: Example plots of several differing scenario types.

these scenarios. Of particular interest are those scenarios in which: 1) the suspect track has little error (Figure 5.5a), 2) the suspect track follows a similar trajectory as the true path but overshoots it (Figure 5.5b), 3) massive error exists in the suspect track (Figure 5.5c), and 4) the entity is slow moving or changing direction (Figure 5.5d). Each of these categories of scenario present unique challenges for a track repair method, and observing the behavior of the proposed method was critical at the proof of concept stage. When little error is present in the initial data, there is a risk of track repair introducing substantial additional error. Conversely, since multiple objectives in the system's GA encourage it to minimize deviations from the original track, there was a risk that the system would fail to correct for scenarios with large drift errors. Slow moving targets and those that are changing direction are more complex types of motion for correction and overshoot scenarios represent a challenging form of drift (due to its inherent similarity to the actual path). In each plot it can be seen that the corrected portion of the tracks (positions 8, 9, and 10) align with the true path almost perfectly, as do the first two predicted positions (11 and 12). While the error in the final three predicted locations is far less than is present in the erroneous tracks, it is clear that the error grows considerably the farther the system predicts into the future. This is due in large part to the compounding of minor prediction errors in the LSTM. As previously noted, the LSTM was not trained on noisy data, so its output was heavily affected by errors at the input. When used in testing, this LSTM statically used 10 input positions when making predictions, meaning that each subsequent future prediction replaced a true position in the input vector used to predict the location after it. As we will see in the discussion of the expanded implementation, this was largely mitigated by developing a more robust RNN for predicting future locations.

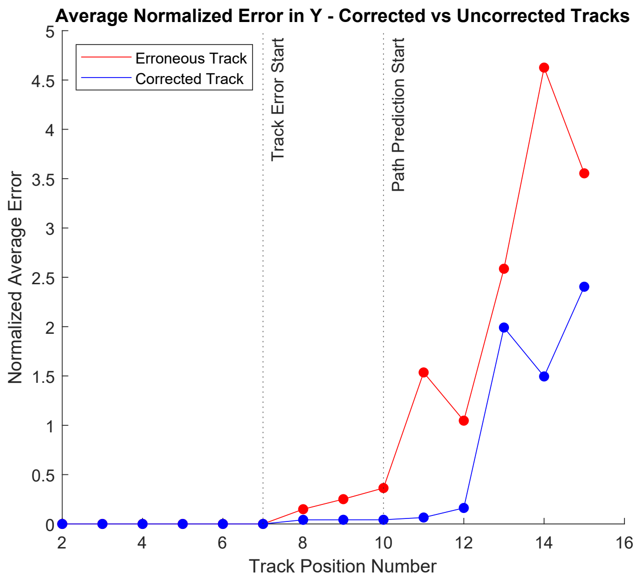
These observations of specific scenarios provide a good indication that the proposed counterfactual correction method successfully generalizes and mitigates drift error in a wide range of scenarios. However, examining individual scenarios is insufficient for determining performance characteristics of such a system. To better quantify this performance, all $1,000$ randomly generated test scenarios were run through the track repair tool and the normalized average error at each track position was calculated. Figure 5.6 depicts this error, broken down between error in the x- and y-dimensions.

The error at each location in both the erroneous and corrected tracks were computed for every test scenario by finding their distances from the corresponding locations in the truth path. Each is then normalized with respect to the truth location and the average of these normalized errors is calculated. For example, if the entity is located at 35 meters from the origin and the track reports the location at 42 meters from the origin, then the normalized error would be $(42 - 35)/35 = 0.2$. This normalization was performed in order to allow for the fair analysis of the system's performance when both very long and very short tracks are being corrected. Otherwise, the long distance scenarios dominate the analysis and could artificially exaggerate the performance of the proposed approach against the baseline erroneous data. Figure 5.6a and Figure 5.6b show the average normalized error in the x- and y-dimensions for each position in the sequence, respectively. From these figures, it is clear that the counterfactual routes proposed by the track repair tool consistently reduced the error in tracks to below 2% and reduced the error present in future predictions made using those paths by an order of magnitude. This clearly demonstrated the potential of the proposed system.

Additional analysis was performed to determine the distribution of the normalized cumulative error across all $1,000$ scenarios. This was done to both examine the spread of the system's error and to determine if any bias was present in either the system or the simulation. Figure 5.7a shows the distribution of the total normalized error across the entire run. Figure 5.7b shows the distribution of the total positional error, while Figures 5.7b and 5.7c show the distribution in the x- and $y-$dimensions, respectively. It is clear from these distributions that the error is not dominated by either the x- or y-dimension, and that the cumulative error of the corrected tracks is both low and tightly distributed. This provides a very strong indication that the system performed consistently across all of the test scenarios. In an unbiased system, we would expect



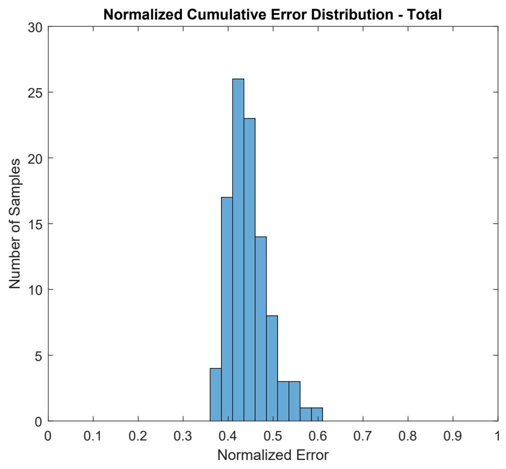
(a) Plot of the average normalized error in the x-dimension at each time step.



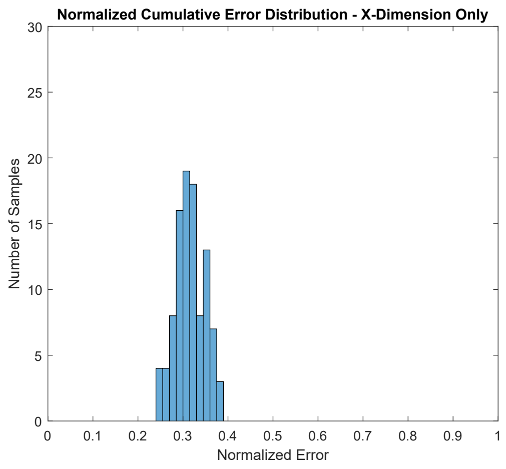
(b) Plot of the average normalized error in the x-dimension at each time step.

Figure 5.6: Average normalized error at each time step for 1,000 independent experiments.

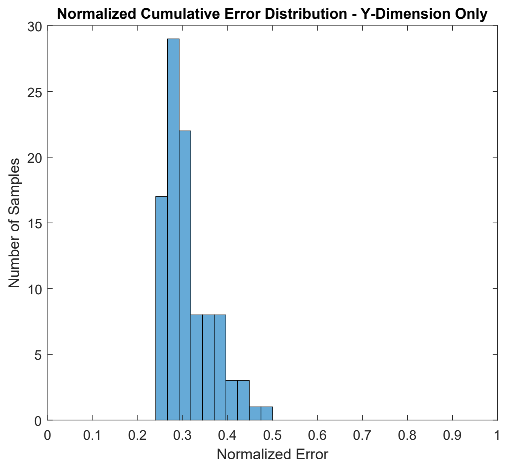
each of the three distributions to be approximately Gaussian. However, this is only true of the x-dimension's distribution. The distribution for both the total error and the y-dimension error are closer to log-normal. This was indicative of some bias in the trained LSTM model, which was subsequently mitigated in the system expansion.



(a) Plot of the total error distribution.



(b) Plot of the error distribution in x.



(c) Plot of the error distribution in y.

Figure 5.7: Plots of the average normalized error at each time step for 1,000 independent experiments.

As well as examining the system's performance on disparate scenarios, testing was performed to determine the consistency of the counterfactual tracks generated with the proposed method. Such analysis was necessary due to the use of a GA. Using the NSGA-II to propose and evaluate the corrected tracks through a continuous space meant that each time the tool was run, a different counterfactual path would almost certainly be selected. This behavior is acceptable so long as these tracks are consistent with one another and closely match the true path. To test this, a random test case was selected and the path correction tool was applied to it independently 100 times. The true path, erroneous track, and each corrected track were stored and used for analysis. Figure 5.8 depicts the selected scenario, including the true path, erroneous track, and a subset of the generated corrected tracks containing the highest and lowest error.

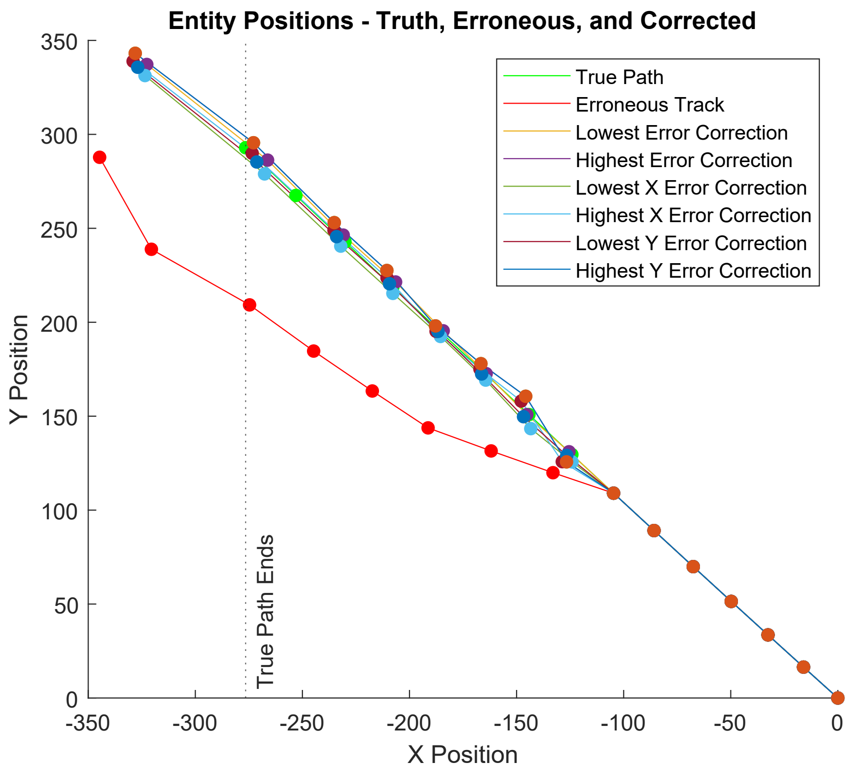


Figure 5.8: Plot of a single scenario run through the path correction tool repeatedly. The true path, erroneous track, and a subset of the corrected tracks with the highest and lowest errors are depicted.

In addition, Figure 5.9 depicts the mean error in each track position normalized with respect to the entity's true location. From this, it can be seen that the corrected tracks were highly accurate, with mean values (μ) sitting at under 2% error for not only the corrected portion of the track but also the first three future locations predicted using these corrected tracks. Further analysis of the data revealed that the standard deviation at each point never exceeded 0.035μ, indicating that the corrected tracks were tightly distributed. Based on both the calculated distribution information and the data plotted in Figure 5.8 it is clear that the tracks generated by multiple independent runs of the proof of concept track repair tool on identical input data were both accurate and highly consistent. It is clear that predictions made using these corrected tracks were also highly consistent and accurate, further demonstrating the validity of my proposed method.

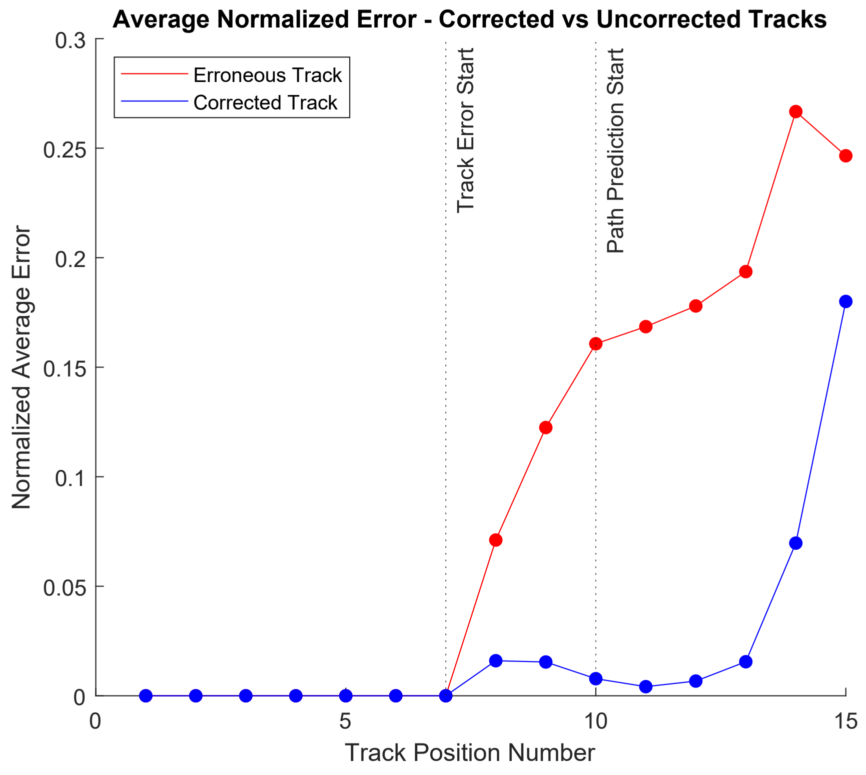


Figure 5.9: Plot of the average normalized error at each time step for 100 repeated runs of a single scenario.

5.5 Expanded Design and Testing

Initial implementation and testing of my counterfactual path correction method provided a solid proof of concept that could consistently mitigate the impacts of drift over a range of simple motion scenarios. It also highlighted some design issues that would need to be explored and overcome. In particular, the LSTM would have to be made more robust, additional physics-based objective functions for the NSGA-II should to be explored, and more expansive testing and characterization would need to be performed with comparison to state-of-the-art methods and more realistic simulation scenarios containing measurement noise. These expansions and extra analysis have been implemented and will now be discussed. I will note here that I worked with two colleagues, Geoffrey Dolinger and Timothy Sharp, on these updates. Specifically, Geoffrey Dolinger made the updates to my original LSTM model and Timothy Sharp created one of the new objective functions used in the NSGA-II. As I discuss the updates in more detail, I will highlight their specific contributions.

While the initial implementation of the counterfactual path repair method was discussed in terms of general spatio-temporal motion (due to its broad applicability to both tracking and navigation), expansion of the system was targeted more specifically towards correcting historic errors in navigational position. This means that all redesign work was done assuming input path data was to be associated with an “ego” entity. Put another way, it is now assumed that the sensor is located on the platform whose position is being observed. Specifically, the updated system is designed to remove errors related to drift in lower-fidelity sensor measurements when receiving an update from a high-fidelity sensor. A very common example of this is a navigation system that relies on a combination of GPS and INS sensors for positioning. The GPS updates rely on interactions with positioning satellites that provide an absolute frame of reference for measurements. This means that while error exists in these measurements, the error in each measurement is independent of the error in every other measurement and can be modeled as i.i.d. zero-mean Gaussian noise. For this reason, these measurements are considered high-fidelity, but the reliance on interactions with the positioning satellites means that updates come less frequently and may be intermittent. The INS, on the other hand, is typically an onboard sensor that updates quickly and regularly. However, it has a localized frame of reference, so INS errors compound over time, resulting in position drift. The data simulation model used for design updates, testing, and analysis is discussed in Section 5.3.1.

In many navigation systems, INSs are used for real-time navigation with periodic GPS updates to correct for drift. While this works well for correcting the error present at the time of GPS update, the historic error present in past measurements (e.g. between GPS updates) can be non-trivial to remove. Simpler state-of-the-art approaches like the various flavors of Kalman filter assume relatively simple motion models that struggle with more complex motions (like sharp turns or curving paths). The longer between GPS correction, the more drift may occur and the more complex the motion paths that may have been traversed. This is a common problem in the navigation domain, particularly when operating in areas where GPS updates may be intermittent or unavailable.

5.5.1 Expanded Approach

The updated design deviated from the original in three ways. First, an EKF was implemented for data preprocessing and system performance was evaluated both with and without the EKF. Second, new objective functions for the NSGA-II were implemented and tested. These new objective functions were largely inspired by the first principles physics behind entity motion. Finally, the LSTM was redesigned to help improve model robustness. Multiple configurations of these changes were examined, but ultimately a combination of the EKF, NSGA-II with five objective functions, and a multi-output LSTM trained on noisy data produced the best results. The final system architecture is depicted in Figure 5.10.

5.5.1.1 Extended Kalman Filter

The EKF was implemented in Matlab using a constant acceleration state update model and backwards recursion for state smoothing. This EKF was used in two ways. First, it was used as a pre-processing step to smooth out the noisy position data, which is consistent with modern navigation systems. In this case the EKF was used to predict the state at each of the first 15 positions, and after each prediction a correction was applied using the noisy measurement. Following the last correction the backwards recursion was applied.

The second use of the EKF was as an alternative path correction tool, in which it served as the current state-of-the-art for a baseline comparison. The EKF was again used to predict the state for each of the first 15 positions and was corrected by the noisy measurements. It was then used for state prediction without correction for positions $16 - 20$ (e.g. the drift region). Correction was not applied through this region to avoid contaminating the EKF's state estimation with low-fidelity data. Finally, it predicted the state at position 21, a correction was performed using the final high-fidelity update, and backwards recursion was applied. An example of both the EKF smoothing and EKF path correction are depicted in Figures 5.11 and 5.12. It should be noted that for the work presented here position measurements with sensor drift were never used to

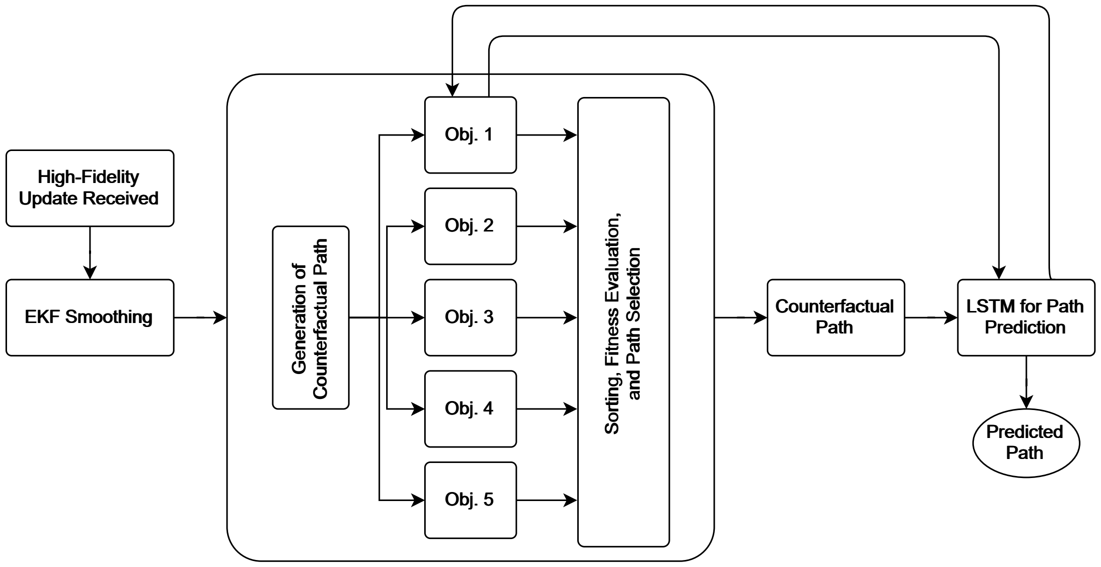

High-Fidelity Update Received
EKF Smoothing
Generation of Counterfactual Path
Obj. 1
Obj. 2
Obj. 3
Obj. 4
Obj. 5
Sorting, Fitness Evaluation, and Path Selection
Counterfactual Path
LSTM for Path Prediction
Predicted Path

Figure 5.10: System architecture for the proposed path correction tool.

correct the state estimation of the EKF.

5.5.1.2 Counterfactual Path Generation

As with the original implementation, the NSGA-II serves to perform the actual path correction. Since it is a multi-objective genetic algorithm, it is designed to generate multiple sequences of positions and then evaluate the fitness of each based on some number of objective functions. Sequences with a higher fitness are used to generate a new population of possible solutions through crossover and mutation. This iterative process continues until a predefined number of generations has been evaluated. Two major modifications have been made to this version of the NSGA-II. First, the length of the output was increased from 3 to 5 positions to match the longer drift region defined in 5.2. This exponentially increases the overall complexity of the solution space that the GA has to explore, making it a critical test of the proposed method's extensibility. The other major modification was the creation of new objective functions. Seven objective functions were developed and evaluated. Ultimately, five of them

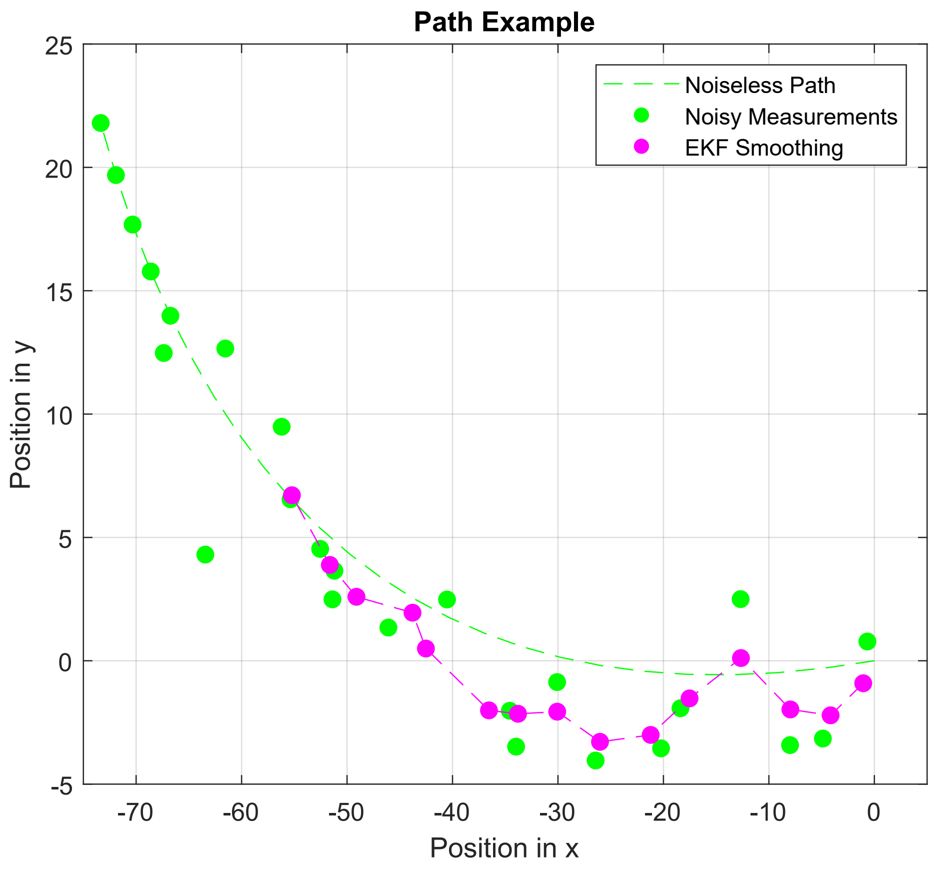


Figure 5.11: Example of noisy position with and without EKF smoothing.

were selected for use in the final system:

$$o_1 = w_1|x_{cor} - x_{pred}| \tag{5.8}$$

$$o_2 = w_2 \frac{\sum_{n=1}^{N} |x_n - x'_n| d_n}{r_n \sum_{n=1}^{N} d_n} \tag{5.9}$$

$$o_3 = w_3 \frac{\sum_{n=1}^{N} |v - v'_n| d_n}{\sum_{n=1}^{N} d_n} \tag{5.10}$$

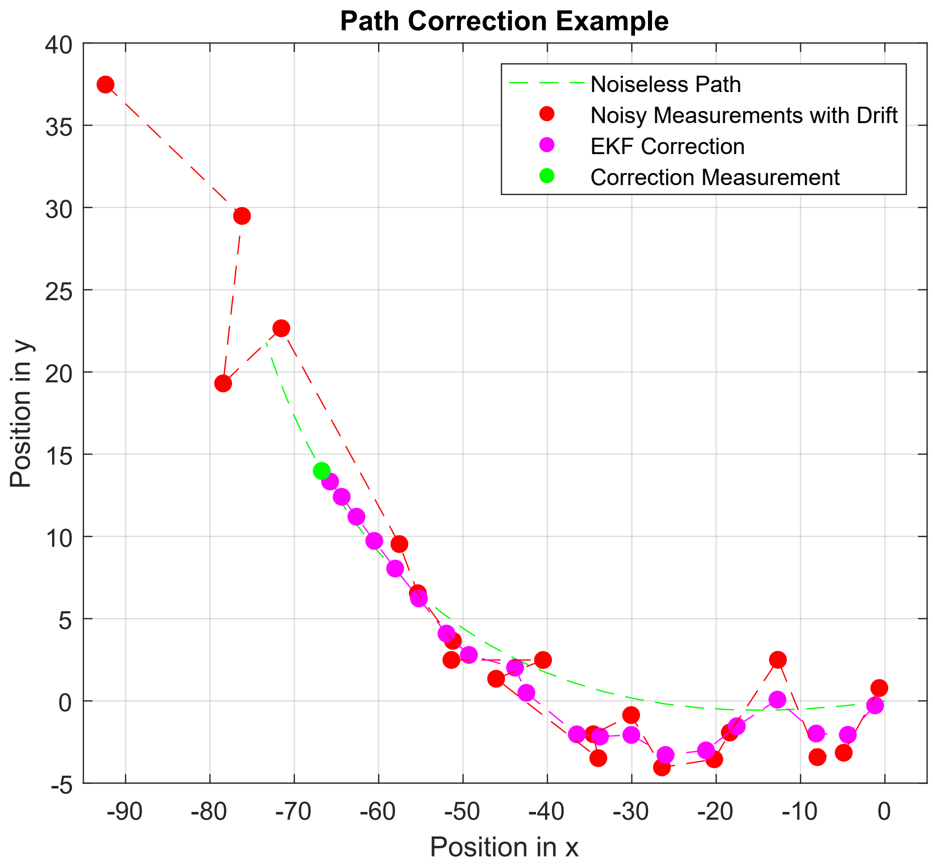


Figure 5.12: Example of EKF Path Correction.

$$o_4 = w_4 \frac{\sum_{n=1}^{N} |a - a'_n| d_n}{\sum_{n=1}^{N} d_n} \tag{5.11}$$

$$o_5 = w_5 \frac{\sum_{n=1}^{N} |a'_n - a'_{n-1}| d_n}{\sum_{n=1}^{N} d_n} \tag{5.12}$$

where $w_1, ..., w_5$ are weighting terms, x_{cor} is the final high-fidelity data-point being used for correction, x_{pred} is the first output of the path prediction LSTM using the proposed counterfactual path, x_n are the observed error data, x'_n are the proposed counterfactual points, d_n is a time-based decay term defined as $d_n = 0.9^n$, r_n is the range of possible locations for the proposed counterfactual, v is the last observed velocity calculated using the two most recent high-fidelity measurements before drift, v'_n is the velocity of the counterfactual data calculated using the current data-point and one previous one, a is the last observed acceleration calculated using the three most recent high-fidelity measurements before drift and a'_n is the acceleration of the counterfactual data calculated using the current data-point and two previous ones.

The first objective function, o_1, calls an LSTM trained to predict future positions of an entity based on a sequence of 20 past positions. For this objective, the first 15 high-fidelity measurements are paired with the 5 proposed counterfactual positions to form an input to the LSTM. The Manhattan distance between first output of the LSTM and the final correction measurement is calculated. The smaller this distance is, the higher the fitness of the solution. This objective is identical to the first objective in the original design and again ensures that the counterfactual paths have a trajectory that is consistent with the final GPS update that initiated the correction.

The second objective, o_2, calculates the Manhattan distance between the original observed track and the proposed counterfactual. Once again, this objective is drawn directly from the original system. However, this objective has one notable difference from its predecessor: a decay factor is applied that causes points later in the sequence to have a lower impact than earlier ones. It encourages the algorithm to generate solutions that are similar to the original observation containing drift, but grants more latitude to deviate from the original observation over time. This reflects the way that drift errors compound over time, where earlier points in the drift region likely contain less error than later points. Objective o_2 was designed based on the knowledge that although compounding errors resulted in drift, the original observations still contain information useful to the algorithm concerning the entity's position and motion characteristics.

The last three objectives use first principles physics concepts to encourage the generation of realistic motion paths. Objective function o_3 seeks to minimize the difference between velocity at each point of the counterfactual path and the velocity at the last high-fidelity update (before entering the drift region). Similarly, o_4 tries to minimize the difference between acceleration at each point of the counterfactual path and the acceleration at the last high-fidelity update. A decay factor was applied to o_3 and o_4 to reflect the increasing likelihood of changes in the motion characteristics the longer the system goes without GPS update. Objective o_5, the last incorporated into the final system, was designed by my colleague Timothy Sharp based on objective o_4. It attempts to minimize the change in acceleration from one time step to the next, again with the same time based decay as Objectives $o_2 - o_4$. This encouraged the algorithm to produce smoother, more realistic paths. Each of these objectives are predicated on the understanding that motion characteristics don't change instantaneously, and so can be used to intelligently restrict the solutions space explored by the GA.

Two additional objective functions were developed and used in early testing of the system. However, each had negative impacts on the performance of the system and were ultimately not used. They are:

$$o_6 = \sum_{n=1}^{N} g_n d_n,\ g_n = \begin{cases} 1,\ \text{if } |x'_n - x_n| \geq 0.1x_n \\ 0,\ \text{otherwise} \end{cases} \tag{5.13}$$

$$o_7 = w \cdot avg(|x_{pred[1,\ldots,5]} - x_{n_{inu}}|)d_n \tag{5.14}$$

Objective o_6 was based on 5.6 from the original implementation and was designed to limit the number of discrete counterfactual positions that deviated from the original observation by more than 10%. However, its inclusion in the updated system caused the generated tracks to follow the original observations with drift too closely, substantially increasing the error. It was evaluated both as a supplement and as an alternative to objective o_2, but negatively impacted performance in both cases. The objective in o_7 tried to make more substantial use of the LSTM than o_1 and was designed by Geoffrey Dolinger. It used the original 15 high-fidelity measurements as inputs to the updated LSTM model which used those to generate 5 predicted positions for the drift region and sought to minimize the distance between this predicted path and the generated counterfactual. It was thought that the inclusion of the LSTM in this way might encourage the system to produce smoother, more consistent counterfactual path. However, this objective function did not make use of either the correction information or the inferred velocity and acceleration characteristics of the entity's motion. This meant that it was not able to effectively identify more complex motions, such as sharp turns during the drift region, resulting in a sharp increase in the system's error. As a result, o_7 was ultimately removed from the system. It is likely, however, that such a method would work well if modified to incorporate some additional state and correction data.

Hyper-Parameter	**Sweep Ranges**	**Optimal Value**
Population Size	$5 - 50$	20
Generations	$20 - 100$	50
Mutation	$10 - 50$	30
Crossover	$1 - 5$	5
Tournament Prob.	$0.1 - 0.99$	0.9
Tournament Part.	$2 - 10$	5

Table 5.2: Hyper-Parameters for the NSGA-II

After identifying appropriate objective functions for the target application space, hyper-parameter sweeps were performed on the various NSGA-II parameters to identify the optimal configuration in terms of the MAE present in the final counterfactual track generated. Table 5.2 shows the hyper-parameters sweep, the ranges of the sweeps, and the values selected for final system testing. It should be noted here that increasing the population size and generation hyper-parameters above the selected values did provide some minor decrease to the output MAE, but at the cost of a drastically increased run time.

5.5.1.3 Path Prediction

Layer	**Type**	**Input Shape**	**Output Shape**
1	Input	d=[1,2]×n=[10,15,20]	d×n
2	LSTM	d×n	d×n×[32,64,100]
3[opt]	LSTM	d×n×[32,64,100]	d×[32,64,100]
4	Dense	d×[32,64,100]	d×[50,100]
5[opt]	Dense	d×[50,100]	d×[25,50]
6	Output	d×[25,50]	d×m = [1,5]

Table 5.3: LSTM Parameter Experimentation

As with the original system, this update made use of an LSTM to predict future states of the entity's path, both for use in the first objective function and for performance analysis. However, the original implementation of the LSTM was trained on noiseless data and produced a single output. This meant that the original LSTM was heavily impacted by errors in the input data. Additionally, being trained to match a single output did not encourage the LSTM to generate predictions with consistent trajectories, increasing prediction errors over time. It also meant that it could be used to predict path sequences, but needed to be run iteratively to do so.

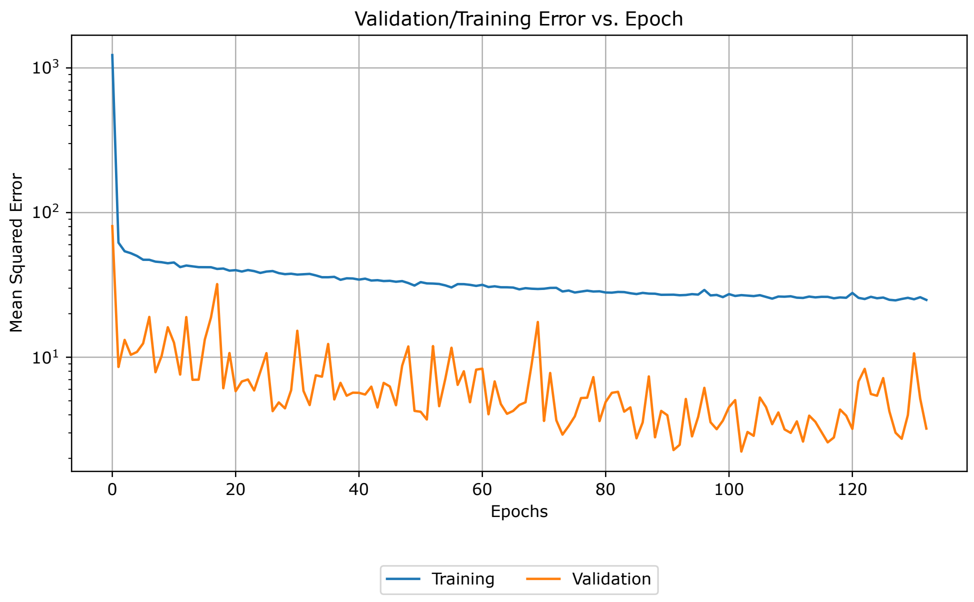
To improve my original path prediction LSTM, my colleague Geoffrey Dolinger took the simulation data with Gaussian distributed noise $N(0,1)$ I provided to him and trained and evaluated 18 different model architectures by exploring configurations of the parameters in Table 5.3. This involved varying the number and size of layers within the LSTM architecture. Each trained model was evaluated based on the MSE of its predicted paths.

Layer	**Type**	**Input Shape**	**Output Shape**
1	Input	1×15	1×15
2	LSTM	1×15	$1\times15\times64$
3	LSTM	$1\times15\times64$	$1\times15\times64$
4	Dense	$1\times15\times64$	1×100
5	Output	1×100	$1\times$m=5

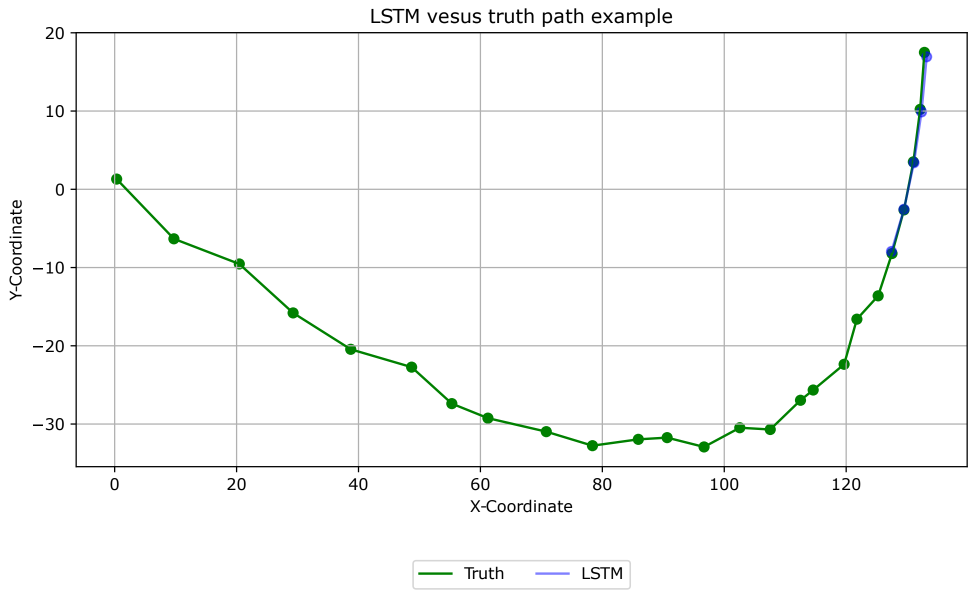
Table 5.4: Updated LSTM Architecture

Ultimately, Geoffrey Dolinger identified the LSTM architecture in Table 5.4 as the optimal configuration for the updated system. This model used exponential linear unit (ELU) activation functions for all layers except the final, linear output layer. The dense layer included a dropout of 10% to prevent overfitting. Training was conducted with a learning rate of 10^{-4}, mean squared error (MSE) as the loss function, and a patience value of 30. Note that the LSTM was trained on noise with lower variance than that that was used to evaluate the performance of the full system. Additionally, training on noisy data made the updated LSTM much more resilient and training it to output a sequence of length 5 significantly reduced the MSE of the model and caused it to produce much smoother outputs.

Figure 5.13a shows the training MSE of the final LSTM model. The model converged quickly and had a low, stable validation curve. Figure 5.13b shows and example path containing both noisy input data in green and the LSTM path prediction in blue. It is clear from this example that the model successfully learned to produce highly accurate path predictions with consistent, realistic trajectories in spite of the noise present at the input.



(a) Training and Validation Loss for LSTM



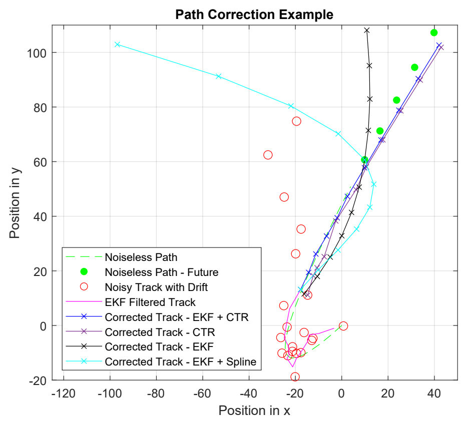
(b) Example Output of Scenario and LSTM

Figure 5.13: Plots Representing LSTM Training and Testing.

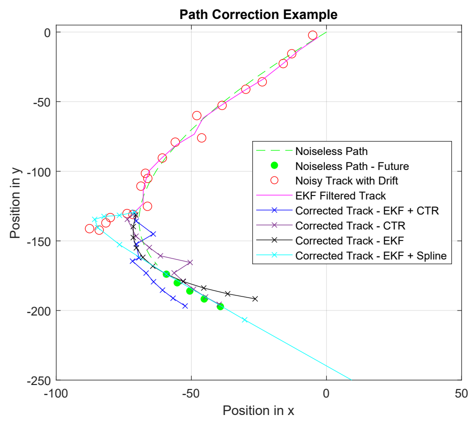
5.5.2 Expanded Testing

Testing of the final updated system was conducted in two parts. For the first, $3,000$ scenarios were generated (as described in Section 5.3.2). Path correction was performed on the drift region of each scenario using four methods: 1) the updated counterfactual method with EKF pre-processing, 2) the updated counterfactual method without EKF pre-processing, 3) EKF correction with backward recursion, and 4) EKF pre-processing coupled with spline interpolation. 1)-3) were described previously in Section 5.5.1. For 4), EKF smoothing was performed on the initial segment of 15 high-fidelity measurements to reduce the impact of the Gaussian noise and then cubic spline interpolation was performed between those and the final correction measurement. The interpolation was performed using Matlab's built in spline interpolation method. Results for each correction method were compared in terms of the MAE present at each position.

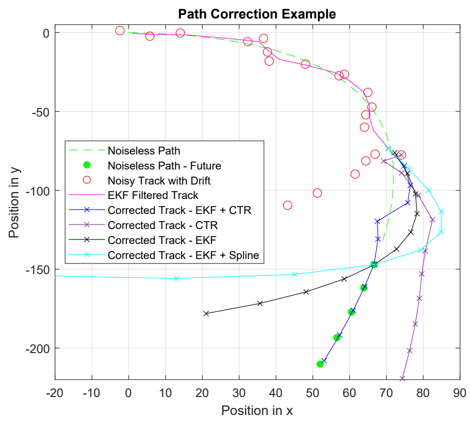
Figure 5.14 shows four examples of path correction for a given scenario. Each one of these scenarios demonstrates different unique motion characteristics that serve to display the utility of the updated method. In 5.14a, the entity immediately performs a tight turn with high relative noise present in the early measurements. Figures 5.14b and 5.14c demonstrate examples where the measurement drift occurs early in a turn, with the former having drift that resembles straight line motion and the latter appearing as a much tighter turn than was actually traversed. Figure 5.14d shows a simpler simulation with more linear motion and low relative measurement noise. As these examples demonstrate, the simpler EKF and spline approaches tend to perform very well in simpler scenarios, but struggle with more complex motion patterns or when the noise present in the non-drift measurement region is relatively high. For those scenarios where the entity motion is more linear, the repaired paths generated by the EKF and the spline interpolation have equivalent or slightly less deviation from the ideal path than the updated counterfactual track repair (CTR) method. However, the CTR method produces tracks far more consistent with the entity's true motions



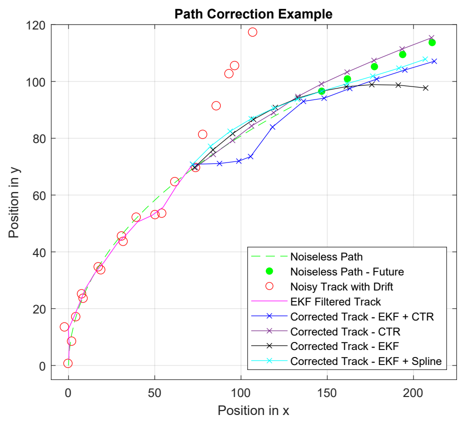
(a) Sample 22.



(b) Sample 93.



(c) Sample 608.



(d) Sample 1970.

Figure 5.14: Plots demonstrating the different correction path correction methods.

than either the EKF or spline methods whenever the motion is significantly nonlinear. This is particularly true when it is used in conjunction with EKF smoothing to reduce noise in the non-drift measurement region. Coupled with the fact that this approach has nearly equivalent performance to the EKF when applied to more linear motion scenarios, this indicates that the proposed approach is much more flexible than more traditional, non-ML approaches.

In addition, it is apparent from examining the paths generated by each method that the updated CTR approach produces paths whose trajectories match those of the entity's real motion more closely than either the EKF or spline approaches. This is further demonstrated by examining the last five positions of each path (the prediction region). In each example, the CTR approach produces path predictions with trajectories matching those of the true motion, while both the EKF and spline predictions tend to rapidly deviate from reality. Figure 5.15 further demonstrates this. This figure shows the average MAE at each point in the sequence over the $3,000$ test simulations. Only minor error can be seen in the first 15 positions resulting from the Gaussian noise. In positions $16-20$, the drift error can be seen to increase linearly over time. In this region, both MAE for the EKF and spline approaches is parabolic, increasing initially before falling to almost 0 at position 21 (the correction point). The MAE for each of these approaches then begins to increase rapidly for the predicted positions $(22-25)$.

The average MAE for the updated CTR approach, on the other hand, is roughly constant throughout both the repaired and predicted regions. This is significant because it indicates that the proposed method produces repaired paths whose motion characteristics are more consistent with the true motion of the entity. In the repaired region of the path (samples $16-20$), the CTR approach has equivalent average MAE to the EKF, but has much lower average MAE than the EKF in the predicted path region (samples $21-25$). This is most likely due to the ability of the CTR approach to better recreate the true motion characteristics of the entity through the drift region, resulting in more realistic predictions of the entity's future locations. Figures 5.16a and 5.16b break out the average MAE at each time step into the x- and y-components, respectively. It should be noted that there is no significant deviation between the MAE present in the x- and y-dimensions, indicating no dimension bias, and that the trends in each dimension are consistent with the full MAE depicted in Figure 5.15. The MAE

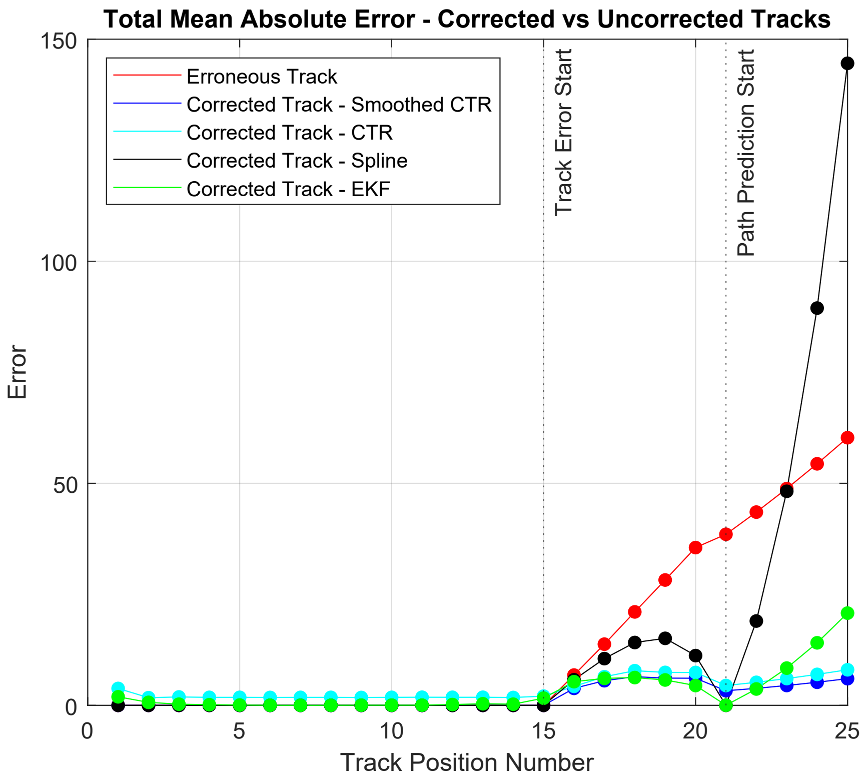
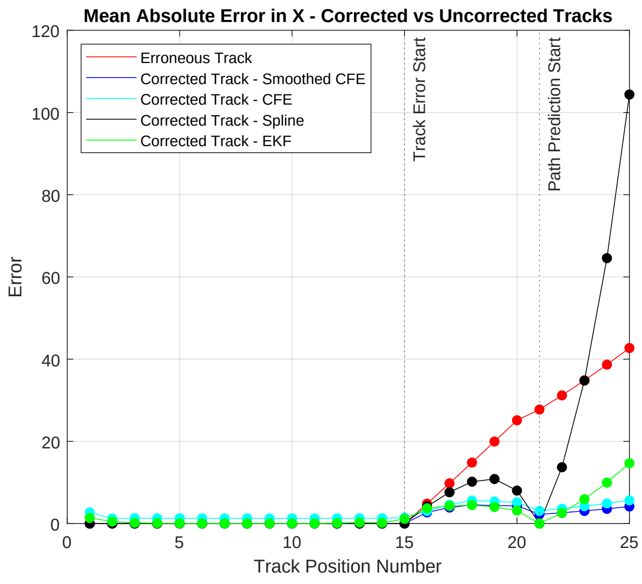


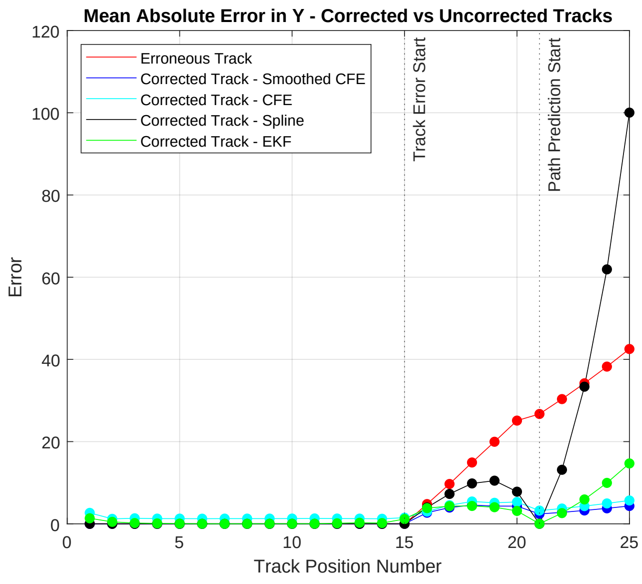
Figure 5.15: Total MAE at each path position, averaged over $3,000$ scenarios.

histograms for the full $3,000$ scenarios are shown for both the CTR with and without EKF pre-processing in Figure 5.17. These histograms serve to highlight that although the CTR will generate unique paths the error is low and tightly distributed. This consistency is desirable and builds confidence that example plots are representative of the tool's performance. Additionally, the mean MAE for all scenarios shows the the EKF smoothing reduces error in the CTR from 4.253 to 3.313 and using the CTR with EKF preprocessing is the optimal method.

The second round of performance testing consisted of running repeated tests on the same scenarios. Four scenarios were selected for this repeat testing. Each was chosen due to either being a challenging scenario (such as those with drift occurring at the



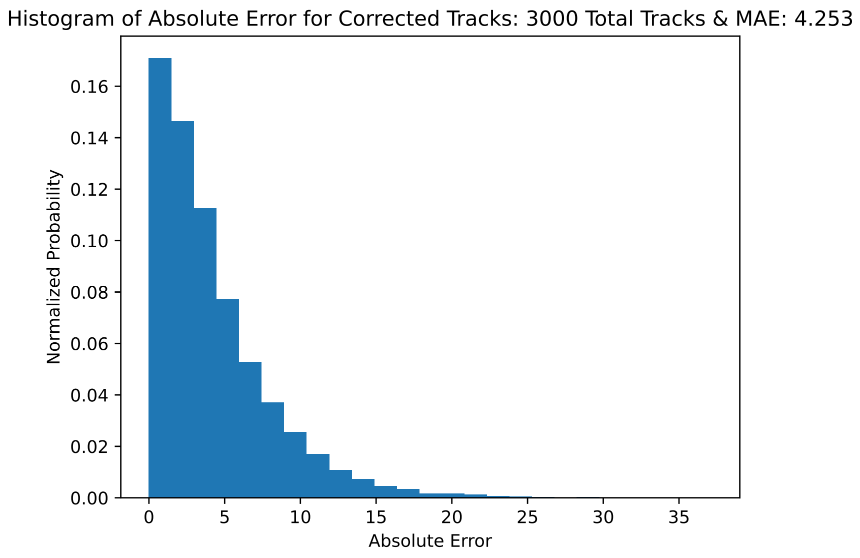
(a) MAE x at each path position.



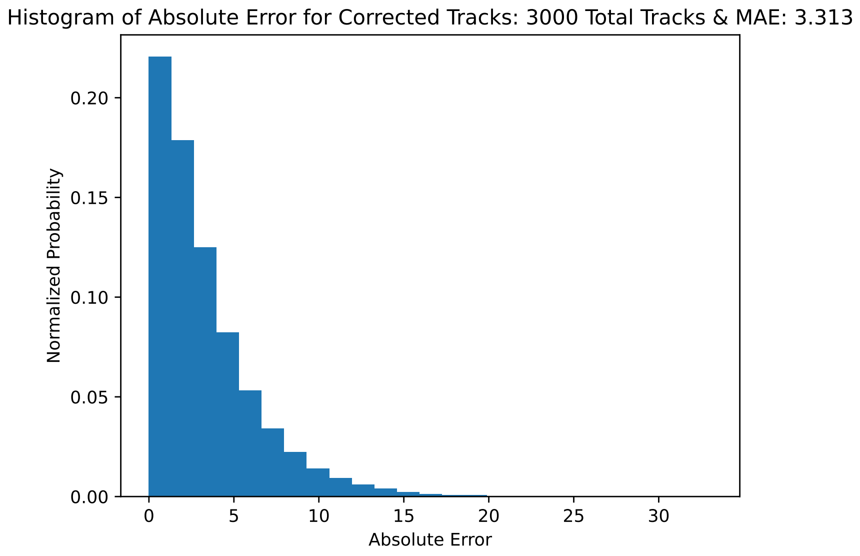
(b) MAE in y at each path position.

Figure 5.16: MAE in x and y at each path position, averaged over $3,000$ scenarios.

beginning of a turn) or because the proposed approach was observed to have high error during the series testing. This provided an interesting set of test cases to observe the repeatability of the updated path correction tool. With these tests, it is of particular



(a) MAE Histogram for CTR.



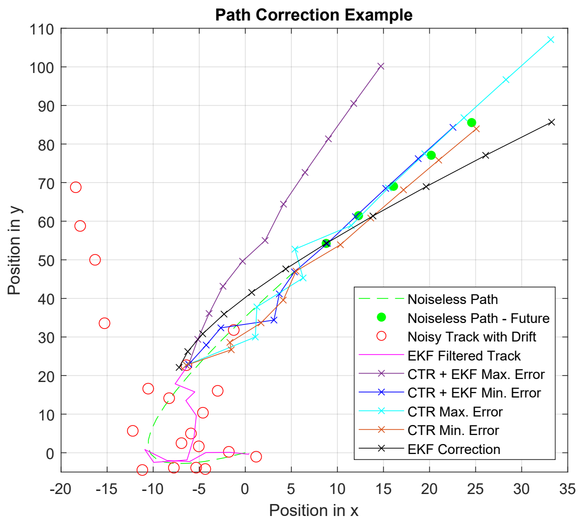
(b) MAE Histogram for EKF+CTR.

Figure 5.17: MAE Histogram over $3,000$ scenarios.

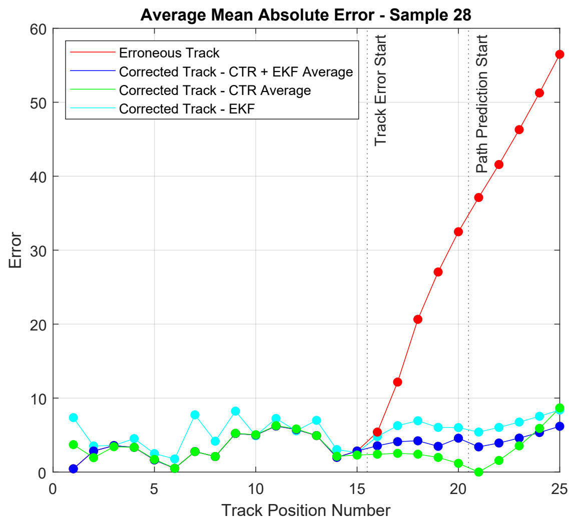
importance to observe the average MAE that arises from repeatedly running the same sample through the tool to verify that it does not vary drastically. Additionally, it is important to observe the maximum and minimum error that may be produced by the CTR approach to determine how repeatable the results are and how the tool's output might vary.

Each of the four scenarios was run through the tool 100 times, both with and without EKF smoothing. The average corrected paths generated by each run were saved and used to compute the average MAE of these tracks. Then, the corrected paths with the highest and lowest MAE for each scenario were identified and recorded. Figures 5.18-5.20 show the results of the repeat testing. Figure 5.18 demonstrates the difference between the results produced by the CTR tool with and without EKF pre-processing. Figure 5.18b demonstrates the impact of the pre-processing on the average MAE. Without being paired with the EKF, the updated CTR tool produces low error paths with average MAE that ranges between $5 - 8$ meters. With the EKF pre-processing, the MAE behavior is similar, but the average MAE is reduced by $1 - 2$ meters at each position. Figure 5.18b shows the scenario along with the highest and lowest error paths produced by the CTR tool both with and without EKF pre-processing. From this, it is clear that the highest error tracks in each case have slightly higher error than the pure EKF correction. However, each of those maximum error paths are smooth and have trajectories that are more consistent with the true path than the pure EKF correction, resulting in MAE that grows at a slower rate with recurrent predictions of future locations. It is also worth noting that the EKF pre-processing causes the CTR tool to produce smoother tracks. This makes sense, given that EKF smoothing of the input reduces the sporadic velocities and accelerations produced by the Gaussian noise present in the initial path measurements. Since several of the CTR's objective functions seek to minimize changes in velocity and acceleration of the corrected path based on those earlier observations, improving the fidelity of that data helps to remove oddities in the corrected paths this approach produces.

Arguably, the key limitation of the proposed path correction approach is in the



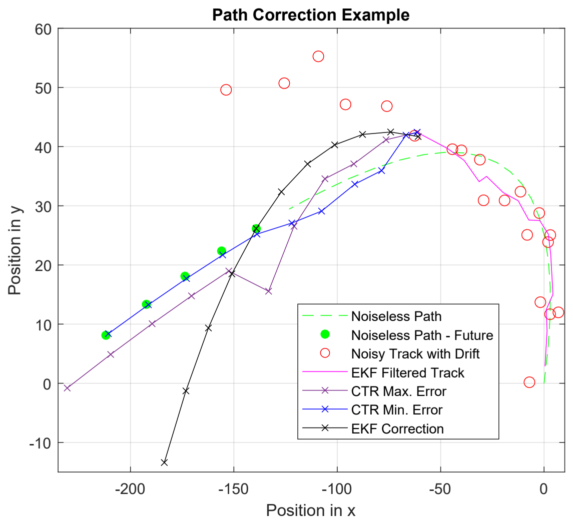
(a) Repeat Sample 614 - Correction Comparison.



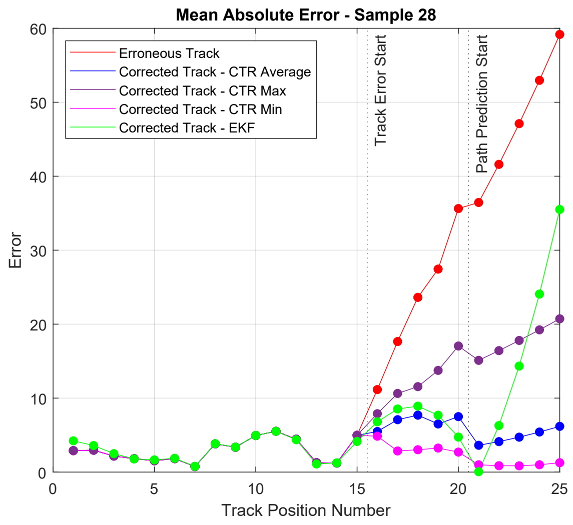
(b) Repeat Sample 614 - MAE Comparison.

Figure 5.18: Plots demonstrating the output of the proposed path correction method with and without EKF pre-processing.

potential variance of the paths produced by the GA. Figure 5.19 shows an example of this variance. Figure 5.19a show the maximum and minimum error paths produced by the CTR tool with EKF pre-processing, along with the path produced by the pure EKF correction, and Figure 5.19b shows the MAE for each. Clearly, there is substantial



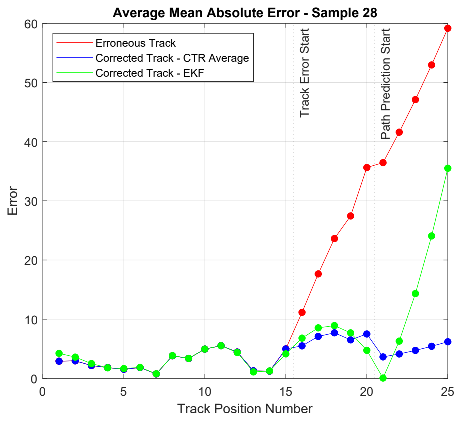
(a) Repeat Sample 28 - Correction Comparison.



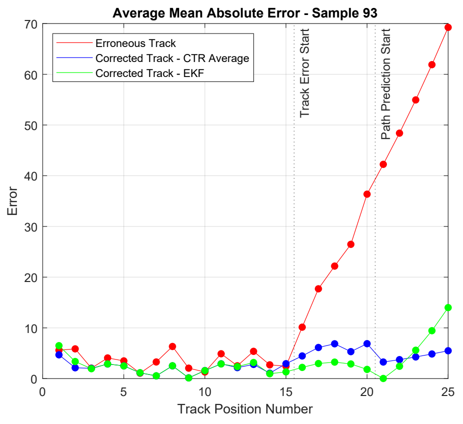
(b) Repeat Sample 28 - MAE Comparison.

Figure 5.19: Plots demonstrating the highest and lowest error outputs of the proposed path correction method.

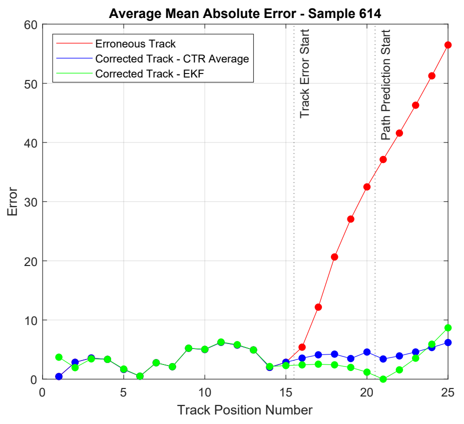
MAE variation in this scenario, with the error in the two paths differing by as much as 15 meters in the correction regions and by as much as 20 meters for the prediction region. However, it is also clear that both the highest and lowest error paths follow roughly the same trajectory, both of which are consistent with the true future path of the entity. This is evident from the sharp increase in the MAE of the EKF corrected path seen in the prediction region, which rapidly exceeds the error present in even the highest error path generated by the CTR approach.



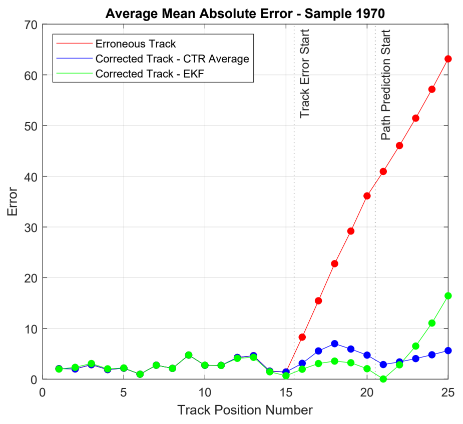
(a) Repeat Sample 28 - Average MAE.



(b) Repeat Sample 93 -Average MAE.



(c) Repeat Sample 614 - Average MAE.



(d) Repeat Sample 1970 - Average MAE.

Figure 5.20: Plots demonstrating the average MAE of 100 repeated runs of the proposed approach on the same sample.

The average MAE for each of the four repeat runs is shown in Figure 5.20. Each plot displays the MAE for the original data and the EKF corrected path, as well as the average MAE for the CTR corrected paths with EKF pre-processing. In each scenario, the average MAE for the CTR-based correction is nearly constant throughout both the correction and prediction regions, as was observed in the series testing. It should again be noted that in each case observed here, the MAE present in the EKF begins to rapidly increase immediately after the correction point, consistently overtaking the error in the CTR tracks within 3 timesteps. Further, the average MAE of the CTR correction in each of the four scenarios is approximately the same, with a mean value of approximately 6 meters and values ranging between $4 - 8$ meters. This is in stark contrast to the comparatively wide range of MAE produced by the EKF correction and in spite of the differences between each of the four scenarios tested.

5.6 Conclusions

In this chapter, I described the development of a new approach to removing drift errors in navigation and tracking scenarios. This method was based on causality aware ML and framed drift correction as a counterfactual evaluation problem. A simpler, proof of concept implementation was developed first using a four objective NSGA-II and an LSTM to propose and evaluate counterfactual paths with reduced drift error. This initial system was trained and tested on simulated data containing only drift error (i.e. no added measurement noise). Initial testing was performed to determine the viability of the proposed method, as well as to characterize its drift reduction and repeatability. An expanded version of the system was then developed that integrated the proposed method with an EKF, added additional objective functions to the NSGA-II, and a new LSTM model trained on more realistic data containing both drift and measurement noise. Testing of the expanded system was conducted to characterize its performance and compare it to that of the EKF and spline interpolation methods.

Testing of the initial implementation demonstrated that the proposed method could be used to substantially reduce drift error present in data over short intervals. It also showcased that the system output consistent tracks. Both findings were crucial to determining the validity of the proposed method. The first because it proved out the baseline required function of the system. The second was important due to the probabilistic nature of the system. The use of a GA was effective for developing a ML system that could propose and evaluate counterfactual scenarios. However, it also posed a general risk due to a lack of repeatability. Each time a GA is run, it will produce a different output, in this case resulting in a different set of tracks being proposed and evaluated each time the system is run. This had the potential to significantly hamper the effectiveness of the proposed method, as small measurement errors could compound over time. The consistency in track outputs when the same input was provided to the system demonstrated that the use of first principles objective functions forced the NSGA-II to restrict its exploration of the solution space such that the deviations between output track was negligible. Additionally, the system design ensures that its behaviors can be trivially explained and its outputs are inherently causal.

However, the trajectories produced in by this prototype were only stable enough to project tracks a few time-steps into the future before heavy degradation was observed. This was almost certainly caused by limitations with the initial objective functions paired with the lack of noise present in the data used to train the system. The original set of objective functions focused heavily on using position information for generating alternative tracks, with only one objective dealing with observed velocity characteristics and none on acceleration. This encouraged the desired reduction in drift error, but did almost nothing to encourage stable trajectories in the proposed tracks. That stability is essential for practical application of a drift correction tool in real applications and was a major focus of the follow up work. Additionally, since the LSTM was trained on ideal data, small errors in the track produced by the GA caused heavy error in the LSTM predictions.

The work to expand the system focused on three major areas. First, a more robust set of objective functions were developed for the GA targeted toward improving the stability of track trajectories. Specifically, objective functions related to minimizing sudden jumps in velocity and acceleration were added. Second, the the system was trained on data containing Gaussian measurement noise to make the LSTM more robust to small track errors. Third, the EKF was implemented both for comparison and for direct integration with the proposed system. Testing of the expansion demonstrated a huge performance increase over the prototype. Drift error was consistently mitigated and projecting tracks generated by the proposed system into the future resulted in low, almost constant error. This meant that not only was this implementation highly effective at removing drift, but that it was able to produce stable trajectories. In fact, the trajectories produced by the updated system were far more stable than those produced by the EKF. That impressive behavior demonstrates the effectiveness of both adding velocity and acceleration based objective functions and of training the LSTM on more representative data. It was also observed that using the EKF to preprocess the noisy input data and then applying the proposed drift correction method produced substantially better results than using either the EKF or the drift correction tool individually. Finally, the expanded testing confirmed that the proposed approach continued to produce consistent tracks for the same set of input data, despite the use of the GA.

It should be noted that while the proposed method demonstrated impressive results, considerable work would need to be done to integrate it into most practical systems. The current iteration is computationally costly and slow compared with existing algorithmic techniques. Additionally, alternatives to the NSGA-II should be investigated.

The objective functions designed for the GA were highly effective at minimizing track variation for the same input data, but the output of the GA is inherently not repeatable. That could heavily restrict the applications to which this method could be applied. I would like to investigate alternative approaches to generating counterfactual tracks, including algorithmic methods with parameter search and other ML methods like neural networks. It is worth noting here that while all ML models have probabilistic behaviors, many of them (including neural networks) do produce perfectly repeatable results when fully trained. Additionally, most of the computational cost and compute time for the current implementation are driven by the GA's search through the possible solution space. Using either an algorithmic or alternative ML method would almost certainly mitigate both issues. Such a substitution will take considerable design work, particularly to maintain equivalent performance and causal awareness.

The work presented in this chapter built directly on my previous research into ML explainability and causality. It also helped me to further develop the idea of creating ML augmented systems by breaking down complex problems into their component elements based on an understanding of the core physical interactions at work in a system. Specifically, the creation of the MOGA and definition of its objective functions allowed me to explore developing learning algorithms that made use of fundamental characteristics of motion. Additionally, the ultimately successful integration of the MOGA, LSTM, and the EKF and the superior performance of this integrated system provided strong supporting evidence for my claims that ML should be used to augment, rather than replace, existing radar detection and tracking algorithms.

Chapter 6

Using Meta-Cognition in Adaptive Detection

6.1 Introduction

In this chapter, I describe my ideation, development, and characterization of a novel ML-based adaptive detection algorithm. Based on a first principles understanding of the challenges that exist in the radar detection domain, I identified that the distribution assumption used to derive most existing adaptive detectors was a critical performance limiter. More specifically, it restrics the ability of radar systems to operate in scenarios containing challenging interference patterns and/or low SNR targets. I was then able to design a detection algorithm that could dynamically adapt in real time to changing clutter distributions based on a two-stage, meta-cognitive approach. The proposed algorithm utilized multiple low-cost, well-orchestrated neural networks to both identify the distribution of clutter and then dynamically set thresholds for existing test statistics. This design was successfully developed and characterized, which was initially published in [14].

Much of my early research into possible applications of ML to the radar domain focused on detection. It quickly became apparent to me that the overwhelming majority of existing detection algorithms relied on the assumptions that the distribution of interference was known and remained relatively consistent (as discussed in Chapter 3). Those adaptive detection algorithms that attempted to operate over a broader range of distributions largely did so by defining a number of test statistics or thresholds that were stored in a lookup table. They would then select from that lookup table by heuristically approximating the distribution. While these methods did provide considerable adaptability, they are ultimately inefficient and scale poorly as the complexity of the clutter scenario increases.

Given the nature of neural networks as universal function approximators, I theorized that integrating these ML techniques directly into existing adaptive detection algorithms could provide a much more general solution for maintaining the crucial CFAR property over a range of interference distributions with minimal loss of detection performance. An initial survey of the literature at that time revealed that a number of researchers had attempted to use ML to replace various detection algorithms and perform detection directly. Others had attempted to use NNs to classify distributions in the aforementioned lookup table detectors [143], and some tried to use NNs to dynamically set the threshold for a given detector and interference distributions [144]. Ultimately, both the attempts to replace existing detection algorithms with ML and to use NNs to effectively classify the distribution of interference in the literature had largely proven unsuccessful. This was due primarily to the fact that in each case the ML model was unable to converge to a high fidelity solution that generalized across distributions. The approaches to dynamic thresholding with NNs had proven somewhat more successful, but was not widely considered a viable technique. These made many of the same assumptions as existing detection algorithms and the CFAR property could not be mathematically proven when using these thresholds (even if CFAR-like behavior could be demonstrated experimentally).

Each of the attempts to use ML for adaptive detection mentioned above made one of several common mistakes when designing the ML models. First, many attempted to use singular models to perform these various tasks. Second, each approach involving interference classification relied on feeding large amounts of data from various interference distributions into the ML model, expecting it to accurately learn to identify the target distribution. Finally, many of these approaches attempted to train the model on real world data, limiting their ability to generalize.

Developing a generalized adaptive detector that can maintain both detection performance and CFAR-like behavior across a broad range of interference distributions is a highly complex problem space that also contains substantial regions of ambiguity. As a result, a singular model, however advanced, will be unlikely to learn a solution that generalizes well. Much of the ambiguity in this problem space is related to the behaviors of the various interference distributions themselves. Take SIRVs as an example: SIRVs such as the K, Pareto, and Weibull distributions have considerably different characteristics depending on the shape and scale parameters used. There are various shape parameter values that can be selected within these SIRVs at which the distributions are indistinguishable. For instance, as the shape parameters become very large each distribution tends towards Gaussian. Attempting to train a NN to broadly distinguish between the Gaussian, K, Pareto, and Weibull distributions is untenable given this degree of ambiguity between them. Additionally, models trained on real world data will be unlikely to generalize well given the limited availability of labeled, high fidelity data containing a broad range of interference distributions. A large amount of work has been done in the literature [66] to fit clutter signals of interest to well-defined distributions and simulate them. Given that such simulation models exist and the need for models to generalize, it is arguably a much better approach to design, train, and characterize ML detection algorithms on simulated data and then test on real world data to demonstrate how well they generalize.

Based on my observations of this existing literature, I have proposed a novel meta-cognitive approach to adaptive detection that leverages a system of ML models to both classify clutter distributions and dynamically set thresholds for an existing adaptive detection algorithm. Rather than attempting to broadly distinguish between distributions, this approach instead identifies regions within distributions with distinct statistical behaviors and trains a classification model to distinguish between them. Additionally, one regression model is trained for each distribution region to set the detector threshold. In the next sections, I will describe the system architecture, its implementation, and the testing used to prove the concept and characterize its performance. It should be noted here that I worked directly with a colleague, Geoffrey Dolinger, to refine, implement, and test this design. I will be careful to distinguish between our respective contributions to this work.

6.2 Related Work

Detection is one of the fundamental functions of any detection systems and is at the core of how autonomous system perceive the world. As a result, it has been the subject of considerable research, particularly in the radar domain. As I discussed in Chapter 3, detection is most commonly framed as a binary hypothesis test. For simple cases where the distribution of each hypothesis is perfectly known, the Neyman-Pearsons criterion can be directly applied. In general, however, the statistical properties of interference are not known and must instead be approximated. Adaptive detection algorithms such as the Reed, Mallet, and Brennan (RMB) detector [52] are derived to allow for improved detection performance by making use of a sample matrix inversion (SMI) technique to estimate a threshold for the rejection of correlated interference. The RMB utilizes a set of training data that is assumed to contain only Gaussian noise to estimate the covariance matrix of the noise and create a matched filter to apply to the signal being evaluated for detection. The output of the matched filter would then be compared against some threshold value based on the probability density function (PDF) of the signal to interference ratio (SIR).

The use of sample data that is assumed to contain only interference when estimating the interference covariance matrix has served as the basis for adaptive radar detection algorithms since. The family of space-time adaptive processing (STAP) methods have leveraged and expanded on the SMI technique to overcome the challenges presented by clutter[145, 146]. STAP dynamically adjusts filtering properties in both space and time [147] to improve the differentiation between target returns and clutter.

Much of the work done in developing adaptive detection algorithms since has been focused on defining the test statistic used to evaluate whether the return signal under evaluation is caused purely by noise/interference or by a combination of interference with a target of interest. Early algorithms focused on the case of homogeneous Gaussian noise with unknown covariance, while more recently work has been done to expand these implementations for use in the partially homogeneous and non-Gaussian cases [53, 54, 55, 56, 57, 58, 59, 60, 61]. This type of non-Gaussian interference has also been observed in additive interference from communications users in a congested spectrum [62, 63, 64].

The key issue with existing adaptive radar detection algorithms that is explored in this work is their sensitivity to the distribution of the interference they encounter. If that interference deviates from assumptions made during derivation of the algorithm, detector properties like false alarm rate may be significantly impacted. While some algorithms, like the ACE, are more resilient to these effects, they still experience this degradation [53, 80, 88, 89, 68, 90, 85, 93, 91, 92]. That resilience also usually comes at a general performance cost. In real world settings, it is not realistic to expect mobile radar systems to encounter consistent interference distributions. This issue is well documented in the literature, and much of modern adaptive detection research is focused on finding ways to overcome this limitation.

The use of ML techniques to augment adaptive detection algorithms is a current research area with significant potential [7, 8]. It is also the focus of the work presented in this dissertation. ML approaches excel at identifying patterns within complex data and classifying it based on those patterns. Previous work in this domain has explored methods for robust threshold selection across a known set of potential interference distributions [7, 8]. A number of other deep learning (DL) approaches to adaptive radar detection have been proposed in recent years. A review of recent DL-based detection algorithms is provided in [148].

Some use neural networks to attempt to replicate or replace existing detection approaches. In [149], Nuhoglu *et al.* propose using an autoencoder with bidirectional LSTM network to de-noise return signals and perform detection. [150] presents a method using a K-nearest neighbors (KNN) approach to combine the AMF and GLRT test statistics for detection in Gaussian noise. The authors of [151] develop a KNN-based detection statistic for use in K-distributed clutter. Both [152] and [153] perform detection using a deep neural network that is trained to replicate the behavior of a cell-averaging CFAR (CA-CFAR) detector on range processed data with Gaussian interference, and [154] uses a neural network trained to perform peak detection in the frequency domain. Mata-Moya *et al.* [144] use a deep neural network (DNN)-based approach for adaptive thresholding in range-Doppler processed data. Both random decision forests and recurrent neural networks (RNNs) for detection based on I/Q data are explored in [155].

Others have used DL to augment existing detection algorithms. In [156], a DNN is prepended to a GLRT detector to identify and remove non-linearities in received signals. The authors in [157] present a multi-headed DL system to estimate the shape and scale parameters of Pareto distributed clutter. Xiang *et al.* [158] propose a method of clutter classification based on the combination of kernel density estimation and a batch orthogonal matching pursuit algorithm. An adaptive detection framework for changing clutter backgrounds is proposed in [143] that uses a cumulative sum algorithm to identify areas where the clutter distribution changes, kernel density estimation to classify the new distribution, and a predefined look-up table of detection thresholds.

Much of the research in the last few years has centered on using DL on range-Doppler (RD) ([159, 160, 161, 162, 163, 164]) or synthetic aperture radar (SAR) processed data ([165, 166, 167, 168, 169]). Many of these use CNN-based architectures, like VGG-16 and U-Nets, to perform target detection and identification ([166, 167, 161, 168, 162, 163, 169, 164]). Others focus on using these architectures for clutter classification [160]. CNNs are so broadly used in this space due to how well they perform at identifying patterns in higher-dimensional data.

Each of the approaches above demonstrate the potential of using DL for detection. However, a number of challenges exist with each. Those that utilize RD processing struggle with detection of slower moving objects that fall near or within the clutter ridge. SAR processing is a specialized technique with high computational cost that is not broadly used for real-time radar detection. Larger DL models like CNNs, RNNs, and autoencoders are computationally costly and can be difficult to train on relatively simple datasets (such as I/Q data) without overfitting. Deep ANNs are interesting for use in detection, due to their speed and ability to generalize well. However, the relative simplicity of these models makes it challenging to train them to perform detection over a broad range of clutter distributions without a high level of model confusion.

Existing techniques using deep neural networks display the same core limitations present in non-DL detectors; either an interference distribution is assumed and used with a single threshold setting network, or a network is used purely for clutter classification and then a look-up table of detectors/thresholds is used. Setting thresholds over a wide range of distributions is too complex a problem for a single neural network to learn, particularly given the overlap in certain distribution regions. But if a distribution is assumed, then the model will not generalize well when presented with other clutter distributions. Look-up tables of detectors/thresholds limit coverage to the distributions and parameters contained within them, and grow rapidly to increase coverage or resolution.

6.3 Proposed Approach

Here I will describe in detail the design and implementation of the proposed meta-cognitive detection algorithm. Figure 6.1 depicts the architecture of the meta-cognitive detector. In this design, raw I/Q data was passed through some simple, heuristic pre-processing to prepare it to both be ingested by the ML models and used in the derivation of the test statistic. The outputs of this preprocessing, which will be described in more detail shortly, consisted of a cell under test, an estimated interference covariance matrix, and a set of down-sampled magnitude ordered statistics (DSMOS) from the background interference. The DSMOS were then passed to both a discriminator and a threshold selector. The discriminator consists of a simple, feed-forward classification network that classifies the interference into one of a set of discrete distribution regions (this concept of distribution regions will be described shortly). Each discrete distribution region that the discriminator has been trained to recognize has a unique threshold setting agent associated with it. These agents consist of feed forward regression networks that have been trained to identify which point within their associated region best fits the data and outputs a threshold for the detector. Finally, the estimated covariance matrix and cell under test are used to calculate the test statistic, which is compared with the threshold in a binary hypothesis to determine whether or not a detection is recorded. The meta-cognitive detector itself was implemented in Python, while the data generation was performed using Matlab. All ML elements of this system were initially developed using Tensorflow and have since been translated into Pytorch.

The data preprocessing block of Figure 6.1 consists of two sets of calculations and has three distinct outputs. The first set of calculations is used to estimate the interference covariance matrix of the cell under test. Since the true interference covariance matrix will never be known in practice, it is estimated from support samples using the method provided in (3.21). For the development of this adaptive detection method, it is assumed that the support samples contain only interference drawn from the same distribution as that present in the cell under test. This assumption is consistent with the existing literature on adaptive detection [13], and it is therefore considered reasonable. The estimated covariance matrix, along with the raw I/Q values for the cell under test,

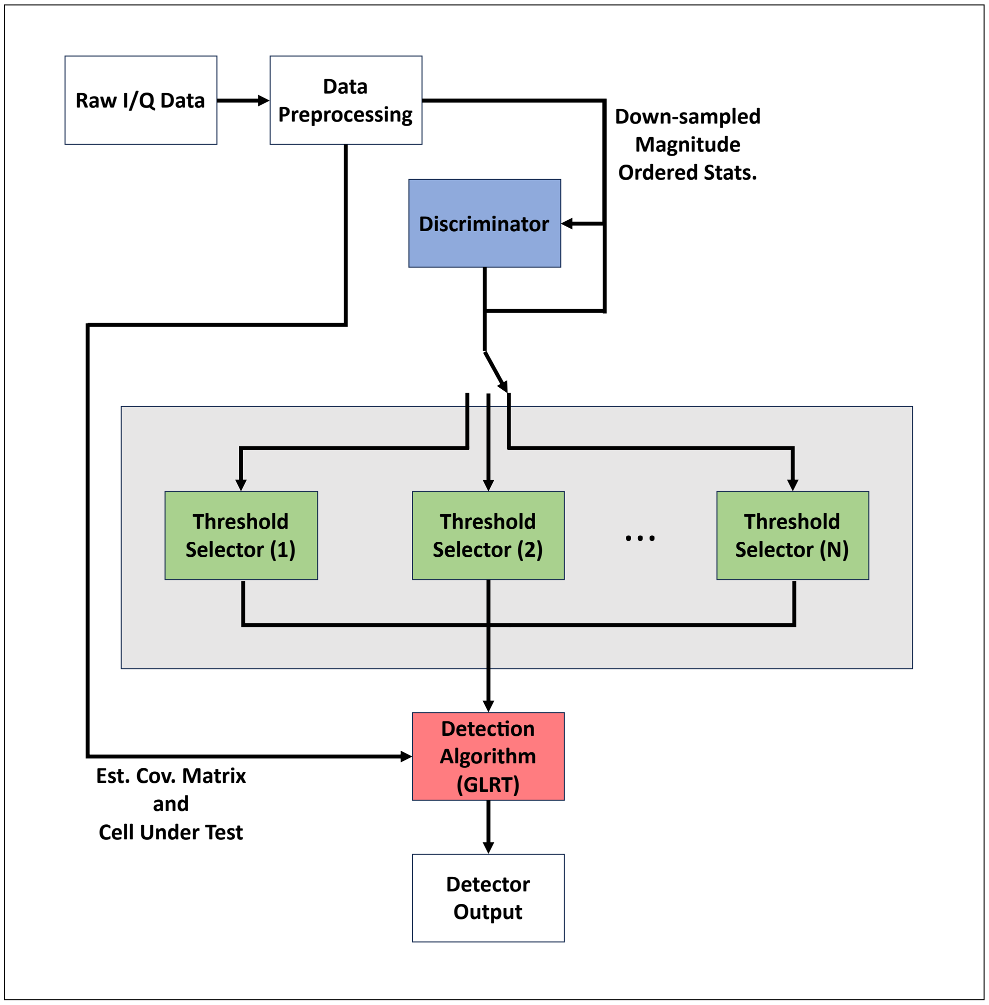


Figure 6.1: System architecture for the proposed meta-cognitive detector.

are output from the preprocessing block and used in the calculation of the detector's test statistic.

The second set of preprocessing calculations is used to take the raw sample support data and convert it into a format that can be ingested by the ML elements of the detector. The type and format of the data provided to an ML model will generally have a strong impact on its performance characteristics and ability to generalize. With that in mind, we initially explored several data formats for the inputs to the discriminator and threshold setting models. Each data format was used to train the various models and were evaluated in terms of model performance, computational overhead, and extensibility to different sample lengths and numbers of support samples. I will note here that the threshold setting agents were implemented and trained by me, while the discriminator agent was implemented and trained by my colleague Geoffrey Dolinger. As a result, we developed, tested, and evaluated these preprocessing calculations collaboratively. Note also that the feed-forward neural networks used in this work require inputs that are real-valued vectors of a fixed length, driving certain commonalities between these preprocessing approaches. We first examined the format requiring the most minimal preprocessing, in which the raw data samples were simply separated into their real and imaginary components, concatenated, and flattened to form a single vector of values:

$$\begin{aligned}\mathbf{z}_{in} &= vec([\mathbf{Re}(\mathbf{z}_k), \mathbf{Im}(\mathbf{z}_k)]) \\ &= [\mathbf{Re}(z_{k,(1)}), \mathbf{Im}(z_{k,(1)}), \ldots, \mathbf{Re}(z_{k,(N*K)}), \mathbf{Im}(z_{k,(N*K)})]^T\end{aligned} \tag{6.1}$$

where $vec\,(\bullet)$ refers to vectorization, $\mathbf{Re}\,(\bullet)$ is the real component of a complex value or vector, and $\mathbf{Im}\,(\bullet)$ is the imaginary component of a complex value or vector. This approach was considered first due to the low computational overhead during preprocessing and the lack of information loss accrued by inputting the raw support data. However, neither the discriminator model nor the threshold setting models consistently converged when training on data in this format. This is likely due to the stochastic nature of the data, which makes it nearly impossible for the NNs to identify consistent features during the learning process.

Next, we considered using a full set of magnitude ordered statistics:

$$\begin{aligned} \mathbf{z}_{in} &= Or(vec(\mathbf{z}_k)) \\ &= \left[|z_{k,(1)}|, |z_{k,(2)}|, \ldots, |z_{k,(N*K)}|\right]^T \end{aligned} \tag{6.2}$$

where $Or\ (\bullet)$ is the order function. This approach was considered for several reasons. First, the nature of clutter models as a combination of texture and speckle (as described in Chapter 3) means that the distribution of the clutter is generally characterized by its magnitude distribution. As a result, it was theorized that the magnitude of the support samples would contain sufficient information about the underlying interference distribution for the ML models to learn. Second, the use of ordered statistics is common practice in the adaptive detection literature ([170]) and provides some additional structure to the model input data, increasing the consistency of the features seen by each model. This structure/consistency in the input data should make it more likely that the models converge to consistent solutions. Additionally, the use of magnitude information removes the need to separate out real and imaginary elements of the input data, reducing its size and the dimensionality of the networks. Both the discriminator and threshold setting models were successfully trained on this input data format.

Finally, a down-sampled set of magnitude ordered statistics was considered:

$$\begin{aligned} \mathbf{z}_{os} &= Or(vec(\mathbf{z}_k)) \\ &= \left[|z_{k,(1)}|, \ldots, |z_{k,(512)}|\right]^T \\ \mathbf{z}_{ds} &= \downarrow_{64} \mathbf{z}_{os} \\ \mathbf{z}_{in} &= \left[z_{ds,(1)}, z_{ds,(2)}, \ldots, z_{ds,(n)}\right]^T \end{aligned} \tag{6.3}$$

where $\downarrow_{64}(\bullet)$ denotes down-sampling to 64 evenly distributed values. This approach again has all of the benefits of using magnitude ordered statistics, but with several additional advantages. Most importantly, this approach makes the ML components of the meta-cognitive detector largely agnostic with respect to the sample length and number of support samples (so long as $N * K \geq 64$). The further reduction of input dimensionality also reduced the size of the NN models and the computational overhead of the meta-cognitive detector. Using an evenly distributed subset of the magnitude ordered statistics during model training also reduces the likelihood of the NNs overfitting, which improves generalization. This data format was ultimately selected for the final system implementation both due to these benefits and the success of using it to train both the classification and regression models. These DSMOS, derived from the support data, the raw sample from the cell under test, and the estimated covariance matrix are the three outputs from the data preprocessing block.

After preprocessing the input data, the DSMOS are passed to the machine learning elements of the meta-cognitive detector. Rather than attempting to train a singular monolithic model, my proposed design consists of two levels of specialized ML models: 1) a set of regression models (here called thresholds selectors) specialized to operate in specific clutter distribution regions and 2) a higher level classification model (here called the discriminator) that identifies the distribution region of the clutter and selects the appropriate regression model. I would now like to draw attention to the concept of a clutter (or interference) distribution region. When discussing random processes, they are often described as broadly characterized by a probability distribution (Gaussian, exponential, lognormal, etc.). However, for more complex distributions, such as those commonly used to model clutter, the distributions have additional parameters that significantly impact their form and behavior. Good examples are the

K, Pareto, and Weibull distributions, which are each defined by a shape and a scale parameter. As its name suggests, the shape parameter alters the shape of the distribution while the scale parameter affects the spread of the distribution. These distributions will change drastically for different values of the shape and scale parameters. This is illustrated in Figure 6.2, which shows plots of the Gaussian distribution and the Weibull distribution with different shape parameters. In this dissertation, I use the term distribution region to refer to a range of shape/scale parameters within a particular distribution. For instance, one such region might be the Weibull distribution with scale $a = 0.5$ to 1 and shape $b = 1$. Note that these distributions often have regions of behavioral overlap, where they become extremely difficult to distinguish from one another. This can be readily seen in Figure 6.2 with the near perfect overlap between the Gaussian distribution and the Weibull distribution with $a = 3$.

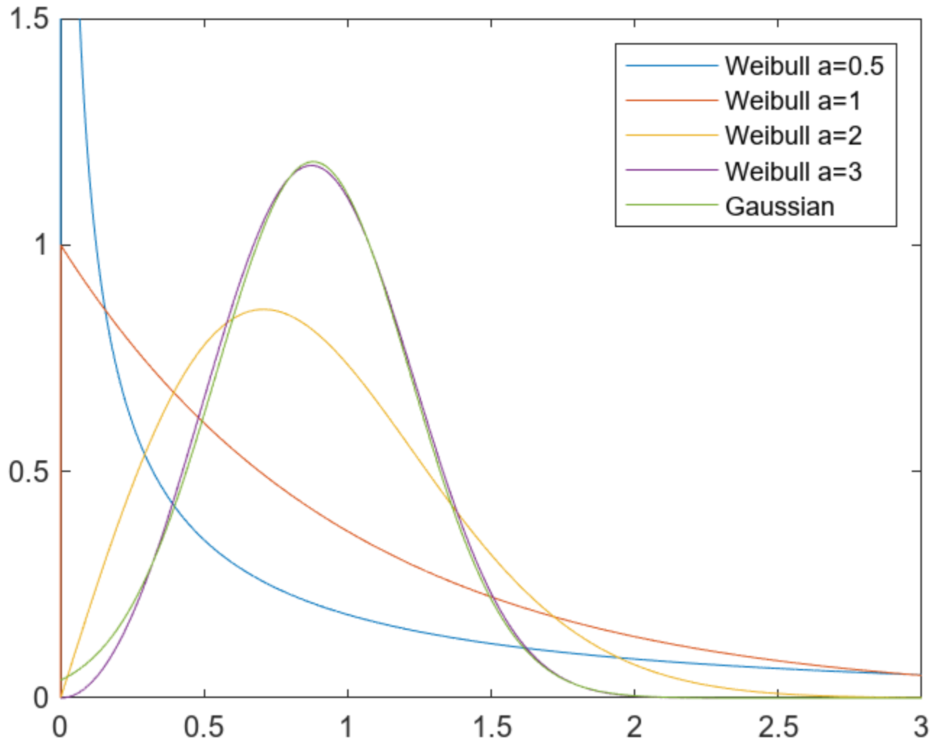


Figure 6.2: Example distribution plots based on Gaussian and Weibull distributions.

Previous attempts to use ML to either classify or operate within a distribution have attempted to broadly train agents to distinguish between distributions such as the Weibull, K, and Gaussian distributions. However, the level of ambiguity across these distributions means that attempting to train a NN on real or simulated data sampled from them will often either fail to converge or have significant confusion between the classification categories. One of the key innovations I have made in this field lies in my proposal to train models based on the notion of statistically unique distribution regions. I define statistically unique distribution regions as parameter ranges within a distribution whose behaviors are significantly different from one another. For instance, the K-distribution can be broadly divided into low shape parameter regions ($a < 1$), where it approximates an exponential distribution, middle shape parameter values ($1 \leq a \leq 3.5$, where it approximates a lognormal distribution, and high shape parameter regions ($a > 3.5$), where it approaches the Gaussian distribution. My core claim is that if one can identify regions of sufficiently unique behavior within the distributions, it is possible to consistently train ML models that will converge with low confusion between the operational regions.

Now one must define how to determine what constitutes "sufficiently unique" regions. Since the goal of this work was to develop an ML enhanced adaptive detection algorithm, in this case the GLRT, I initially used the threshold behavior of the GLRT to define these regions. These threshold values are set to maintain a desired P_{fa} for the detection algorithm and are directly related to the statistical properties of the interference distribution, making it a good metric to use for this application. Figures 6.3 and 6.4 show the GLRT threshold curves at a $P_{fa} = 10^{-4}$ for the K- and Pareto distributions, respectively. These threshold values were determined experimentally and show the strong, monotonically decreasing relationships between the threshold values and each distribution's shape parameter. Note also that each of these curves approaches the same static value (0.45) as the shape parameter gets large and the distributions approach Gaussian behavior. This supports my claim that the threshold behavior of the GLRT across each distribution is a sufficiently good metric for this application.

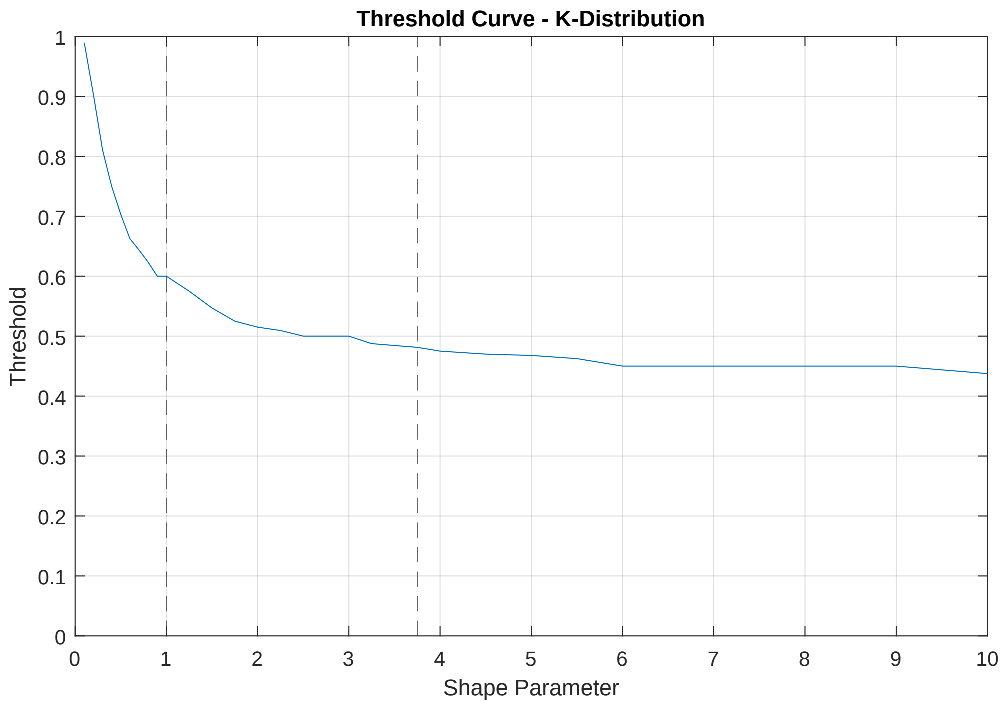


Figure 6.3: GLRT threshold curve for the K-distribution at $P_{fa} = 10^{-4}$.

Using the threshold behavior to determine the unique distribution regions that the classification and regression models are trained to operate over also has key design benefits for the NNs. It is clear from Figures 6.3 and 6.4 that the K- and Pareto distribution curves can be divided into three regions (signified by the vertical dashed lines) with very different slope characteristics. Regression NNs essentially learn mapping functions between input data and continuous output values. They are inherently more difficult to train and are generally less stable than classification networks. The more consistent the desired feature-output relationship (in this case reflected by the slope of the threshold to shape parameter curve), the smaller the network size required to learn it and the more accurate the outputs are. Both of these are critical elements in the design of regression models used to set thresholds in detection algorithms, which must both operate in real time and whose P_d and P_{fa} characteristics are extremely sensitive to threshold error. Additionally, the more similar the distribution regions, the more similar their threshold values, and the lower the impact of an incorrect classification become. It should be noted here that my colleague, Geoffrey Dolinger, was able to verify this claim using the Jensen-Shannon (JS) divergence. He also used the JS divergence to create a more generalized test for distribution uniqueness that we subsequently used for this line of research.

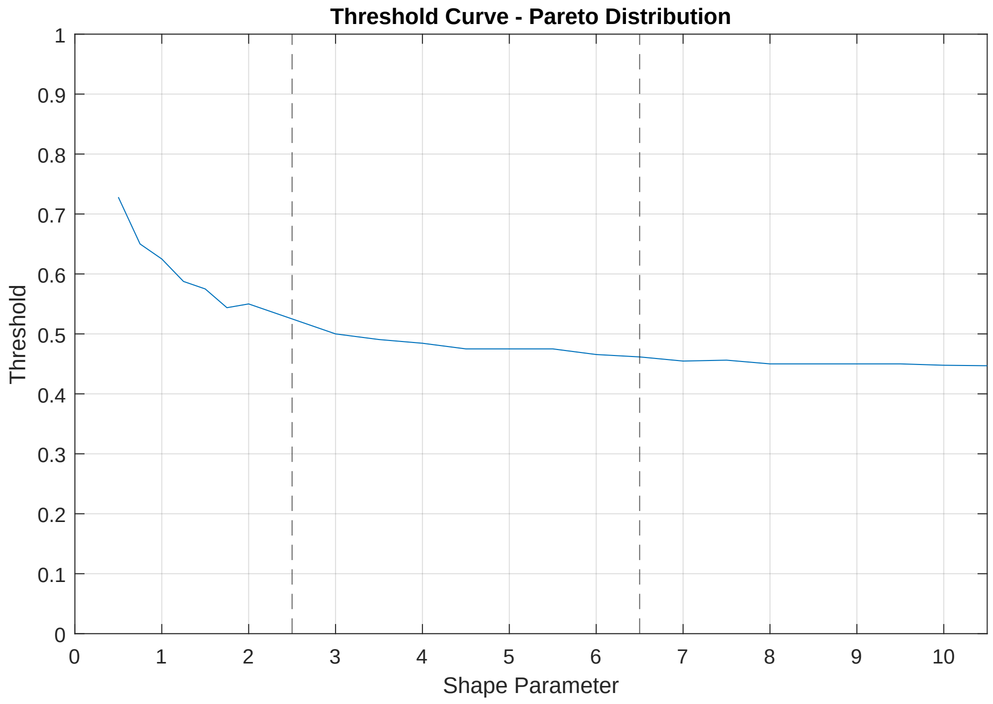


Figure 6.4: GLRT threshold curve for the Pareto distribution at $P_{fa} = 10^{-4}$.

After establishing the concept of statistically unique distribution regions, seven interference regions drawn from three distributions were selected for designing and characterizing a proof of concept implementation of the meta-cognitive detector. The Gaussian distribution was selected as a baseline, given that its behaviors are well understood, and the performance of the baseline GLRT is optimal in the presence of correlated Gaussian clutter. Three regions were then identified in each of the K- and Pareto distributions. As discussed in Chapter 3, the K- and Pareto distributions were selected because they are often used to model complex radar interference scenarios, like ocean clutter. The three regions within each distribution correspond to the regions displayed in Figures 6.3 and 6.4. These regions are listed in Table 6.1.

Classes	Gaussian	K-Dist	K-Dist	K-Dist
a Regions	-	0.1 to 1	1.25 to 3.5	3.75 to 10
a Training label	0	1	2	3
Classes	-	Pareto	Pareto	Pareto
a Regions	-	.5 to 2.5	3 to 6.5	7 to 10.5
a Training label	0	4	5	6

Table 6.1: Clutter Ranges and Labels

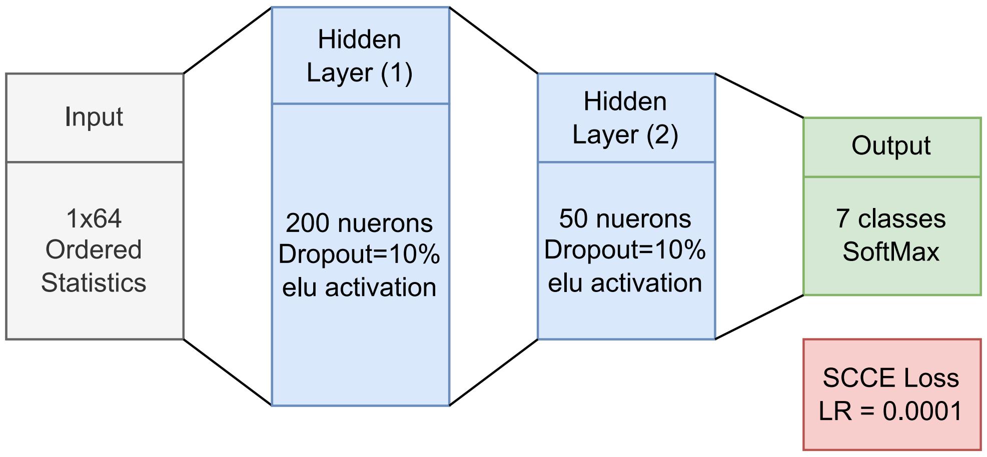


Figure 6.5: Discriminator Model

I will now describe the implementation and training of each level of the ML system. I would like to note, however, that while I personally implemented, trained, and characterized each of the threshold selection models, my colleague Geoffrey Dolinger implemented, trained, and characterized the discriminator model. I then performed the full system integration and oversaw its characterization. Dolinger's discriminator model consists of a classification DNN designed to distinguish between the seven clutter distribution regions described above. It takes as an input the support sample DSMOS and outputs a set of seven probabilities (that will sum to 1) based on its determination of how likely it is that the input data came from each of the seven regions. This is a DNN with two hidden layers, each of which uses the exponential linear unit (ELU) activation function. The final activation layer is a softmax, and the loss function is sparse categorical cross-entropy (SCCE). Figure 6.5 [14] depicts the architecture of Dolinger's discriminator.

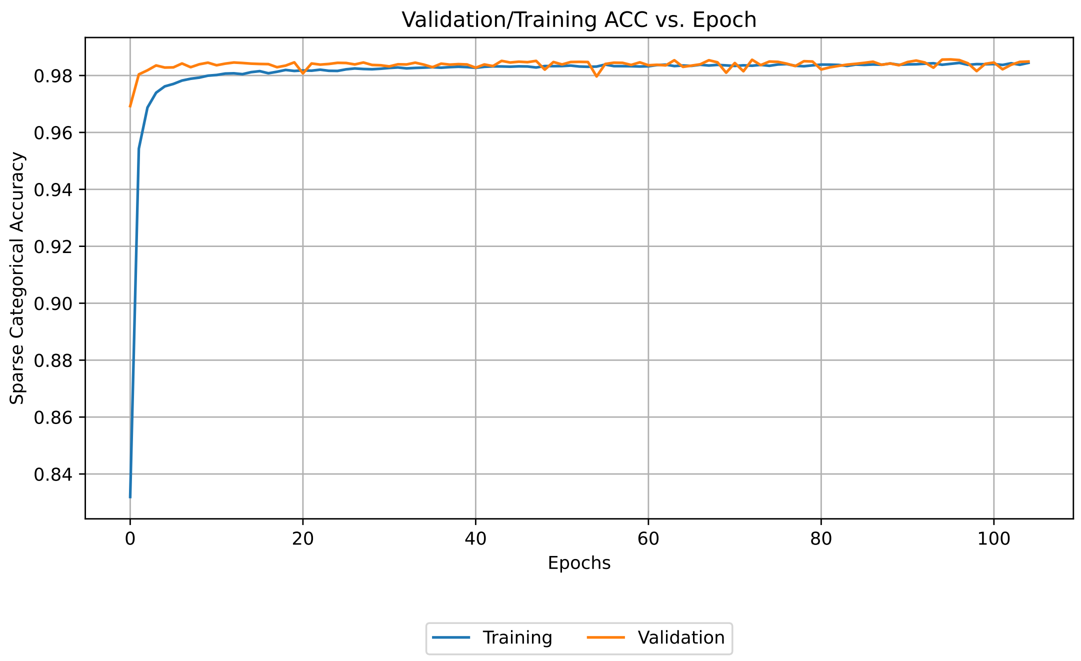


Figure 6.6: Discriminator Training Accuracy

To train the discriminator model, $20,000$ data samples consisting of the length 64 DSMOS (z_{in}) were generated for each of the 7 distribution regions. For each region other than Gaussian, these samples were generated using the shape parameter

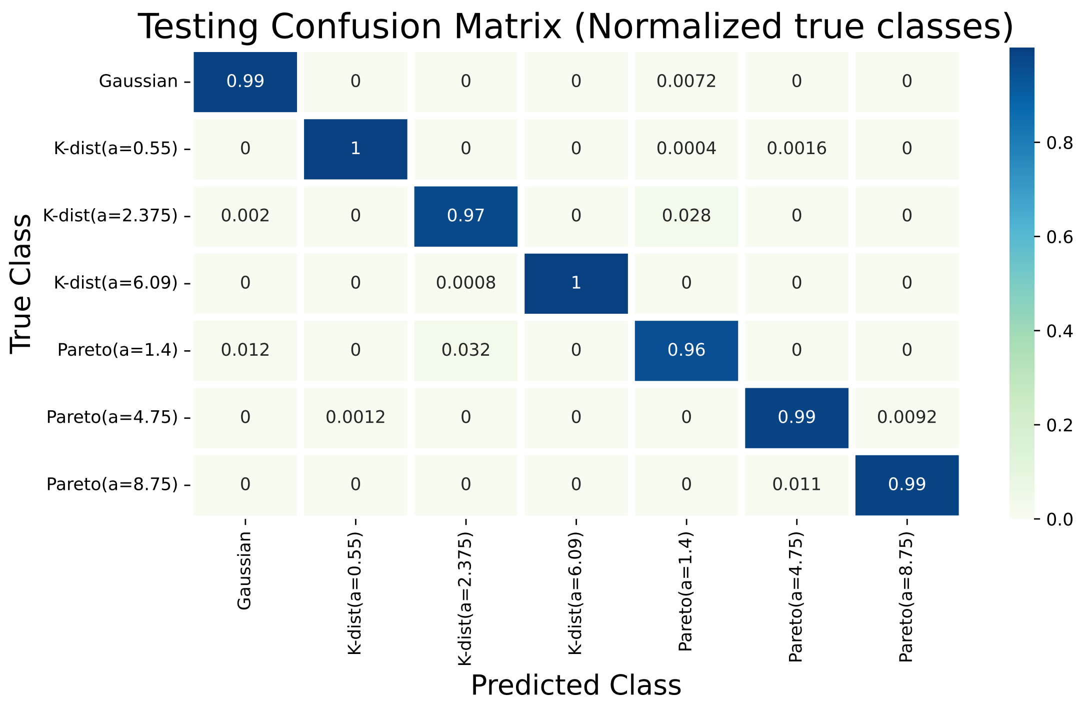


Figure 6.7: Discriminator Confusion Matrix

at the center of the region. Data labels for each region are listed in Table 6.1. The training data set was divided into 75% for training and 12.5% each for validation and testing. Training was conducted with a learning rate of 10^{-4} and a patience value of 30. This discriminator trained to a classification accuracy of 98% and displayed very little confusion between classes. These performance characteristics are critical for the meta-cognitive system, as misclassifications made by the discriminator will quickly drive up the false alarm rate. The training curve for Dolinger's discriminator is depicted in Figure 6.6, and its confusion matrix is shown in Figure 6.7

As was discussed above, the distribution space was subdivided into seven regions in order to develop a system that could detect targets in diverse clutter distributions while also maintaining CFAR-like performance. Each had a threshold selector implemented and trained to perform within that region with the Gaussian GLRT detector.

The first region was the simple Gaussian case for which the GLRT was designed. In this region, the threshold was selected using the closed form expression given by:

$$\lambda_0^{GLRT} = \frac{1}{(P_{fa})^{\frac{1}{K-N+1}}} \tag{6.4}$$

The other six regions were pulled from the K- and Pareto distribution models as listed in Table 6.1. Regression DNNs were used to set the thresholds for the GLRT operating in each region. Each of these DNNs had an identical model structure but were uniquely trained only on clutter data pulled from their target distribution region. They each take as an input the same length 64 DSMOS as the discriminator model and output a single continuous threshold value. These models contained three hidden layers, each of which used the ELU activation function. ELU was also used as the final activation function with Mean Absolute Error (MAE) as the loss function. This model structure is depicted in Figure 6.8.

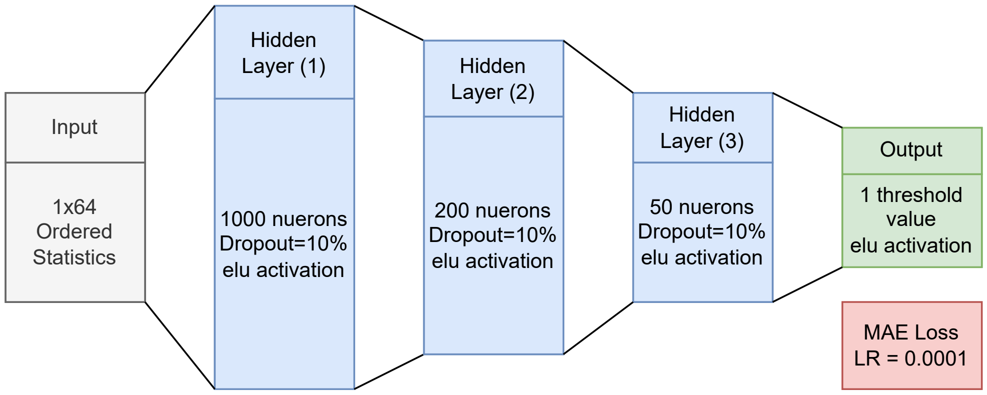

Input
1x64 Ordered Statistics
Hidden Layer (1)
1000 nuerons Dropout=10% elu activation
Hidden Layer (2)
200 nuerons Dropout=10% elu activation
Hidden Layer (3)
50 nuerons Dropout=10% elu activation
Output
1 threshold value elu activation
MAE Loss LR = 0.0001

Figure 6.8: Threshold Selector Model

Each of the threshold setting models was trained on data samples generated from the distribution/shape parameter pairs listed in Table 6.2. Since regression models learn to approximate continuous transfer functions, I was able to train these models at discrete points distributed throughout each target region with the expectation that the final transfer function would allow it to operate over the full region. This was verified during characterization, which will be discussed later. Training labels for the threshold setting models were drawn from the ideal threshold values shown in Figures 6.3 and 6.4 which were generated for a target $P_{fa} = 10^{-4}$. Again, these ideal threshold values were found experimentally using Monte Carlo analysis. This is commonly done in the literature on covariance inversion detectors due to the lack of closed form expressions for calculating thresholds in non-Gaussian clutter.

Filter	**Shape Parameter (a) for Threshold Labels**
K_1	$[0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00]$
K_2	$[1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50]$
K_3	$[3.75, 4.00, 4.25, 4.50, 5.00, 5.50, 6.00, 7.00, 8.00, 9.00, 10.00]$
P_1	$[0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 2.50]$
P_2	$[3.00, 3.50, 4.00, 4.50, 5.00, 5.50, 6.00, 6.50]$
P_3	$[7.00, 7.50, 8.00, 8.50, 9.00, 9.50, 10.00, 10.50]$

Table 6.2: Shape Parameter (a) for Threshold Labels

For each of the six threshold selection models, the training data set consisted of $20,000$ DSMOS (z_{in}) and associated labels generated at each of the shape parameter values listed in Table 6.2. 75% of the data was used for training, with 12.5% used for both validation and testing. These models were trained with a learning rate of 10^{-4} and a patience value of 30. The training loss curves for the Threshold Selectors can be found in Figures 6.9 and 6.10.

While these training curves do provide a good indication of whether or not the regression models converged to low error solutions, they give no direct indication of how well the models will perform when integrated with the GLRT. Given the tight performance requirements for detectors and the need to maintain CFAR-like behavior without substantially reducing detections, it was essential to characterize the P_{fa} and

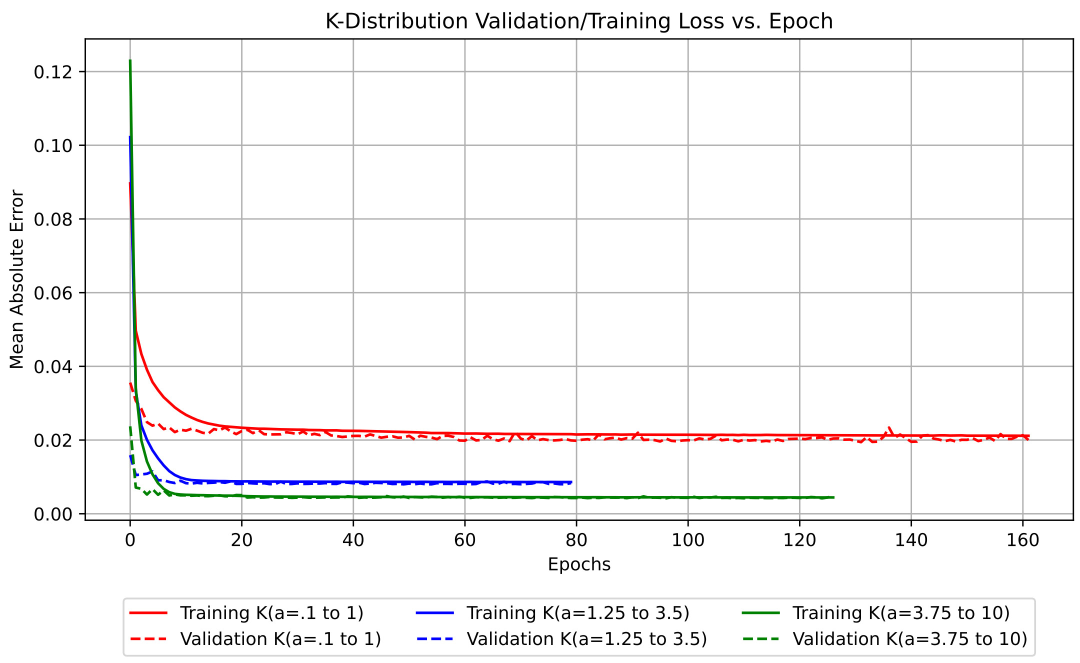


Figure 6.9: K-Distribution Training Loss

P_d performance of the GLRT when its threshold was being set by the trained regression models. This was done prior to full system integration to isolate potential sources of error. After being trained, each threshold selector was integrated with the GLRT, and its performance was verified using $100,000$ pairs of H_0 and H_1 test samples and associated support samples at each distribution/shape parameter listed in table 6.2. This data was generated with $N = 16$ and $K = 32$. Figures 6.11 and 6.12 depict the P_{fa} performance, and Figures 6.13 and 6.14 show the detection performance. In each figure the transition from one threshold selector model to another is denoted by a vertical, dashed black line. The region to the left of the vertical, dashed red line in Figures 6.11 and 6.13 is an extremely heavy-tailed region of the K-distribution that proved to be particularly challenging. Significant analysis of the behaviors of both the proposed meta-cognitive detector and various classical covariance inversion detectors has been done and will be discussed later in Section 6.4.

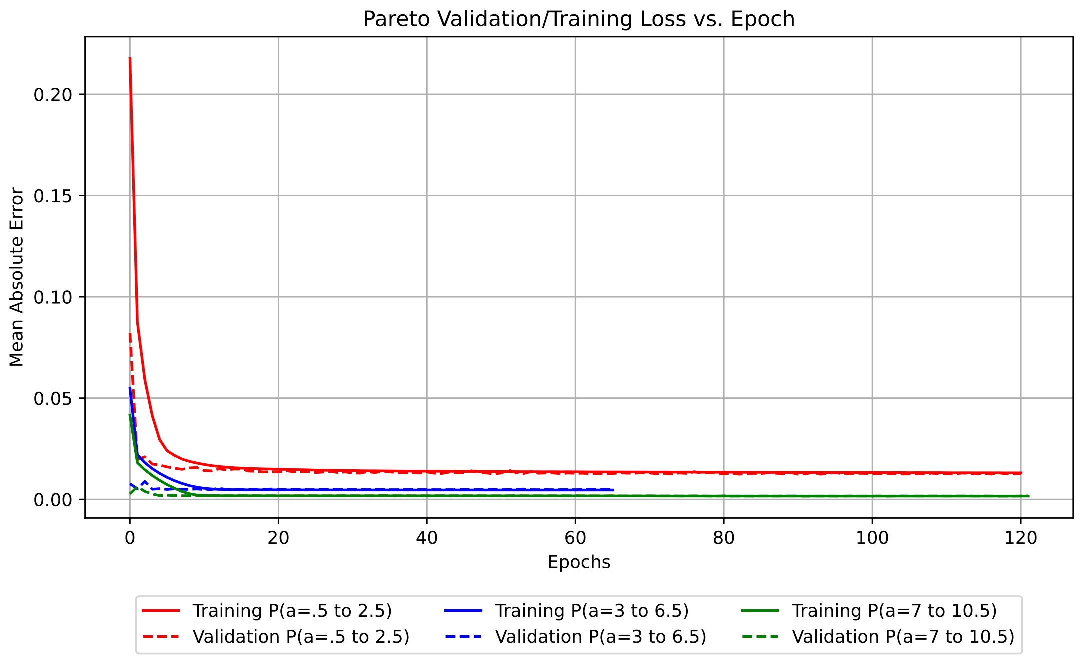


Figure 6.10: Pareto Training Loss

Based on the P_{fa} plots in Figures 6.11 and 6.12, the most important observation is that curves are almost perfectly flat at the desired 10^{-4}. The key exception to this observation is the extremely heavy-tailed portion of the low shape parameter K-distribution. However, even in this area, the P_{fa} drops closer to 10^{-5} rather than jumping higher, which is a preferable failure mode. Additionally, when examining the ideal threshold curve for the K-distribution (Figure 6.3), it is clear that the target threshold displays some asymptotic behavior in this region and approaches 1, which is the maximum bound on the GLRT threshold. The asymptotic behavior likely makes it difficult for the regression network to learn a consistent transfer function across the full region. Breaking the region into two and training an additional threshold selector would likely mitigate this issue. That asymptotic behavior coupled with the fact that the ideal threshold values approach 1 at the bottom of the low shape parameter K-distribution region also explains the severe drop in the P_d observed in this region.

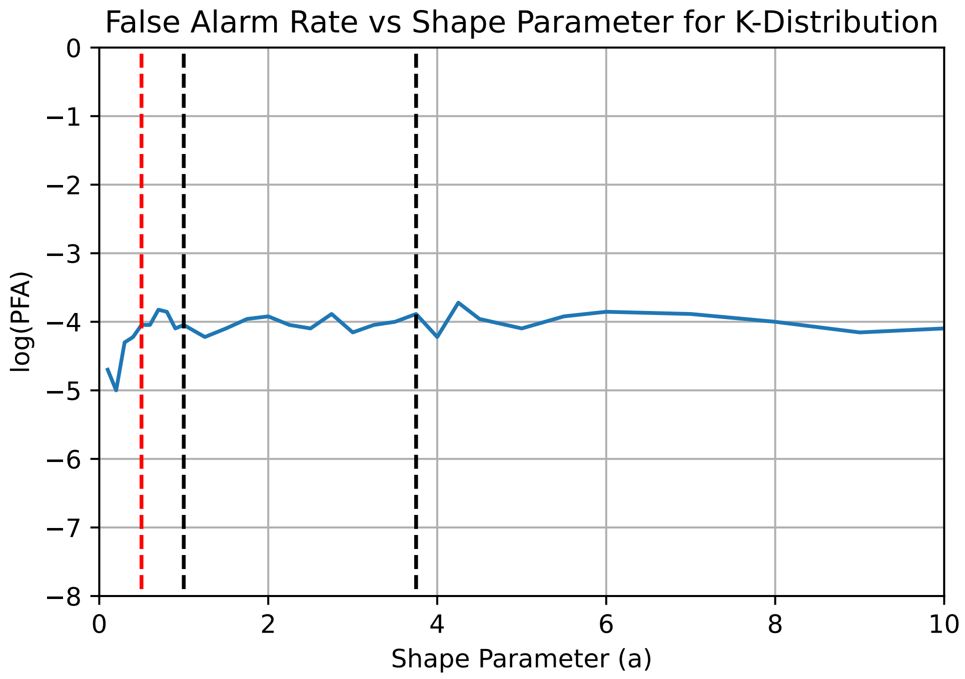


Figure 6.11: False Alarm Rate for K-Distribution Threshold Agents.

6.4 Testing and Analysis

Characterization of the full system was conducted in several stages. Each stage was designed to determine the system's performance characteristics, limitations, and benefits. First, its performance was verified over the full range of clutter distributions/shape parameters. Then it was characterized on a new clutter distribution on which no portion of the meta-cognitive detector was trained. Next, the system was reimplemented with 9 distribution regions (rather than the original 7) that include variations in the scale parameter b, and the expanded system was characterized. Finally, testing was conducted to evaluate how the meta-cognitive detector performs with different sample lengths, sample support values, and one-lag clutter correlation coefficients. To perform a comprehensive analysis of this approach's fidelity in each distribution region,

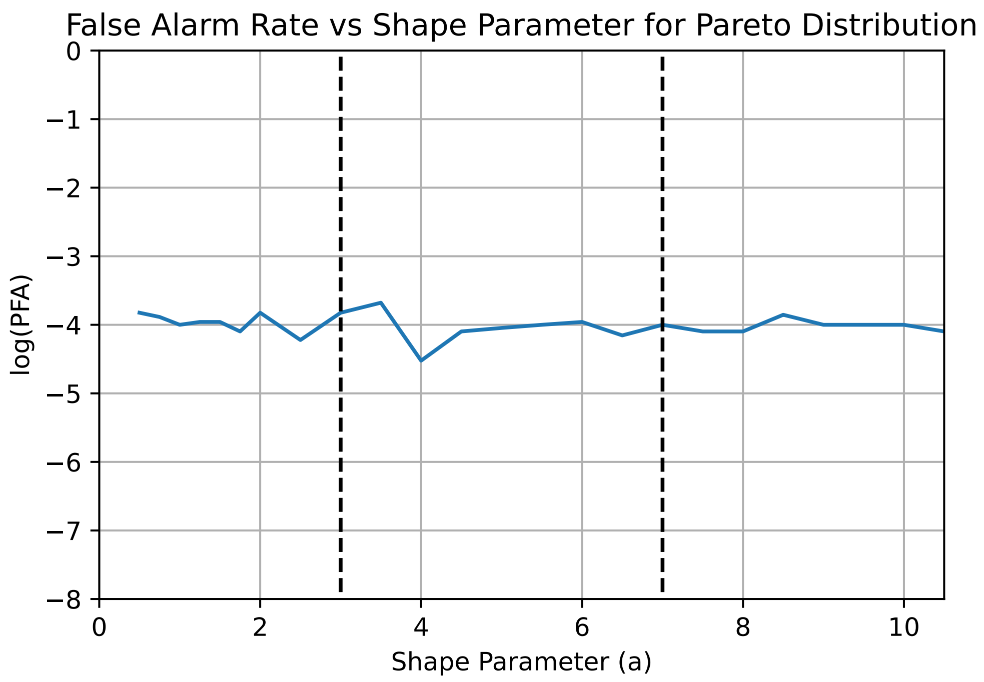


Figure 6.12: False Alarm Rate for Pareto Distribution Threshold Agents.

all clutter data used for training and characterization were generated directly with the desired distributions. Unless otherwise stated, all testing was conducted with clutter data generated using a sample length of $N = 16$, sample support of $K = 2N$, a one-lag clutter correlation coefficient of $\rho = 0.9$, and shape parameter $b = 1$. Any static P_{fa} testing was performed with $P_{fa} = 10^{-4}$, and P_{fa} sweeps were conducted over $P_{fa} = 10^{-4}$ to 10^{-1}. The results of the testing were captured in terms of the observed P_d, P_{fa}, and Receiver Operating Characteristic (ROC) curve. The ROC curve typically consists of a plot of the P_d (y-axis) versus the P_{fa} (x-axis), and it is commonly used in the literature on adaptive detection algorithms to compare the performance of different detectors. In this work, I use two minor variants of the ROC curve; 1) P_{fa} versus Target P_{fa} and 2) P_d versus Target P_{fa}. I use these variants because the proposed meta-cognitive approach to adaptive detection was designed to allow existing

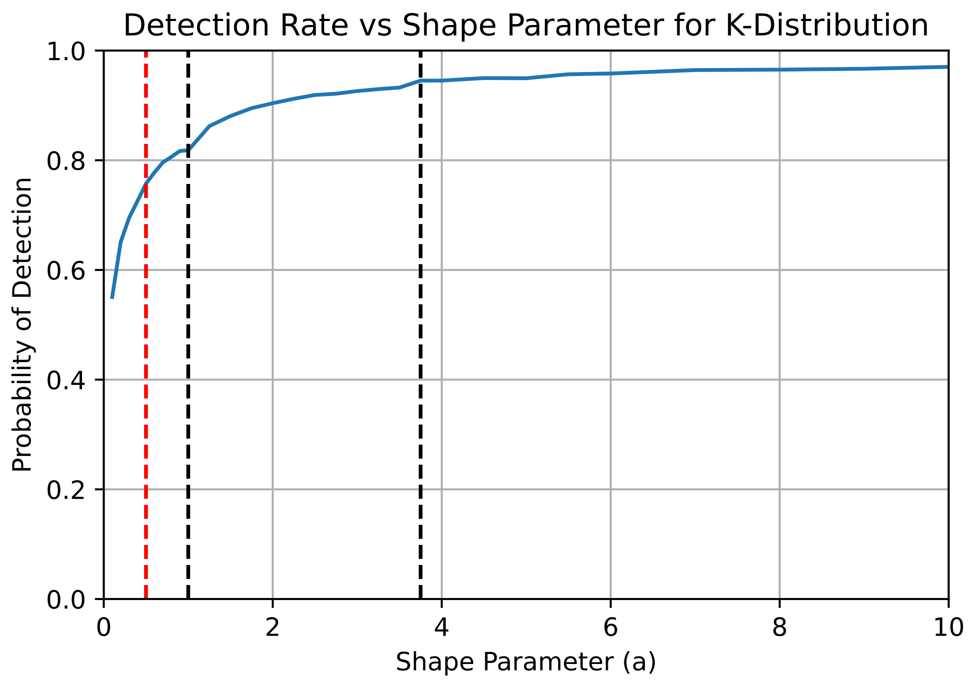


Figure 6.13: Detection Probability for K-Distribution Threshold Agents at SIR = 35dB.

detection algorithms to operate over an arbitrarily broad range of distributions, and one of its key characteristics is the ability to maintain CFAR-like behavior when other detectors fail to do so.

Four additional detection algorithms are used at various stages of the characterization testing for comparison with the proposed approach. These are the standard Gaussian GLRT (3.22), the Gaussian AMF (3.26), the DL-based detector described in both [152] and [153], and the Gaussian GLRT with threshold selector trained on low shape parameter $(0.1-1.0)$, K-distributed clutter (K_1 filter). The Gaussian GLRT and AMF serve as a comparison with the state-of-the-art in non-ML-based adaptive detection. The alternative DL detector is used as a comparison with modern DL detection approaches. This approach uses range processed (i.e., matched filtered) return signals

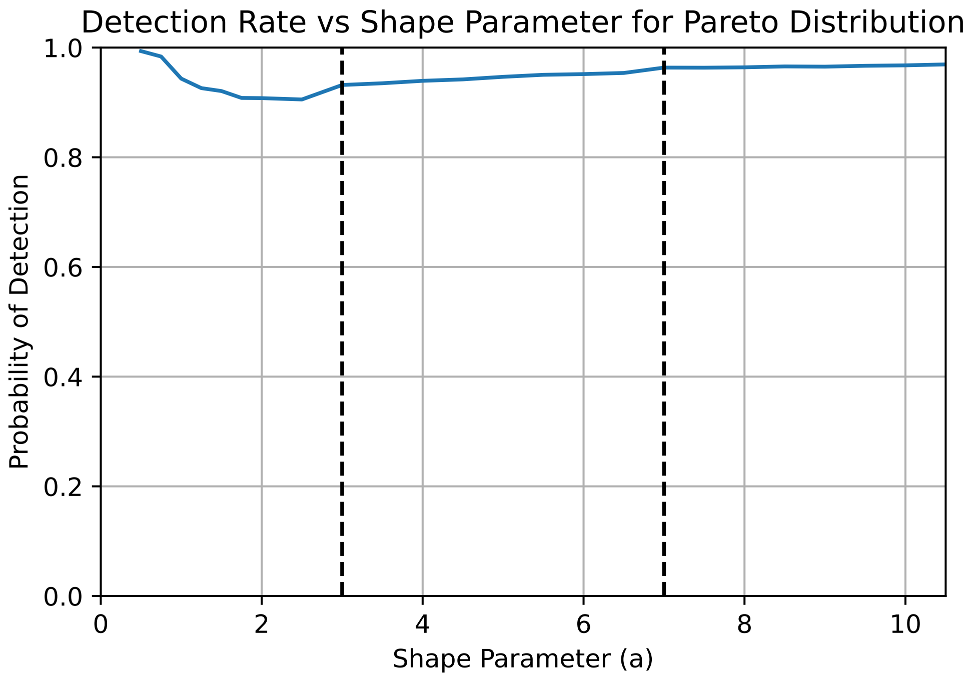


Figure 6.14: Detection Probability for Pareto Distribution Threshold Agents at SIR = 35dB.

as the input to a single neural network trained to perform detection directly. It was selected for comparison due to the similarity in the data model and level of processing, allowing for a more direct performance comparison. This also provides an interesting comparison against a purely ML-based detection system. The implementation and training of this alternative DL approach were performed exactly as detailed in [153]. The low shape parameter K-distribution is the most non-Gaussian distribution region considered here, and the threshold selector trained in this region consistently has the highest threshold values, making it a natural choice for comparison with the ensemble system. The Gaussian GLRT paired with the heavy-tailed K threshold setting ML model is referred to as the K_1 GLRT for the rest of this section.

6.4.1 System Characterization in Gaussian, K, and Pareto Clutter

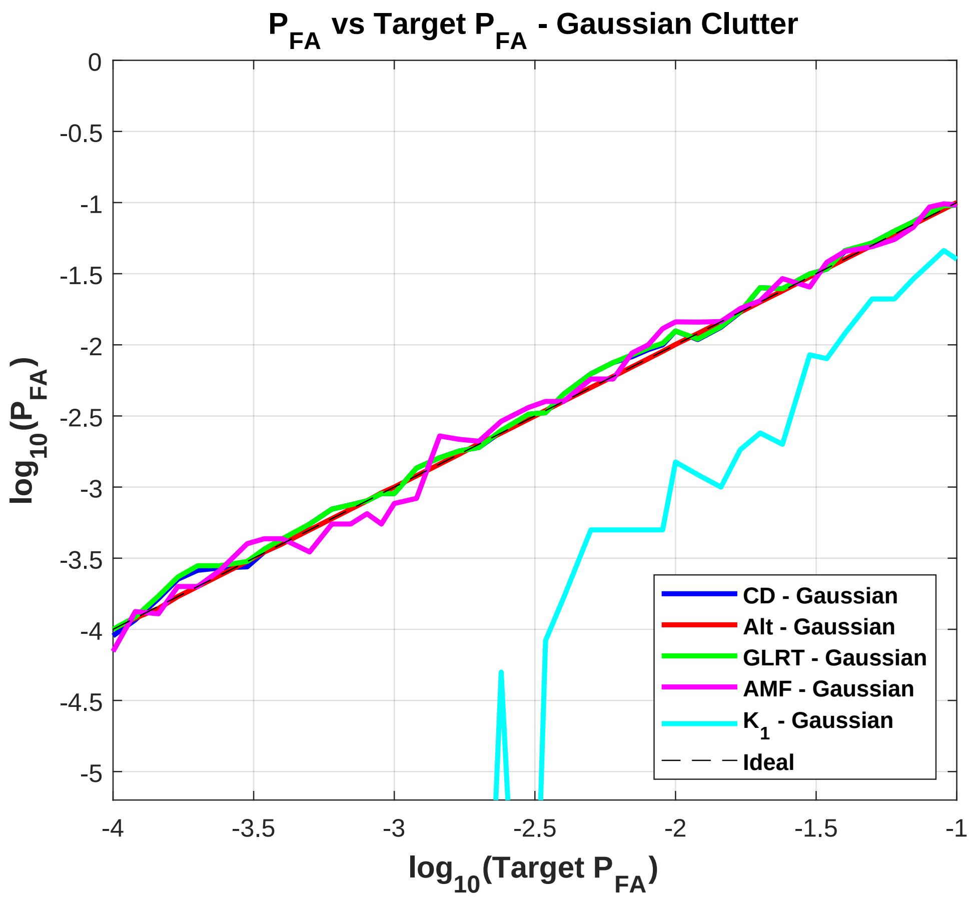


Figure 6.15: ROC Curve - P_{FA} vs Target P_{FA} - Gaussian Clutter.

The first set of large scale system testing was conducted to characterize the performance of the meta-cognitive detector over the full set of clutter distribution regions for which it was designed. As was alluded to in Section 6.3, the K-distribution clutter region where $a = 0.1$ to 0.5 was an especially difficult region and was analyzed separately. I will first present the results of the broader region testing, and then I will discuss the results of the testing targeted at the extremely low shape parameter K-distribution region. The full system was first tested using $500,000$ test samples (with associated sample support) spread across the full region of distributions and

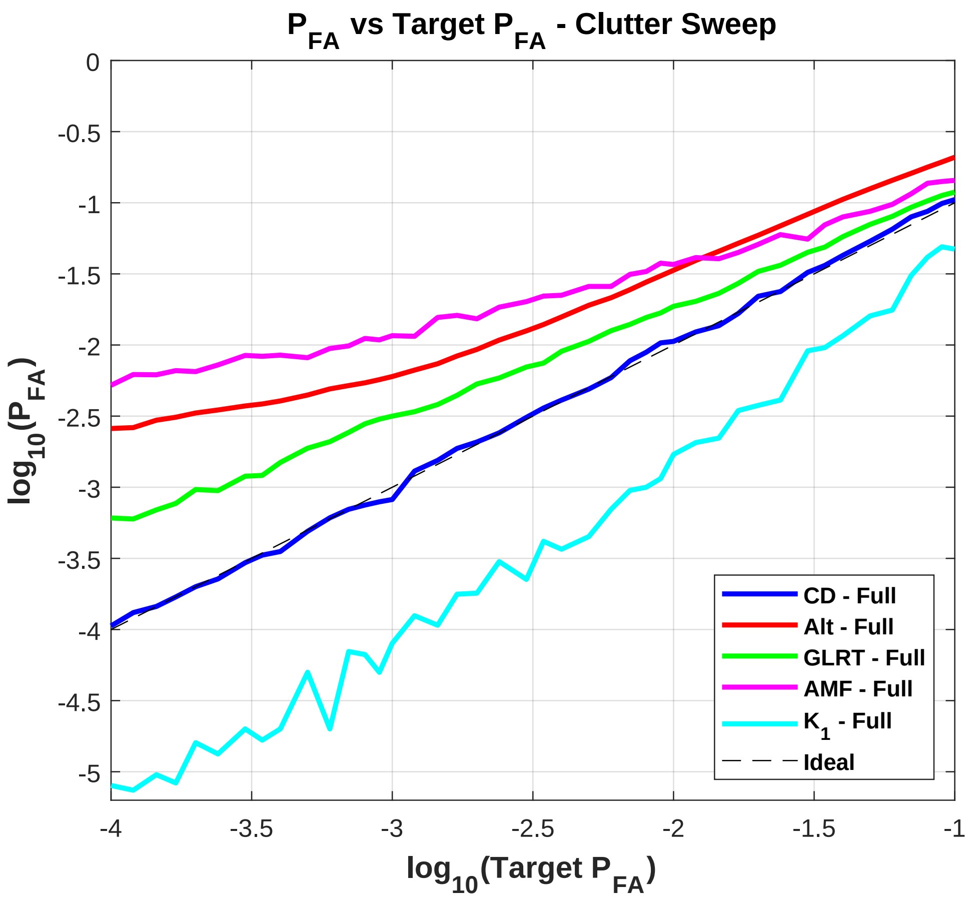


Figure 6.16: ROC Curve - P_{FA} vs Target P_{FA} - Clutter Sweep.

shape parameters of interest. $200,000$ of these samples were generated using the K-distribution clutter model, $200,000$ were generated using the Pareto model, and the remaining $100,000$ were generated with the Gaussian model. For each of the K- and Pareto distributed clutter data, 100 shape parameters were randomly selected (between 0.5 and 10 for K- and between 0.5 and 10.5 for Pareto), and each shape parameter was used to generate $2,000$ samples. It should be noted here that while the shape parameters generated for this testing fell within the regions for which the threshold selectors were trained, virtually none matched the training values exactly. This observation was important because it served to demonstrate how effectively the system generalizes to

Detector	**Simulation Number**	**False Alarms**	**Detections**	$\mathbf{P_{fa}}$	$\mathbf{P_d}$
			$Gaussian$		
Meta-Cog. Det.	$100,000$	9	$98,615$	$\mathbf{0.000090}$	$\mathbf{0.986150}$
Gaussian GLRT	$100,000$	10	$98,692$	0.000100	0.986920
AMF	$100,000$	7	$98,645$	0.000070	0.986450
K_1 GLRT	$100,000$	0	$72,236$	0.000000	0.722360
DL Detector	$100,000$	10	$99,759$	0.000100	0.997590
			$K(0.5-10)$		
Meta-Cog. Det.	$200,000$	21	$186,996$	$\mathbf{0.000105}$	$\mathbf{0.934980}$
Gaussian GLRT	$200,000$	162	$194,636$	0.000810	0.973180
AMF	$200,000$	841	$197,664$	0.004205	0.988220
K_1 GLRT	$200,000$	3	$127,548$	0.000015	0.637740
DL Detector	$200,000$	280	$198,975$	0.001200	0.994875
			$Pareto(0.5-10.5)$		
Meta-Cog. Det.	$200,000$	23	$187,825$	$\mathbf{0.000115}$	$\mathbf{0.939125}$
Gaussian GLRT	$200,000$	132	$194,355$	0.000660	0.971775
AMF	$200,000$	1756	$198,013$	0.008780	0.990065
K_1 GLRT	$200,000$	1	$112,982$	0.000005	0.564910
DL Detector	$200,000$	$1,004$	$198,027$	0.005020	0.990135
			Full Test - $Gaussian + K(0.5-10) + Pareto(0.5-10.5)$		
Meta-Cog. Det.	$500,000$	53	$473,436$	$\mathbf{0.000106}$	$\mathbf{0.946872}$
Gaussian GLRT	$500,000$	304	$487,683$	0.000608	0.975366
AMF	$500,000$	2604	$494,292$	0.005208	0.988584
K_1 GLRT	$500,000$	4	$312,766$	0.000008	0.625532
DL Detector	$500,000$	$1,294$	$496,761$	0.002588	0.993522
			$Lognormal$		
Meta-Cog. Det.	$100,000$	52	$91,693$	0.000520	0.916930
Gaussian GLRT	$100,000$	378	$95,890$	0.003780	0.958900
AMF	$100,000$	4398	$99,640$	0.043980	0.996400
K_1 GLRT	$100,000$	14	$87,914$	0.000140	0.879140
DL Detector	$100,000$	$1,390$	$98,765$	0.013900	0.987650

Table 6.3: Verification Test Results - Full Range of Clutter Distributions and Additional Lognormal Case

the distribution regions.

These $500,000$ samples were first generated as clutter only (H_0) samples and were used to evaluate the false alarm rate of the detection algorithm. Those same samples were then used to generate (H_1) samples by adding a target as in Equation (3.3) to determine the P_d for the detector over different distributions and shape parameters. Both the P_{fa} and P_d were tested at a target $P_{fa} = 10^{-4}$. A static steering vector was used for all H_1 test samples with $\mathbf{p}_i = \exp^{-j2\pi(i-1)0.2}$. H_1 testing was conducted at an SIR value of 35dB. α was set to ensure the desired SIR for the SUT using [94]:

$$SIR = \alpha^2 \times \mathbf{p}^H \mathbf{M}^{-1} \mathbf{p} \tag{6.5}$$

This baseline performance comparison was conducted using the Gaussian GLRT, AMF, Carretero *et al.*'s DL detector, and the K_1 GLRT. For the remainder of this chapter, I will refer to Carretero *et al.*'s DL detector as simply the DL detector.

Table 6.3 shows the results for the meta-cognitive detector, Gaussian GLRT, AMF, K_1 GLRT, and the DL detector from the full region of clutter distributions. This table shows the P_{fa} and P_d results for each distribution individually as well as the aggregate across all three distributions. Several impressive outcomes can be observed. First, the proposed meta-cognitive detector matched the performance of the Gaussian GLRT for Gaussian interference. Second, it successfully maintained the target P_{fa} of 10^{-4} across the full sweep of K- and Pareto distributed data while also achieving a very high probability of detection. Across all of the distributions the K_1 GLRT, which was the most restrictive filter, maintained a very low P_{fa}, but it did not display CFAR-like behavior and had substantially degraded P_d. Both the AMF and DL detector utterly failed to maintain CFAR-like behavior in both the K- and Pareto distributed clutter with P_{fa}s at least an order of magnitude higher than the target. The full data set results show the optimal performance of the meta-cognitive detector as it maintains a $P_{fa} \approx P_{fa,Target}$ and had an over all $P_d \approx 94.5\%$. By comparison the Gaussian

GLRT, AMF, and DL detector each had much worse P_{fa} ($P_{fa,GLRT} \approx 6 \times 10^{-4}$, $P_{fa,AMF} \approx 5 \times 10^{-}3$, $P_{fa,DL} \approx 2.5 \times 10^{-3}$). On the other extreme, the restrictive K_1 GLRT was below the P_{fa} target, but it had much worse $P_d \approx 72.2\%$

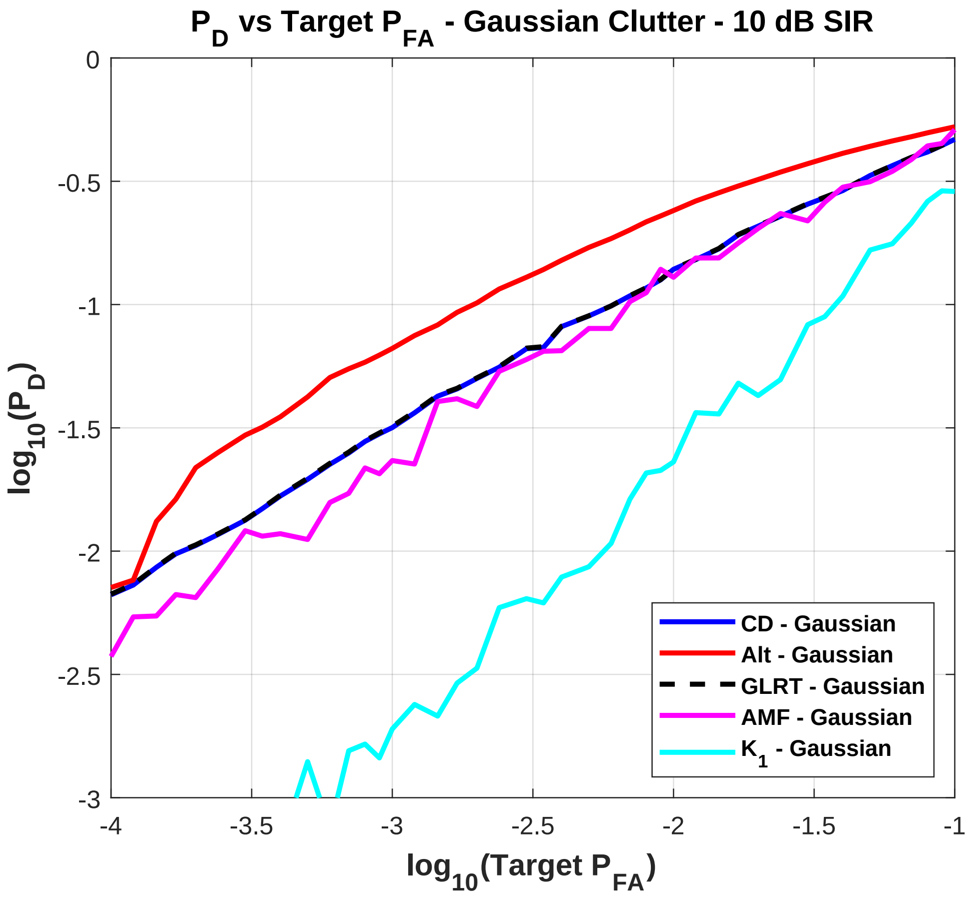


Figure 6.17: ROC Curve - P_d vs Target P_{fa} - Gaussian Clutter.

Additional testing was conducted to generate ROC curves for each detector. The Gaussian-only and Gaussian/K/Pareto sweep data sets were each run through the detectors over a range of target P_{fa}s from 10^{-4} to 10^{-1}. The P_d and P_{fa} for each detector were recorded and plotted against the target P_{fa} in Figures 6.15-6.20. H_1 testing was conducted at two distinct SIR values: 35 dB and 10 dB. For both the meta-cognitive detector and DL detector, this required retraining of the ML elements of each system

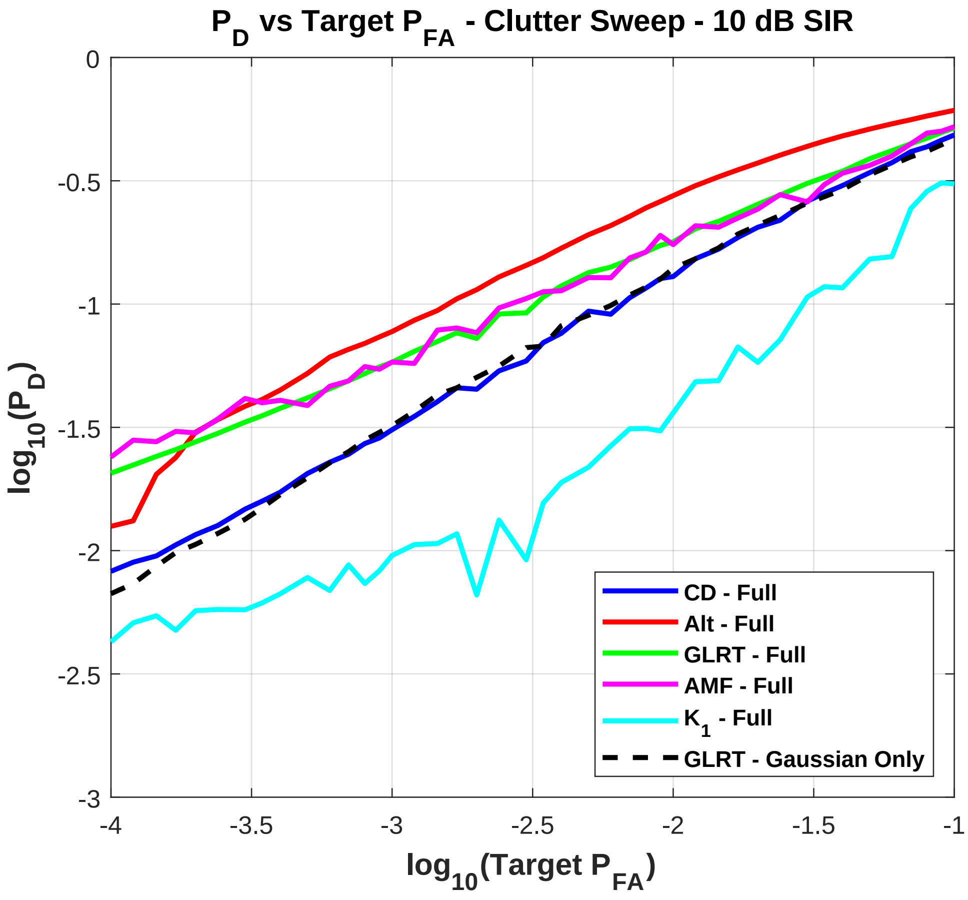


Figure 6.18: ROC Curve - P_d vs Target P_{fa} - Clutter Sweep.

for each target P_{fa}. In each ROC plot, the meta-cognitive detector is labeled as CD and the DL detector as Alt.

The ROC curves generated by this testing further demonstrate the superiority of the meta-cognitive approach to adaptive detection. Figure 6.15 depicts the P_{fa} versus target P_{fa} curve in purely Gaussian clutter. Clearly, all detectors except for the K_1 GLRT maintained the target P_{fa} throughout. Conversely, the K_1 GLRT consistently had drastically lower P_{fa} than the target. Figure 6.16 shows the P_{fa} versus target P_{fa} curve for the full distribution sweep. Only the meta-cognitive detector continues to maintain the target P_{fa} with no observable change in behavior. Every other detector displayed a drastic increase in the number of false alarms.

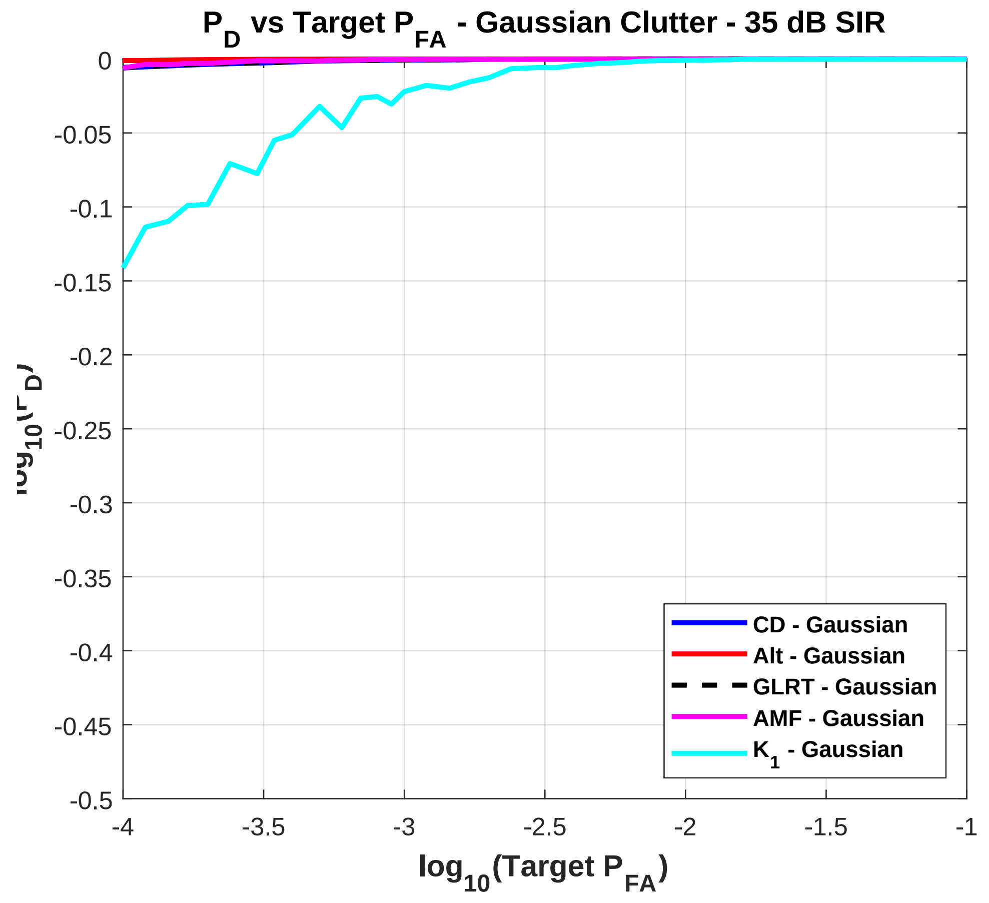


Figure 6.19: ROC Curve - P_D vs Target P_{FA} - Gaussian Clutter.

Figures 6.17 and 6.18 show the P_d vs target P_{fa} curves when the SIR is 10 dB. The Gaussian-only case is shown in Figure 6.17, and the case where all distributions are included is contained in Figure 6.18. When only Gaussian clutter is present, the meta-cognitive detector and AMF both have approximately identical detection performance when compared to the GLRT. The K_1 GLRT has relatively poor performance, while the DL detector actually has better performance than the GLRT. Examining the full sweep curve, an important observation can be made: the detection performance of the meta-cognitive detector remains almost perfectly unchanged. Each of the other

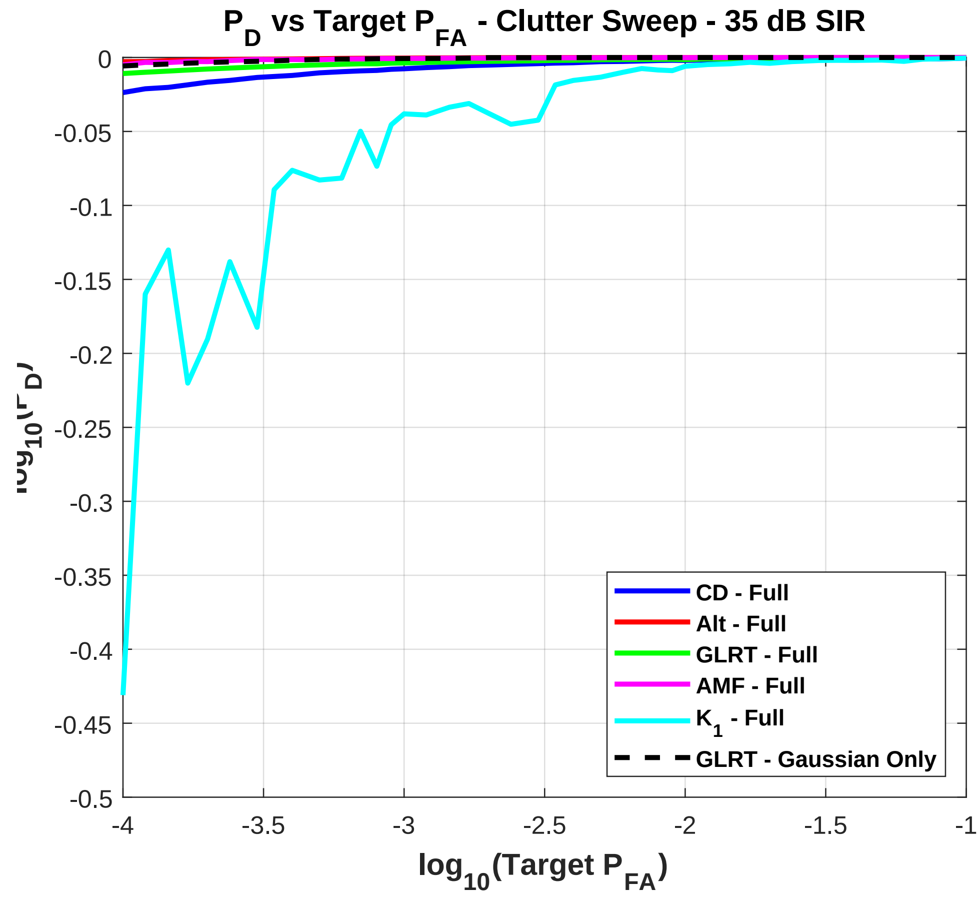


Figure 6.20: ROC Curve - P_d vs Target P_{fa} - Clutter Sweep.

detectors evaluated displays a jump in P_d corresponding to the previously observed jumps in P_{fa}. When the SIR is increased to 35 dB, as shown in Figures 6.19 and 6.20, all approaches other than the K_1 GLRT have excellent detection performance (as would be expected).

As the the K_1 GLRT expects extremely heavy tailed clutter, it was designed to set high threshold values, resulting in the lower P_{fa} than the target. However, this loss of sensitivity resulted in a corresponding low P_d. Conversely, the GLRT, AMF, and DL detectors do not adapt their thresholds to the K- and Pareto distributions, and therefore they use high thresholds in heavy-tailed clutter that result in both higher P_{fa} and

P_d than expected. It is extremely important that the proposed detector demonstrates no variation in either the P_d or P_{fa} between the Gaussian-only and full distribution sweeps for both the low SIR and high SIR cases. That level of consistency demonstrates that this novel form of meta-cognition enables the proposed system to maintain the desired performance characteristics across all of the various clutter types on which it was trained. There is also no performance penalty observed for using this meta-cognitive technique over an ideal detector in the presence of purely correlated Gaussian clutter.

Targeted system testing was performed in the most heavy-tailed clutter region considered here: the aforementioned, low shape parameter K-distribution. $500,000$ samples were used in this evaluation, $100,000$ generated from each of the following five shape parameters: 0.1, 0.2, 0.3, 0.4, and 0.5. The same clutter data was again used to generate both H_0 and H_1 test samples for determining the P_{fa} and P_d (respectively) of the algorithm at a target $P_{fa} = 10^{-4}$ and SIR of 35 dB. This more targeted performance comparison was conducted using only the Gaussian GLRT and the K_1 GLRT.

The results for the heavy-tailed clutter region of the K-distribution with extremely low shape parameter values ($a = 0.1$ to 0.5) are shown in Table 6.4. Clearly, as the shape parameters drop below 0.5 the meta-cognitive detector loses its CFAR-like behavior. However, it does still outperform the Gaussian GLRT. Interestingly, the targeted K_1 GLRT detector is able to maintain the desired P_{fa}. Given that the K_1 GLRT uses the same threshold selector designed for the low shape parameter K-distribution region as the main meta-cognitive system, this clearly indicates that the discriminator model fails to correctly classify the clutter in this sub-region. Looking at the ideal threshold curve in Figrue 6.3, I will again point out the asymptotic behavior of the curve as the shape parameter drops below 1.0. This behavior likely indicates that

Detector	**Simulation Number**	**False Alarms**	**Detections**	$\mathbf{P_{fa}}$	$\mathbf{P_d}$
$K(0.1 - 10)$					
Meta-Cog. Det.	$200,000$	220	$185,276$	$\mathbf{0.001100}$	$\mathbf{0.926380}$
Gaussian GLRT	$200,000$	782	$194,478$	0.003910	0.972390
K_1 GLRT	$200,000$	1	$126,488$	0.000005	0.632440
$K(0.5)$					
Meta-Cog. Det.	$100,000$	10	$75,520$	$\mathbf{0.000100}$	$\mathbf{0.755200}$
Gaussian GLRT	$100,000$	$1,277$	$95,054$	0.012770	0.950540
K_1 GLRT	$100,000$	10	$75,504$	0.000100	0.755040
$K(0.4)$					
Meta-Cog. Det.	$100,000$	12	$72,840$	$\mathbf{0.000120}$	$\mathbf{0.728400}$
Gaussian GLRT	$100,000$	$2,040$	$94,896$	0.020400	0.948960
K_1 GLRT	$100,000$	9	$72,807$	0.000090	0.728070
$K(0.3)$					
Meta-Cog. Det.	$100,000$	43	$69,407$	$\mathbf{0.000430}$	$\mathbf{0.694070}$
Gaussian GLRT	$100,000$	$3,502$	$94,837$	0.035020	0.948370
K_1 GLRT	$100,000$	10	$69,031$	0.000100	0.690310
$K(0.2)$					
Meta-Cog. Det.	$100,000$	981	$68,772$	$\mathbf{0.009810}$	$\mathbf{0.687720}$
Gaussian GLRT	$100,000$	$7,358$	$95,110$	0.073580	0.951100
K_1 GLRT	$100,000$	8	$64,859$	0.000080	0.648590
$K(0.1)$					
Meta-Cog. Det.	$100,000$	$13,995$	$83,488$	$\mathbf{0.139950}$	$\mathbf{0.834880}$
Gaussian GLRT	$100,000$	$19,290$	$96,204$	0.192900	0.962040
K_1 GLRT	$100,000$	4	$54,825$	0.000040	0.548250

Table 6.4: Verification Test Results - Extremely Heavy-Tailed K-Distributed Clutter

the statistical properties of the K-distribution change too rapidly in this region for the discriminator model to effectively generalize across it, resulting in instability in the trained model. It is very likely that the loss of CFAR-like performance in the mega-cognitive detector could be mitigated by training a new discriminator model with this region further subdivided.

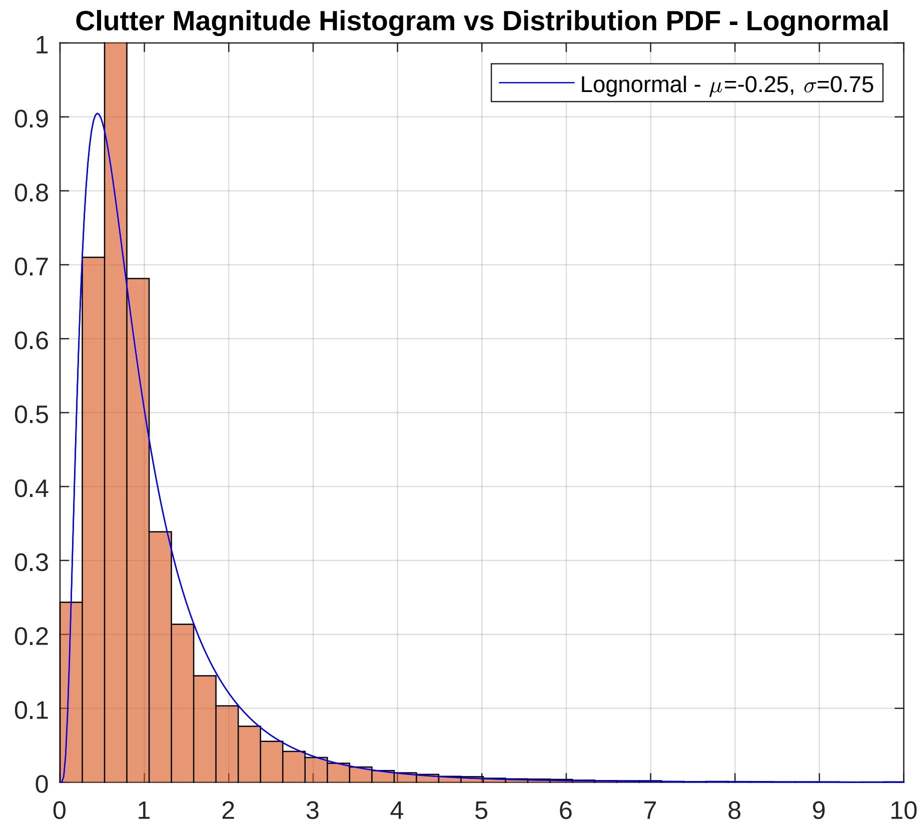


Figure 6.21: Distribution PDF vs Data Histogram - Lognormal Clutter.

6.4.2 System Characterization in Lognormal Clutter

The second set of large scale testing was conducted to determine how well the proposed meta-cognitive detector performs on unknown distributions. This consisted of repeating the full characterization at the target $P_{fa} = 10^{-4}$ on clutter with a lognormal distribution as well as generating equivalent ROC curves. The meta-cognitive detector was not trained on lognormal data prior to running these tests, making this a good test of how well the approach generalizes across new distributions. The clutter was generated as a set of correlated, complex lognormal random variables. Figure

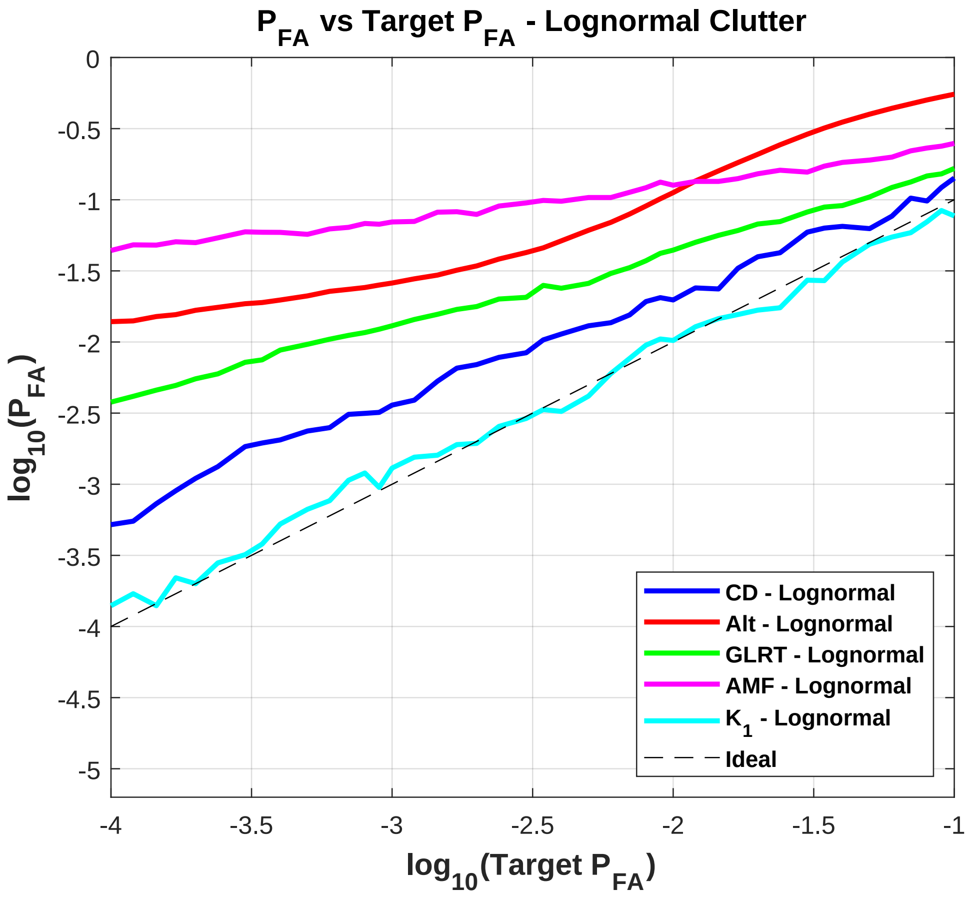


Figure 6.22: ROC Curve - P_{fa} vs Target P_{fa} for Lognormal clutter.

6.21 shows the histogram of the magnitude of the generated lognormal clutter coupled with the PDF of the distribution. This distribution does not fall within the category of SIRVs, but it does closely resemble a heavy-tailed distribution similar to the low shape parameter K-distribution.

A test set of $100,000$ lognormal samples and associated sample support was generated for these tests. Once again, both H_0 and H_1 SUTs were generated and run through the meta-cognitive, GLRT, AMF, K_1 GLRT, and DL detectors. The P_{fa} and P_d were measured with a target $P_{fa} = 10^{-4}$ and SIR of 35 dB. These results are recorded in Table 6.3. This process was then repeated over a range of P_{fa} values ranging from

10^{-4} to 10^{-1} to generate ROC curves at SIR values of 35 dB and 10 dB. The ROC curves for this testing are shown in Figures 6.22 - 6.24.

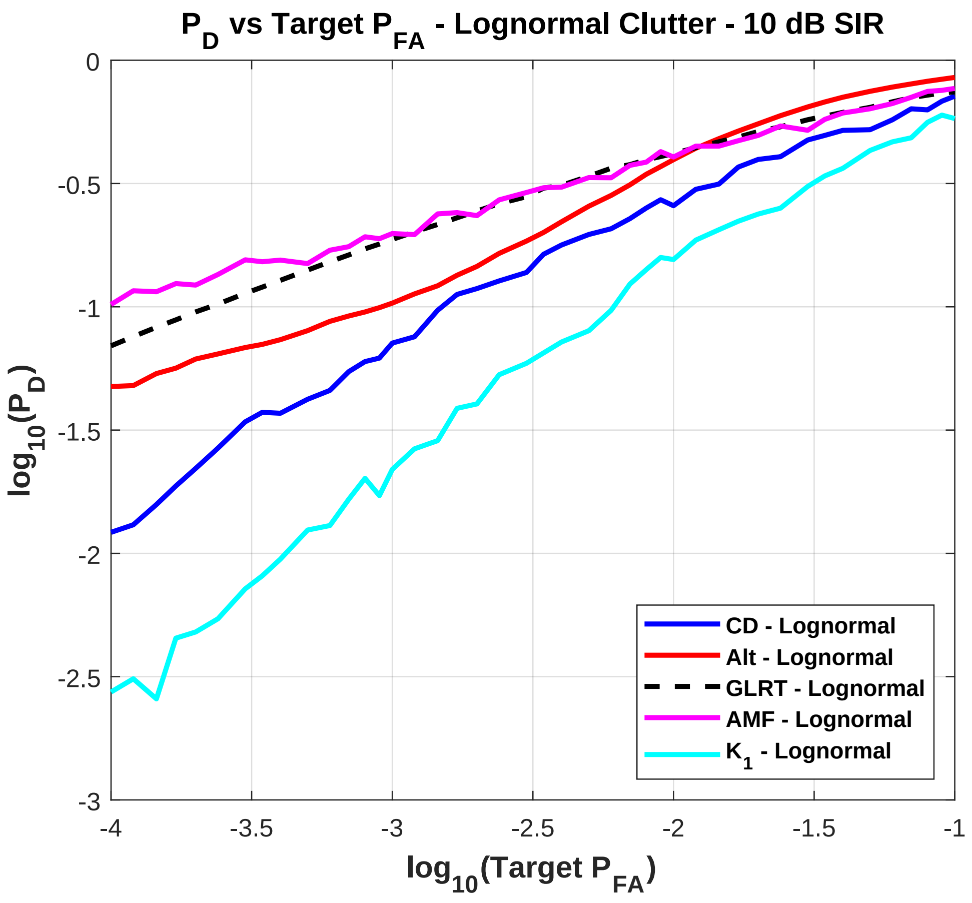


Figure 6.23: ROC Curve - P_d vs Target P_{fa} for Lognormal clutter with 10 dB SIR.

Clearly, the P_{fa} performance of the proposed detector degrades when presented with lognormal data. However, the full meta-cognitive detector substantially outperforms the GLRT, AMF, and DL detectors. Only the K_1 GLRT is able to maintain the target P_{fa} when presented with the lognormal clutter. Interestingly, the K_1 GLRT is simply the GLRT with the dynamic thresholding agent trained on low shape parameter K-distribution data. This makes sense given the previously noted similarity between the lognormal distribution and heavy-tailed K. Given that the K_1 threshold selector

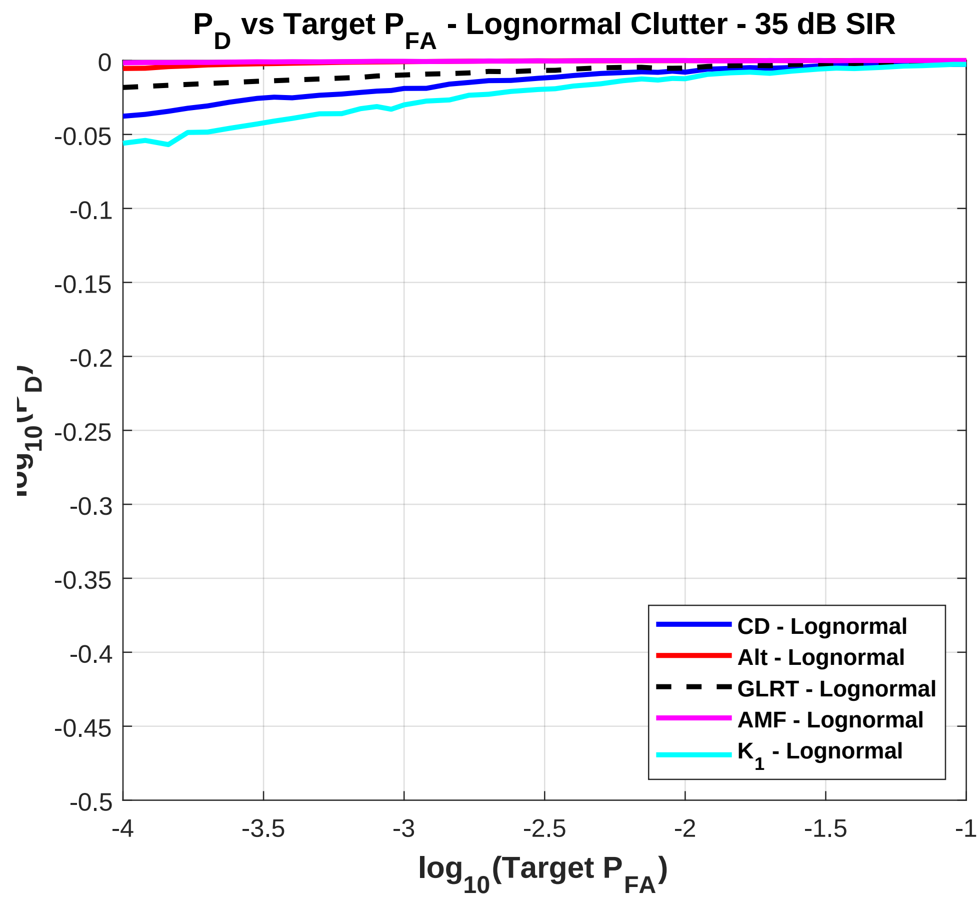


Figure 6.24: ROC Curve - P_d vs Target P_{fa} for Lognormal clutter with 35 dB SIR.

was part of the full meta-cognitive system, along with the demonstrated instability of the discriminator model in the heavy-tailed K-distribution region, this provides strong evidence of the generalizability of the proposed meta-cognitive approach. A more robust discriminator agent would almost certainly have allowed the meta-cognitive detector to maintain its CFAR-like behavior in the lognormal clutter. Additionally, it lends credence to two of my core claims: 1) the idea that distributions can be grouped into statistically unique regions for training ML systems, and 2) a fully adaptive ML-based detector can be designed using multiple, coordinated ML agents specifically trained to operate in statistically unique distribution regions. Note that in Figures 6.23

and 6.24 the performance is plotted against the *target* P_{FA}, not the *actual* P_{FA}. Therefore, while the AMF, GLRT, and alternative DL method will have better sensitivity for a *desired* probability of false alarm, their actual probability of false alarm will be the much higher value shown in Figure 6.22.

6.4.3 Distribution Scale Parameter Testing

After establishing a baseline performance of the proposed meta-cognitive detection algorithm, I conducted some additional testing to characterize its behavior further. Broadly speaking, the additional testing can be separated into two categories: 1) testing the detector response to changes in the clutter parameters and 2) testing the detector response to changes in the radar parameters. Clearly, much of the previous testing I have described for this detector evaluated its behavior as the clutter distribution changed, but there were two notable parameters that were held constant.

The first parameter was the correlation coefficient used to define the clutter covariance. I initially made the decision not to explore this parameter due to the fact that I was still using the GLRT, which approximates the CCM and uses it to whiten the clutter during detection. Due to this behavior, I felt it was fair to assume that the correlation coefficient would not impact overall behavior of the new detection algorithm. However, after proving out the concept I felt it would be worthwhile to prove that assumption experimentally.

The second parameter that I did not evaluate during initial testing was the scale parameter for the K- and Pareto distributions. This was largely due to my determination that testing over the scale parameter would be redundant. Like the shape parameter, the scale parameter alters the baseline behavior of the distributions parameterized by

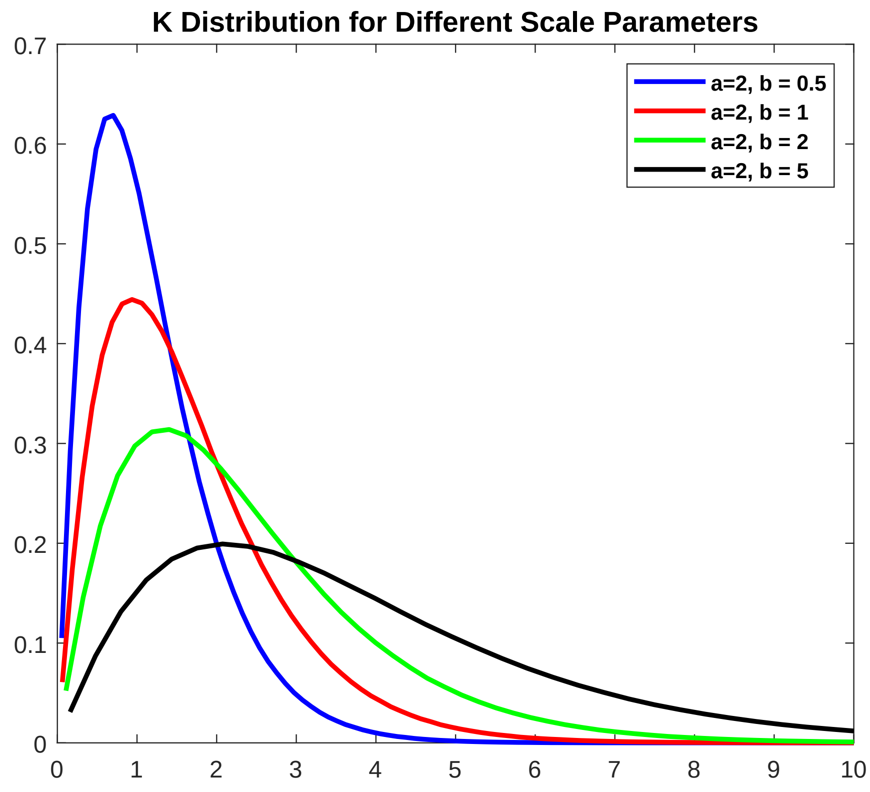


Figure 6.25: Example K-distribution plots for different scale parameter values.

it. As a result, it was reasonable to assume that the distributions could be further subdivided into statistically unique regions according to both the shape and scale parameters. In which case, the meta-cognitive detector can trivially be trained to compensate for those new regions. In fact, for SIRV distributions like the ones tested here, those parameters are often combined into a single parameter used to define the distributions.

The impact of the scale parameter can be observed in Figures 6.25 and 6.26. It is worth noting again that the K- and Pareto distributions used for testing in this work were generated as a compound Gaussian function according to the process outlined in Chapter 3.3, with the K-distribution consisting of a Gaussian modified by a Gamma distribution and the Pareto consisting of a Gaussian modified by an inverse Gamma

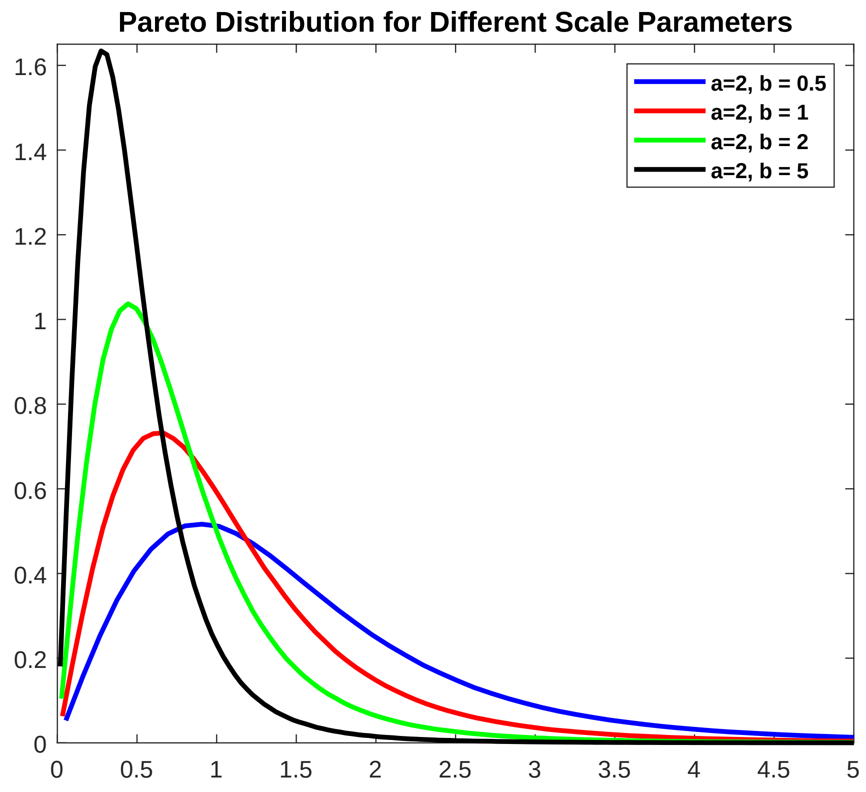


Figure 6.26: Example Pareto distribution plots for different scale parameter values.

distribution. In each case, I talk about the shape and scale parameters of the Gamma distribution used to modify the correlated Gaussian distribution. It can be observed in Figures 6.25 and 6.26 that varying the scale parameter discussed here with a constant shape parameter does impact how heavy-tailed the clutter distribution is, which should have a noticeable impact on the false alarm rate of a detector.

On the other hand, the scale parameter for a given shape parameter has a significantly lower impact on the behavior of the distribution than varying the shape parameter for a given scale parameter value. As a result, I expected that the meta-cognitive detector would be effective for those scale parameter values that produced distribution regions similar to those the system was trained for (i.e. when $b = 1$). However, it was also expected that most scale parameter values would produce distribution regions that were statistically distinct from those for which the system was designed. In those regions, the detector would naturally produce degraded performance as the distribution characteristics stopped resembling those for which the threshold setting and discriminator models were trained.

To test the response of the meta-cognitive detector over different values of the correlation coefficient and scale parameter, two sets of large scale testing were performed. In each case, tests were run to determine the false alarm rate of the meta-cognitive detector with a target $P_{fa} = 10^{-4}$ over a range of parameter values. For the correlation coefficient, values of $\rho = 0.9, 0.5, 0.1$ were used. This testing was performed on the original detector used in all previous testing without modification or retraining. Values of $b = 0.5, 1, 2, 5$ were used for the scale parameter with a fixed shape parameter of $a = 2$. $500,000$ tests were run for each parameter value being examined, with $100,000$ from the Gaussian distribution, $200,000$ from the K-distribution and $200,000$ from the Pareto distribution. For this testing, the sample length was held fixed at $N = 16$ with sample support of $K = 2N$. The results of this testing are presented in Table 6.5.

Several observations can be made from the results in Table 6.5. The first is that the proposed meta-cognitive detector did indeed maintain the desired P_{fa} regardless of the value of the correlation coefficient. This was to be expected, as the proposed detector uses the classic GLRT as its core detection algorithm. As I have discussed previously, adaptive detectors like the GLRT use the estimated clutter covariance matrix to whiten the sample under test, which make them largely agnostic to different covariance structures. At a minimum, it can be argued that my proposed approach to meta-cognitive detection generalizes as well as the core adaptive detection algorithm to different covariance structures. That fact coupled with the meta-cognitive detector's

Distribution	**Parameter**	**Value**	**Simulation Number**	**False Alarms**	$\mathbf{P_{fa}}$
			$CorrelationTesting$		
Gaussian	ρ	0.9	$100,000$	9	$\mathbf{0.000090}$
K	ρ	0.9	$200,000$	21	$\mathbf{0.000105}$
Pareto	ρ	0.9	$200,000$	23	$\mathbf{0.000115}$
Total	ρ	0.9	$500,000$	53	$\mathbf{0.000106}$
Gaussian	ρ	0.5	$100,000$	7	$\mathbf{0.000070}$
K	ρ	0.5	$200,000$	21	$\mathbf{0.000105}$
Pareto	ρ	0.5	$200,000$	27	$\mathbf{0.000135}$
Total	ρ	0.5	$500,000$	55	$\mathbf{0.000110}$
Gaussian	ρ	0.1	$100,000$	11	$\mathbf{0.000110}$
K	ρ	0.1	$200,000$	26	$\mathbf{0.000130}$
Pareto	ρ	0.1	$200,000$	28	$\mathbf{0.000140}$
Total	ρ	0.1	$500,000$	65	$\mathbf{0.000130}$
			$ScaleParameterTesting$		
Gaussian	b	0.5	$100,000$	11	$\mathbf{0.000110}$
K	b	0.5	$200,000$	18	$\mathbf{0.000090}$
Pareto	b	0.5	$200,000$	44	$\mathbf{0.000220}$
Total	b	0.5	$500,000$	73	$\mathbf{0.000146}$
Gaussian	b	1	$100,000$	9	$\mathbf{0.000090}$
K	b	1	$200,000$	21	$\mathbf{0.000105}$
Pareto	b	1	$200,000$	23	$\mathbf{0.000115}$
Total	b	1	$500,000$	53	$\mathbf{0.000106}$
Gaussian	b	2	$100,000$	10	$\mathbf{0.000100}$
K	b	2	$200,000$	26	$\mathbf{0.000130}$
Pareto	b	2	$200,000$	36	$\mathbf{0.000180}$
Total	b	2	$500,000$	72	$\mathbf{0.000144}$
Gaussian	b	5	$100,000$	12	$\mathbf{0.000120}$
K	b	5	$200,000$	55	$\mathbf{0.000275}$
Pareto	b	5	$200,000$	49	$\mathbf{0.000245}$
Total	b	5	$500,000$	116	$\mathbf{0.000232}$

Table 6.5: Verification Test Results - Varying Correlation Coefficient and Scale Parameter

observed behavior provides further evidence of my assertion that by intelligently integrating ML techniques with classical signal processing it is possible to net the benefits of both approaches. However, to fully evaluate how the proposed approach generalizes, it will be critical to test it with more realistic clutter models, particularly those with complex-valued covariance matrices. Additionally, this method should be tested on real-world data.

The other observations I will make are related to the shape parameter. For the K-distributed clutter tests, the meta-cognitive detector maintains its target false alarm rate for low shape parameter values and only shows a significant increase in the observed P_{fa} as b becomes very large. On the other hand, the detector behavior is quite different as the shape parameter of the Pareto distributed clutter changes. For those tests, degraded performance could be observed for both very small and very large values of b. To explain this disparity, I will refer back to the threshold curves found in Figures 6.3 and 6.4. Recall that for this testing the shape parameter value of both the K- and Pareto distributions was held constant at $a = 2$. However, that parameter value corresponds to very different behavioral regions for the K- and Pareto distributions.

For the K-distribution, the threshold behavior of the GLRT is stable and linear for the distribution region near $a = 2$ when $b = 1$. That region corresponds to the "medium shape parameter" threshold selector. Referring to Figure 6.25, as the value of b decreases, the distribution becomes more heavy-tailed and more closely resembles the low shape parameter K-distribution region the detector was initially trained on. That region's threshold setting agent sets higher thresholds, helping to maintain the target P_{fa}. Then, as the value of the scale parameter becomes large, the distribution flattens out but does not begin to closely resemble a Gaussian distribution. In effect, it stops resembling any distribution regions for which the meta-cognitive detector was trained. That causes the classifier to improperly select the threshold setting agents associated with the high shape parameter K and Gaussian distribution regions. Those set lower thresholds, which drives up false alarms due to the skewed nature of this new distribution region.

The Pareto distribution, on the other hand, is still in a more volatile, heavy-tailed region when $a = 2$ and $b = 1$. This corresponds to the distribution region that the meta-cognitive detector was trained to recognize as the low shape parameter Pareto region. Referring to Figure 6.26, for small values of b the Pareto distribution flattens out while continuing to be skewed. As noted previously for the K-distribution, this behavior is unlike anything the meta-cognitive detector was trained to handle. This again caused the discriminator model to select thresholding agents that set lower thresholds, driving up the false alarm rate due to the skewed nature of the distribution. Low values of the scale parameter caused the distribution to become extremely heavy-tailed. The classifier selected the correct thresholding agent for this region, but that model was not trained to set thresholds high enough to maintain the target P_{fa} with this new behavior.

Classes	K-Dist	K-Dist	K-Dist	K-Dist	K-Dist
a Regions	0.1 to 1	1.25 to 3.5	1.25 to 3.5	1.25 to 3.5	3.75 to 10
b Value	1	0.5	1	2	1
Training label	1	2	3	4	5
Classes	Gaussian	Pareto	Pareto	Pareto	
a Regions	-	.5 to 2.5	3 to 6.5	7 to 10.5	
b Value	-	1	1	1	
Training label	0	6	7	8	

Table 6.6: Clutter Ranges and Labels

These observations align with my assertion that the scale parameter simply alters the statistical properties of the distribution in much the same way that the shape parameter does. That in turn provides strong evidence that the proposed meta-cognitive approach to adaptive detection can be trivially extended to this additional parameter. To prove this further, I designed a nine region implementation of the detector (as opposed to the original seven region implementation) and trained and tested it. This new version of the detector had regions as defined in Table 6.6. Note that only two additional regions were defined and trained to prove out the concept. Both of these regions were in the K-distribution and defined for the medium shape parameter value region. This new implementation of discriminator and threshold setting models was trained according to the process outlined in Chapter 6.3. After training the new system, a new batch of P_{fa} testing was run to evaluate the performance impact. The results of that testing are presented in Table 6.7. Note that since the number of models was increased and the discriminator was now being required to classify across a larger number of regions, it was important to test the performance of the updated system across all regions, not just the newly added ones.

Distribution	**Parameter**	**Value**	**Simulation Number**	**False Alarms**	$\mathbf{P_{fa}}$
Gaussian	b	1	$100,000$	10	$\mathbf{0.000100}$
K	b	$0.5, 1, 2$	$200,000$	21	$\mathbf{0.000105}$
Pareto	b	1	$200,000$	17	$\mathbf{0.000085}$

Table 6.7: Verification Test Results - Varying Correlation Coefficient and Scale Parameter

In Table 6.7, it can clearly be seen that the newly trained meta-cognitive detector does indeed match the target false alarm rate with the inclusion of the new scale parameter regions. These results demonstrate several important performance characteristics of the proposed method. First, it proves the earlier assertion that the system extends well to cover scale parameter regions. In addition, it demonstrates that larger systems can be trained without observable degradation to the performance of the system. This is important, as I designed this meta-cognitive detection algorithm to be readily customizable and extensible. This testing demonstrated that such modification was indeed possible. However, a much more detailed analysis needs to be performed to determine the limits of the discriminator model and the computational costs associated with increasing the number of threshold setting models. Classification models today can be designed to distinguish between vast numbers of categories, so in theory, the discriminator model should be able to extend far beyond its current state with some minor redesign and optimization. Since the threshold setting models are kept in parallel and only one model is ever selected by the discriminator model, there shouldn't be a significant increase in the algorithm's runtime or computational requirements. On the other hand, each additional model increases the storage requirements of the detector. Additionally, it took considerable expert input to redesign and retrain this system. However, much of the analysis used to identify statistically unique regions, model design, data generation, and model training could be readily automated. Such automation would make the generation of custom meta-cognitive detectors much faster and allow for easy production-level usage of the proposed method in real radar systems. My colleague Geoffrey Dolinger has been working on expanding this work to implement such an automatic configuration tool.

6.4.4 Variable Data Size and Sample Support Testing

Finally, I characterized the impact of changing the sample size and the level of sample support provided to the detector. Real radar systems are often configurable, and both of these parameters are likely to change significantly depending on the particular application and domain. This testing was important to demonstrate how readily the proposed method could be applied to real-world systems. I did not initially test these parameters because I determined them to be of secondary importance. The core challenge I wanted to explore was the creation of a ML-aided detector that could operate and maintain CFAR across distributions. Additionally, I anticipated that only minor modifications would be required to make the proposed meta-cognitive detector. In theory, the fact that the system was designed to take in down-sampled ordered statistics for the ML components means that the system as a whole should continue to function regardless of sample length, with the GLRT itself serving as the primary limiter as the sample length increases. For the same reason, the level of sample support should have a minimal impact on the ML models' performance, but it should heavily impact the GLRT. While the ML components of the meta-cognitive detector should continue to function regardless of sample length and sample support, it should be noted that the threshold used to maintain a target P_{fa} with the GLRT is affected by both parameters. This means that I did not expect the meta-cognitive detector to be able to maintain optimal behavior as these parameters changed without modification.

To test my hypothesis and characterize the detector's response, two sets of tests were conducted. Once again, the detector was run on clutter only samples and the false alarm rate was measured for different values of each parameter with a target $P_{fa} = 10^{-4}$. For the sample length, values of $N = 16, 32$ were evaluated. Sample support values of $K = 2N, 4N$ were tested. In each case, the shape parameter was swept across the full set of regions defined in Table 6.1 and the scale parameter was held constant at $b = 1$. $500,000$ tests were run for each parameter value being examined, with $100,000$ from the Gaussian distribution, $200,000$ from the K-distribution and $200,000$ from the Pareto distribution. The results of this testing are presented in Table 6.8.

I observed that increasing either the sample length or the level of sample support causes the detectors false alarm rate to decrease significantly. While it can be tempting to view this as a positive, recall that maintaining a constant, known P_{fa} is desirable because the P_d and P_{fa} have a trade-off relationship. So, when the P_{fa} falls below

Distribution	Parameter	Value	Simulation Number	False Alarms	$\mathbf{P_{fa}}$
$SampleSupportTesting$					
Gaussian	K	32	$100,000$	9	$\mathbf{0.000090}$
K	K	32	$200,000$	21	$\mathbf{0.000105}$
Pareto	K	32	$200,000$	23	$\mathbf{0.000115}$
Total	K	32	$500,000$	53	$\mathbf{0.000106}$
Gaussian	K	64	$100,000$	10	$\mathbf{0.000100}$
K	K	64	$200,000$	0	$\mathbf{0.000000}$
Pareto	K	64	$200,000$	23	$\mathbf{0.000115}$
Total	K	64	$500,000$	33	$\mathbf{0.000066}$
$SampleLengthTesting$					
Gaussian	N	16	$100,000$	9	$\mathbf{0.000090}$
K	N	16	$200,000$	21	$\mathbf{0.000105}$
Pareto	N	16	$200,000$	23	$\mathbf{0.000115}$
Total	N	16	$500,000$	53	$\mathbf{0.000106}$
Gaussian	N	24	$100,000$	10	$\mathbf{0.000100}$
K	N	24	$200,000$	0	$\mathbf{0.000000}$
Pareto	N	24	$200,000$	13	$\mathbf{0.000065}$
Total	N	24	$500,000$	23	$\mathbf{0.000046}$

Table 6.8: Verification Test Results - Varying Sample Length and Sample Support

the target value, the detection performance of the detector is negatively impacted. The observed behavior makes sense when we consider how adaptive detection algorithms work. They are designed to approximate the covariance matrix of the interference present in a region and then derive a whitening filter to decorrelate the interference in the sample under test. Increasing the level of sample support used improves the approximation of the covariance matrix. Increasing the sample length while maintaining the same relative amount of sample support (i.e. $K = 2N$) increases the granularity of the approximation, producing a more realistic approximation of the matrix. This behavior is consistent with the Gaussian threshold calculation derived by Kelley in [13]. I will note, however, that increasing the sample length also increases the number of sample support required to accurately approximate the covariance matrix. This significantly increases the time required to collect support data and the computational requirements to run the adaptive detection algorithm.

As anticipated, changing the values of N and K had a significant impact on the false alarm rate of the detector. This makes sense when we consider Equation (3.20) that was derived by Kelly to set the ideal threshold for correlated Gaussian clutter. That threshold is impacted by both the sample length and the level of sample support. Since the threshold setting agents are trained on data with fixed values of N and K, it is unsurprising that the threshold values they set do not extend well to other sample length/sample support values. There are two relatively simple approaches that can be applied to resolve this issue. The sample models can be retrained on variable length data. As I noted before, the data preprocessing used in this system that orders and down-samples the data allows the models to continue functioning regardless of the input data size (so long as there are at least 64 values). By adding additional inputs telling the threshold setting models what sample length and level of sample support were used, these models could be trained to approximate threshold functions that take these parameters into account. Alternatively, the behavior of the system for different values of N and K can be characterized, and a simple fitting function can be derived that adjusts the output threshold provided to the GLRT. The first approach is likely more robust, but requires significantly more data generation and training. Additionally, it requires some level of redesign of the models. The second approach is relatively fast and easy to implement and does not require significant redesign of the ML components of the system.

To prove that the proposed system could be easily modified to support different sample sizes and sample support levels, I ran a halving algorithm in Matlab to generate GLRT threshold curves (like those in Figures 6.3 and 6.4) for values of sample length in the range $N = 8$ to 64 and sample support in the range $K = 2N$ to $6N$. I then used those curves to generate a three dimensional surface with N and K serving as the x- and y-dimensions, and the threshold value serving as the z-dimension. Finally, I used Matlab's built-in "fit" function to derive a fitting function to translate between points along the surface. I coded that function into Python and then modified the original system implementation to call it immediately after the threshold setting agent to adjust the threshold passed to the GLRT. This allowed me to take the existing system, which was trained to produce thresholds for a particular x/y point on the surface and adjust the thresholds produced for any given values of N and K on the surface. This should allow the system to maintain the desired P_{fa} across a large range of sample sizes and sample support. The derived fitting function was:

$$\begin{aligned} \lambda_i &= 0.3475 + 2.139\lambda - 0.05194N - 1.357\lambda^2 - 0.04715\lambda N + 0.002065N^2 \\ &\quad +0.0811\lambda^2 N - 0.001269\lambda N^2 - 0.00001438N^3 \\ \lambda_{new} &= 0.8509 + 0.8281\lambda_i - 0.03626K + 0.09224\lambda_i^2 - 0.005008\lambda_i K \\ &\quad +0.0004824K^2 + 0.007174\lambda_i^2 K - 0.00007838\lambda_i K^2 - 0.000001889K^3 \end{aligned} \tag{6.6}$$

where λ is the threshold output by the ML models, λ_i is an intermediate threshold value, λ_{new} is the adjusted threshold, N is the sample length, and K is the number of support samples. Some initial testing was then performed to evaluate this system modification. The results of that testing are presented in Table 6.9.

The application of the fitting function had the desired effect, causing the meta-cognitive detector to maintain the target P_{fa} across a large range of sample lengths and sample support values. The consistent behavior of the modified detector provides a strong indication that the threshold setting agents learned to effectively approxi-

Distribution	**Parameter**	**Value**	**Simulation Number**	**False Alarms**	$\mathbf{P_{fa}}$
		$SampleSupportTesting$			
Gaussian	K	32	$100,000$	9	$\mathbf{0.000090}$
K	K	32	$200,000$	21	$\mathbf{0.000105}$
Pareto	K	32	$200,000$	23	$\mathbf{0.000115}$
Total	K	32	$500,000$	53	$\mathbf{0.000106}$
Gaussian	K	48	$100,000$	10	$\mathbf{0.000100}$
K	K	48	$200,000$	24	$\mathbf{0.000120}$
Pareto	K	48	$200,000$	23	$\mathbf{0.000115}$
Total	K	48	$500,000$	57	$\mathbf{0.000114}$
Gaussian	K	64	$100,000$	10	$\mathbf{0.000100}$
K	K	64	$200,000$	19	$\mathbf{0.000095}$
Pareto	K	64	$200,000$	20	$\mathbf{0.000100}$
Total	K	64	$500,000$	49	$\mathbf{0.000098}$
		$SampleLengthTesting$			
Gaussian	N	16	$100,000$	9	$\mathbf{0.000090}$
K	N	16	$200,000$	21	$\mathbf{0.000105}$
Pareto	N	16	$200,000$	23	$\mathbf{0.000115}$
Total	N	16	$500,000$	53	$\mathbf{0.000106}$
Gaussian	N	24	$100,000$	9	$\mathbf{0.000090}$
K	N	24	$200,000$	21	$\mathbf{0.000105}$
Pareto	N	24	$200,000$	21	$\mathbf{0.000105}$
Total	N	24	$500,000$	51	$\mathbf{0.000102}$
Gaussian	N	36	$100,000$	11	$\mathbf{0.000110}$
K	N	36	$200,000$	22	$\mathbf{0.000110}$
Pareto	N	36	$200,000$	20	$\mathbf{0.000100}$
Total	N	36	$500,000$	53	$\mathbf{0.000106}$

Table 6.9: Modified System Test Results - Varying Sample Length and Sample Support

mate the desired threshold curves without significant gaps or anomalous behaviors. It further indicates that both the discriminator models and threshold setting models are largely agnostic to the size of the input data. If the discriminator model's performance were impacted, then there would have been a significant increase in misclassifcations. If the thresholding agents were impacted, then they would have set threshold values that did not closely align with the threshold curves on which they were trained. In either case, applying the threshold adjustment, which was based on experimentally derived ideal threshold curves for the GLRT, would not have helped the system to maintain the target false alarm rate. This adjustment provided direct mappings between threshold curves for different values of N and K, so reduced performance of the models would not have been improved by such a mapping. Additionally, the simplicity of this solution demonstrates the flexibility of the proposed ML-enhanced method.

While this initial testing provides a strong indication that my meta-cognitive approach to adaptive detection can be readily applied in real-world scenarios, considerable additional testing needs to be performed to fully characterize it. In particular, I have not conducted P_d testing to determine the full ROC performance over N and K. Additionally, I was not able to explore training the ML models on multiple data sizes. Finally, thorough analysis needs to be performed to determine the impact to the computational cost of this method as the sample support and sample length are increased. Since the ML models are largely agnostic to these parameters, the covariance matrix approximation should be the primary driver of the algorithms computational requirements, meaning we can expect roughly equivalent performance to existing state-of-the-art. However, the preprocessing required to acquire the down-sampled ordered statistics will have some modest impact that will increase with both the sample size and level of sample support provided.

6.5 Conclusions

In this chapter, I have described the ideation, development, and characterization of a novel meta-cognitive adaptive detection algorithm. Traditional adaptive detection algorithms perform a binary hypothesis test comparing a test statistic with a threshold value. The test statistics are derived from likelihood ratios between the H_1 (target present) and H_0 (target absent) cases. They make assumptions about the statistical distribution of the interference present in a sample under test, and they are designed to maximise the probability of detection while maintaining a constant rate of false alarms when those assumptions hold true. Unsurprisingly, these algorithms perform extremely well when the statistical characteristics of the interference closely match the distribution used to derive the test statistic, but they quickly deteriorate as they diverge. The method presented in this chapter combined ML models used to interpret the statistical characteristics of input clutter and dynamically set thresholds with an existing adaptive detection algorithm derived to maintain a consistent P_{fa} for a known distribution.

My initial design concept for the meta-cognitive detector consisted of two tiers of ML models followed by Kelley's GLRT. The first tier of models consisted of a set of specialized regression models trained to set the threshold for the GLRT for different distribution regions identified to be statistically unique. The second tier was a single classification model designed to identify the distribution region of input interference data and select the appropriate regression model for that distribution. I use the term "meta-cognitive" to refer to this tiered structure of a higher-level ML agent orchestrating a set of lower-level, more specialized ML agents. I based this design on a first principles understanding of the problem space and knowledge of the limitations of neural networks. This work served as an ambitious test case for the core premise of my dissertation: the intelligent integration of first principles-based ML with traditional signal processing algorithms can be used to substantially improve the performance of radar systems. Adaptive detection was an especially challenging sub-domain within radar due to the tight error tolerances needed to maintain CFAR-like behavior and the broad array of well known and thoroughly characterized signal processing algorithms.

At a base level, the goal of any detector is to determine whether or not a target is present in a given sample under test. To do so, the detector needs to both calculate a test statistic and produce a reliable threshold. The GLRT produces a well understood test statistic that is designed to be correlation agnostic. The threshold that must be used to maintain a target P_{fa} will change significantly depending on the statistical properties of the interference. To set an appropriate threshold for the GLRT across a range of clutter distributions, the algorithm needs to be able to both identify the distribution of the interference and set a threshold for it. However, the behaviors of distributions for different parameter values (like shape and scale) vary wildly, and the threshold needed to maintain a target P_{fa} varies with them. Such varied behavior makes it difficult for a single regression neural network to approximate the threshold function across an entire distribution accurately enough to maintain P_{fa} values on the order of 10^{-4}. Additionally, there is considerable overlap in the statistical characteristics of different distributions with certain parameter values. This can make two distributions with particular parameter values virtually indistinguishable from one another. That confusion makes it difficult for a classification neural network to broadly distinguish between distributions like the K and Pareto distributions.

I was able to overcome these challenges for the Gaussian, K, and Pareto distributions by dividing the distributions into statistically unique regions in which the threshold behaviors are relatively consistent and training one specialized regression model for each region. This allowed the regression models to closely approximate the threshold function for their given regions and the classification model to reliably distinguish between the regions. In addition, it minimized the impact of errors in the ML models by simplifying the problem space they were tasked with solving. For instance, when the discriminator misclassified, it would generally select a region that neighbored the ideal region and used a regression model that often produced a threshold similar to the target threshold. Additionally, this design allowed for the use of low-cost ANNs, minimizing the computational overhead of the ML portion of the system. The resulting detector was one that maintained CFAR-like behavior across a range of distributions for which the core adaptive detection algorithm was not derived.

The meta-cognitive detector was first implemented with three main elements: 1) data preprocessing to produce a set of 64 DSMOS from the raw support samples, 2) ML models to identify the distribution region of the interference and set the detection threshold, and 3) the GLRT to compute the test statistic and determine whether a detection was made. The ML portions of the detector were trained on clutter data from the K- and Pareto distributions over a wide range of shape parameter values, while the GLRT selected was derived assuming correlated Gaussian clutter. Large scale testing was conducted to characterize both the false alarm rate and the ROC of this detector when presented with Gaussian, K-, and Pareto clutter. Its behavior was compared with the Gaussian GLRT, Gaussian AMF, and an alternative ML detector presented in [152, 153].

Importantly, the fully trained meta-cognitive detector was able to match the performance of the Gaussian GLRT when presented with only Gaussian interference, and it was the only detector that successfully maintained the target P_{fa} across all three interference distributions. That testing demonstrated that the inclusion of the ML models to set the threshold for the GLRT allowed the detection algorithm to remain performant when the assumptions used to derive it were violated. This was achieved without negatively impacting its performance when presented with the ideal case for which it was designed. This was an extremely important finding, as it provides evidence that this approach to integrating ML into traditional detection algorithms produces a detector with improved performance without any significant trade-off. However, considerable additional testing would be required to determine how robust the approach is and how well it could be applied to real-world data.

Further testing was conducted to determine how well the proposed detector could generalize in different scenarios. First, it was tested on clutter data generated from a lognormal distribution that it had never seen before. While the detector did experience degraded performance in the lognormal interference, it had considerably better P_{fa} and ROC characteristics than the comparison detectors. Next, the model's P_{fa} performance was evaluated when four data parameters were changed: 1) the correlation coefficient used to model the clutter covariance, 2) the clutter distribution scale parameter, 3) the sample length, and 4) the number of support samples used. The proposed method displayed no change to the P_{fa} when the clutter correlation coefficient was changed, indicating that this parameter had little to no effect on the ML models used in the system. The GLRT was originally designed to be agnostic of the clutter correlation, so it was expected that this testing would not impact that algorithm. In fact, it is necessary for the P_{fa} of adaptive detection algorithms to not be dependent on clutter correlation in order for them to be proven to have the CFAR property. This testing was crucial, as it demonstrated that the integration of the ML elements of the detector did not violate this core assumption of the GLRT.

Varying the scale parameter, on the other hand, had a substantial impact on the P_{fa}. This was to be expected, given that the scale parameter alters the behavior of the distribution similar to the way the shape parameter does. The Gaussian GLRT itself cannot operate as desired in different distributions and must rely entirely on the adaptive thresholding provided by the ML models to maintain performance. In order to maintain target P_{fa} over this parameter, it would need to be included in the identification of statistically unique regions, and the ML components of the detector would need to be trained to account for it. I was able to demonstrate this on a new implementation of the detector trained with two additional regions which were targeted towards a range of scale parameter values for the K-distribution. The retrained system was able to maintain the desired P_{fa}, demonstrating both that the meta-cognitive approach could be readily expanded to include new distribution regions and that these regions could be easily defined by a combination of shape and scale parameters.

Finally, it was found that altering either the sample length or the level of sample support impacted the false alarm rate of the meta-cognitive detector. I was able to identify that this behavior was caused by an aspect of the GLRT (and adaptive detection algorithms in general). Altering either of these parameters impacts the quality of the quasi-whitening filter used by the adaptive algorithm, which in turn changes the threshold value needed to maintain a target P_{fa}. Thanks to the design of the ML system and the preprocessing used, the models were not impacted by the size of the input data. However, the ML models used to generate thresholds were trained on values that assumed a specific sample length and number of support samples. As a result, they would need to either be retrained or have their outputs modified to allow the system to achieve the desired performance. To demonstrate, I was able to derive a simple fitting function from the ideal threshold curves for different parameter values. It mapped the outputs of the thresholding agents to target thresholds based on the values of the sample length and sample support. Applying this function allowed the meta-cognitive detector to maintain the target P_{fa} when these parameters were varied without retraining or redesigning any of the ML models. Demonstrating that this detector could be generalized to different sample lengths and levels of sample support without prohibitive levels of redesign or retraining was critical to proving that my proposed method could feasibly be applied to real-world detection systems.

The work presented here was a critical step for my research into the first principles design of ML-enhanced systems for radar. Detection is an extremely well-researched topic whose community is understandably skeptical of ML. It presents a unique challenge due to the need to maintain extremely low rates of false alarm and tight performance requirements. I was able to prove the core claim of this dissertation by identifying a deficiency in existing detection algorithms and designing low-cost, target ML models to add flexibility to existing algorithms without negatively impacting their base performance.

Chapter 7

Augmenting Adaptive Filtering with ML

7.1 Introduction

In this chapter, I describe my ideation, development, and characterization of a method for using generative ML to approximate adaptive filtering in detection algorithms with low sample support. By evaluating the underlying issues at play with adaptive detection algorithms and their reliance on support data, I was able to draw a parallel with techniques used in image generation models. I was first able to demonstrate that by using U-Net and conditional GAN structures it was possible to train a generator model to approximate CCMs that closely matched the ideal matrices. These same models were then trained to instead directly approximate the inverted CCMs used for adaptive filtering, reducing computational requirements and processing time. In addition, the generator was integrated with simple GLRT detection algorithm, and its impact on the performance of the algorithm was tested against an existing approximation technique. Finally, I was able to evaluate how this generative ICM approximation method performed with lower sample support. The proposed method significantly outperformed the comparison approximation method with less than half of the theoretical minimum amount of sample support required by the current state-of-the-art. This work was initially published in [15].

With the creation of the meta-cognitive detection algorithm, I was able to demonstrate that ML could be used effectively to overcome the Gaussian assumption of standard radar detection algorithms without significantly impacting performance. This proved the core thesis of my research: ML carefully designed with first principles in mind can be integrated with existing methods to improve their performance in challenging domains like radar. That work demonstrated that ML could be effectively used to augment, rather than replace, algorithmic techniques, and that it could be used in applications with tight error tolerances. I then looked into other ways in which ML could be used to improve the performance of radar detection algorithms.

My investigation began with a review of major limitations and pain points in the field of adaptive radar detection. Two limitations in particular caught my attention as promising areas for the application of ML. The first was the issue of mixed clutter backgrounds and the second was the burden of sample support. Mixed clutter backgrounds refer to those scenarios in which the background is inconsistent, and different regions of the clutter fit different distribution models. They are challenging due to the reliance on clutter covariance approximation in adaptive detection and the need to assume a clutter distribution when deriving test statistics. Sample support requirements for clutter covariance approximation and inversion drives much of the compute requirement and processing time for adaptive detection algorithms. The large number of support samples required can also severely restrict a detector's ability to adapt to variable clutter backgrounds. Corrupt returns in the support sample set can cause a degradation in detector performance or even cause the approximated CCM to no longer be inevitable.

While both of these issues are open challenges in radar detection, the sample support issue is where I chose to focus my work. There were two main reasons for this. First, I consider it to be a more fundamental problem. Reducing sample support requirements improves the overall performance of the algorithms, and the speed with which they can update the whitening filter used to decorrelate background clutter. That speed would help to mitigate the effects of mixed clutter backgrounds. The second reason is that there are numerous methods for approximating the distribution of clutter and dynamically adjusting detection thresholds that could be applied to overcome a mixed clutter background. In fact, there is a strong theoretical argument to be made that the meta-cognitive detector described in Chapter 6 is one such approach if it were trained on a robust set of distributions. As such, I wanted to examine the application of ML to a slightly different problem space in radar detection.

When evaluating how best to approach using ML to reduce sample support requirements in adaptive detection, it was important to consider how this problem is normally formulated. In traditional adaptive filtering, the CCM is approximated from a set of radar return signals that are assumed to contain only correlated interference. Put another way, this problem can be considered as taking a set of randomly distributed samples encoded with correlation information and using it to generate a structured CCM. Further, the structures within the CCM are defined by the correlation information encoded into the interference data. This is virtually indistinguishable from the function of image generator models, which are typically trained to take in a matrix of randomly distributed noise encoded with target feature information and generate an image whose structures are defined by the encoded feature information.

With this similarity in mind, I began work on designing a generative model that could generate CCM approximations from raw interference samples. My earlier attempts were using variational autoencoders (VAEs), which are commonly used to synthesize signal information and reproduce images. However, the VAE models either failed to converge well or did not generalize well once trained. This was likely due to some issues with the mapping of the correlation information in the interference samples to the VAE latent distribution, potentially rooted in the structure of my loss functions. Further investigation into the use of VAEs may be worthwhile, but I ultimately decided to move away from the VAE and attempt to use convolutional GANs. GANs rely on an adversarial training method to ensure that the generator model's latent space is complete and continuous. This means that once a GAN converges, its generator is generally robust and generalizes well. However, the training process itself is delicate and introduces substantial risk. Convolutional GANs were selected due to the nature of the spatial relationships to be learned. In particular, a U-Net style generator was selected with a classical CNN for the discriminator. The U-Net generator was chosen for two reasons. First, the structure follows an encoder-decoder style with skip connections, ensuring that the latent features are learned without a substantial loss of structural information. Second, the U-Net can be designed with a level of input-output dimensional symmetry that should allow the same model to work with inputs of variable size.

During this iteration, I was able to consistently train a GAN generator to produce covariance matrices that appeared realistic, but they did not correctly contain the correlation information in the input data. This was due to a well known issue with classic GAN architectures: the discriminator is only provided with an image and tasked with identifying it as "real" or "generated". This encourages the generator to create outputs that look like they could be a real matrix, but has no mechanism for forcing it to learn strict associations with the underlying features encoded in the input noise. To resolve this issue, I turned to a variant of the classical GAN architecture: the conditional GAN. Conditional GANs differ from classical GANs only in the information that is provided to the discriminator model. Whereas a traditional GAN's discriminator sees only an image (or other target output) either from a training set or produced by the generator and is trained to classify them, the conditional GAN also provides some form of limiting information to the discriminator that contextualizes the image that it sees. In many cases, the input to the generator is used as this limiting information. In effect, the input that would provided to a generator for a given output would be concatenated onto the image provided to the discriminator (both for real and generated images). This allows GANs to be designed such that the generator is forced to learn targeted feature associations.

For my purposes, this meant that I could provide the discriminator model with the support samples that would be used to approximate a given CCM as well as the CCM itself, allowing it to learn what a realistic CCM looked like for a given set of data with particular correlation characteristics. The conditional GAN paired with a U-Net generator model was very successful at training to produce high quality facsimiles of real CCMs that contained the appropriate structures associated with the data's correlation information. I then took this design one step further and developed a method for training the conditional GAN to directly generate the ICM. Directly generating he ICM was an important modification, as it removed the need to invert the generated matrix for filtering. Removing that step made the approach more robust, since there would be no inversion errors, and it also removed the considerable computational burden associated with inverting large CCMs. Testing was then performed to determine the impact of reducing the amount support data used to train and test the generator model. I found that using this method I could reduce the amount of support data used for ICM approximation could be reduced by roughly half of the theoretical minimum used in classical methods. The reduction in support data requirements coupled with the removal of the need to invert the covariance matrix when producing the adaptive filter offset the computational burden associated with use an ML generator.

Early testing used to prove out the concept was performed using a simple clutter model with a real-valued covariance matrices containing simple structures. To demonstrate the validity of the approach to more challenging data, a Nitzberg [171] model with complex-valued covariance structures was used to generate data and train the generator. The GAN structure struggled with the higher complexity, largely due to the limitations associated with the training loop and its dependence on the discriminator model. It is likely that a discriminator model could be designed to function effectively in this scenario. However, I theorized that the GAN structure itself was unnecessary and that it was the U-Net generator that provided most of the value in the ICM approximation. This led me to remove the generator model from the GAN, modify it to produce complex-valued outputs, and implement a supervised training method with custom loss functions to generate ICMs for adaptive filtering. This approach proved to be highly effective. It will be crucial to conduct additional testing of the method presented here, particularly on real-world data, but the initial results are extremely promising and demonstrate the potential of ML to enhance adaptive detection algorithms and reduce data and computational burden associated with it.

7.2 Related Work

The development of methods enabling adaptive detection with limited sample support has been a major focus of the detection research community for several decades.

This is largely driven by two problems. First, the time it takes to collect and process the quantities of support data needed can be prohibitive [172]. Second, in airborne and maritime radar systems it is challenging to gain consistent access to large quantities of support data that is homogeneous with a given sample under test [172, 173].

In [174], Aboutanios and Mulgrew propose a variation of traditional adaptive detection algorithms that relies only on the SUT and steering vectors to suppress interference. This single data set (SDS) approach was based on the adaptive filtering techniques presented in [175, 176]. More recently, several hybrid methods that combine the SDS method with traditional support data methods have been developed [177, 178, 179].

The multi-stage Wiener filter (MWF) proposed by Goldstein *et al.* in [180] has also been adapted for use in STAP detection [172]. This method is considered relatively robust and computationally efficient. However, it requires that the number of filtering stages be selected based on the rank of the clutter being filtered [181]. This generally requires either the use of rank estimation methods [181] or data augmentation [182, 183]. In [184, 185], a hybrid detector combining the SDS and MWF methods was proposed.

Various recursive methods have also been proposed to reduce sample support requirements. [111, 186] present recursive methods for covariance approximation that assumes the clutter can be modeled as a SIRVs. [112] builds on these works and proves several important statistical properties of the recursive methods in [111, 186]. In [187], Conte and De Maio derive a recursive estimation method assuming persymmetric CCMs.

Other methods have been developed under various assumptions about the nature of the interference signals and the CCM. These include low-rank assumptions [188, 189], the spiked covariance model [190, 191, 192], and applying a unit circle roots property to detection algorithms [193, 194].

7.3 Data Model

As with almost all signal processing and ML problems, the first thing to consider when developing a new technique is the data that will be used to evaluate it. Similar to the adaptive detection problem presented in Chapter 6, adaptive filtering has tight error tolerances and has a number of existing methods derived from the assumed distribution of the clutter. When developing new adaptive filters or detection algorithms, authors in the literature consistently begin by characterizing their performance on simulated data whose statistical properties are perfectly known, rather than on real-world data. This removes any potential impact of the data and is crucial to provide a realistic analysis of the behaviors demonstrated by the proposed method or algorithm. Several common models are used for generating correlated clutter signals in the literature. I used two of them at different stages of this research. The first is a simple model widely used in the adaptive detection literature when newly derived detection algorithms are being characterized [13]. This model assumes the clutter can be modeled as a correlated, Gaussian-distributed signal such that:

$$\mathbf{c} = \mathcal{CN}(0, \mathbf{M}) \tag{7.1}$$

where M is the clutter covariance matrix defined as:

$$\mathbf{M} \sim \mathbb{R}^{N,N} \; s.t. \; M_{i,j} = \rho^{|i-j|} \tag{7.2}$$

where N is the length of a sampled interference signal generated by the clutter, ρ is a real-valued correlation coefficient, and i, j are matrix indices in the range $1 \leq i, j \leq N$.

While this model appears simplistic, it provides a strong basis for the development and characterization of adaptive detection and filtering methods. For my research, I selected it for two of these reasons. First, the correlation information in the CCM can be tightly controlled. That enabled me to examine how different CCM configurations impacted the resulting ICM used for adaptive filtering. Additionally, it allowed for thorough evaluation of the performance of the generative ML. That analysis was important at this stage, given that generative ML has not, to my knowledge, previously been used for directly approximating adaptive whitening filters. It further allowed for thorough evaluation of the impacts of reducing sample support on both the proposed method and the comparison method. The second reason for this model's selection was that it is also easy to implement and allows for rapid generation of large quantities of statistically relevant data. This meant that the approach could be implemented, trained, and evaluated rapidly to determine its validity.

After proving that a generative model could be trained to produce high quality ICM approximations with limited sample support, I then needed to test how well the approach generalized to more complex clutter models. For that evaluation, I selected the Nitzberg model from [171]. This model again assumes that the clutter can be modeled as a complex-valued, correlated Gaussian distribution, as in Equation (7.1). However, the CCM is now defined according to a combination of the clutter correlation and a velocity-based Doppler shift:

$$\mathbf{M} = \mathbf{A}\mathbf{R}\mathbf{A}^H \sim \mathbb{C}^{N,N} \tag{7.3}$$

$$\mathbf{R} \sim \mathbb{C}^{N,N} \; s.t. \; R_{i,j} = (\sigma_{cl}^2)(\rho_l^{|i-j|})(e^{j2\pi(i-j)f_d}) \tag{7.4}$$

$$\mathbf{A} = \mathbf{I}(N) \tag{7.5}$$

where N is the length of the length of a sampled interference signal generated by the clutter, σ_{cl} is the internal variance of the clutter, ρ_l is the clutter correlation coefficient, f_d is the Doppler frequency shift, $\mathbf{I}(\bullet)$ is the identify matrix of the given dimension, and i, j are matrix indices in the range $1 \leq i, j \leq N$.

The Nitzberg clutter model, like the Ward model [145], is popular in the radar literature [195, 177] due to the ability to model more realistic CCM structures that include correlation information related to velocity elements in the background. Such information is important in more challenging clutter scenarios like those encountered in space-time adaptive processing (STAP) or with ocean backgrounds. Clutter with a Doppler component is far more challenging to filter out and to disambiguate from a legitimate target whose relative velocity is similar to that of the clutter. The CCMs produced by this model are higher fidelity that the model found in [13] and are complex-valued. However, the CCM was still defined directly and used to generate clutter samples, meaning that the CCM was perfectly known in each test case and could be used for system analysis during testing. It served as a strong intermediate step before attempting to test the system on high-fidelity simulations or on real-world data, where the true CCM is generally not available and must be approximated for a given scenario.

Table 7.1: GAN Network

	Type	**Kernel Num.**	**Kernel Shape**	**Kernel Stride**	**Input Shape**	**Output Shape**	**Batch Norm.**
				Generator			
1	Conv.	16	3×3	1	16×16×2	14×14×16	N/A
2	Conv.	32	3×3	1	14×14×16	12×12×32	0.8
3	Conv.	64	3×3	1	12×12×32	10×10×64	0.8
4	Conv.	128	3×3	1	10×10×64	8×8×128	0.8
5	Tr. Conv.	32	3×3	1	8×8×128	10×10×64	0.8
6	Tr. Conv.	32	3×3	1	10×10×64	12×12×32	0.8
7	Skip	N/A	N/A	N/A	12×12×32	12×12×64	N/A
8	Tr. Conv.	32	3×3	1	12×12×64	14×14×32	0.8
9	Tr. Conv.	32	2×2	1	14×14×32	14×14×16	0.8
10	Skip	N/A	N/A	N/A	14×14×16	14×14×32	N/A
11	Tr. Conv.	32	5×5	1	14×14×32	18×18×32	0.8
12	Conv.	1	3×3	1	18×18×32	16×16×1	0.8
				Discriminator			
1	Conv.	16	3×3	1	16×16×3	14×14×16	N/A
2	Conv.	16	3×3	2	14×14×16	6×6×16	0.8
3	Conv.	32	3×3	2	6×6×16	2×2×32	0.8
4	Flatten	N/A	N/A	N/A	2×2×32	1×128	N/A
5	Linear	N/A	N/A	N/A	1×128	1×256	N/A
6	Linear	N/A	N/A	N/A	1×256	1×1	N/A

7.4 Proof of Concept Design

7.4.1 Model Architecture

For this work, I wanted to explore using generative ML models to directly infer correlation information from a set of clutter samples and construct a corresponding ICM for use in adaptive radar detection. In adaptive detection, support samples are generally used to approximate an ICM that is then used in a quasi-whitening filter (this is what makes them adaptive). To maximize efficiency and minimize information loss, raw IQ data is used for both the ICM generation and the adaptive detection algorithm.

To prove out the method, training and testing IQ data was simulated in Matlab with the known covariance structures defined in Equations (7.1) and (7.2). As I have described previously in Chapter 3.5, most modern covariance estimation techniques require that the number of support samples (K) used to approximate the ICM be greater than twice the sample length (i.e. $K > 2N$). For the generative ML approach presented here, the sample support was set to $K = N$ for all training and testing. However, I will note here that the earliest design iterations of the ML generator used $K = 2N$ and I subsequently reduced the sample support prior to final implementation and testing.

Currently, standard ML layers in Python are not designed to process complex-valued data. This is largely due to the activation and loss functions that are used, almost all of which were derived assuming real-valued inputs and outputs. At the time of writing this dissertation, Pytorch had been updated to allow the propagation of complex-valued tensors through its layers, but did not have support activation or loss functions. Various custom loss functions had been developed by the open source community, but each simply split the complex tensors into its real and imaginary components, performed calculations on each component separately, and then recombined them later. For convolutional networks like the ones I used here, this is not substantively different than simply splitting the input data to the model into its real and imaginary components and then concatenating them together along the channel dimension before sending them through the model. As a result, the real and imaginary components of the complex IQ samples used for ICM generation were separated into distinct channels. The resulting input matrix $\mathbf{Z}_{in}$ to the generative model had the form: $(N \times N \times 2)$

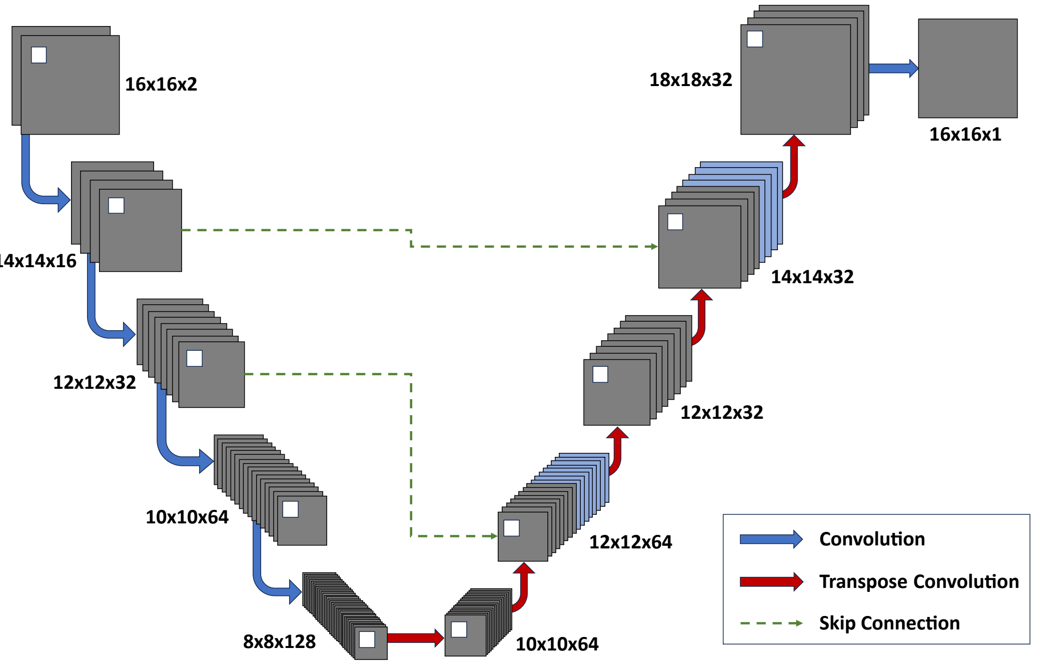
$$\mathbf{Z}_{in} = [Re(\mathbf{Z}_k), Im(\mathbf{Z}_k)] \tag{7.6}$$

where $Re(\bullet)$ refers to the real component of a complex matrix, $Im(\bullet)$ is the the imaginary component of a complex matrix, and $\mathbf{Z}_k$ is the set of K support samples. For this work, it was assumed that the CCM was real valued and hermitian and the model output is the associated ICM, $\mathbf{Y}_{out}$, which is also real-valued and hermitian with dimensions $(N \times N \times 1)$.

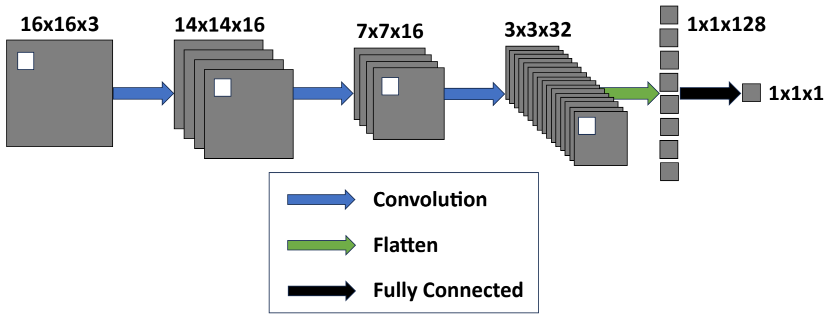
The prototype generative model developed for this research consisted of a GAN created in Pytorch. GANs are neural networks whose architectures consist of both a generator model and a discriminator model that must train in tandem. The architecture of the GAN presented here is depicted in Figure 7.1 and specific layer parameters are provided in Table 7.1. A U-Net implementation was used for the generator model. This model consists of an encoder-decoder structure with a series of convolutional layers in the encoder to identify key features in the input data and a series of transpose convolutional layers in the decoder to construct the output. Skip connections from the encoder to the decoder are also used to prevent the loss of key structural (or positional) information when constructing the output. Leaky Rectified Linear Unit (Leaky ReLU) activation layers with negative slope factors of 0.2 were used after each convolutional and transpose convolutional layer and Tanh was used as the final activation layer of the model. Additionally, batch normalization with momentum of 0.8 was applied during model training after each Leaky ReLU other than the first.

U-Nets are desirable for this application for several reasons. First, ICMs for correlated clutter are highly structured and those structures are dependent on positional information in the clutter samples. The use of convolutional layers in both the encoder and decoder are ideal for inferring such relationships, making it an architecture that will likely generalize to a broad range of ICM structures. The encoder-decoder structure of the U-Net also aids with the ability to generalize. Encoder structures in neural networks are designed to learn characteristic features of input data while the decoder takes those features and constructs the appropriate output. The added skip connections in U-Nets help minimize the loss of positional information caused by the encoding process. This allows them to perform consistently even on large or relatively sparse data sets. Additionally, U-Nets can be designed with inherent symmetry that allows for paired input/output dimensions regardless of the size of the input data. This means that a single generator model can be designed and trained to output ICMs for a broad range of sample length values. Note also that convolutional layers train filters that are each applied across every channel of the input data. So, while the dimensions of the input data used to characterize this prototype were kept static at $(N \times N \times 2)$, by increasing the number of channels at the model input and using data trimming/-padding the model can trivially be trained to support almost any amount of sample support without structural modification to the model.

The discriminator model was designed as a convolutional neural network (CNN) classifier with a binary output. The model receives as an input an ICM (either real or generated) concatenated with the input matrix to the generator ($\mathbf{Z}_{in}$), which is an $(N \times N \times 3)$ matrix. The input is passed through three convolutional layers, each with Leaky ReLU activation with negative slope factors of 0.2. The second and third convolutional layers have batch normalization with momentum of 0.8 applied during training. The output of the third convolutional layer is flattened and passed through two linear layers with ReLU activation. The final activation function for the discriminator was sigmoid and binary cross entropy was used for the loss function. The two output states for the discriminator model correspond to “real” or “generated” classifications for the observed ICM.



(a) GAN Generator Architecture.



(b) GAN Discriminator Architecture.

Figure 7.1: GAN architecture used for ICM generation.

7.4.2 GAN Training

The generative model used in this work was trained on 8×10^5 randomly generated sets of correlated Gaussian modeled using Equations (7.1) and (7.2). After training data was loaded, it was shuffled and divided in 80% training samples, 10% validation samples, and 10% testing samples. Each noise set consisted of $K = 16$ samples, each of length $N = 16$. The training noise samples were generated with covariance matrices of the form in (7.2). 2×10^5 samples were generated with correlation coefficients at $\rho = 0.1, 0.5, 0.7, 0.9$. The ideal ICM was calculated for each noise sample set and scaled down by a constant so that the values in any given matrix ranged between -1 and 1 to align with the output of the generator's final Tanh activation. Note that the same scaling factor was used for every training set, and the ICMs output by the generator were scaled up by the same value.

The generator and discriminator models were trained in parallel using Pytorch Lightning. Training was conducted for 100 epochs with a batch size of 32, Adam optimization, and learning rate of 10^{-4} for both the generator and discriminator models. At each training step, the generator was presented with a single set of noise samples and output a generated ICM. Next, two inputs were provided to the discriminator, one constructed from the noise samples concatenated with the ideal ICM and the other from the noise samples concatenated with the generated ICM. For each of the two inputs, the discriminator identifies if the observed ICM is ideal or generated. The average loss for the two classifications is calculated and used to update the discriminator weights. If the discriminator successfully classifies the output of the generator model, the generator loss function is calculated and its weights are updated. Otherwise no update is made to the generator model.

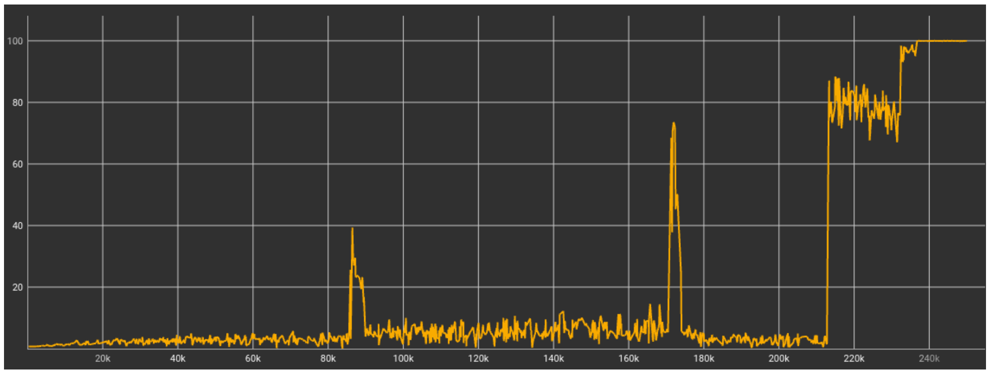
I want to again highlight that much of the difficulty with successfully training a GAN rests in the need to maintain approximate parity in the performance of the two models. If either model trains significantly faster than the other, the training process breaks down and typically does not recover. This requires substantial tuning of the model hyper-parameters and architectures to ensure mutual convergence. For the prototype system presented here, mutual convergence was achieved in roughly two out of three training runs. In each case where training failed, it was due to the discriminator learning too quickly to distinguish between the real and generated ICMs. This was almost certainly due to the comparatively high complexity of the generator's task.

Traditionally, GAN discriminators are trained using only the output of the generator or the real data the generator was being trained to emulate. However, this only trains the discriminator to broadly classify the data it observes as "realistic" or "unrealistic". Such a generic classification is insufficient for applications like this one, where the validity of the generator output is tied to the characteristics of the specific input clutter. This issue caused early iterations of my design, in which I only provided the ideal and generated ICMs as the inputs to the discriminator, to produce ICMs that had the correct general structure of an ideal ICM but did not properly reflect the correlation information of the input data. While researching ways to mitigate this issue, I discovered the conditional GAN architecture. The conditional GAN variant provides additional information to the discriminator that provides extra context so that it learns what a realistic input looks like for a given scenario. I was able to update my design such that the noise samples used by the generator are also passed to the discriminator. Passing both the support samples and the observed ICM to the discriminator allowed it to learn the noise characteristics that lead to specific ICM structures, as opposed to just learning whether an ICM looks generally realistic. This in turn forced the generator to learn to recognize the noise characteristics of a set of support data and generate the appropriate ICM structures for that noise data.

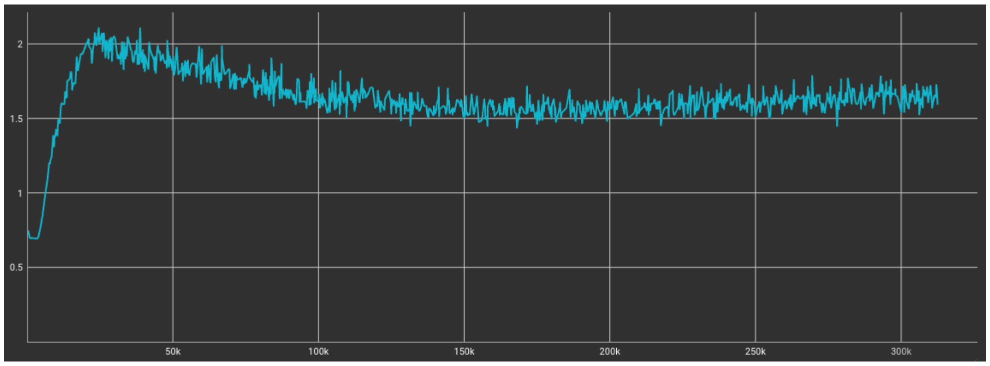
7.5 Testing and Results

My ultimate goal with this research was to determine whether ML could be used to reduce sample support requirements for the process of approximating ICMs used in adaptive radar detection algorithms. This required that I demonstrate that the model could consistently generate close approximations of CCMs and that those approximations were sufficiently high-fidelity to not significantly impact the performance of an adaptive detection algorithm. Two major forms of testing were conducted on the prototype system to both prove out the concept and determine its applicability to the detection domain. First, the quality of the generated ICMs were evaluated against both the ideal matrices found by inverting the true covariance matrix and against estimated matrices found by inverting covariance matrices approximated using the method of Equation (3.61) (described in Chapter 3.5). Second, preliminary testing was done to integrate the prototype ICM generator into the well known GLRT detection algorithm and characterize its impact on detection and false alarm rates. Unless otherwise stated, all generated ICMs were generated using $K = N$ support samples and all estimated ICMs used $K = 2N$. This is an important distinction, as it means that the comparison method was given twice the data as the generative model. Further, the number of support samples used by the generative model to approximate ICMs was not sufficient for existing alternatives to consistently approximate an invertible CCM.

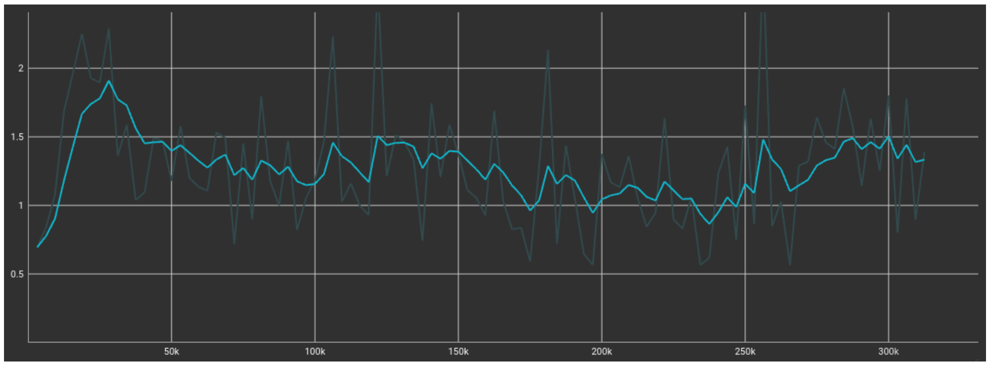
The conditional GAN was trained multiple time for 100 epochs. All told, roughly 200 models were trained and evaluated. During training, both the generator and discriminator loss were monitored. Model checkpoints were saved periodically based on the generator loss. Each trained model produced ten checkpoints that corresponded to the ten epochs with the lowest generator model validation loss. This was evaluated



(a) Generator training curve for an unconverged model.



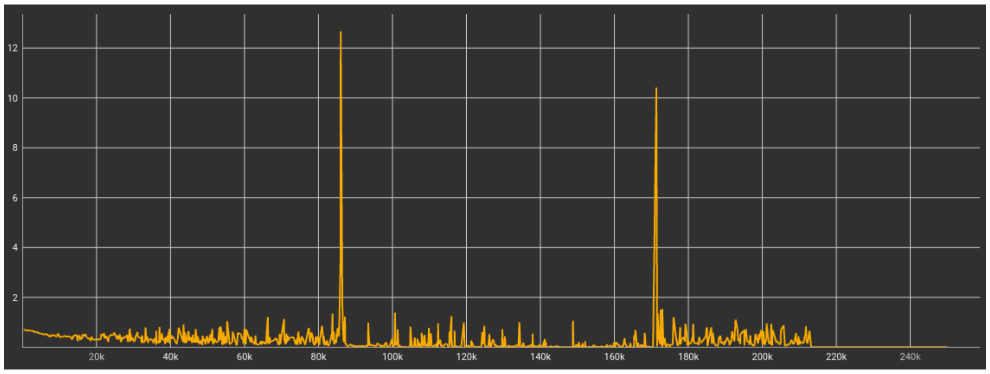
(b) Generator training curve for a converged model.



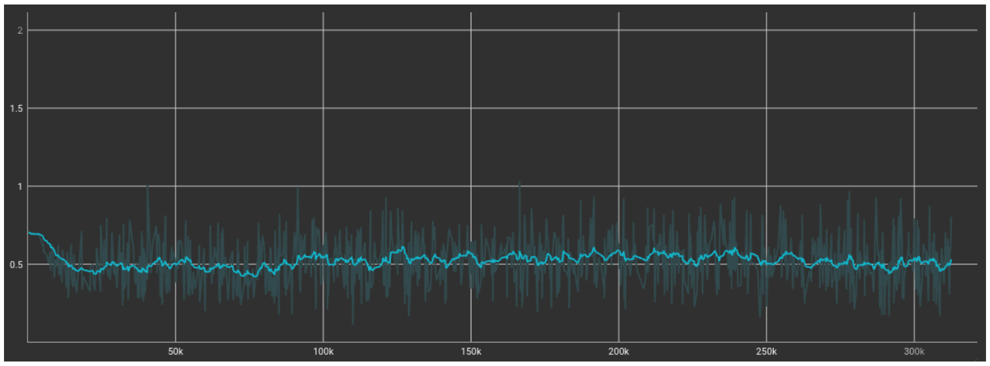
(c) Generator validation curve for a converged model.

Figure 7.2: Example GAN generator model training curves.

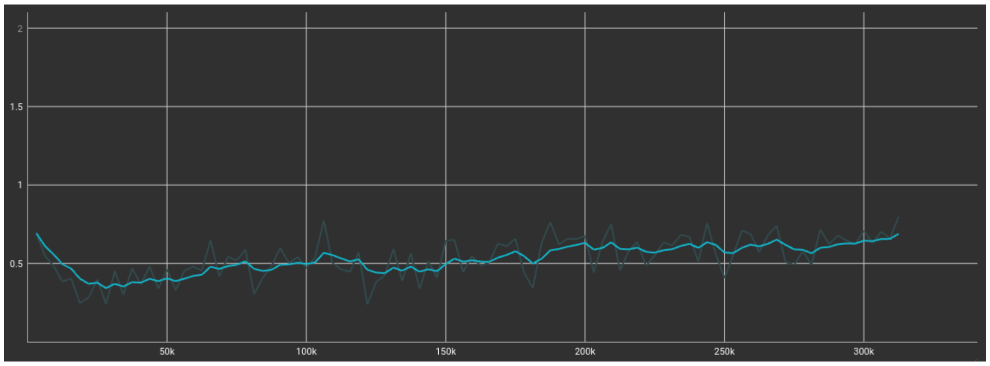
after each epoch, so initially the first ten epochs were saved by default and then if any subsequent epochs produced a lower validation loss, it would replace the worst performing existing checkpoint. Example training curves for the GAN are shown in Figures 7.2 and 7.3. Note that Figures 7.3a and 7.2a show curves for the discriminator



(a) Discriminator training curve for an unconverged model.



(b) Discriminator training curve for a converged model.



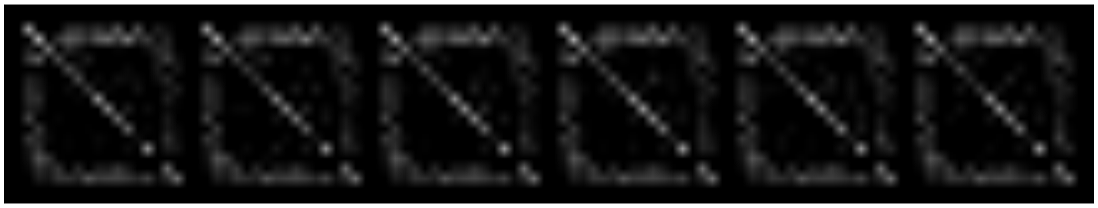
(c) Discriminator validation curve for a converged model.

Figure 7.3: Example GAN discriminator model training curves.

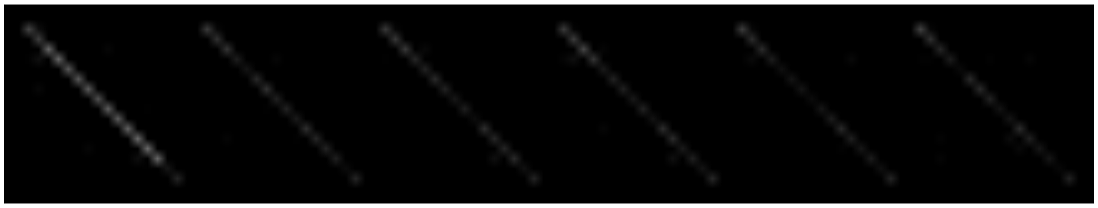
and generator loss, respectively, for a GAN whose generator failed to converge. Figures 7.2b, 7.2c, 7.3b, and 7.3c, conversely show those loss curves for a model whose generator did converge. There is some interesting behavior that can be observed in the training curves for each case. In particular, it should be observed that for the GAN that failed to converge, the generator loss continued to grow steadily with several regions of unstable training (marked by large spikes in loss) until the generator reached a point of instability in training from which it did not recover. The discriminator, on the other hand, maintained relatively low loss that dropped to zero once the generator failed. This demonstrates clearly what happens when the discriminator learns too quickly and the generator cannot reach a convergence point. For the GAN that did converge, the generator loss initially rose as the discriminator learned to differentiate between real and generated images. However, both the generator and discriminator loss curved quickly reached a stability point after which their loss curves displayed periodic, inverse cycles of increase and decrease. This behavior is indicative of a stable adversarial training process in which each model improves and then pushes the other model to improve in response.

Models whose generator loss successfully converged were pulled and examined using a tool called Tensorboard. This tool allows the training of a model to be observed using log files both in real time and after the fact. In my Python code, I set up training logs to monitor the discriminator loss, generator loss, epoch number, step number, and to save example model outputs from the testing data set. Any models whose training curves maintained parity, whose generator loss converged to low values, and whose output images appeared to contain the expected structures for an ICM were chosen as candidates for the final generator. Ten models were chosen as candidates and three checkpoints for each of those candidate models were pulled down for further evaluation. Figure 7.4 shows examples of the logged generator outputs from different stages of GAN training. Note that the generator output initially produces low-valued, almost perfectly random outputs, as shown in Figure 7.4a. In Figure 7.4b, one can see the early stages of a generator beginning to learn the structures of the

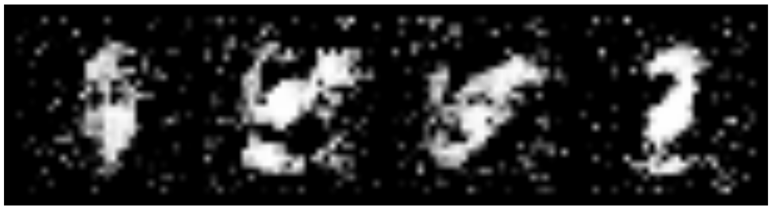
(a) Example generator outputs at the beginning of training.



(b) Example generator outputs in early training epochs.



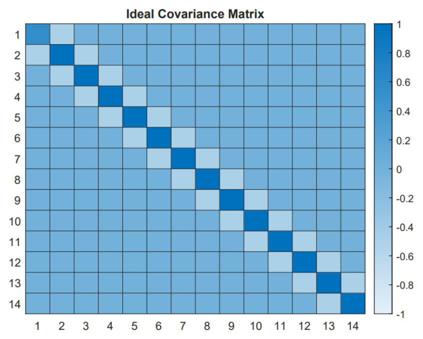
(c) Example generator outputs of converging model.



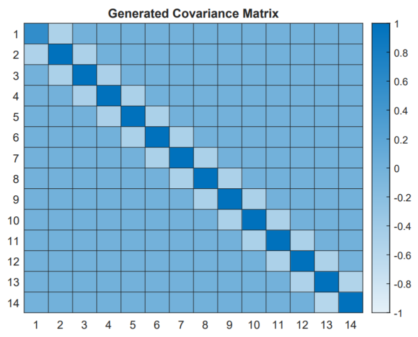
(d) Example generator outputs when model fails to converge.

Figure 7.4: Example generator outputs at different stages of GAN training.

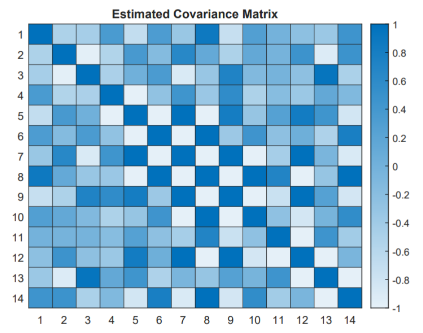
ICMs. Figure 7.4c shows the outputs produced by a model a model that successfully converged and produced realistic ICMs approximations. For comparison, Figure 7.4d shows example outputs from later epochs of a GAN whose generator failed to converge. Unsurprisingly, the model did not continue to output random noise, but instead experimented with different structures to see if it could fool the discriminator. Since the discriminator trained to effectively too quickly, the generator experimented with increasingly pronounced, though unfocused, structures.



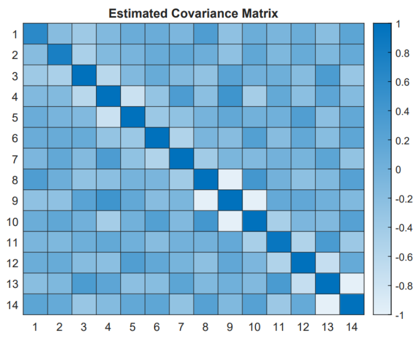
(a) Ideal ICM for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.



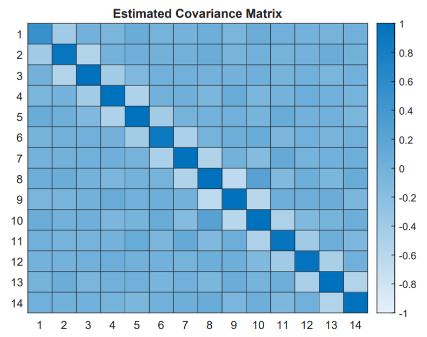
(b) GAN generated ICM for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.



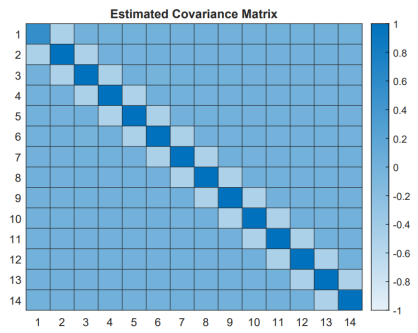
(c) ICM estimated from $K = N$ support samples for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.



(d) ICM estimated from $K = 2 \times N$ support samples for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.



(e) ICM estimated from $K = 10 \times N$ support samples for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.



(f) ICM estimated from $K = 500 \times N$ support samples for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.

Figure 7.5: Heat maps of inverse covariance matrix for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.

In general, convergence of a neural network to low loss values during training is a good indication that it identified consistent patterns within the training data and produced some kind of meaningful output. However, loss/reward values are not sufficient metrics for determining the success of a model, particularly when the output is complex and high-dimensional. To evaluate the quality of the trained models used in this research, the Frobenius distance between generated and ideal ICMs was used. Each candidate checkpoint of the trained GAN was evaluated in this way. Figure 7.5 shows an example of the ideal ICM, generated ICM, and ICMs estimated from increasing levels of sample support. For each of the covariance structures evaluated here, the ideal ICM peak was always located along the primary diagonal, the trough was always located along the first off-diagonals, and the rest of the ICM elements were comparatively near to zero.

Metric	ρ **- 0.9**	ρ **- 0.7**	ρ **- 0.5**	ρ **- 0.1**
GAN FD Full	0.8699	0.9057	0.7190	0.5663
GAN FD Diag.	0.0542	0.1608	0.1355	0.0891
GAN FD Off Diag.	0.6216	0.1968	0.1213	0.0655
GAN FD Res.	0.6058	0.8674	0.6937	0.5552
Est. FD Full - 2N	25.8287	8.6238	1.7248	0.3498
Est. FD Diag. - 2N	9.8307	3.3759	0.7008	0.1932
Est. FD Off Diag. - 2N	8.1024	3.3646	0.6755	0.1021
Est FD Res. - 2N	22.3421	7.1605	1.4172	0.2717
Est. FD Full - 10N	1.0046	0.3533	0.1900	0.1013
Est. FD Diag. - 10N	0.3932	0.1303	0.0714	0.0448
Est. FD Off Diag. - 10N	0.3909	0.1256	0.0672	0.0336
Est FD Res. - 10N	0.8319	0.3017	0.1619	0.0840

Table 7.2: F_{dist} Against Ideal ICM for Gen. and Est. ICMs

Based on those consistent structural patterns in the matrices, four Frobenius distances were calculated and used to select the final model. Each one targeted a different region within the ICMs and were found by applying masks to the matrices before calculating the distance. First, the distance between the full generated and ideal matrices were calculated. Second, the generated and ideal matrices were multiplied by an identity matrix mask and the distance between the diagonals was found. Then, each value in the matrices other than the first off-diagonals were set to zero, and the distance was calculated. Finally, the primary diagonal and first off-diagonals were masked to zero for each matrix and the distance was calculated. These four distances allowed me to examine the similarity of each key region within the ICMs as well as for the full matrices. Each model was evaluated using these four metrics at four distinct correlation coefficient values and the model with the lowest overall distances was selected for full testing. The distances for this model, along with the distances between the standard approximation method with $K = 2N$ and $K = 10N$ sample support and the ideal matrices are shown in Table 7.2. It should be highlighted here that the generative model only ever sees interference samples drawn from a distribution with given covariance structure, and it was only characterized on new data samples. This means that the GAN would have been unable to overfit to particular interference data and the quality of its outputs were entirely determined by its ability to recognize the covariance information in each sample and apply well trained filters to construct the desired ICM.

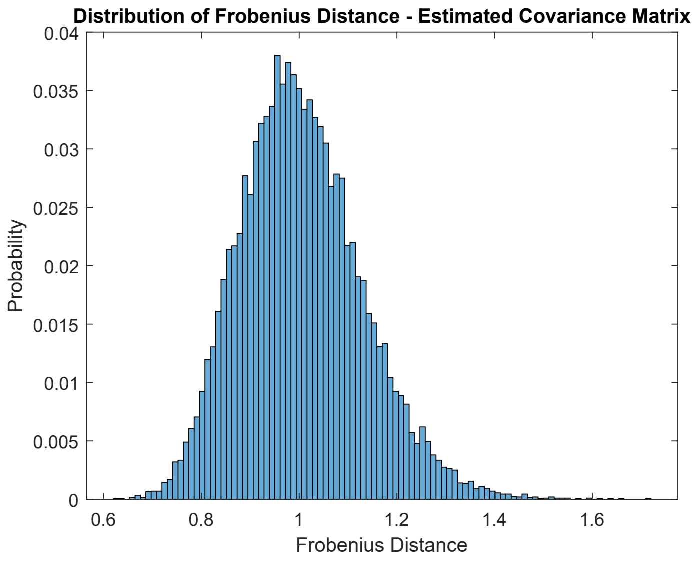
Note that the distance values for the estimated matrices are high for low sample support and more correlated clutter, and reduce as both the correlation drops and sample support increases. This is to be expected, as less correlated noise produces ICMs with simpler structures that are easier to estimate, approaching a scalar matrix as the correlation coeffiecient approaches zero, and it is well established that the estimation method used here is dependent on the number of support samples. Conversely, the the generative approach maintains much more consistent distance values, indicating that it learned the structures associated with particular interference sets well and consistently generated ICMs that closely matched the ideal. Additionally, Figure 7.6 depicts an example of the distribution of the total Frobenius distance over 2×10^4 test sets. Figures 7.6a and 7.6b show the distance distributions for the estimated ICMs with $K = 10N$

and the generated ICMs, respectively. Clearly, both distributions are roughly Gaussian and tightly distributed, further indicating the consistency of the proposed approach.

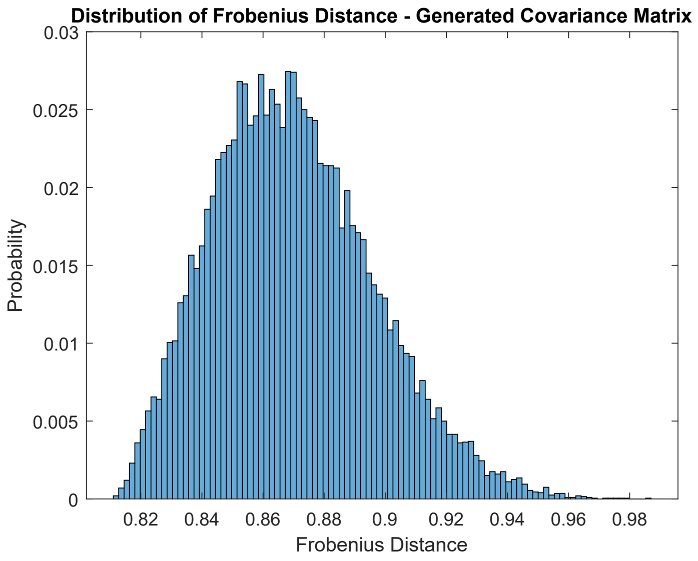
I performed some additional testing at this stage with variable sizes of input data to the generator. Specifically, I wanted to verify my initial hypothesis that a U-Net generator could be designed with sufficient symmetry between the encoder and decoder that it would be able to produce ICMs from data of almost any size without alteration. This is enable because these models consist entirely of convolutional, pooling, and transpose convolutional layers whose only trainable parameters are filter of fixed dimensions. So, as long as the input data matrix is sufficiently large such that the dimensionality reduction through the encoder side of the network doesn't reduce the dimensions of the data tensors below the dimensions of any of the model's filters, the generator model I designed should successfully produce an output of the same spatial dimensions (matrix height and width) as the input. To test this, I ran datasets with dimensions ranging from 12×12 to 36×36 through a generator model trained on 16×16 data and evaluated the results. For each of these datasets, the model successfully produced an output of the expected dimensions. Additionally, for most of these datasets the basic structure of the ICM could be observed in the output matrices.

However, most contained some combination of large gaps or artifacts in the generated matrices and had significantly degraded Frobenius distance values. For datasets of similar dimensions to those used to train the model (14×14 to 18×18), the gaps and artifacts in the ICM structures were minor, but they grew rapidly after that. This is almost certainly due to a combination of a lack of latent features related to variable sized inputs and a decoder not trained to produce variable sized outputs. From this, two important observations can be made. First, the generator structures can support almost any size input data and produce a symmetric output. Second, in order to produce consistent, meaningful results for different dimension of input, the networks needs to be trained on variable sized data. I explored doing variable data size training and ran into an issue with the GAN. While the generator model was dimension agnostic, the discriminator was not. It contains linear layers, whose parameter numbers must be defined and are not compatible with variable sized inputs. So, to train the generator on dataset of various size, I needed to create a discriminator for each data size. This is not a viable long-term solution, which limits the current design significantly. Fortunately, this testing also provided a reasonable indication that the U-Net used in the generator could be trained for a range of input sizes if it was removed from the GAN structure and provided a sufficiently large and general training dataset with variable dimensions.

It must again be highlighted here that the generative model was evaluated under a far more stringent set of conditions than the traditional estimation method. The traditional estimation method was only ever evaluated with $K \geq 2N$. This level of sample support is necessary for the approximated CCMs to be consistently invertible and achieve reasonable performance. The trained generator model, by contrast, only ever saw $K = N$ support samples. Despite only having access to half the level of sample support, the generative model was able to produce ICMs that were of equivalent or better quality than those produced using the comparison method. That demonstrates that there is sufficient information in a small sample set for the model encoder to identify the underlying correlation information of the interference signals such that the model decoder can reconstruct a close facsimile of the true ICM. With such low sample support, this is not possible with existing covariance approximation techniques. Reducing sample support requirements to half of the current minimum would provide a massive increase to levels of flexibility and agility in radar detection systems (particularly those on mobile platforms). If this method can be shown to generalize well on real world, it will provide a major step forward in the field of adaptive detection.



(a) Distribution of estimated ICM with $K = 10N$.



(b) Distribution of generated ICM with $K = N$.

Figure 7.6: F_{dist} distribution over 2×10^4 tests of ICM approximation for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.9$.

After evaluating the quality and consistency of the generated ICMs and selecting the best performing model, I integrated that model with an adaptive detection algorithm to determine whether the generated ICMs could be used effectively for adaptive filtering. This was done by replacing $\mathbf{S}^{-1}$ in Equation (3.22) of the GLRT with the generated matrices and evaluating the impact to the detector's performance. This testing was conducting following the same process outlined in Chapter 6.4.1 for characterizing detector performance with ROC curves. Correlated complex Gaussian interference data was generated with covariance defined by (7.2). 1×10^5 sets of support samples were generated for each covariance structure tested, and both H_0 and H_1 SUT were generated for each. For the H_1 SUTs, a steering vector of $\mathbf{p}_i = \exp^{-j2\pi(i-1)0.2}$ was used. The SIR used in each H_1 test was set using (6.5). The P_{fa} and P_d characteristics of the GLRT were measured with an SIR of 3dB using the ideal ICMs ($\rho = 0.9$), estimated ICMs ($K = 2N$, $\rho = 0.9$), and the generated ICMs ($K = N$, $\rho = 0.9, 0.7, 0.5$) and used to generated receiver operating characteristic (ROC) curves for each. Figure 7.7 shows these curves. Note that, at this stage, for the generative case the thresholds used to achieve each target P_{fa} were determined experimentally. It is clear that the ROC curves for the GLRT augmented with the GAN display improvements consistent with what could be achieved by other approximation methods using higher sample support. This clearly demonstrates that the model successfully generates ICMs consistent enough with reality to be used in adaptive filtering. It also provides further indication that the proposed method substantially reduces sample support requirements for adaptive filtering.

Next, the target's normalized Doppler frequency, F_d, was swept from 0.05 to 0.35 while the detector steering vector was held constant with $F_d = 0.2$. The P_d for the GLRT using both the estimated and generated ICMs was recorded with $\rho = 0.5$ and

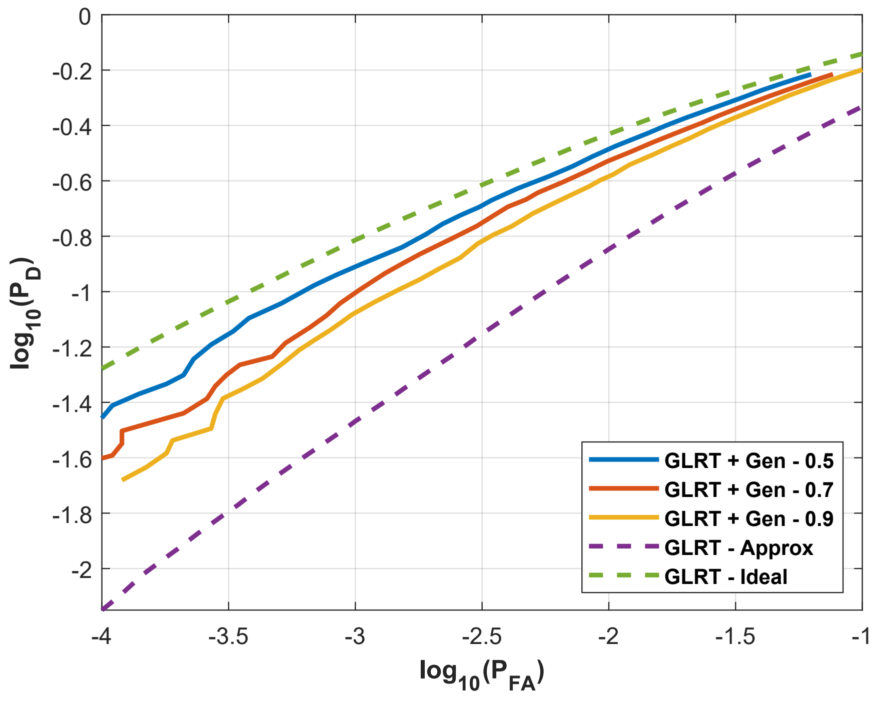


Figure 7.7: ROC curves showing the P_D vs P_{FA}.

SIR of 10dB. Figure 7.8 shows the plot of the P_d versus F_d for each. It is clear that the performance of the detector using generated ICMs is consistent with that of the estimated ICMs, with some improvement to both P_d of targets aligned with the steering vector and rejection of targets outside of the steering vector. These results are particularly exciting, as they demonstrate that the filter performance with the generated ICMs is consistent with that of an estimated ICM using the widely used comparison method with higher sample support. This provides strong evidence that the ICM generation method functions as desired. Substantially degraded detector performance would indicate that the method was not viable for the application. Conversely, ideal performance of the filter would be suspect and likely indicate that the generator had overfit on data and would not generalize well. The observed performance, coupled with previous observations of the Frobenius distances, supports my hypothesis that a generative model can be used to replace traditional CCM estimation methods in adaptive detection algorithms.

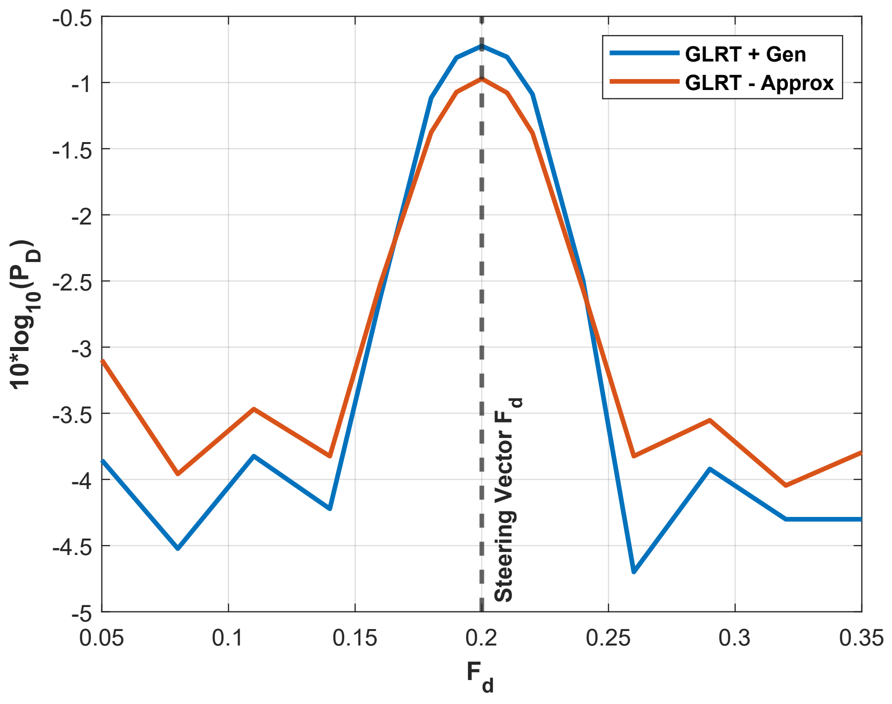
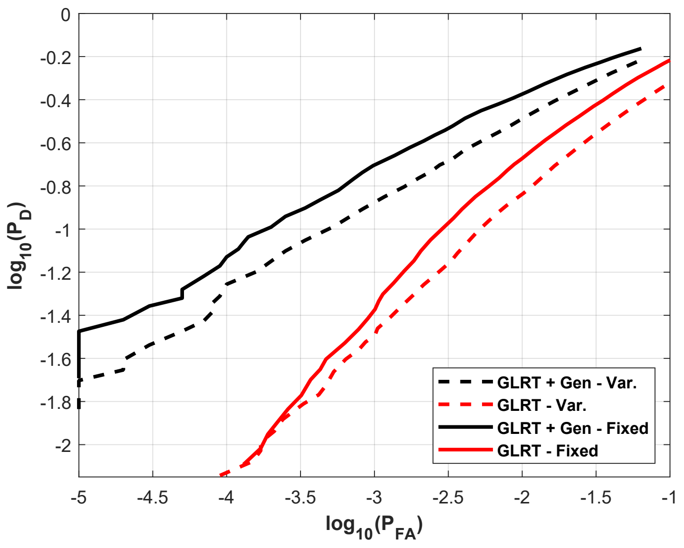


Figure 7.8: P_D vs normalized f_d at SIR = 10dB and steering vector $f_d = 0.2$.

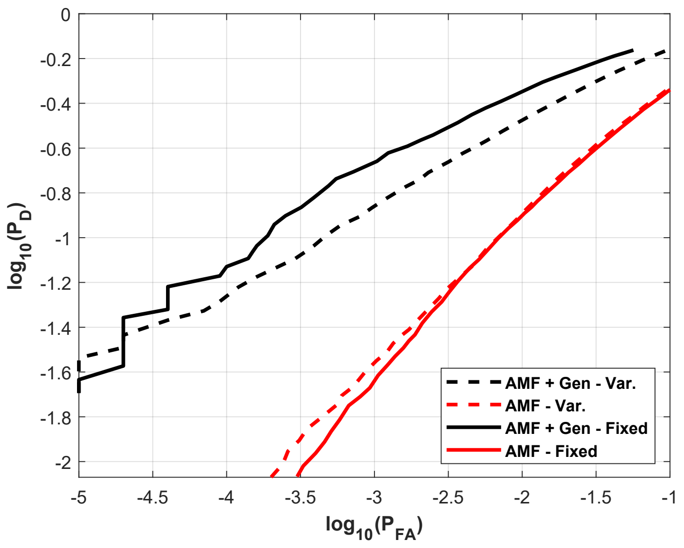
Up to this point, testing had been conducted to determine the accuracy and consistency of the prototype system's generated ICMs. However, it was also important to verify that the model could generalize to different covariance structures. I will note here that there is a strong theoretical basis for expecting that the U-Net generator could be trained to generalize to a broad range of CCMs. This is due to the fact that CCMs are highly structured, those structures are defined by correlation information which is reflected in positional relationships in the clutter data used as inputs to the model, and U-Nets are specifically designed to identify such positional features in input data and construct highly structured outputs corresponding to those features. That basis, paired with the initial demonstration that the clutter samples contained sufficient information to train the generator model, gave me high confidence in the ability of this method to be broadly generalizable. However, the prototype was also trained on a tightly controlled and relatively simple subset of CCMs. To be fully generalizable it would require training on a much larger set of data with far more varied correlation parameters. To verify that this method had the potential to generalize, additional testing was performed to determine how well it could adapt to more complex covariance structures that it had never encountered.

For this testing, an initial covariance matrix was generated using (7.2) with $\rho = 0.5$. Then, certain values within the covariance matrix were set to higher values, adding higher covariance perturbations into the correlated interference. Two forms of perturbation were added: 1) fixed and 2) variable. For the fixed perturbations, the higher covariance values were set to static locations in the covariance matrix. These locations were $[i_1, j_1] = [7, 13]$ and $[i_2, j_2] = [j_1, i_1]$ and the value of the covariance at this position was randomly set to be between $0.1 - 0.4$ for each sample set (increased from 0.0156 without perturbation). In the variable case, perturbations were placed randomly with i_1 ranging between $13 - 15$ and j_1 between $5 - 7$. Again $[i_2, j_2] = [j_1, i_1]$ and the value of the covariance at this position was randomly set to be between $0.1 - 0.4$. Note that in each case, care was taken to maintain the Hermitian nature of the matrix. Data generated with these interference covariance matrices were then run through the GLRT with SIR of 3dB and ROC curves were generated and plotted in Figure 7.9. In both cases, the GLRT showed modest performance increase when using the generated covariance structures. This provided evidence supporting my expectation that the model could be trained to effectively generalize to a wide range of CCM structures.

This prototype implementation of my proposed generative ML-enhanced ICM approximation method demonstrated a number of key findings. First, it proved that there is sufficient information in a set of raw clutter samples for a ML encoder to identify and learn a set of latent features corresponding to clutter correlation patterns. Further, it demonstrated that this could be accomplished with as little as $K = N$ support samples. Second, this testing proved that a ML decoder could be trained to generate high fidelity approximations of the ideal ICMs from the latent features learned by the encoder. Third, I was able to demonstrate that the behavior of the ICM generator model was consistent enough to be directly integrated with the well-known GLRT adaptive detection algorithm without negatively impacting its performance. Finally, I was able to determine that the the generative model was flexible enough to adapt to new CCM structures it had never seen before, providing a strong indication of its generalizability. However, it also highlighted several weaknesses with the initial design, and considerable work needs to be done to refine this method and determine how generally applicable it is to the adaptive filtering domain. In particular, while the U-Net appears to be ideal for use in this application, the basic GAN structure is fragile and will likely interfere with training the generator on larger, more general datasets. Additionally, the current implementation does not include any loss function incentives to encourage the model to produce invertible matrices, potentially impacting the quality of any filter that uses its generated ICMs. Finally, considerable work still needed to be done to verify that the proposed method could generalize enough for use on real data. In an attempt to improve this method and perform a more robust characterization of it, I made several major modifications to the prototype system design and continued this line of research. I will now present those updates and some expanded testing used to evaluate them.



(a) GLRT ROC curves.



(b) AMF ROC curves.

Figure 7.9: ROC curves for $M_{ij} = \rho^{|i-j|}$ and $\rho = 0.5$ with perturbations.

7.6 Updated Implementation

After successfully proving out the initial concept on simple, real-valued covariance structures, I redesigned the system for use in more realistic scenarios with complex-valued CCMs. There were three primary modifications: 1) the data model used to simulate training and testing data was updated, 2) the GAN structure was replaced with a generative U-Net, and 3) custom loss functions were developed for model training. Updates to the data generation models allowed me to simulate clutter backgrounds that were complex-valued and contained more challenging CCM structures associated with Doppler components in the background. The GAN structure proved to work well for the simple scenarios initially considered, but didn't generalize well. This is largely because it limited my ability to control the learning process and encourage the model to learn more complex ICM structures. The custom loss functions were developed to ensure that the model could both generalize well and to encourage it to learn to generate ICMs that could be inverted stably. I will now describe each update in detail before analyzing the performance impacts.

7.6.1 System Updates

7.6.1.1 Clutter Model

As previously noted, the clutter model was updated to a more challenging and realistic one. Specifically, the model for the CCM was updated so that it was complex and modeled Doppler components in clutter. This was accomplished using the Nitzberg CCM model [171]. All training and testing data for this phase of development was simulated in Matlab assuming a correlated Gaussian distribution with zero mean and covariance defined by Equations (7.3)-(7.5).

7.6.1.2 ML Architecture

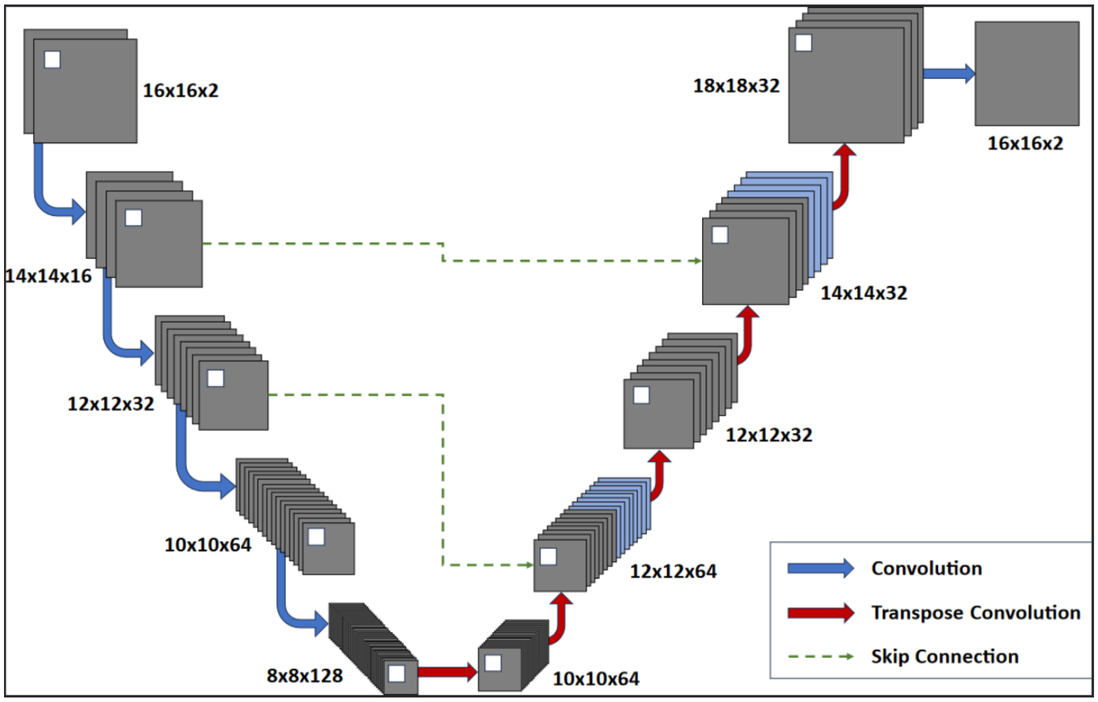


Figure 7.10: U-Net architecture used for ICM generation.

One of the key observations I made when testing the initial implementation of the ICM generator was that the GAN structure inhibited the ability of the network to generalize effectively in more complex clutter scenarios. The key reason for this is related to the structure of the loss functions in a GAN, specifically that the loss used to train the generator depends heavily on the ability of the discriminator to distinguish between real and generated ICMs. This dependency required that the two structures be trained in tandem and that they maintain approximate parity with one another. If either one trained too quickly, the training process would break down in an unrecoverable way and learning would cease. This meant that the discriminator was a critical failure point during training, which had a substantial impact as the complexity of the possible solution space increased. Also, it restricted the ability to consistently apply custom loss functions to encourage specific behaviors.

Table 7.3: U-Net Architecture

	Type	**Kernel Num.**	**Kernel Shape**	**Kernel Stride**	**Input Shape**	**Output Shape**	**Batch Norm.**
1	Conv.	16	3×3	1	$16{\times}16{\times}2$	$14{\times}14{\times}16$	N/A
2	Conv.	32	3×3	1	$14{\times}14{\times}16$	$12{\times}12{\times}32$	0.8
3	Conv.	64	3×3	1	$12{\times}12{\times}32$	$10{\times}10{\times}64$	0.8
4	Conv.	128	3×3	1	$10{\times}10{\times}64$	$8{\times}8{\times}128$	0.8
5	Tr. Conv.	32	3×3	1	$8{\times}8{\times}128$	$10{\times}10{\times}64$	0.8
6	Tr. Conv.	32	3×3	1	$10{\times}10{\times}64$	$12{\times}12{\times}32$	0.8
7	Skip	N/A	N/A	N/A	$12{\times}12{\times}32$	$12{\times}12{\times}64$	N/A
8	Tr. Conv.	32	3×3	1	$12{\times}12{\times}64$	$14{\times}14{\times}32$	0.8
9	Tr. Conv.	32	2×2	1	$14{\times}14{\times}32$	$14{\times}14{\times}16$	0.8
10	Skip	N/A	N/A	N/A	$14{\times}14{\times}16$	$14{\times}14{\times}32$	N/A
11	Tr. Conv.	32	5×5	1	$14{\times}14{\times}32$	$18{\times}18{\times}32$	0.8
12	Conv.	1	3×3	1	$18{\times}18{\times}32$	$16{\times}16{\times}2$	0.8

Fortunately, the core benefits of the original design for the ICM generator were due to the use of a U-net for the generator design, rather than the GAN structure itself. As such, it was a relatively simple matter to remove the discriminator and implement a supervised training method with custom loss functions. Minor modifications were also made to the generator network itself in order to allow for the generation of complex-valued ICMs. The updated ICM generator design is detailed in Table 7.3 and shown in Figure 7.10.

7.6.1.3 Loss Functions

Two custom loss functions were designed and used to train the U-net architecture. Both of these loss functions were based on the Frobenius distances between the generated matrices and the ideal ones. For the first loss function, the average of the Frobenius distances for three sub-regions was used. These three sub-regions were the primary diagonal, each of the first off-diagonals, and the residual regions of the matrix. The sub-regions were selected due to their unique characteristics within the ICM structure. I theorized that this would help teach the model to generate more consistent and realistic models. I will refer to this loss function as the one-stage loss function. It can be represented as:

$$L_1 = \frac{F_{diag} + F_{off} + F_{res}}{3} \tag{7.7}$$

where L_1 is the one-stage loss value, F_{diag} is the Frobenius distance along the primary diagonal, F_{off} is the Frobenius distance along the first off-diagonals, and F_{res} is the Frobenius distance in the residual regions. The distances for each sub-region was found by first masking the matrix to set every entry outside of the target region to zero and then calculating the distance value.

The second custom loss function that I developed is a two-stage loss function. In the first stage, this loss is calculated identically to Equation 7.7. Then, after T_1 training steps, a secondary loss component is introduced. To calculate the secondary loss, the output matrix from the U-net is first inverted. Then the same set of Frobenius distances used in L_1 are calculated for the inverted matrix and averaged. The average distance is clipped to ensure that poor matrix inversions do not trigger errors or push the model weights out of a stable training region. Once the second loss term is introduced, it is multiplied by an inverse scaling factor that decays over time. This results in that term having a suppressed impact on loss early in training that gradually grows over time. Note that after T_2 training steps, the inverse scale factor is set to a constant value. The first stage of this loss function is intended to allow the U-net to learn how to generate realistic ICM structures with the second stage intended to improve the generator's outputs by making them more stable. The two-stage loss function can be represented as:

$$L_2 = \begin{cases} L_1, & t \leq T_1 \\ L_1 + Min\left[a, b\left(\frac{F^{inv}_{diag} + F^{inv}_{off} + F^{inv}_{res}}{3}\right)\right], & t > T_1 \end{cases} \tag{7.8}$$

$$b = Min\left(\frac{1}{\beta}, \frac{t}{\beta T_2}\right) \tag{7.9}$$

where L_2 is the two-stage loss value, $Min[\bullet]$ is a function that returns the minimum of a set of values, t refers to the training step, T_1 is the training step at which the secondary loss component is added, T_2 is the training step at which the inverse scaling factor for the secondary loss component is minimized, β is a scaling factor, F^{inv}_{diag} is the Frobenius distance along the primary diagonal for the inverted matrix, F^{inv}_{off} is the Frobenius distance along the first off-diagonals for the inverted matrix, and F^{inv}_{res} is the Frobenius distance in the residual regions for the inverted matrix. For this loss function, the values of a, β, T_1 and T_2 that optimize training will be heavily dependent on the size of the output matrices being generated and the model's hyper-parameters. They can be treated as additional hyper-parameters that can be tuned during the training process. In the work presented here, they were set to ensure that the secondary loss term ranged between 0 to 0.5. I will note here that the use of T_1 and T_2 for determining when to introduce secondary loss term and when to stop suppressing its effects are not optimal. These terms should be replaced with performance-based thresholds, rather than time based ones. This will make the loss function for effective and more generalizable.

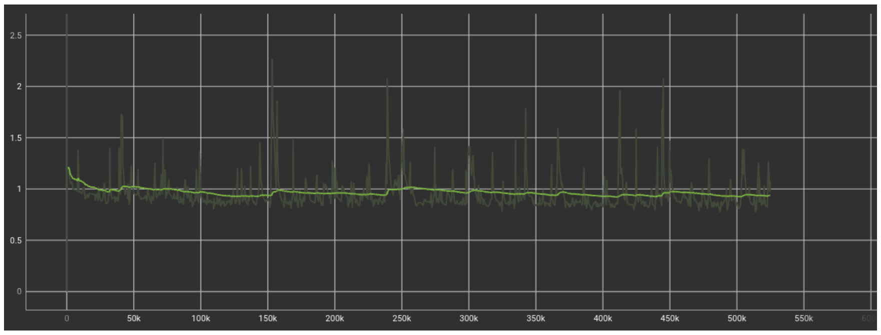
7.6.2 Testing and Results

As with the initial ICM generator, the goal of testing this updated implementation was to demonstrate that it could learn to produce ICMs from limited support samples accurately enough that they could be used within adaptive detection algorithms with minimal performance degradation. Full testing is still underway at the time writing this dissertation, but my initial findings are extremely promising and will presented here. At this stage, I have successfully made the system updates described in Chapter 7.6.1 and trained the U-Net structure on correlated Gaussian data with complex covariance structures using each of the two new custom loss functions. I have also tested models trained with each loss function to determine the quality of the ICMs they generate. However, I have not yet directly integrated the updated generator models with an adaptive detector to determine their impact on detection performance.

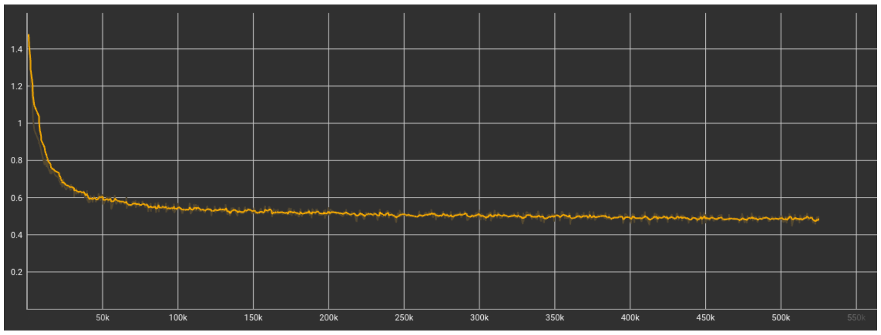
The training of the updated models was conducted similarly to the initial implementation. Each U-Net was trained to produce $N \times N \times 2$ ICMs based on $K = N$ input interference samples of length $N = 16$. Note that the number of output channels for the generator model was increased from one to two in this implementation. One channel is now trained to produce the real component of the ICM, while the other is trained to produce the imaginary component. Training data was generated in Matlab as described in Chapter 7.6.1 with $\sigma_{cl} = 1$, four distinct values of ρ_l ranging from 0.1 to 0.9, and eleven values of f_d ranging from 0 to 0.5. This resulted in 44 distinct scenarios used to train the generators. Multiple models were trained on each of the two loss functions over 100 epochs. Model convergence was achieved consistently after some slight hyper-parameter turning, largely thanks to the shift toward supervised training.

Examples of the training and validation curves for the U-Net are shown in Figures 7.11 and 7.12. Figures 7.11a and 7.12a show these curves for a model that failed to converge, Figures 7.11b and 7.12b show them for a model that successfully converged using the one-stage loss function, and Figures 7.11c and 7.12c show them for a model that successfully converged using the two-stage loss function. Note that the average loss remains largely constant for the model that failed to converge, while for the others the loss steadily decreases before reaching a stable convergence near 0.5. Observe that the training curves for each loss function are initially indistinguishable but the curves for the model trained with the two-stage loss function eventually display much larger loss spikes than those observed in the one-stage loss model. This is indicative of the introduction and subsequently growing impact of the secondary loss term. It should also be noted that the validation and training curves ultimately converge to roughly the same value. This indicates that the models trained successfully without over-fitting on the training data. Avoiding over-fitting is necessary to train a model that can generalize and be used in real systems. Two models (one trained with each loss function) were selected for further testing based on a combination of their loss values and observation of their outputs. Figure 7.13 depicts example model outputs for both loss functions generated during training.

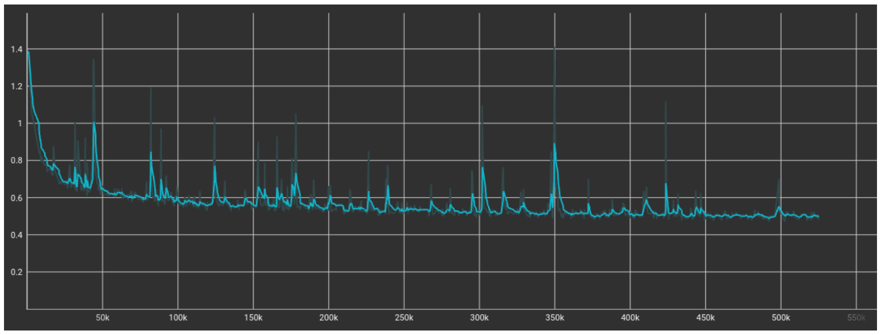
The quality of the ICMs generated by the trained models has been evaluated using the same method used for the initial GAN implementation (as presented in Chapter 7.5). To perform this analysis, $88,000$ test samples were generated with 88 unique test cases. This was done using the method described in Chapter 7.6.1 with $\sigma_{cl} = 1$, eight distinct values of ρ_l ranging from 0.1 to 0.9, and eleven values of f_d ranging from 0 to 0.5. The test cases used for analysis include a broad range of CCM configurations, half of which the generator model never saw during training. This was done to evaluate



(a) Example training curve for an unconverged model.



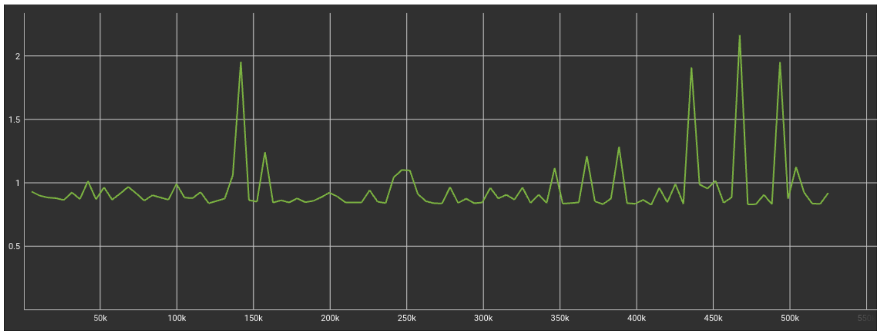
(b) Example training curve for a converged model with the 1-stage loss function.



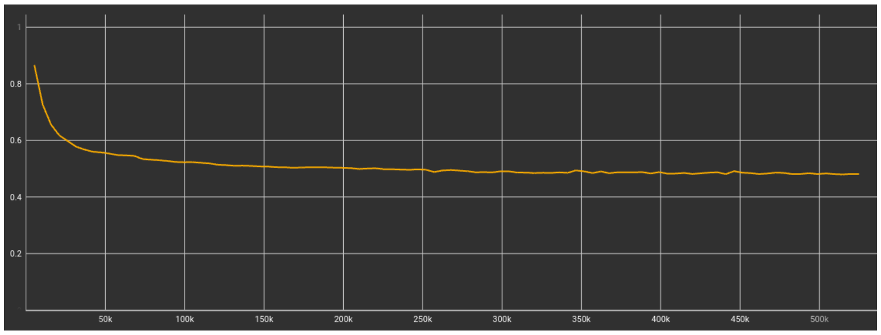
(c) Example training curve for a converged model with the 2-stage loss function.

Figure 7.11: Example U-Net training curves.

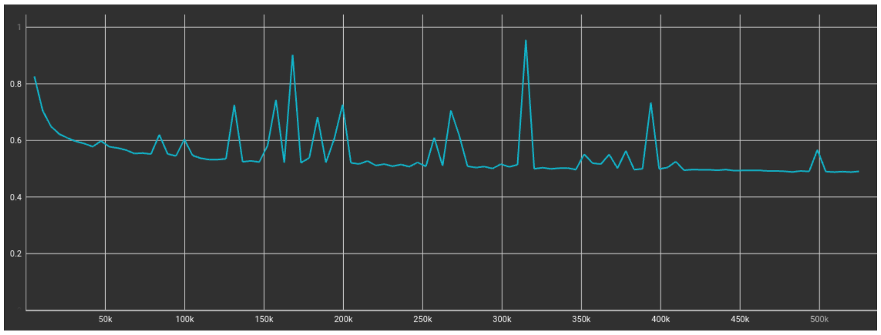
how well the generator models could generalize. Each of the $88,000$ test samples were run through the two generator models and the output ICMs were collected for analysis. In addition, the generated ICMs were each inverted and analyzed to determine the stability and quality of the generator outputs. At the same time, the classic CCM



(a) Example validation curve for an unconverged model.



(b) Example validation curve for a converged model with the 1-stage loss function.

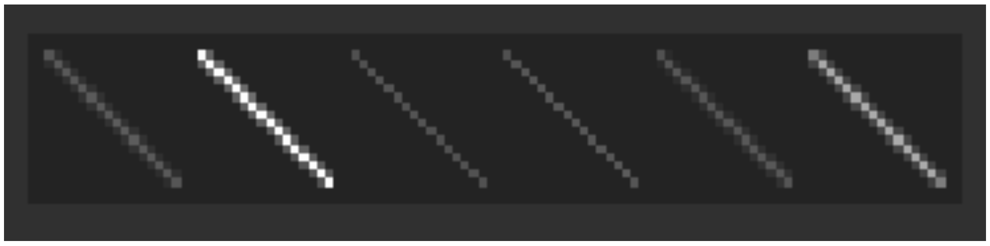


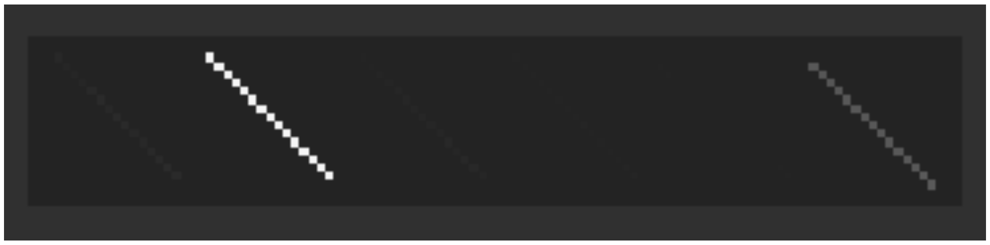
(c) Example validation curve for a converged model with the 2-stage loss function.

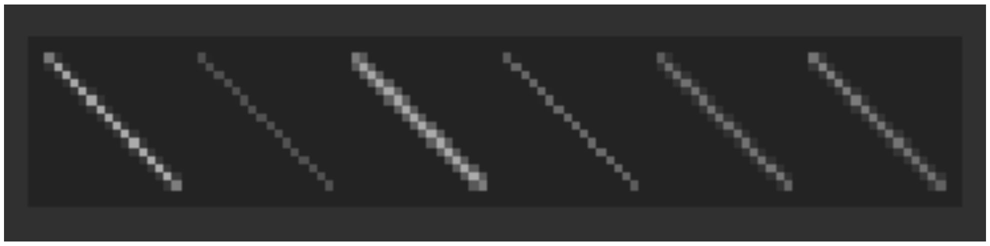
Figure 7.12: Example U-Net validation curves.

estimation method provided by Equation (3.61) (described in Chapter 3.5) was used to generate approximations of the ICMs and CCMs for comparison.

The same four Frobenius distance calculations described in Chapter 7.5 were used to evaluate the quality of the generated ICMs and their inversions. These included the


(a) Example U-Net real outputs for converged model with 1-stage loss function.


(b) Example U-Net imaginary outputs for converged model with 1-stage loss function.


(c) Example U-Net real outputs for converged model with 2-stage loss function.

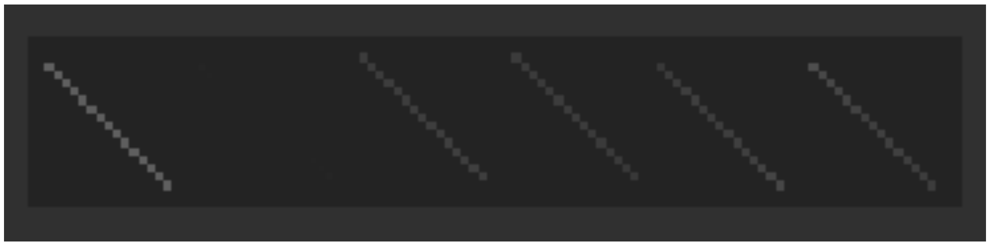

(d) Example U-Net imaginary outputs for converged model with 2-stage loss function.

Figure 7.13: Example U-Net outputs for each loss function.

Frobenius distances for the full matrices and three targeted sub-regions. Each distance was calculated with respect to the ideal CCM/ICM. Table 7.4 shows the average of these values for each model and the comparison estimation method. Additionally, Figure 7.14 shows the distribution of the Frobenius distances for the full matrices. As with the testing described in Chapter 7.5, each generative model was given $K = N$ support samples while the comparison CCM estimation method was given $K = 2N$.

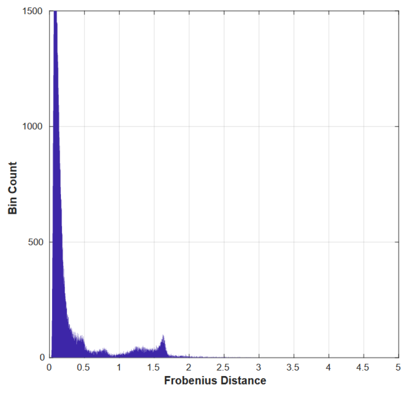
There are several key observations to be made here. First, the ICM Frobenius distance values were extremely low for both of the generative models evaluated here. They were each lower than the equivalent distances for both the ICM and CCM from the comparison method. More impressively, the generated matrices were all invertible and the Frobenius distances for the inversions were lower than those of the comparison method's ICM. This indicates that the error resulting from inverting the generated matrices is less than that of the classic estimation method. Additionally, the distribution of the distances for the generated matrices is tightly grouped at low values. Together, these observation provides a strong indication that this updated generative method consistently produces high quality approximations of the ICM that capture the phase information needed to decorrelate samples. This is further demonstrated in Figures 7.15-7.26. These figures show side-by-side heatmaps comparing the ideal and generated ICMs and their inversions from six test cases with different CCM structures. It is clear from these images that the proposed method is capable of producing a broad range of ICMs associated with complex CCM structures and that those complex structures are accurately reproduced by inverting the generated ICMs.

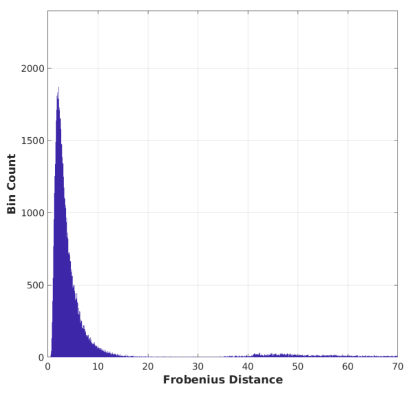
It should also be observed that there was very little performance difference between the generative model trained with the one-stage loss function and the one trained with the two-stage loss. In fact, the model trained with the one-stage loss function performed slightly better. This was surprising, as I expected the inclusion of the inversion loss to improve the overall quality of the generated ICMs. However, it appears that the one-stage loss was sufficient to learn the necessary features, at least for the selected clutter covariance model. In the future I would like to compare the two against real-world data to see how well they each generalize in that scenario. Additionally, I would like to explore whether tuning the parameters of the two-stage loss function can further improve the model's performance.

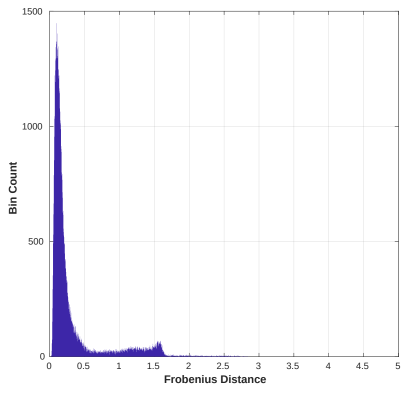
Metric	**Unet -** $K = N$ **One Stage Loss**	**Unet -** $K = N$ **Two Stage Loss**	**Est. -** $K = 2N$
ICM FD Full	0.3241	0.3260	28.2474
ICM FD Diag.	0.2004	0.1998	13.7057
ICM FD Off Diag.	0.2353	0.2384	11.5019
ICM FD Res.	0.0144	0.0137	21.7815
CCM FD Full	13.2781	17.1628	2.7966
CCM FD Diag.	4.3218	5.5523	0.6878
CCM FD Off Diag.	6.5119	8.2996	0.9474
CCM FD Res.	2.7900	13.5087	2.5319

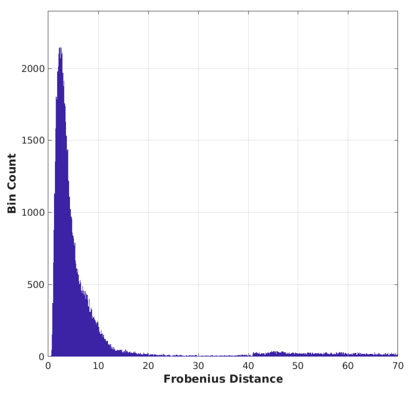
Table 7.4: F_{dist} comparison - Unet versus Estimation

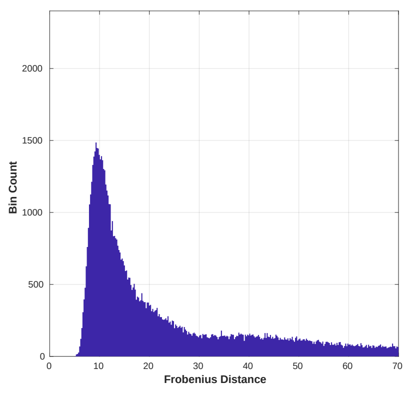
The initial results presented here are extremely positive. It appears as though the U-Net structure learned the correlation information well and was successfully adapted to generate complex ICMs with very little sample support. It will be crucial to integrate this variation of the proposed generative ICM estimation method with an adaptive detection algorithm and fully characterize its performance. However, given the success of the initial implementation when integrated with the GLRT and the superior performance demonstrated by the updated generator, it is reasonable to expect that this modification with provide considerable performance improvements to adaptive detectors reliant on matrix approximation and inversion techniques. Further testing will also need to be conducted on high-fidelity simulations and real-world data.


(a) ICM - One-Stage Loss.


(b) CCM - One-Stage Loss.


(c) ICM - Two-Stage Loss.


(d) CCM - Two-Stage Loss.


(e) ICM - Estimated.

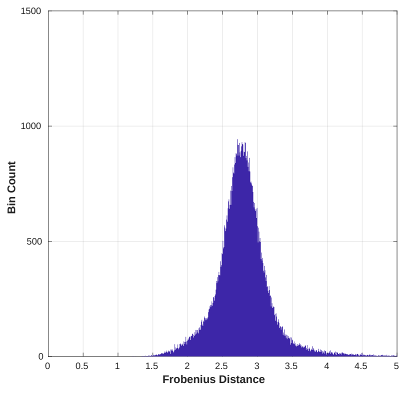
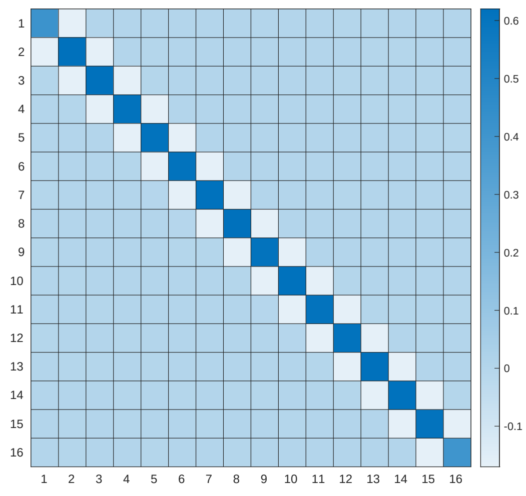
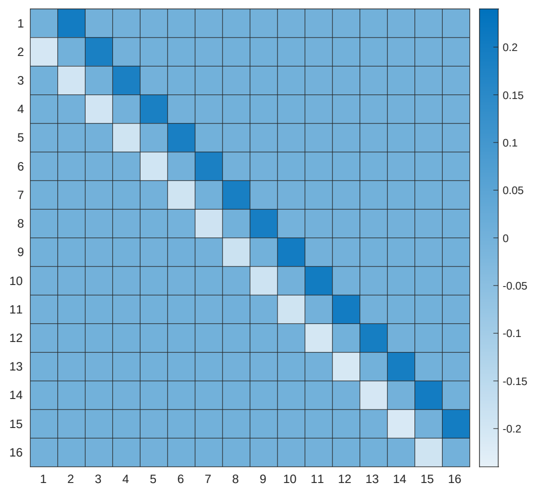
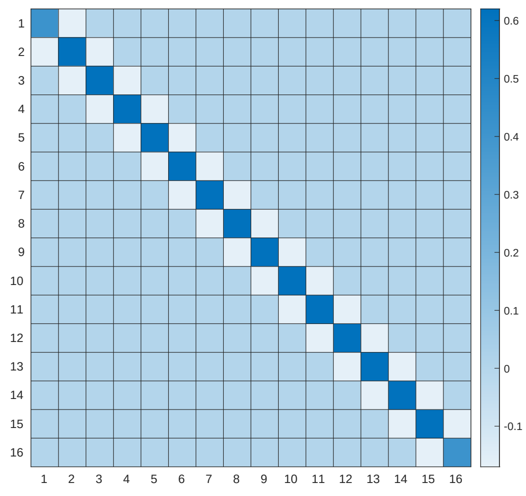

(f) CCM - Estimated.

Figure 7.14: F_{dist} distribution - generated with K=N versus estimated with K=2N.


(a) Real component of generated ICM.


(b) Imaginary component of generated ICM.


(c) Real component of true ICM.

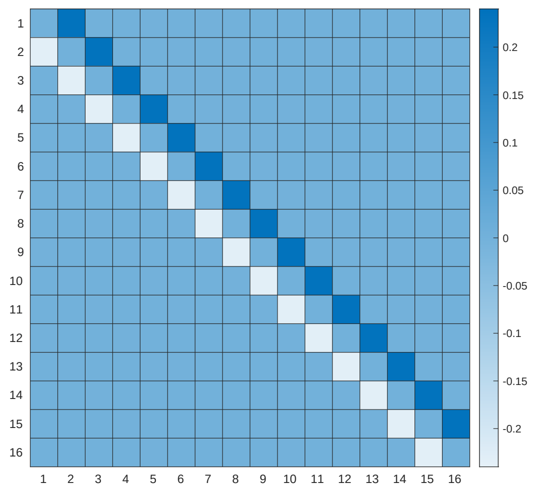
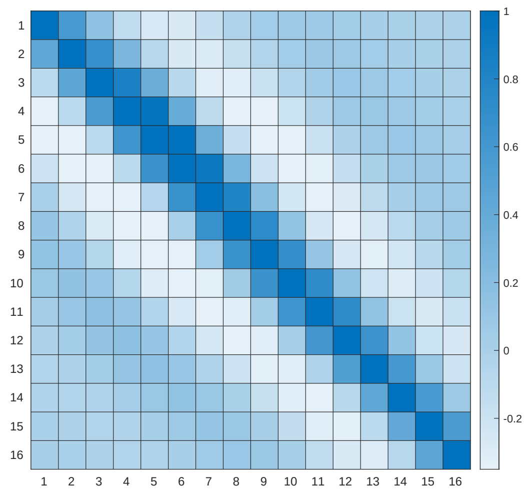
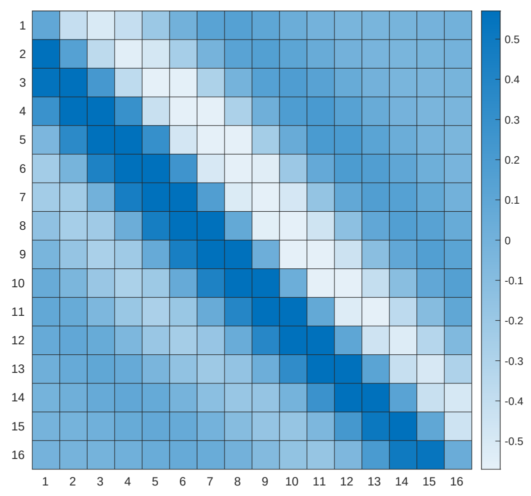

(d) Imaginary component of true ICM.

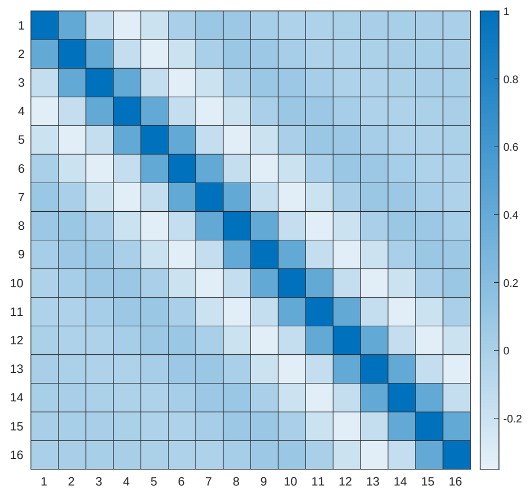
Figure 7.15: Test Case 1 - Generated versus True ICM.



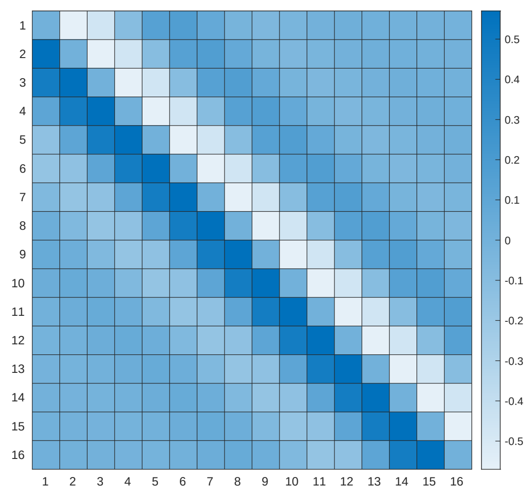
(a) Real component of generated CCM.



(b) Imaginary component of generated CCM.

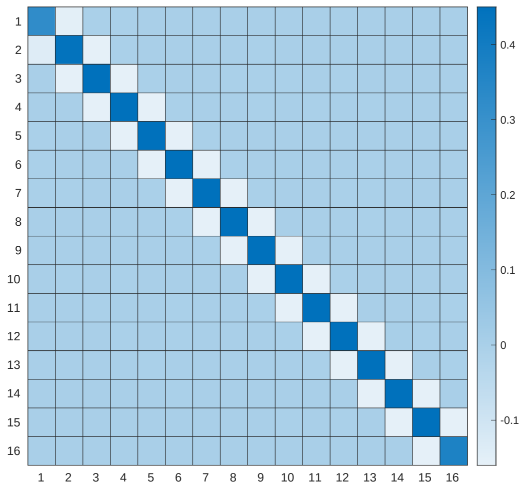


(c) Real component of true CCM.

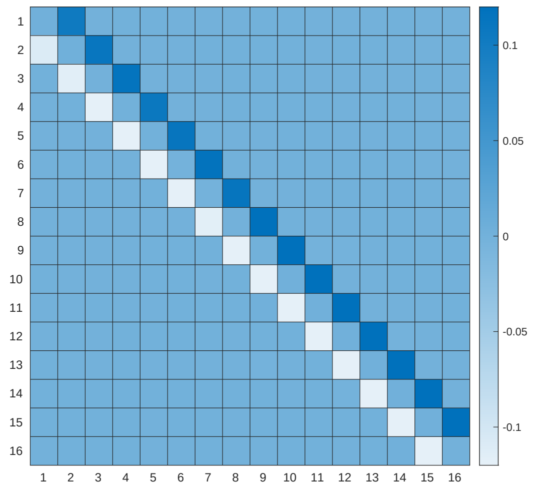


(d) Imaginary component of true CCM.

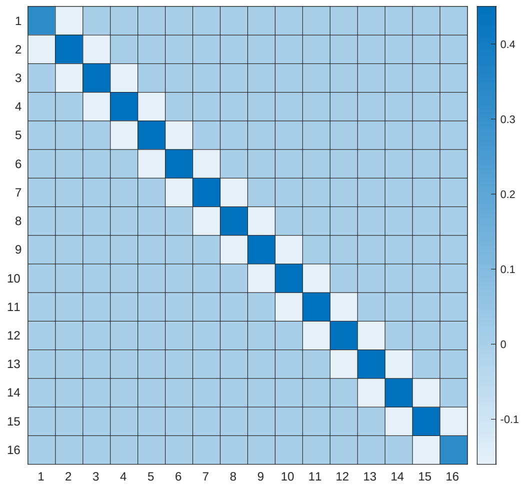
Figure 7.16: Test Case 1 - Generated versus True CCM.



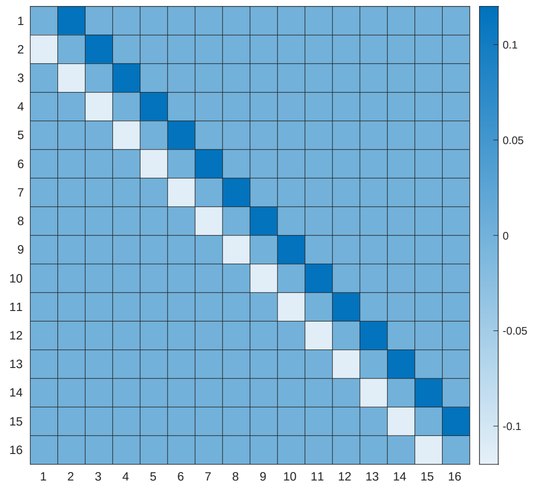
(a) Real component of generated ICM.



(b) Imaginary component of generated ICM.

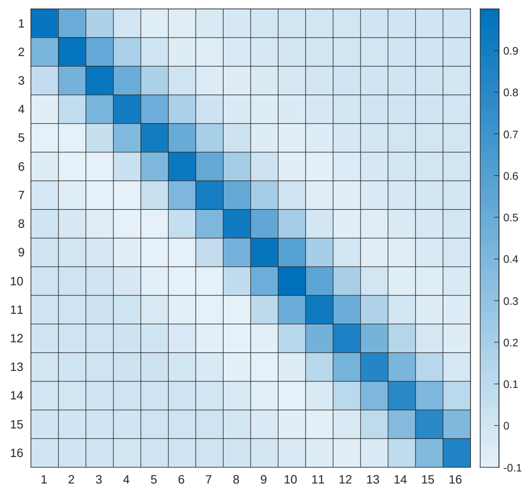


(c) Real component of true ICM.

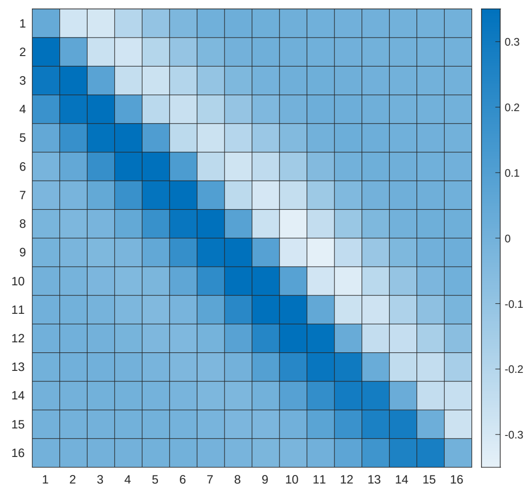


(d) Imaginary component of true ICM.

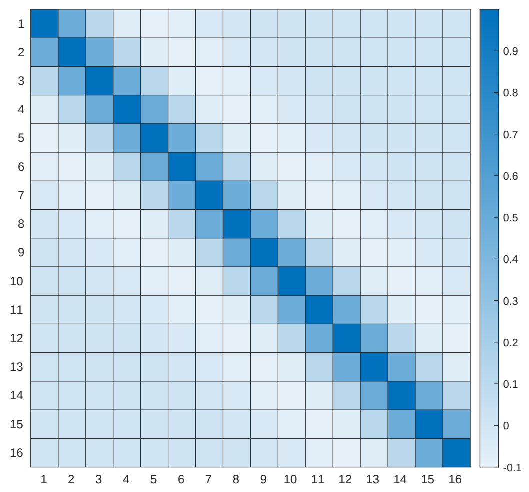
Figure 7.17: Test Case 2 - Generated versus True ICM.



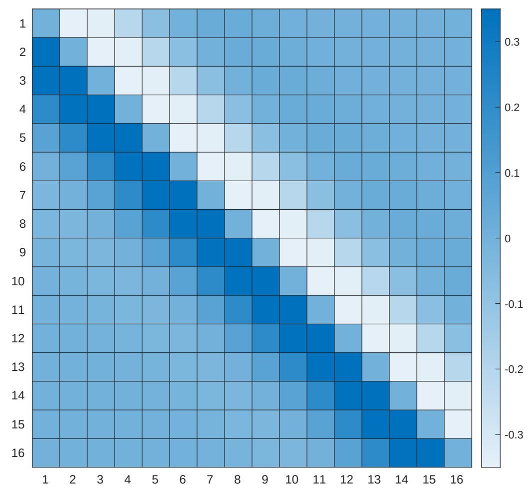
(a) Real component of generated CCM.



(b) Imaginary component of generated CCM.

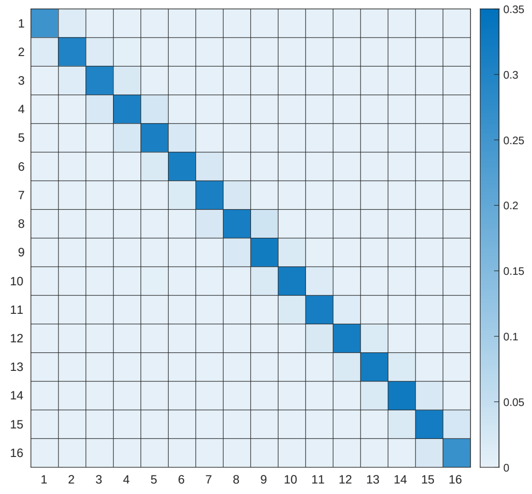


(c) Real component of true CCM.

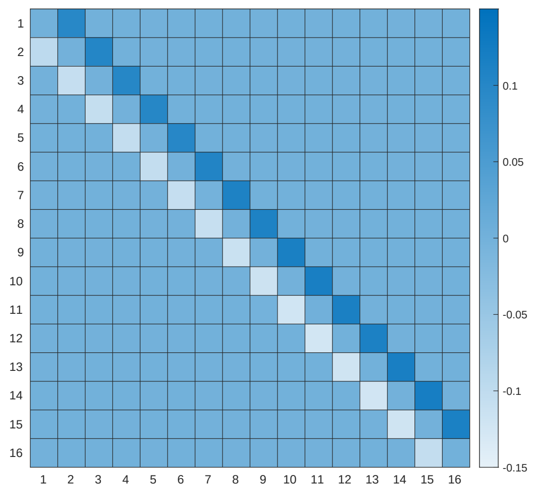


(d) Imaginary component of true CCM.

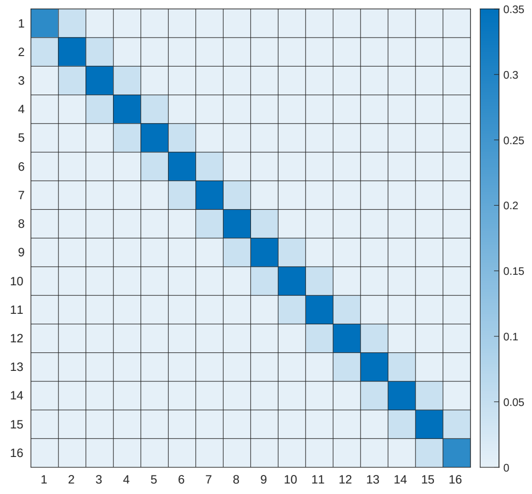
Figure 7.18: Test Case 2 - Generated versus True CCM.



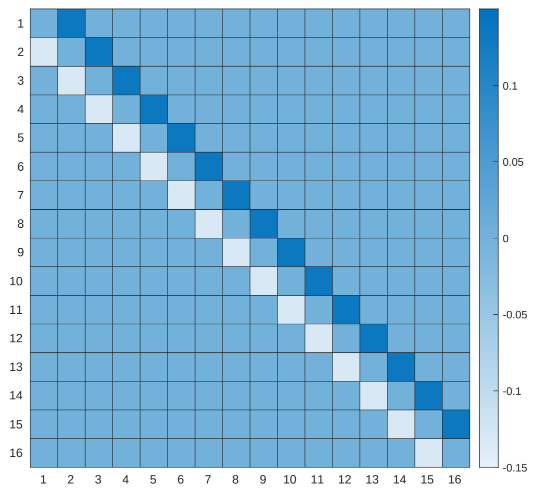
(a) Real component of generated ICM.



(b) Imaginary component of generated ICM.

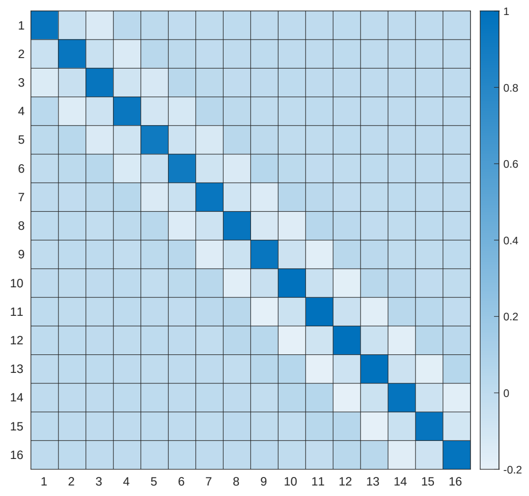


(c) Real component of true ICM.

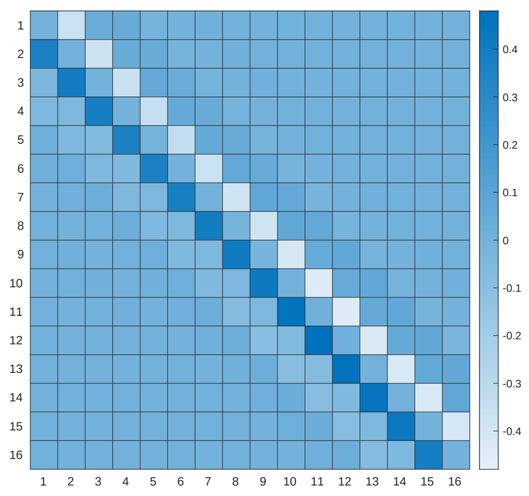


(d) Imaginary component of true ICM.

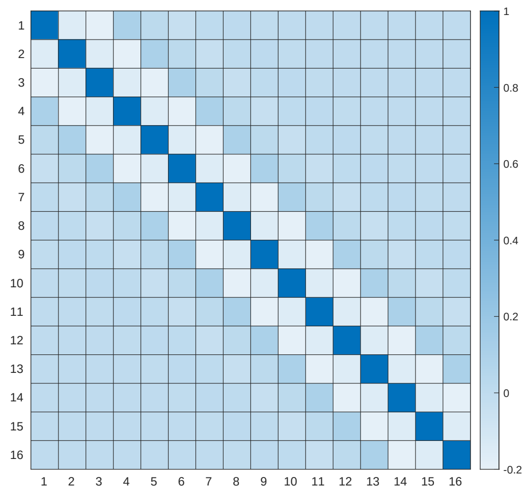
Figure 7.19: Test Case 3 - Generated versus True ICM.



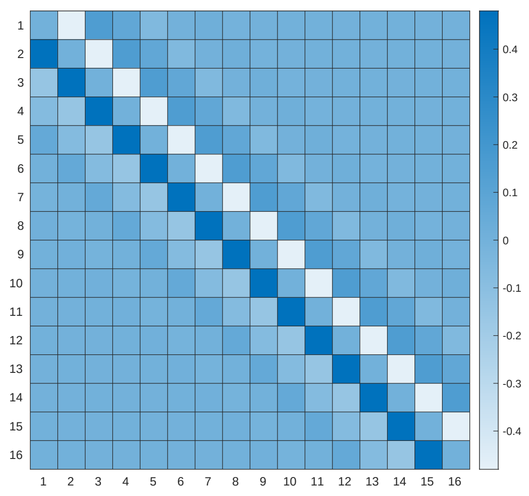
(a) Real component of generated CCM.



(b) Imaginary component of generated CCM.

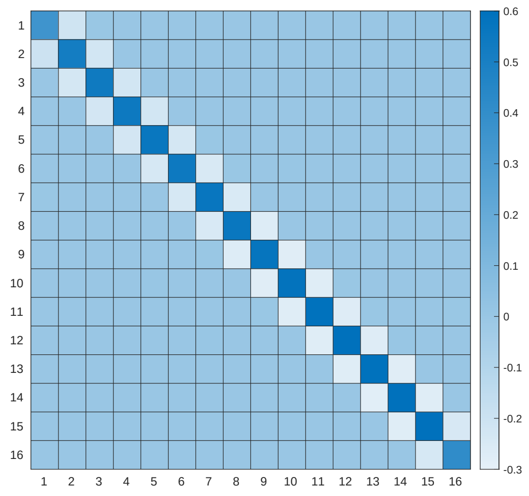


(c) Real component of true CCM.

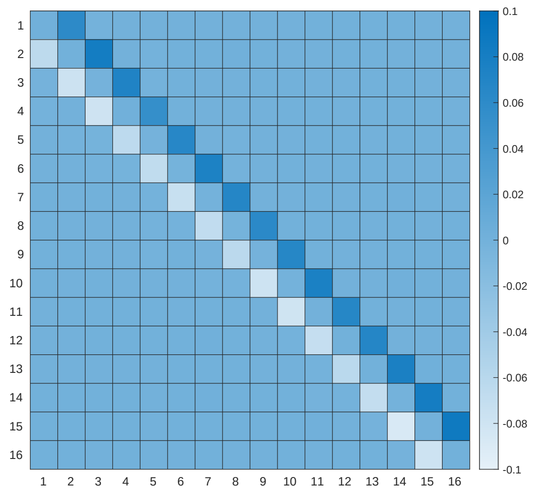


(d) Imaginary component of true CCM.

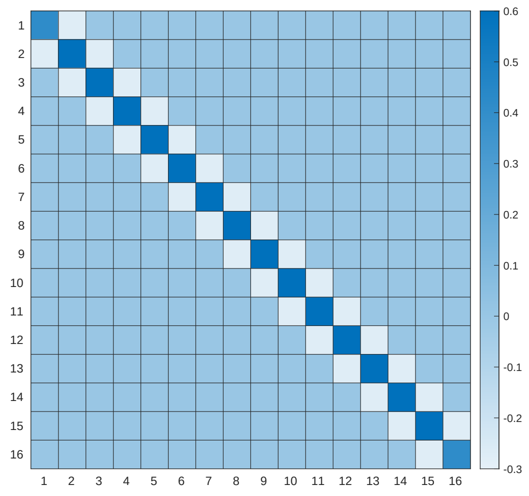
Figure 7.20: Test Case 3 - Generated versus True CCM.



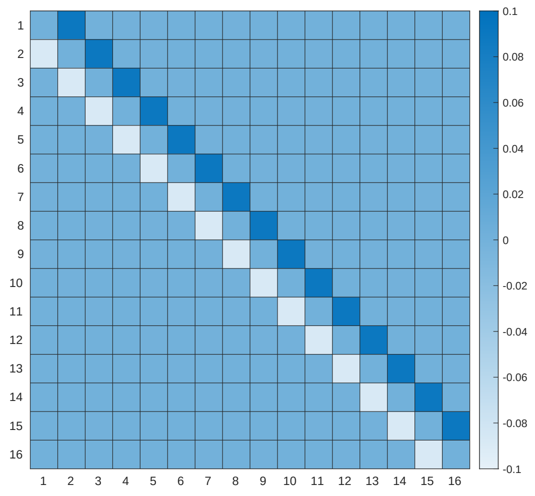
(a) Real component of generated ICM.



(b) Imaginary component of generated ICM.

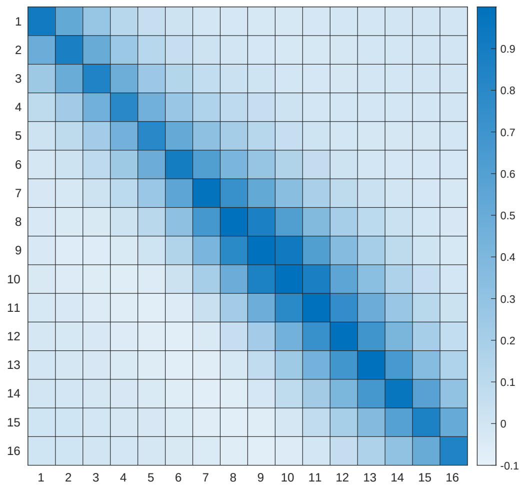


(c) Real component of true ICM.

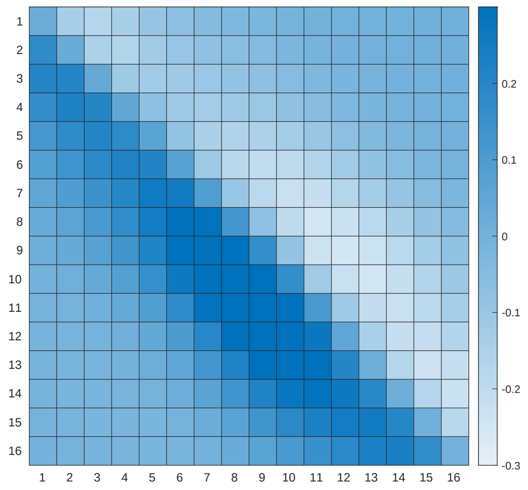


(d) Imaginary component of true ICM.

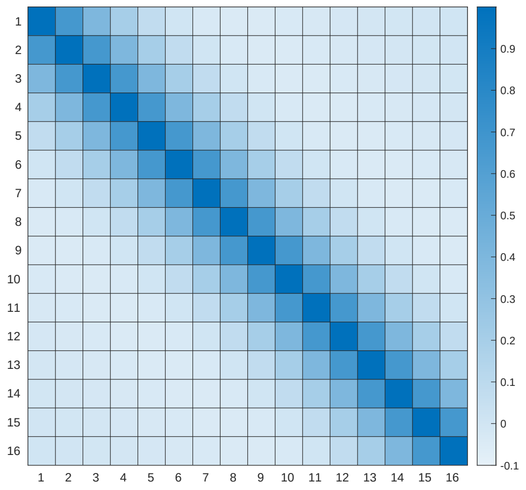
Figure 7.21: Test Case 4 - Generated versus True ICM.



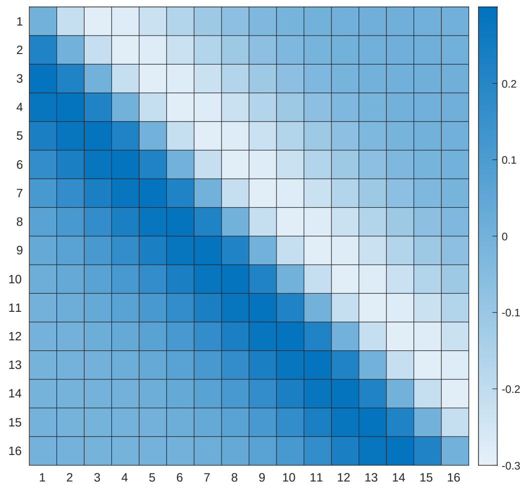
(a) Real component of generated CCM.



(b) Imaginary component of generated CCM.

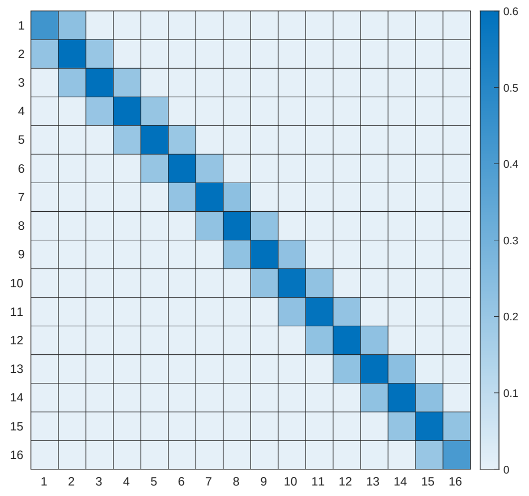


(c) Real component of true CCM.

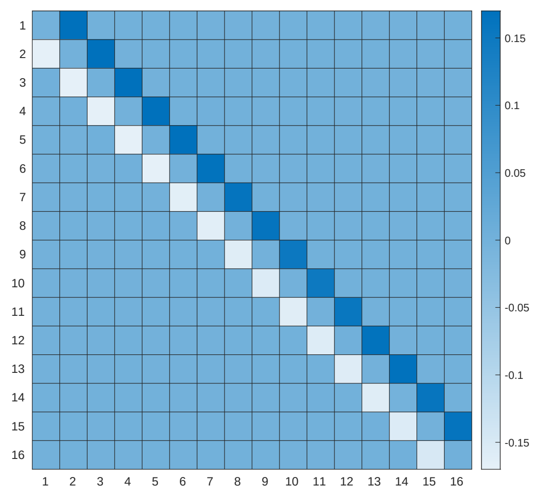


(d) Imaginary component of true CCM.

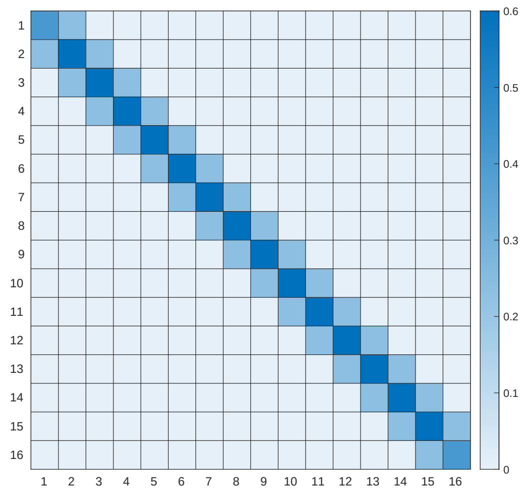
Figure 7.22: Test Case 4 - Generated versus True CCM.



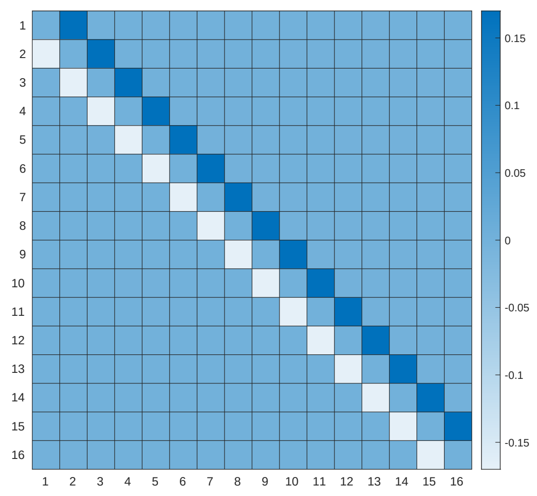
(a) Real component of generated ICM.



(b) Imaginary component of generated ICM.

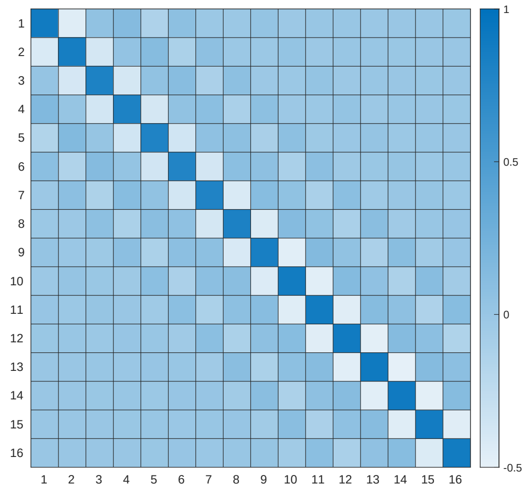


(c) Real component of true ICM.

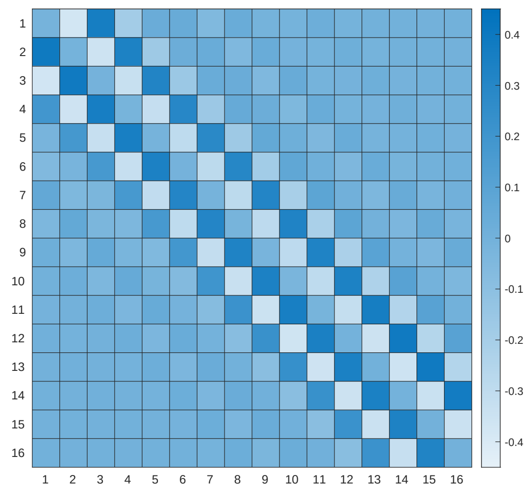


(d) Imaginary component of true ICM.

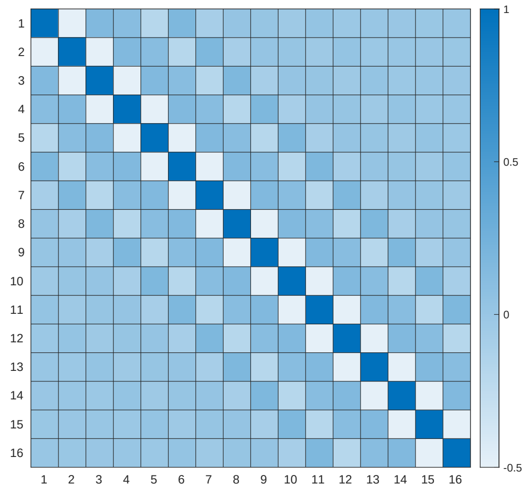
Figure 7.23: Test Case 5 - Generated versus True ICM.



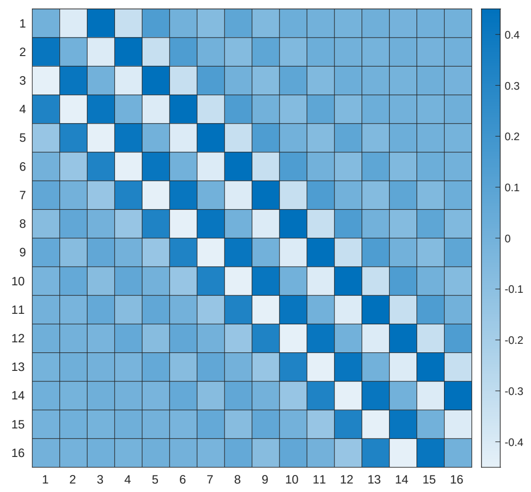
(a) Real component of generated CCM.



(b) Imaginary component of generated CCM.

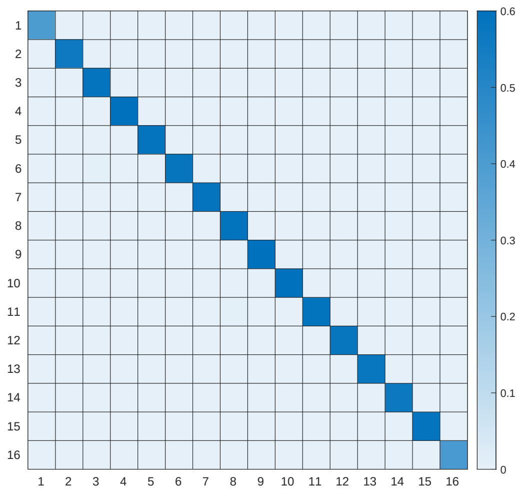


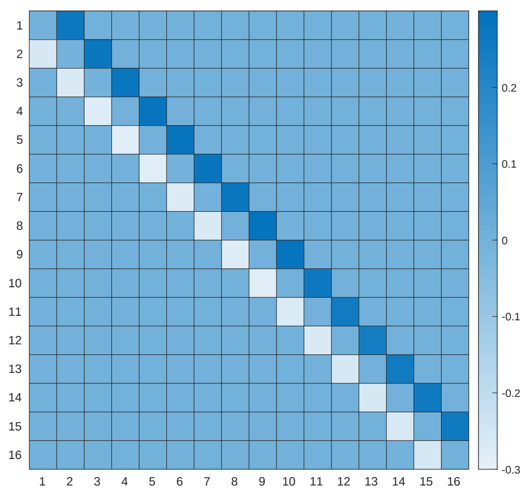
(c) Real component of true CCM.

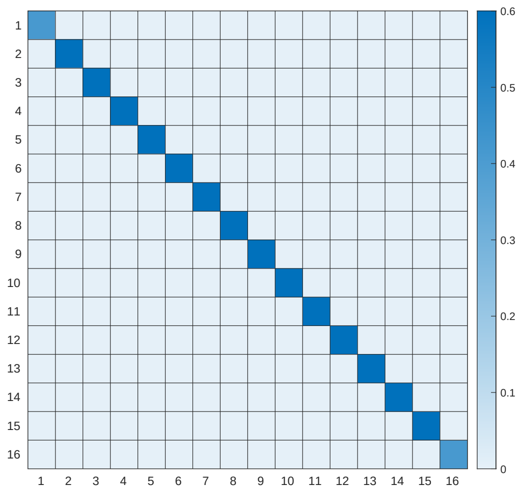


(d) Imaginary component of true CCM.

Figure 7.24: Test Case 5 - Generated versus True CCM.


(a) Real component of generated ICM.


(b) Imaginary component of generated ICM.


(c) Real component of true ICM.

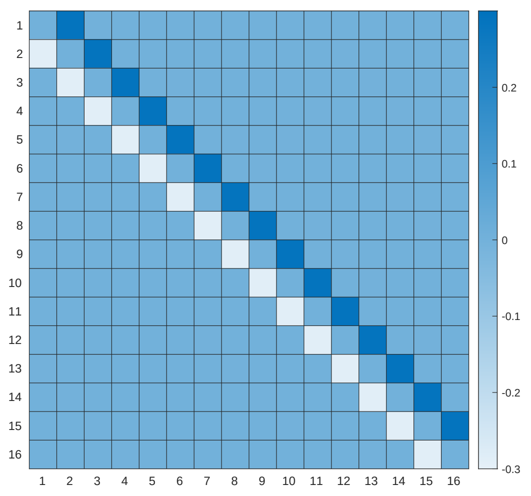
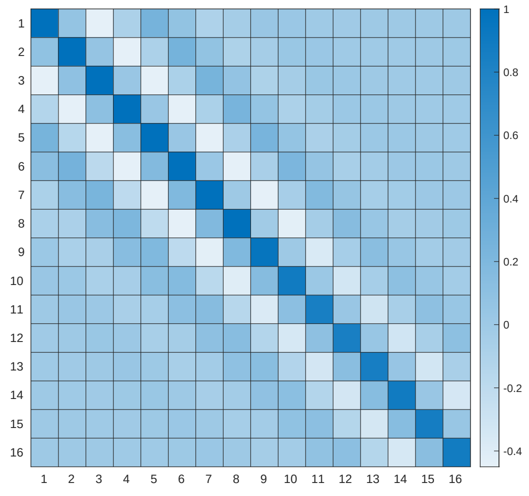
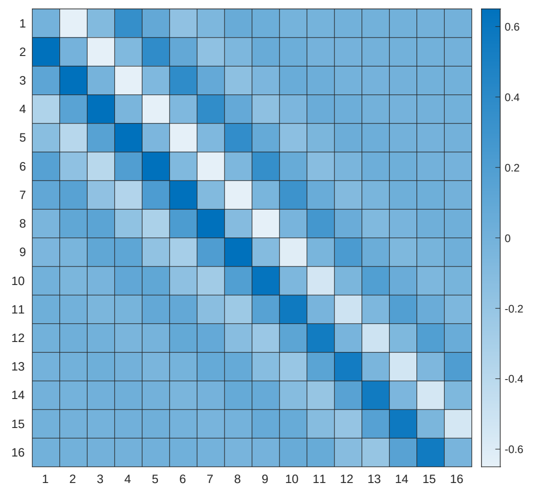
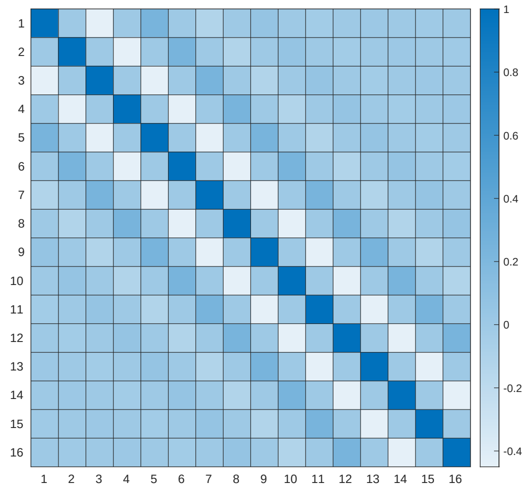

(d) Imaginary component of true ICM.

Figure 7.25: Test Case 6 - Generated versus True ICM.


(a) Real component of generated CCM.


(b) Imaginary component of generated CCM.


(c) Real component of true CCM.

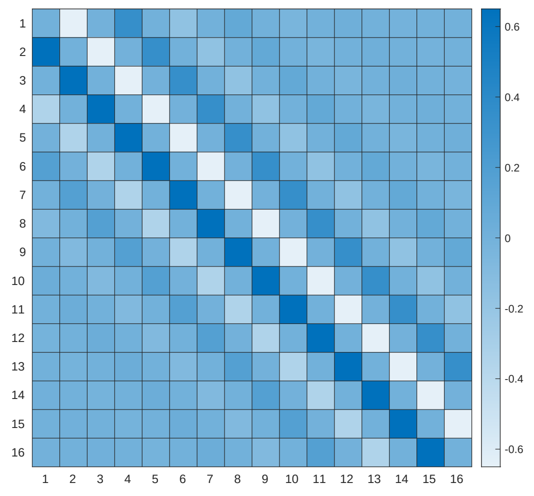

(d) Imaginary component of true CCM.

Figure 7.26: Test Case 6 - Generated versus True CCM.

7.7 Conclusion

In this chapter, I have presented an approach to directly creating adaptive whitening filters for use in detection algorithms based on generative ML. Adaptive detection algorithms rely heavily on covariance matrix approximation and inversion techniques to decorrelate interference signals in a SUT before applying a hypothesis test. Existing methods for approximating CCMs either rely on large quantities of support data or use assumptions about CCMs structures to reduce their rank. The latter risks discarding useful covariance information, while the former requires long sampling times and assumes clutter homogeneity. Estimated CCM must then be inverted to effectively decorrelate the interference present in a return signal. That inversion can be computationally costly, Additionally, poorly approximated CCMs may not be invertible. This dependence on sample support and CCM estimation is a major limitation for adaptive detection algorithms. The method presented here frames the problem as an image-to-image generation task and applies modern ML approaches to accomplish it.

In generative ML, image-to-image generation is performed using encoder-decoder style neural networks. The encoder can be used to identify the underlying statistical characteristics of an input image and map them to an intermediate vector space called a latent space. The decoder can then be trained to generate output images with specific structural elements associates with particular latent features. In the context of radar detection, I have argued that a matrix of support samples containing interference can be equated to an image whose underlying statistical characteristics include the covariance information of the clutter. Under this assumption, a generative model can be trained to identify covariance features in interference data and directly produce an approximation of the inverse clutter covariance matrix. Such a model could be trained to approximate these matrices with relatively little sample support and would remove the need for matrix inversion, drastically improving the performance of adaptive detection algorithms.

I initially explored neural network architectures including the VAE, transformer, and conditional GAN for generating adaptive filters from interference data. Of these, the conditional GAN showed the most promise. This architecture uses two networks trained together in an adversarial process. The first network, known as the generator, is trained to produce an output image from an input. The second model, called the discriminator, is presented with both real and generated images and tasked with identifying to which category a given image belongs. For a conditional GAN, the discriminator is given additional contextual information along with the images that allows it to determine if an image is realistic for the current scenario. This quickly forces the generator to learn to use context features (like covariance characteristics) when generating images.

My GAN design consisted of a U-Net for the generator and a CNN for the discriminator. The U-Net was selected because it performs well at the image-to-image translation task and is inherently symmetrical, allowing for different sizes of input and output matrices. A CNN was used for the discriminator because it is one of the most commonly used ML architectures for image classification problems. The support samples used as the input to the generator were also passed to the discriminator so that it could learn to identify if a generated ICM was consistent with the dataset's covariance features. I was able to successfully implement and train the GAN generator model to produce real-valued ICMs for a set of correlated, Gaussian distributed clutter samples. To evaluate the quality of the generator's outputs, $20,000$ test scenarios were run through the model with $K = N$ support samples. Frobenius distances were then calculated between the ideal ICM and the output of the generator. These were compared to a commonly used CCM estimation and inversion method using sample support values of $K = 2N$ and $K = 10n$. It was found that the average distance for the generated ICMs was equivalent to that of the comparison estimation method using $K = 10N$ support samples. This was an impressive performance give the low level of sample support provided to the generator.

In order to verify that the proposed ICM generator could be used in adaptive detection systems, the generator model was directly integrated with both the Gaussian GLRT and Gaussian AMF. These integrated systems were then tested to characterize the impact of the generator on each detector's performance. The ROC and clutter rejection performance of the detectors was evaluated both when using the proposed ICM generator with $K = N$ support samples and when using the comparison estimation method with $K = 2N$ support samples. In each test case, the ICM generator provided modest performance improvements over the classic estimation method while using half as many support samples.

These results were extremely promising and proved both that generative models could learn latent covariance features from limited support samples and that they could consistently produce ICM approximations accurate enough to be used in adaptive detection algorithms. However, the clutter covariance model used to generate the training and testing data was simplistic and did not model the phase information found in more realistic detection scenarios. Additionally, the adversarial training of the conditional GAN was found to be inconsistent and sensitive. As a result, it was unclear how well the proposed method would generalize.

To address this limitation, several modifications were made to the prototype design. First, the clutter covariance model was updated to include complex values and phase shifts resulting from localized clutter velocity components. The GAN structure was then discarded in favor of supervised training of the U-Net generator, which was modified slightly to produce complex-valued outputs. Finally, two custom loss functions were developed and used to train the generator. The first used only the Frobenius distances the model output and ideal ICM for three sub-regions within the matrices. The second loss function gradually introduced a secondary loss term over time that was based on Frobenius distances between the inversion of the model output and the ideal CCM.

I have successfully implemented these updates and conducted initial testing to evaluated the quality of the ICMs generated by models trained using each of the custom loss functions. Currently, testing has been limited to running test scenarios through the ICM generators with $K = N$ support samples and measuring the Frobenius distances between the generated and ideal ICMs. The test scenarios used for this testing included 88 different clutter ICM configurations, 44 of which the model had not encountered during training. These results were again evaluated against those of the comparison estimation method using $K = 2N$ support samples.

It was found that each of these generator models was able to consistently produce ICM approximations for every test case with lower average error than the comparison method. It was also found that the generated ICMs were universally invertible. Further, the Frobenius distances between the inversions and the ideal CCMs were low. The combination of these results provides strong evidence that the proposed ICM generation method generalizes well and can learn to identify and reproduce critical phase information with limited support samples. However, the new generator models have not yet been integrated and tested with an adaptive detection algorithm. That evaluation will be a critical step in determining the effectiveness of this method. Additionally, there was little to no performance difference between the generator models trained on each custom loss function.

The ICM generation method presented in this chapter showcases the power of using a first principles design approach to integrate ML with existing signal processing algorithms. It has the potential to significantly reduce sample support requirements for adaptive detection algorithms. It also directly produces the ICM from data rather then approximating and inverting the CCM. This would make adaptive detection algorithms significantly faster and more reliable. Early testing has demonstrated that the generative ML model can learn the underlying features associated with clutter covariance and use relatively few support samples to generate highly accurate approximations of the ICMs used in adaptive filtering. It also proved that this method can be effectively integrated into existing adaptive detection algorithms, allowing operation in low sample support scenarios without negatively impacting performance. Considerable work still need to be done to fully characterize this method. Testing needs to be performed using higher-fidelity simulations and real-world radar data. Testing should also be conducted to evaluate how well this approach can be trained to distinguish between different clutter distributions (K, Pareto, Weibull, etc.). Additionally, the run-time and computational requirements need to be examined for different sizes of input and output matrices. Finally, performance comparisons should be conducted with a broader range of state-of-the-art CCM estimation methods.

Chapter 8

Conclusions and Future Work

8.1 Summary of Work

This dissertation explores a challenging problem in modern radar: how can ML best be utilized to improve system performance? RSP has strict performance requirements that are commonly viewed as antithetical to the approximate and sometimes unpredictable nature of ML models. However, when properly designed and implemented, ML methods offer flexibility that may allow radar systems to maintain performance in traditionally challenging scenarios.

In Chapter 1, the problem of integrating ML with RSP was introduced. Issues related to historic ML approaches in radar were discussed. Additionally, the proposed first principles design approach and integration of ML with RSP algorithms was presented. A summary of the contributions in this work was also presented.

Chapter 2 provided an overview of ML concepts used in this work. This began with a discussion of the different types of learning, with particular attention paid to supervised learning concepts. Neural networks were introduced, followed by discussions of attention mechanisms and generative models. Finally, this chapter provided an overview of ML interpretability, explainability, causality, and genetic algorithms.

Radar concepts relevant to the research presented in this dissertation are introduced in Chapter 3. This chapter focused heavily on adaptive detection, starting with a discussion of hypothesis testing and the Neyman-Pearson lemma. This was followed by an overview of the radar detection problem space including definitions of target signal returns and interference. Common interference models were then described, focusing on the family of distributions called SIRVs. A survey of adaptive detectors and covariance estimation methods was discussed. Finally, an overview of the Kalman filter used in navigation and tracking was given.

Chapter 4 presented the design and early implementation of SEDA. This architecture provided a framework for the development of explainable ML systems through the integration of ML models, non-ML algorithms, and attention elements. Verification testing and early characterization of SEDA were performed using simple, well-defined datasets including MNIST.

A ML-based method for removing sensor measurement drift in navigation and tracking systems was described in Chapter 5. This discussion began by describing the issue of sensor drift and the motion models used for simulating training and testing data in this work. A reframing of the problem as counterfactual evaluation then provided. An initial design of the proposed method using a combination of the NSGA-II and an LSTM was discussed next, along the testing used to characterized it. Finally, an updated design and charaterization testing were provided. This updated design used the EKF in conjunction with the ML elements to improve performance.

In Chapter 6, a meta-cognitive adaptive detection algorithm was presented. It integrated an orchestrated, multi-layered ML system with Kelly's Gaussian GLRT. This produced a ML-enhanced detector that could maintain CFAR-like across interference distributions and could be easily retrained for additional interference models. The proposed design was presented, along with the clutter models used in simulation for training and testing, the data analysis used in the design of the detector, and an initial implementation. Testing and ROC analysis of the implemented detector on a range of clutter distributions was then described. Further testing was presented characterizing the performance of the ML-enhanced detection algorithm as a number of system and clutter distribution parameters were varied to evaluate how well the proposed method generalizes.

Finally, Chapter 7 described a ML method for reducing sample support requirements in covariance inversion-based adaptive detectors. This method used generative models trained to directly generate inverse clutter covariance matrices based on a limited set of support samples. This chapter began with a discussion of the clutter covariance models used for training and testing the system. An initial design using a GAN structure was then described. This initial design was integrated with the GLRT to evaluate its impact on the detector's performance. Tesing of that design on Gaussian clutter was described next. Lastly, an updated design was presented that used a generative Unet architecture trained on two custom loss functions developed for this work.

8.2 Conclusions

This dissertation presents a design approach to developing ML-enhanced RSP algorithms that are more robust and adaptable to changing scenarios. It also provides a set of these algorithms for use in adaptive detection, tracking, and navigation. Established radar signal processing algorithms used today, like the GLRT and EKF, have been derived under key assumptions about the environment and targets being observed. They maintain near optimal performance when these assumption hold, but experience severe performance degradation when they are violated.

ML models can be used to mitigate that performance degradation. However, much of the historic research into applications of ML to radar has focused on using large, singular models with lots of data to replace existing algorithms. This has led to strong push-back the radar community due to the poorly understood and often unrepeatable behaviors present in large scale ML models. The design approach presented here addresses this issue by using a combination of domain expertise, an understanding of the first principles physics at play in radar, and basic engineering design principles to integrate ML elements directly into existing RSP algorithms. The resulting ML-enhanced algorithms can maintain performance across a wider range of scenarios with minimal loss of performance.

A ML architecture for explainability was presented. This architecture allowed for the simultaneous generation of actions and corresponding explanations in a human readable format. This architecture also allowed for the seamless integration of ML models with non-ML algorithms. It was initially demonstrated on several simple, well understood datasets and has since been applied to data fusion for tracking and computer vision problems. Its development provides a methodology for logically breaking down complex problems and orchestrating ML-enhanced systems to solve them.

A method for reducing measurement drift errors in navigation and tracking systems using counterfactual evaluation was presented. It was demonstrated that this method not only reduced the drift error, but that it produced more stable trajectories when doing so than the EKF. Importantly, the best performance was achieved when the EKF was directly integrated as a preprocessing step for the proposed counterfactual method. This provided supporting evidence for the claim that well designed ML techniques can be used in conjunction with existing methods to improve overall performance. It also provided clear methods for evaluating the performance impacts of the ML components by using the unmodified exiting methods as a baseline for comparison. However, the nature of the genetic algorithm meant that the results were not perfectly repeatable, with a slightly different track being produced each time the same data was run though the system. Additionally, the MOGA added significant computational overhead compared with the EKF.

A meta-cognitive adaptive detection algorithm was presented. This algorithm used two stages of ML to dynamically set thresholds for existing adaptive detectors (like the GLRT) based on the distribution of interference present in the scenario. The lower-level ML consisted of multiple models trained to set thresholds within a particular distribution category. The higher-level model was trained to classify distribution categories from support samples and select the corresponding thresholding agent. The distribution categories were defined by the detector threshold characteristics. Testing and characterization of this method was done using the Gaussian GLRT. It was found that this method allowed the adaptive detector to maintain CFAR-like behavior across a wide range of distributions with minimal impact to the probability of detection. It was also found that there was no observable performance impact to the GLRT when the clutter was Gaussian. The primary limitation of the proposed method was in its intensive design and training process.

Finally, this work provided a method for reducing the sample support requirements of covariance inversion-based adaptive detectors using generative ML. A generative model was trained on a limited set of support samples to identify the correlation information of the interference and directly produce ICMs to be used in whitening filters. Both GANs and generative Unets were examined, and two custom loss functions were developed for training the generative model. The generative model was directly integrated with adaptive detection algorithms to evaluate its performance impact. It was found that this method could reduce support sample requirements by at least half while providing modest performance improvements to both clutter rejection and ROC behavior. Additionally, it removed the need for CCM inversion, which is a computationally costly operation.

Using the design approach presented here allows one to take advantage of the benefits of both traditional signal processing techniques and ML. It also serves as an excellent intermediate step to begin integrating ML capabilities into new fields. Breaking down challenging problems and identifying areas where ML can be applied in a targeted fashion alongside established methods helps to provide theoretical basis for its use. It also limits the potential for undesirable behaviors and improves user trust in the ML elements. If the integration is successful, and if it makes sense to do so, the ML elements can be expanded to take on more operations. Such an iterative approach minimizes the potential risk of deploying ML and serves to improve the overall design of any tools using it.

8.3 Future Work

RSP and ML are both vast topics, and there are various open areas of research into how they might be better integrated. Considerable work remains to be done to develop more quantifiable metrics for measuring the quality and consistency of ML models applied to the radar domain. Measures of user trust in ML-enhanced algorithms should also be defined and evaluated. These are both key enablers to the development of ML-enhanced radar systems.

The development of a MOGA-based algorithms for measurement drift mitigation highlighted that many ML methods lack repeatability, which may introduce considerable risk to radar systems. Future work in this field should focus on the development of methods to characterize the potential variation in outputs from ML-enhanced radar systems when repeatedly presented with identical input data. This approach also took considerably longer to run than the EKF it was compared against, largely due to the use of the genetic algorithm. Work should be done to explore alternative methods to the MOGA for generating and evaluating tracks. Of particular interest is the generative transformer, whose latent space can be trained to recognize latent features from multiple modalities and along many dimensions. This could allow them to be trained to achieve similar performance without the computational overhead and lack of repeatability present in the current system. Additionally, new objective functions should be developed using alternative data types.

The design, development, and modification of a meta-cognitive detector like the one presented in this work requires considerable expertise in both detection and ML. It also requires considerable time to evaluate and identify which regions within a set of distributions are statistically unique. Work should be done to automate this process so that custom detection algorithms can be implemented rapidly. Considerable work should also be done to characterize performance of the proposed method on a broader range of clutter distributions. Further, work should be done to explore how well the method generalizes to different radar systems and parameter configurations.

In the future, the proposed U-Net generator should be integrated with various adaptive detection algorithms in order to fully characterize its performance impacts. Additionally, this method should be evaluated with different interference distributions to see how well it identifies covariance properties across distributions. It should also be evaluated with variably-sized input-output pairings to see how well it generalizes across data sizes. Both the meta-cognitive detection algorithm and the ICM generation method should be evaluated on high-fidelity simulations and using real-world data. It is also worth characterizing the computational requirements of these approaches for different input data sizes.

References

[1] S. M. Kay, “Fundamentals of statistical processing, volume 2: Detection theory,” 2009.

[2] E. Mason, B. Yonel, and B. Yazici, “Deep learning for radar,” in *2017 IEEE Radar Conference (RadarConf)*, 2017, pp. 1703–1708.

[3] K. V. Mishra, M. Bhavani Shankar, V. Koivunen, B. Ottersten, and S. A. Vorobyov, “Toward millimeter-wave joint radar communications: A signal processing perspective,” *IEEE Signal Processing Magazine*, vol. 36, no. 5, pp. 100–114, 2019.

[4] F. Engels, P. Heidenreich, M. Wintermantel, L. Stäcker, M. Al Kadi, and A. M. Zoubir, “Automotive radar signal processing: Research directions and practical challenges,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, pp. 865–878, 2021.

[5] M. Shaghaghi and R. S. Adve, “Machine learning based cognitive radar resource management,” in *2018 IEEE Radar Conference (RadarConf18)*, 2018, pp. 1433–1438.

[6] S. A. Flandermeyer, R. G. Mattingly, and J. G. Metcalf, “Deep reinforcement learning for cognitive radar spectrum sharing: A continuous control approach,” *IEEE Transactions on Radar Systems*, vol. 2, pp. 125–137, 2024.

[7] J. Metcalf, S. Blunt, and B. Himed, “A machine learning approach to distribution identification in non-gaussian clutter,” in *Proc. IEEE Radar Conf*, May 2014, pp. 0739–0744.

[8] J. Metcalf, S. D. Blunt, and B. Himed, “A machine learning approach to cognitive radar detection,” in *Proc. IEEE Radar Conf. (RadarCon)*, May 2015, pp. 1405–1411.

[9] A. Stringer, B. Sun, Z. Hoyt, L. Schley, D. Hougen, and J. K. Antonio, “Seda: A self-explaining decision architecture implemented using deep learning for onboard command and control,” in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, pp. 1–10.

[10] D. Hogue, T. Sharp, J. Karch, G. Dolinger, A. Stringer, L. Schley, A. Bowersox, C. Weaver, and D. Hougen, "Using informative ai to understand camouflaged object detection and segmentation," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pp. 1–9.

[11] A. Stringer, B. Sun, L. Schley, D. F. Hougen, and J. K. Antonio, "Causality-aware machine learning for path correction," in *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*. IEEE, 2022, pp. 1–10.

[12] A. Stringer, G. Dolinger, T. Sharp, D. Hogue, J. Karch, L. Borowska, and J. Metcalf, "Improving predictive navigation through the optimization of counterfactual track evaluation," in *2023 IEEE/ION IEEE/ION Position, Location and Navigation Symposium (PLANS) (In Review)*. IEEE, 2023, pp. 1–12.

[13] E. Kelly, "An adaptive detection algorithm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-22, no. 2, pp. 115–127, 1986.

[14] A. Stringer, G. Dolinger, D. Hogue, L. Schley, and J. Metcalf, "A meta-cognitive approach to adaptive radar detection," in *IEEE Transactions on Aerospace and Electronic Systems (In Review)*. IEEE, 2023, pp. 1–12.

[15] A. Stringer, T. Sharp, G. Dolinger, S. Howell, J. Karch, J. Metcalf, and A. Bowersox, "Application of generative machine learning for adaptive detection with limited sample support," in *2024 IEEE Radar Conference (RadarConf24), Accepted*, 2024.

[16] M. L. Minsky and S. A. Papert, *Perceptrons*. MIT Press, 1969.

[17] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997.

[19] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020.

[20] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[21] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz, "Mocogan: Decomposing motion and content for video generation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1526–1535.

[22] Y. Saito, S. Takamichi, H. Saruwatari, Y. Saito, S. Takamichi, and H. Saruwatari, "Statistical parametric speech synthesis incorporating genera-

tive adversarial networks," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 26, no. 1, p. 84–96, jan 2018.

[23] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," 2018.

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[25] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015.

[26] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," 2016.

[27] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," 2016.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[29] D. Kingma and M. Welling, "Performance comparison of NSGA-II and NSGA-III on various many-objective test problems," in *2014 International Conference on Learning Representations (ICLR)*, 2014.

[30] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[31] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[32] X. Li, A. Alkhateeb, and C. Tepedelenlioğlu, "Generative adversarial estimation of channel covariance in vehicular millimeter wave systems," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1572–1576.

[33] P. Isola, J. Zhu, T. Zhou, and A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 5967–5976.

[34] T. Miyato and M. Koyama, "cgans with projection discriminator," 2018.

[35] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[36] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a "right to explanation"," *AI Magazine*, vol. 38, no. 3, pp. 50–57, Oct. 2017.

[37] M. Fisher, V. Mascardi, K. Y. Rozier, B.-H. Schlingloff, M. Winikoff, and N. Yorke-Smith, "Towards a framework for certification of reliable autonomous systems," *Autonomous Agents and Multi-Agent Systems*, vol. 35, no. 1, 2020.

[38] S. Mohseni, M. Pitale, V. Singh, and Z. Wang, "Practical solutions for machine learning safety in autonomous vehicles," 2019.

[39] J.-X. Mi, A.-D. Li, and L.-F. Zhou, "Review study of interpretation methods for future interpretable machine learning," *IEEE Access*, vol. 8, pp. 191 969–191 985, 2020.

[40] H. Zhang, Q. Zhang, S. Shao, T. Niu, and X. Yang, "Attention-based LSTM network for rotatory machine remaining useful life prediction," *IEEE Access*, vol. 8, pp. 132 188–132 199, 2020.

[41] L. A. Dennis, M. Fisher, N. K. Lincoln, A. Lisitsa, and S. M. Veres, "Practical verification of decision-making in agent-based autonomous systems," *Automated Software Engineering*, vol. 23, no. 3, pp. 305–359, 2016.

[42] P. Bremner, L. A. Dennis, M. Fisher, and A. F. Winfield, "On proactive, transparent, and verifiable ethical reasoning for robots," *Proceedings of the IEEE*, vol. 107, no. 3, pp. 541–561, 2019.

[43] J. Pearl, "An introduction to causal inference," *The International Journal of Biostatistics*, vol. 6(2), no. 7, 2010.

[44] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Toward causal representation learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 612–634, 2021.

[45] J. Pearl, M. Glymour, and N. Jewell, *Causal Inference in Statistics: A Primer*. Wiley, 2016.

[46] J. H. Brenas and A. Shaban-Nejad, "Health intervention evaluation using semantic explainability and causal reasoning," *IEEE Access*, vol. 8, pp. 9942–9952, 2020.

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[48] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, "Performance comparison

of NSGA-II and NSGA-III on various many-objective test problems," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 3045–3052.

[49] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[50] M. A. Richards, *Fundamentals of radar signal processing*. Tata McGraw-Hill Education, 2005.

[51] J. Ward, "Space time adaptive processing for airborne radar," Massachusetts Institute of Technology - Lincoln Laboratory, Tech. Rep., 1994.

[52] I. Reed, J. Mallett, and L. Brennan, "Rapid convergence rate in adaptive arrays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-10, no. 6, pp. 853–863, 1974.

[53] E. Conte, M. Lops, and G. Ricci, "Asymptotically optimum radar detection in compound-gaussian clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 617–625, 1995.

[54] F. Gini and M. Greco, "Suboptimum approach to adaptive coherent radar detection in compound-gaussian clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 3, pp. 1095–1104, 1999.

[55] P. Lombardo, D. Pastina, and T. Bucciarelli, "Adaptive polarimetric target detection with coherent radar. ii. detection against non-gaussian background," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 37, no. 4, pp. 1207–1220, 2001.

[56] D. Pastina, P. Lombardo, and T. Bucciarelli, "Adaptive polarimetric target detection with coherent radar. i. detection against gaussian background," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 37, no. 4, pp. 1194–1206, 2001.

[57] F. Gini and A. Farina, "Vector subspace detection in compound-gaussian clutter. part i: survey and new results," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 4, pp. 1295–1311, 2002.

[58] F. Gini, A. Farina, and M. Montanari, "Vector subspace detection in compound-gaussian clutter. part ii: performance analysis," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 4, pp. 1312–1323, 2002.

[59] P. Wang, K. J. Sohn, H. Li, and B. Himed, "Performance evaluation of parametric rao and glrt detectors with kassper and bistatic data," in *2008 IEEE Radar Conference*, 2008, pp. 1–6.

[60] K. J. Sangston, F. Gini, and M. S. Greco, “Coherent radar target detection in heavy-tailed compound-gaussian clutter,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 64–77, 2012.

[61] D. Ciuonzo, A. De Maio, and D. Orlando, “On the statistical invariance for adaptive radar detection in partially homogeneous disturbance plus structured interference,” *IEEE Transactions on Signal Processing*, vol. 65, no. 5, pp. 1222–1234, 2017.

[62] D. P. Zilz and M. R. Bell, “Statistical modeling of wireless communications interference and its effects on adaptive-threshold radar detection,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 2, pp. 890–911, Apr. 2018.

[63] N. Nartasilpa, S. Shahi, A. Salim, D. Tuninetti, N. Devroye, D. Erricolo, D. P. Zilz, and M. R. Bell, “Let’s share commrad: Co-existing communications and radar systems,” in *Proc. IEEE Radar Conf. (RadarConf18)*, Apr. 2018, pp. 1278–1283.

[64] J. G. Metcalf and S. Flandermeyer, “On spectrum sharing for pulse-doppler radar and ofdm communications,” in *2020 IEEE Radar Conference (RadarConf20)*, 2020, pp. 1–6.

[65] K. Sangston and K. Gerlach, “Coherent detection of radar targets in a non-gaussian background,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 2, pp. 330–340, 1994.

[66] M. Rangaswamy, D. Weiner, and A. Ozturk, “Computer generation of correlated non-gaussian radar clutter,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 1, pp. 106–116, 1995.

[67] A. Balleri, A. Nehorai, and J. Wang, “Maximum likelihood estimation for compound-gaussian clutter with inverse gamma texture,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 2, pp. 775–779, 2007.

[68] N. Pulsone and C. Rader, “Adaptive beamformer orthogonal rejection test,” *IEEE Transactions on Signal Processing*, vol. 49, no. 3, pp. 521–529, 2001.

[69] J. Liu, Z.-J. Zhang, P.-L. Shui, and H. Liu, “Exact performance analysis of an adaptive subspace detector,” *IEEE Transactions on Signal Processing*, vol. 60, no. 9, pp. 4945–4950, 2012.

[70] C. H. Gierul, “On the receiver operating characteristics of adaptive radar detectors,” University of Zurich, Department of Informatics, Tech. Rep., 2013.

[71] D. Ciuonzo, A. De Maio, and D. Orlando, “A unifying framework for adaptive radar detection in homogeneous plus structured interference— part i: On the

maximal invariant statistic," *IEEE Transactions on Signal Processing*, vol. 64, no. 11, pp. 2894–2906, 2016.

[72] D. Ciuonzo, A. De Maio, and D. Orlando, "A unifying framework for adaptive radar detection in homogeneous plus structured interference— part ii: Detectors design," *IEEE Transactions on Signal Processing*, vol. 64, no. 11, pp. 2907–2919, 2016.

[73] W. Liu, Y.-L. Wang, J. Liu, L. Huang, and C. Hao, "Performance analysis of adaptive detectors for point targets in subspace interference and gaussian noise," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 1, pp. 429–441, 2018.

[74] E. Kelly, "Performance of an adaptive detection algorithm; rejection of unwanted signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 25, no. 2, pp. 122–133, 1989.

[75] E. Conte and G. Ricci, "Sensitivity study of glrt detection in compound-gaussian clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 1, pp. 308–316, 1998.

[76] P.-J. Chung and K. M. Wong, "A full generalized likelihood ratio test for source detection," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 2445–2448.

[77] P. Wang, H. Li, and B. Himed, "A simplified parametric glrt for stap detection," in *2009 IEEE Radar Conference*, 2009, pp. 1–5.

[78] D. P. Bruyere and N. A. Goodman, "Adaptive detection and diversity order in multistatic radar," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 4, pp. 1615–1623, 2008.

[79] K. J. Sohn, H. Li, and B. Himed, "Parametric glrt for multichannel adaptive signal detection," *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5351–5360, 2007.

[80] L. Scharf and L. McWhorter, "Adaptive matched subspace detectors and adaptive coherence estimators," in *Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers*, 1996, pp. 1114–1117 vol.2.

[81] F. Robey, D. Fuhrmann, E. Kelly, and R. Nitzberg, "A cfar adaptive matched filter detector," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 28, no. 1, pp. 208–216, 1992.

[82] J. Roman, M. Rangaswamy, D. Davis, Q. Zhang, B. Himed, and J. Michels, "Parametric adaptive matched filter for airborne radar applications," *IEEE*

Transactions on Aerospace and Electronic Systems, vol. 36, no. 2, pp. 677–692, 2000.

[83] A. De Maio, “A new derivation of the adaptive matched filter,” *IEEE Signal Processing Letters*, vol. 11, no. 10, pp. 792–793, 2004.

[84] J. Liu, H. Li, and B. Himed, “Threshold setting for adaptive matched filter and adaptive coherence estimator,” *IEEE Signal Processing Letters*, vol. 22, no. 1, pp. 11–15, 2015.

[85] A. De Maio, “Rao test for adaptive detection in gaussian interference with unknown covariance matrix,” *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3577–3584, 2007.

[86] K. J. Sohn, H. Li, and B. Himed, “Parametric rao test for multichannel adaptive signal detection,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 3, pp. 921–933, 2007.

[87] A. De Maio and S. Iommelli, “Coincidence of the rao test, wald test, and glrt in partially homogeneous environment,” *IEEE Signal Processing Letters*, vol. 15, pp. 385–388, 2008.

[88] S. Kraut and L. Scharf, “The cfar adaptive subspace detector is a scale-invariant glrt,” *IEEE Transactions on Signal Processing*, vol. 47, no. 9, pp. 2538–2541, 1999.

[89] S. Kraut and J. Krolik, “Application of maximal invariance to the ace detection problem,” in *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers (Cat. No.00CH37154)*, vol. 1, 2000, pp. 417–420 vol.1.

[90] S. Kraut, L. Scharf, and R. Butler, “The adaptive coherence estimator: a uniformly most-powerful-invariant adaptive detection statistic,” *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 427–438, 2005.

[91] J. Liu, Z.-J. Zhang, Y. Yang, and H. Liu, “A cfar adaptive subspace detector for first-order or second-order gaussian signals based on a single observation,” *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5126–5140, 2011.

[92] Y. Gao, G. Liao, S. Zhu, X. Zhang, and D. Yang, “Persymmetric adaptive detectors in homogeneous and partially homogeneous environments,” *IEEE Transactions on Signal Processing*, vol. 62, no. 2, pp. 331–342, 2014.

[93] F. Bandiera, O. Besson, and G. Ricci, “An abort-like detector with improved mismatched signals rejection capabilities,” *IEEE Transactions on Signal Processing*, vol. 56, no. 1, pp. 14–25, 2008.

[94] C. Richmond, "Performance of a class of adaptive detection algorithms in nonhomogeneous environments," *IEEE Transactions on Signal Processing*, vol. 48, no. 5, pp. 1248–1262, 2000.

[95] N. Pulsone and M. Zatman, "A computationally efficient two-step implementation of the glrt," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 609–616, 2000.

[96] F. Bandiera, O. Besson, D. Orlando, and G. Ricci, "An improved adaptive sidelobe blanker," *IEEE Transactions on Signal Processing*, vol. 56, no. 9, pp. 4152–4161, 2008.

[97] M. Rangaswamy, P. Chakravarthi, D. Weiner, L. Cai, and H. Wang, "Signal detection in correlated gaussian and non-gaussian radar clutter," KAMAN SCIENCES CORP UTICA NY, Tech. Rep., 1993.

[98] A. Wiesel, "Unified framework to regularized covariance estimation in scaled gaussian models," *IEEE Transactions on Signal Processing*, vol. 60, no. 1, pp. 29–38, 2012.

[99] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE Signal Processing Magazine*, vol. 13, no. 4, pp. 67–94, 1996.

[100] P. Wirfält and M. Jansson, "On kronecker and linearly structured covariance matrix estimation," *IEEE Transactions on Signal Processing*, vol. 62, no. 6, pp. 1536–1547, 2014.

[101] Y. Li and Y. Chi, "Compressive parameter estimation with multiple measurement vectors via structured low-rank covariance estimation," in *2014 IEEE Workshop on Statistical Signal Processing (SSP)*, 2014, pp. 384–387.

[102] Y. Sun, P. Babu, and D. Palomar, "Robust estimation of structured covariance matrix for heavy-tailed elliptical distributions," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3576–3590, 2016.

[103] D. Wikes and M. Hayes, "Iterated toeplitz approximation of covariance matrices," in *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 1988, pp. 1663–1666 vol.3.

[104] G. Pailloux, P. Forster, J. Ovarlez, and F. Pascal, "Persymmetric adaptive radar detectors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 4, pp. 2376–2390, 2011.

[105] J. Liu, W. Liu, B. Tang, J. Zheng, and S. Xu, "Distributed target detection exploiting persymmetry in gaussian clutter," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1022–1033, 2019.

[106] A. De Maio and D. Orlando, “An invariant approach to adaptive radar detection under covariance persymmetry,” *IEEE Transactions on Signal Processing*, vol. 63, no. 5, pp. 1297–1309, 2015.

[107] A. Zoubir, V. Koivunen, Y. Chakhchoukh, and M. Muma, “Robust estimation in signal processing: A tutorial-style treatment of fundamental concepts,” *IEEE Signal Processing Magazine*, vol. 29, no. 4, pp. 61–80, 2012.

[108] S. Sen, “Low-rank matrix decomposition and spatio-temporal sparse recovery for stap radar,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 8, pp. 1510–1523, 2015.

[109] M. Azizyan, A. Krishnamurthy, and A. Singh, “Extreme compressive sampling for covariance estimation,” *IEEE Transactions on Information Theory*, vol. 64, no. 12, pp. 7613–7635, 2018.

[110] Y. Chen, Y. Chi, and A. Goldsmith, “Exact and stable covariance estimation from quadratic sampling via convex programming,” *IEEE Transactions on Information Theory*, vol. 61, no. 7, pp. 4034–4059, 2015.

[111] E. Conte, A. De Maio, and G. Ricci, “Recursive estimation of the covariance matrix of a compound-gaussian process and its application to adaptive cfar detection,” *IEEE Transactions on Signal Processing*, vol. 50, no. 8, pp. 1908–1915, 2002.

[112] F. Pascal, Y. Chitour, J. Ovarlez, P. Forster, and P. Larzabal, “Covariance structure maximum-likelihood estimates in compound gaussian noise: Existence and algorithm analysis,” *IEEE Transactions on Signal Processing*, vol. 56, no. 1, pp. 34–48, 2008.

[113] M. Greco, P. Stinco, F. Gini, and M. Rangaswamy, “Impact of sea clutter nonstationarity on disturbance covariance matrix estimation and cfar detector performance,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 46, no. 3, pp. 1502–1513, 2010.

[114] A. Ali, N. González-Prelcic, and R. Heath, “Spatial covariance estimation for millimeter wave hybrid systems using out-of-band information,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 12, pp. 5471–5485, 2019.

[115] B. Kang, S. Gogineni, M. Rangaswamy, J. Guerci, and E. Blasch, “Adaptive channel estimation for cognitive fully adaptive radar,” *IET Radar, Sonar & Navigation*, vol. 16, no. 4, pp. 720–734, 2022.

[116] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 03 1960.

[117] I. Arasaratnam and S. Haykin, “Cubature kalman filters,” *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1254–1269, 2009.

[118] P. S. Madhukar and L. B. Prasad, “State estimation using extended Kalman filter and unscented Kalman filter,” in *2020 International Conference on Emerging Trends in Communication, Control and Computing (ICONC3)*, 2020, pp. 1–4.

[119] S. Julier, J. Uhlmann, and H. Durrant-Whyte, “A new method for the nonlinear transformation of means and covariances in filters and estimators,” *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477–482, 2000.

[120] S. Julier and J. Uhlmann, “Unscented filtering and nonlinear estimation,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[121] Y. LeCun, C. Cortes, and C. Burges, “The MNIST Database of Handwritten Digits.”

[122] B. Eskridge and D. Hougen, “Prioritizing fuzzy behaviors in multi-robot pursuit teams,” in *2006 IEEE International Conference on Fuzzy Systems*, 2006, pp. 1119–1125.

[123] Y. Zhang *et al.*, “Siren’s song in the ai ocean: A survey on hallucination in large language models,” *ArXiv*, vol. abs/2309.01219, 2023.

[124] D. N. Banerjee and S. S. Chanda, “Ai failures: A review of underlying issues,” *ArXiv*, vol. abs/2008.04073, 2020.

[125] D. V. Nam and K. Gon-Woo, “Robust stereo visual inertial navigation system based on multi-stage outlier removal in dynamic environments,” *Sensors (Basel, Switzerland)*, vol. 20, no. 10, p. 2922, 2020.

[126] A. E. Mahdi, A. Azouz, A. E. Abdalla, and A. Abosekeen, “A machine learning approach for an improved inertial navigation system solution,” *Sensors (Basel, Switzerland)*, vol. 22, no. 4, p. 1687, 2022.

[127] N. Al Bitar and A. Gavrilov, “A novel approach for aiding unscented kalman filter for bridging gnss outages in integrated navigation systems,” *Navigation (Washington)*, vol. 68, no. 3, pp. 521–539, 2021.

[128] L. Semeniuk and A. Noureldin, “Bridging gps outages using neural network estimates of ins position and velocity errors,” *Measurement science & technology*, vol. 17, no. 10, pp. 2783–2798, 2006.

[129] N. Al Bitar and A. Gavrilov, “A new method for compensating the errors of integrated navigation systems using artificial neural networks,” *Measurement*, vol. 168, p. 108391, 2021.

[130] A. Abosekeen and A. Abdalla, “Improving the navigation system of a uav using multi-sensor data fusion based on fuzzy c-means clustering,” *International Conference on Aerospace Sciences and Aviation Technology*, vol. 14, pp. 1–12, 2011.

[131] A. V. Kanhere, S. Gupta, A. Shetty, and G. Gao, “Improving gnss positioning using neural-network-based corrections,” *NAVIGATION: Journal of the Institute of Navigation*, vol. 69, no. 4, 2022.

[132] N. I. Ziedan, “Multipath channel estimation and pattern recognition for environment-based adaptive tracking,” in *Proceedings of the 25th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2012)*, 2012, pp. 394–407.

[133] Y. Zhang, “A fusion methodology to bridge gps outages for ins/gps integrated navigation system,” *IEEE Access*, vol. 7, pp. 61 296–61 306, 2019.

[134] R. Sun, L. Fu, G. Wang, Q. Cheng, L.-T. Hsu, and W. Y. Ochieng, “Using dual-polarization gps antenna with optimized adaptive neuro-fuzzy inference system to improve single point positioning accuracy in urban canyons,” *Navigation*, vol. 68, no. 1, pp. 41–60, 2021.

[135] H. fa Dai, H. wei Bian, R. ying Wang, and H. Ma, “An ins/gnss integrated navigation in gnss denied environment using recurrent neural network,” *Defence Technology*, vol. 16, no. 2, pp. 334–340, 2020.

[136] A. Siemuri, H. Kuusniemi, M. S. Elmusrati, P. Välisuo, and A. Shamsuzzoha, “Machine learning utilization in gnss—use cases, challenges and future applications,” in *2021 International Conference on Localization and GNSS (ICL-GNSS)*, 2021, pp. 1–6.

[137] P. H. Eichel and C. Jakowatz, “Phase-gradient algorithm as an optimal estimator of the phase derivative,” *Optics letters*, vol. 14, no. 20, pp. 1101–1103, 1989.

[138] D. E. Wahl, P. Eichel, D. Ghiglia, and C. Jakowatz, “Phase gradient autofocus-a robust tool for high resolution sar phase correction,” *IEEE Transactions on Aerospace and electronic systems*, vol. 30, no. 3, pp. 827–835, 1994.

[139] M. Greco, K. Kulpa, G. Pinelli, and P. Samczynski, “Sar and insar georeferencing algorithms for inertial navigation systems,” in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2011*, vol. 8008. SPIE, 2011, pp. 499–507.

[140] M. Greco, S. Querry, G. Pinelli, K. Kulpa, P. Samczynski, D. Gromek, A. Gromek, M. Malanowski, B. Querry, and A. Bonsignore, “Sar-based aug-

mented integrity navigation architecture," in *2012 13th International Radar Symposium*. IEEE, 2012, pp. 225–229.

[141] G. Franceschetti and R. Lanari, *Synthetic aperture radar processing*. CRC press, 2018.

[142] A. Zhang and M. M. Atia, "An efficient tuning framework for kalman filter parameter optimization using design of experiments and genetic algorithms," *NAVIGATION: Journal of the Institute of Navigation*, vol. 67, no. 4, pp. 775–793, 2020.

[143] Y. Xiang, M. Akcakaya, S. Sen, and A. Nehorai, "Point detection, learning, and adaptation," *Circuits, Systems, and Signal Processing*, vol. 40, p. 233–261, 2021.

[144] D. Mata-Moya, N. del Rey-Maestre, V. M. Peláez-Sánchez, M.-P. Jarabo-Amores, and J. M. de Nicolás, "Mlp-cfar for improving coherent radar detectors robustness in variable scenarios," *Expert Systems with Applications*, vol. 42, no. 11, pp. 4878–4891, 2015.

[145] J. Ward, "Space-time adaptive processing for airborne radar," in *IEE Colloquium on Space-Time Adaptive Processing (Ref. No. 1998/241)*, 1998, pp. 2/1–2/6.

[146] Z. Yang, X. Li, H. Wang, and W. Jiang, "On clutter sparsity analysis in space–time adaptive processing airborne radar," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 5, pp. 1214–1218, 2013.

[147] W. Melvin, "A stap overview," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 19–35, 2004.

[148] W. Jiang, Y. Ren, Y. Liu, and J. Leng, "Artificial neural networks and deep learning techniques applied to radar target detection: A review," *Electronics*, vol. 11, no. 1, 2022.

[149] M. A. Nuhoglu, Y. K. Alp, and F. C. Akyon, "Deep learning for radar signal detection in electronic warfare systems," in *2020 IEEE Radar Conference (RadarConf20)*, 2020, pp. 1–6.

[150] A. Coluccia, A. Fascista, and G. Ricci, "Robust cfar radar detection using a k-nearest neighbors rule," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 4692–4696.

[151] A. Coluccia, A. Fascista, and G. Ricci, "A knn-based radar detector for coherent targets in non-gaussian noise," *IEEE Signal Processing Letters*, vol. 28, pp. 778–782, 2021.

[152] J. Akhtar and K. E. Olsen, "A neural network target detector with partial ca-cfar supervised training," in *2018 International Conference on Radar (RADAR)*, 2018, pp. 1–6.

[153] M. V. i. Carretero, R. I. A. Harmanny, and R. P. Trommel, "Smart-cfar, a machine learning approach to floating level detection in radar," in *2019 16th European Radar Conference (EuRAD)*, 2019, pp. 161–164.

[154] S. Wunsch, J. Fink, and F. K. Jondral, "Improved detection by peak shape recognition using artificial neural networks," in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, 2015, pp. 1–5.

[155] E. V. Carrera, F. Lara, M. Ortiz, A. Tinoco, and R. León, "Target detection using radar processors based on machine learning," in *2020 IEEE ANDESCON*, 2020, pp. 1–5.

[156] R. Sahay, S. Appadwedula, D. J. Love, and C. G. Brinton, "A neural network-prepended glrt framework for signal detection under nonlinear distortions," *IEEE Communications Letters*, vol. 26, no. 9, pp. 2161–2165, 2022.

[157] T. H. Kerbaa, A. Mezache, F. Gini, and M. S. Greco, "Multi-headed deep learning-based estimator for correlated-sirv pareto type ii distributed clutter," *EURASIP Journal on Advances in Signal Processing*, 2023.

[158] Y. Xiang, M. Kelsey, H. Wang, S. Sen, M. Akcakaya, and A. Nehorai, "A comparison of cognitive approaches for clutter-distribution identification in nonstationary environments," in *2018 IEEE Radar Conference (RadarConf18)*, 2018, pp. 0467–0472.

[159] R. Ahmed and H. Deng, "A multi-feature based machine learning approach for ground-moving radar target detection," in *2022 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting (AP-S/URSI)*, 2022, pp. 533–534.

[160] X. Peng, Y. Tian, P. Wang, J. Yu, and W. Ren, "Clutter classification for cognitive radar with a deep convolutional neural network," in *IET International Radar Conference (IET IRC 2020)*, vol. 2020, 2020, pp. 46–51.

[161] C. Wang, J. Tian, J. Cao, and X. Wang, "Deep learning-based uav detection in pulse-doppler radar," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–12, 2022.

[162] C.-H. Lin, Y.-C. Lin, Y. Bai, W.-H. Chung, T.-S. Lee, and H. Huttunen, "Dl-cfar: A novel cfar target detection method based on deep learning," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, 2019, pp. 1–6.

[163] D. Brodeski, I. Bilik, and R. Giryes, "Deep radar detector," in *2019 IEEE Radar Conference (RadarConf)*, 2019, pp. 1–6.

[164] S. M. D. Rizvi, S. Ahmad, K. Khan, A. Hasan, and A. Masood, "Deep learning approach for fixed and rotary-wing target detection and classification in radars," *IEEE Aerospace and Electronic Systems Magazine*, vol. 37, no. 3, pp. 32–42, 2022.

[165] J. Ai, X. Yang, J. Song, Z. Dong, L. Jia, and F. Zhou, "An adaptively truncated clutter-statistics-based two-parameter cfar detector in sar imagery," *IEEE Journal of Oceanic Engineering*, vol. 43, no. 1, pp. 267–279, 2018.

[166] J. Ai, R. Tian, Q. Luo, J. Jin, and B. Tang, "Multi-scale rotation-invariant haar-like feature integrated cnn-based ship detection algorithm of multiple-target environment in sar imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 12, pp. 10 070–10 087, 2019.

[167] X. Chen, N. Su, Y. Huang, and J. Guan, "False-alarm-controllable radar detection for marine target based on multi features fusion via cnns," *IEEE Sensors Journal*, vol. 21, no. 7, pp. 9099–9111, 2021.

[168] J. Ai, Y. Mao, Q. Luo, M. Xing, K. Jiang, L. Jia, and X. Yang, "Robust cfar ship detector based on bilateral-trimmed-statistics of complex ocean scenes in sar imagery: A closed-form solution," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 3, pp. 1872–1890, 2021.

[169] J. Ai, Z. Pei, B. Yao, Z. Wang, and M. Xing, "Ais data aided rayleigh cfar ship detection algorithm of multiple-target environment in sar images," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 2, pp. 1266–1282, 2022.

[170] H. Rohling, "Radar cfar thresholding in clutter and multiple target situations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-19, no. 4, pp. 608–621, 1983.

[171] R. Nitzberg, "An effect of range-heterogeneous clutter on adaptive doppler filters," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 3, pp. 475–480, 1990.

[172] J. Goldstein, I. Reed, and P. Zulch, "Multistage partially adaptive stap cfar detection algorithm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 2, pp. 645–661, 1999.

[173] M. K. McDonald and D. Cerutti-Maori, "Coherent radar processing in sea clutter environments, part 2: adaptive normalised matched filter versus adaptive

matched filter performance," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1818–1833, 2016.

[174] E. Aboutanios and B. Mulgrew, "A stap algorithm for radar target detection in heterogeneous environments," in *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, 2005, pp. 966–971.

[175] J. Li and P. Stoica, "An adaptive filtering approach to spectral estimation and sar imaging," *IEEE Transactions on Signal Processing*, vol. 44, no. 6, pp. 1469–1484, 1996.

[176] P. Stoica, H. Li, and J. Li, "A new derivation of the apes filter," *IEEE Signal Processing Letters*, vol. 6, no. 8, pp. 205–206, 1999.

[177] E. Aboutanios and B. Mulgrew, "Hybrid detection approach for stap in heterogeneous clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 46, no. 3, pp. 1021–1033, 2010.

[178] E. Aboutanios and L. Rosenberg, "Single snapshot coherent detection in sea clutter," in *2019 IEEE Radar Conference (RadarConf)*, 2019, pp. 1–6.

[179] E. Aboutanios and L. Rosenberg, "Hybrid detection approaches using the single data set algorithm," in *2020 IEEE Radar Conference (RadarConf20)*, 2020, pp. 1–6.

[180] J. Goldstein, I. Reed, and L. Scharf, "A multistage representation of the wiener filter based on orthogonal projections," *IEEE Transactions on Information Theory*, vol. 44, no. 7, pp. 2943–2959, 1998.

[181] R. Gray, E. Aboutanios, and L. Rosenberg, "Determination of the number of stages of the multistage wiener filter," in *2023 IEEE International Radar Conference (RADAR)*, 2023, pp. 1–6.

[182] J. D. Hiemstra and J. S. Goldstein, "Robust rank selection for the multistage wiener filter," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 2002, pp. III–2929–III–2932.

[183] J. Hiemstra, M. Weippert, H. Nguyen, and J. Goldstein, "Insertion of diagonal loading into the multistage wiener filter," in *Sensor Array and Multichannel Signal Processing Workshop Proceedings, 2002*, 2002, pp. 379–382.

[184] M. H. K. Wong, E. Aboutanios, L. Rosenberg, and A. Spargo, "Reduced rank hybrid target detectors for maritime environments," in *International Conference on Radar Systems (RADAR 2022)*, vol. 2022, 2022, pp. 272–277.

[185] M. Wong, E. Aboutanios, and L. Rosenberg, "A stap detection scheme for low

sample support maritime environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 59, no. 5, pp. 5671–5683, 2023.

[186] F. Gini and M. Greco, "Covariance matrix estimation for cfar detection in correlated heavy tailed clutter," *Signal Processing*, vol. 82, no. 12, pp. 1847–1859, 2002.

[187] E. Conte, A. De Maio, and C. Galdi, "Statistical analysis of real clutter at different range resolutions," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 903–918, 2004.

[188] I. Kirsteins and D. Tufts, "Adaptive detection using low rank approximation to a data matrix," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 1, pp. 55–67, 1994.

[189] M. Rangaswamy, F. Lin, and K. Gerlach, "Robust adaptive signal processing methods for heterogeneous radar clutter scenarios," in *Proceedings of the 2003 IEEE Radar Conference (Cat. No. 03CH37474)*, 2003, pp. 265–272.

[190] L. Yang, M. R. McKay, and R. Couillet, "High-dimensional mvdr beamforming: Optimized solutions based on spiked random matrix models," *IEEE Transactions on Signal Processing*, vol. 66, no. 7, pp. 1933–1947, 2018.

[191] X. Ding and F. Yang, "Spiked separable covariance matrices and principal components," 2020.

[192] S. Jain, V. Krishnamurthy, M. Rangaswamy, B. Kang, and S. Gogineni, "Radar clutter covariance estimation: A nonlinear spectral shrinkage approach," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[193] J. Smith and A. Shaw, "Unit circle roots property for sensor array signal processing," in *NAECON 2021 - IEEE National Aerospace and Electronics Conference*, 2021, pp. 210–216.

[194] J. Smith, A. Shaw, and A. Hassanien, "A new approach to moving target detection using unit circle roots constrained adaptive matched filter," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, pp. 1091–1097.

[195] W. Melvin, "Space-time adaptive radar performance in heterogeneous clutter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 2, pp. 621–633, 2000.