UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

QUESTION-ANSWERING FOR SEGMENT RETRIEVAL ON PODCAST TRANSCRIPTS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

ANDREW ELARYAN

Norman, Oklahoma

2022

QUESTION-ANSWERING FOR SEGMENT RETRIEVAL ON PODCAST TRANSCRIPTS

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Dr. Christan Grant, Chair

Dr. Dean Hougen

Dr. Amy McGovern

Dedication

This thesis is dedicated to my advisor Dr. Christan Grant for his guidance these past few years. Thank you for pushing me out of my comfort zone, for believing in my abilities, and, most importantly, for your patience.

I would also like to thank my wonderful committee members, Dr. Dean Hougen and Dr. Amy McGovern. Dr. Hougen introduced me to research my sophomore year, sparking an initial interest in the field of machine learning which has stayed with me to this day. Dr. McGovern's fantastic Artificial Intelligence class provided me with the theoretical background that was monumental in my ability to properly review and understand the literature behind this thesis. Thank you both.

Thank you to all of the friends I've made these past five years at OU, as well as my peers in the School of Computer Science and Department of Mathematics. Specifically, I'd like to thank Keegen Hart and Trey Crump, who are currently working on defending Masters theses of their own, for helping me navigate this entire process and motivating me to make this work the best that I can.

I would like to thank my family for supporting me in my education and providing me the opportunity to pursue the life that I want. Although Oklahoma is quite far away, and California will be even further, I know that I'll always be a phone call away from home in Virginia.

Finally, I would like to thank the fantastic canines in my life, Brad and Giorgio, may he rest in peace, for their unwavering loyalty, priceless companionship, and intellectual contributions to this thesis.

Acknowledgements

The computing for this project was performed at the OU Supercomputing Center for Education & Research (OSCER) at the University of Oklahoma (OU). Thank you to Henry Neeman, Horst Severini, Jason Speckman, and all of the wonderful OSCER staff members for their valuable technical expertise. Thank you to NIST and the TREC Podcast Track maintainers for providing the dataset, test topics, and task guidelines for this project. Thank you to Spotify for providing their transcripts, transcript indices, and baseline results. Thank you to the developers and maintainers of the Bertserini library, which was used extensively for retrieval.

Table of Contents

List Of Tables

List Of Figures

Abstract

Podcasting has rapidly ascended as one of the primary forms of spoken-word media in the 21st century. The Spotify Podcast Dataset has compiled transcripts of over 100,000 podcast episodes, making it one of the largest repositories of spoken word data. The segment retrieval task aims to find the most relevant segments to a given query from the set of episode transcripts. This thesis presents a two-stage approach to segment retrieval using an end-to-end question-answering (QA) deep learning architecture with an additional step to expand answers to segments. Standard BM25 retrieval on an index of predetermined segments from each episode serves as a baseline retrieval system. Experiments for both approaches involved producing and evaluating a ranked list of 20 relevant segments for 50 test topics. Comparison between the two retrieval methods shows that the QA retriever trails the baseline in nDCG@10 by 0.128, precision@10 by 0.184, and average segment relevance score by 0.461. QA retrieval slightly outperforms the baseline by 0.024 in recall@10 while slightly underperforming it by 0.102 in average segment relevance score when discounting irrelevant segments. The results suggest that the QA retrieval approach in this thesis can adequately identify and rank relevant segments within a relevant input text. However, for some queries, it may struggle to find enough relevant candidate documents during the first stage of retrieval. QA retrieval shows promise in handling informational queries for the user goal of answering a question. Future work includes improving processes such as candidate document retrieval, answer span expansion, and data annotation.

Chapter 1

Introduction

1.1 Motivation

The advent of the Internet has catalyzed the rapid expansion of novel entertainment mediums. One such medium that gained prominence in the late 2000s, and continues to grow into the 2020s, is podcasting. Podcasts are a form of program-oriented spoken-word audio, generally available on-demand and online. Podcasts boasted an estimated 104 million monthly consumers in the United States in 2020 (Edison Research, 2020). ListenNotes, a podcast aggregator, reports over 2.7 million shows and 124 million episodes as of the start of 2022 (Listen Notes, 2022). This large, and continually growing, catalog of podcasts presents challenges in the fields of information retrieval, natural language processing, and search.

1.2 Problem Statement

Conventional methods for podcast search rely on metadata such as show title, show description, episode title, and episode description. This metadata, usually provided by the podcast creator, often lacks information on the salient points of the podcast episode (Clifton et al., 2020). Evidence in the literature suggests that search methods that include the episode transcript significantly outperform those which rely solely on metadata (Yu et al., 2020). Advances in automated speech recognition have unlocked the podcast

domain as a corpus of spoken-word documents so that podcast search can be approached as a text search problem (Chelba, Hazen, and Saraclar, 2008). Thus, researchers are interested in finding effective and efficient ways to integrate podcast transcripts into the search process.

1.3 Proposed Approach

Users often turn to query-based search to find information in podcasts when they have a specific question (Besser, Hofmann, Larson, et al., 2008). They may wish to not only find a podcast episode covering the topic but the specific portion of the episode's content which answers their question as well. These segments derived from the original episode should represent an episode's discussion on the query topic while holding comprehensible meaning on their own. The task of finding relevant segments to a query from a large corpus of text is referred to as *segment retrieval* in the TREC podcasts track (Jones, Carterette, et al., 2021; Karlgren et al., 2021). Working towards this user goal motivated the development of this thesis.

Existing segment retrieval approaches focus on ranking or reranking predetermined segments (Jones, Carterette, et al., 2021; Karlgren et al., 2021; Yu et al., 2020). However, arbitrarily predefining segments based on audio time markers or word count may not capture the optimal segment for a given query. The segment may start too early, including irrelevant content, or start too late, missing out on valuable context. The same applies to segment ending times. Additionally, predefined segments are isolated from the context of the rest of their source episode. This allows for segments to be ranked highly even if the episode containing the segment is mostly irrelevant, which is not necessarily desirable.

This thesis aims to address these drawbacks by implementing a question-answering (QA) approach for segment retrieval on a large corpus of full podcast episode transcripts

and investigating its effectiveness for the task. The QA paradigm effectively models the aforementioned user journey for the segment retrieval task, aiming to extract the best answer span in relation to a given query. Recent developments in natural language processing, particularly the Transformer and BERT (Vaswani et al., 2017; Devlin et al., 2018), have advanced the state-of-the-art[1] for modern QA systems.

The answer spans extracted by existing QA pipelines are often short and lack context from the rest of the document. As such, they are not well suited to act as standalone segments for the segment retrieval task. The retrieval system implemented in this thesis adds an additional step after extraction to expand answers into longer, individually comprehensible segments using the surrounding context.

With answer expansion, a segment's location is flexible, defined by the location of the best answer in the document rather than a predefined segment. Additionally, retrieval on full episode transcripts allows for the inclusion of episode-level context in ranking segments, prioritizing relevant segments derived from relevant episodes.

Thus, this thesis proposes an extended end-to-end question-answering pipeline to perform segment retrieval on podcast episode transcripts and presents a discussion around experimental results. The contributions of this thesis are as follows:

- Segment retrieval is conducted by extracting answer spans from full podcast transcripts and expanding them to into comprehensive segments (Section 3.3). This differs from existing approaches which operate on predefined segments.

- Experiments involved segment retrieval on the set of test topics from TREC 2020 (Chapter 4). The proposed approach's effectiveness is compared to the commonly used BM25 information retrieval ranking function.

[1] `https://rajpurkar.github.io/SQuAD-explorer/`

- Analysis of experimental results identified limitations, future work, and applications to user goals for the proposed approach (Chapter 6, Chapter 7).

Chapter 2 provides a general overview of podcasts as well as the theoretical basis in ranking and language modeling for QA. Chapter 3 describes the implementation details of the proposed QA segment retrieval system. Chapter 4 specifies the methods for experimentation with results presented in Chapter 5. Chapter 6 discusses outcomes and takeaways from the experimental results, with closing remarks and ideas for future work in Chapter 7.

Chapter 2

Background

2.1 Podcasts

A major reason to study podcasts, as they relate to search, is that they provide an extensive corpus of spoken word. One prominent example, the Spotify Podcast Dataset[1], claims to be "orders of magnitude larger than previous speech corpora used for search" (Clifton et al., 2020), with over 100,000 podcast episode transcriptions representing 60,000 hours of audio. This vast collection of podcast transcripts and audio reflects the rising popularity of podcasts and provides researchers with ample data for work in numerous fields, such as search and summarization.

Podcasts may appear in a diverse set of formats and contain a variety of content (Clifton et al., 2020). Audio recordings classified as podcasts do not all come in the same shape and size. They can be long or short, formal or informal, and consist of varying levels of audio quality. Many popular podcasts involve a discussion or interview with multiple speakers, while others are monologues by a single voice. Additionally, podcasts may also be informal and unscripted. These characteristics can lead to difficulties in transcription and challenge existing search methods (Clifton et al., 2020).

Furthermore, podcasts cover a vast array of topics, some of which are niche or obscure. This is especially relevant if the speaker uses terms that are out-of-vocabulary for a speech model but are related to the podcast episode's subject (Chelba, Hazen, and

[1] https://podcastsdataset.byspotify.com/

Saraclar, 2008). In fact, these are often the terms that provide the finest granularity in search. Thus, podcasts can serve to test search methods for contexts that are not in the domain of the evaluating model.

Blogs are often referred to as a comparison for podcasts, as user goals for blog search and podcast search show similarities (Besser, Hofmann, Larson, et al., 2008; Jones, Zamani, et al., 2021; Jones, Carterette, et al., 2021). Podcasts "can be viewed as audio blogs" (Jones, Zamani, et al., 2021), with both mediums falling under the category of user-generated content for amateur producers. However, most other user-generated content is characterized by short documents (Weerkamp, Balog, and Rijke, 2009). Podcasts are generally long-form content, leading to wordier transcripts. Longer podcast transcripts may challenge information retrieval methods and models originally intended for use with shorter documents, such as blogs (Jones, Zamani, et al., 2021).

Users may wish to search for information both on the repository of podcast episodes and within a podcast episode itself. This latter goal is especially relevant for podcasts measured on the scale of hours, as it can be difficult to pinpoint the location of information in an audio stream. Some creators add timestamps in the description of their episode to delimit topics or chapters, but these are not guaranteed to reflect the granularity of information that a user is searching for. The desire for specific information from a podcast, rather than combing through an entire episode, serves as the motivation for exploring segment retrieval, the discovery of information within an episode.

2.2 Ranking Evaluation

To properly evaluate a search engine or recommendation system, it is important to assess the results provided by the system through multiple metrics. In this thesis, the focus of evaluation is the retrieval system's effectiveness, which "measures the ability of the

search engine to find the right information" (Croft, Metzler, and Strohman, 2010). This process involves comparing the ranking of documents retrieved by the search engine with ground-truth relevancy judgments from human evaluators (Croft, Metzler, and Strohman, 2010). The following sections discuss three common effectiveness metrics for search evaluation: precision, recall, and normalized discounted cumulative gain, all of which are suggested measures for evaluating podcast search (Jones, Zamani, et al., 2021).

2.2.1 Precision and Recall

Precision and recall assume binary relevance, meaning that a retrieved document is seen as either relevant or not relevant. The set A is defined to include all relevant documents, while the set B is the set of documents actually retrieved. Then, precision and recall are defined as follows (Croft, Metzler, and Strohman, 2010):

$$\text{Precision} = \frac{|A \cap B|}{|B|} \tag{2.1}$$

$$\text{Recall} = \frac{|A \cap B|}{|A|} \tag{2.2}$$

Thus, "precision is the proportion of retrieved documents that are relevant", while "recall is the proportion of relevant documents that are retrieved" (Croft, Metzler, and Strohman, 2010).

Since the goal of podcast segment retrieval is not to find all possible relevant documents, and instead to return a ranked list of a specified length with the most relevant documents early on in the list, it is useful to evaluate precision and recall at different rank positions (Croft, Metzler, and Strohman, 2010). For example, if the entire returned document list is of length 20, one can evaluate the precision and recall of sublists, starting from rank position 1, of length 5, 10, 15, and 20, the entire list. This method of

evaluation helps to differentiate the ranking effectiveness of the top-ranked documents from the effectiveness over the entire set of retrieved documents. Note that precision can increase and decrease as rank position increases, but recall is monotonically increasing.

2.2.2 Normalized Discounted Cumulative Gain (nDCG)

In practical use of a search engine, the documents at the top of the ranked list carry more weight than those at the bottom. For cases such as web search, question-answering, and navigational search, it is unlikely that a user will look at all retrieved documents, and they may only look at the top one or two (Croft, Metzler, and Strohman, 2010). A relevant segment that is highly ranked during retrieval is much more useful than a relevant segment that shows up later in the rankings. Evaluating precision and recall at earlier ranking positions can help with examining document retrieval performance with an emphasis on the relevancy of earlier documents.

However, document relevancy may not be a binary function, as it is treated in calculating precision and recall, and some documents may be classified as more relevant than others. For example, a highly ranked document at a given rank position should contribute more to the search engine's effectiveness score than a less relevant segment at the same position. Thus, the normalized discounted cumulative gain metric (nDCG) is used to evaluate the non-binary relevance of retrieved segments along with the accuracy of their retrieved order.

Discounted cumulative gain aims to measure the cumulative effectiveness of a ranking list up to a specified ranking position using human-judged relevancy grades. It is calculated as follows (Croft, Metzler, and Strohman, 2010):

$$\mathrm{DCG}_p = \mathrm{rel}_1 + \sum_{i=2}^{p} \frac{\mathrm{rel}_i}{\log_2 i} \tag{2.3}$$

where $p \geq 2$ is the ranking being evaluated and rel_i is a human graded relevance score for the document at rank position i. The denominator $\log_2 i$ is used to reduce the impact of later documents on the ranking's DCG score. This reduction factor can be modified to increase or decrease the discounting rate in the function (Croft, Metzler, and Strohman, 2010).

The ideal DCG, or IDCG, is calculated the same way as standard DCG. However, the ranking list is sorted in decreasing relevance order (Croft, Metzler, and Strohman, 2010). This is the hypothetical perfect relevance order for this set of retrieved documents. IDCG is used to normalize the nDCG score among retrieval runs that retrieve different numbers of relevant documents. Thus, nDCG can be seen as a measure of how closely the retrieved ranking order resembles the perfect ranking order of those documents. nDCG is calculated as follows (Croft, Metzler, and Strohman, 2010):

$$\text{nDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p} \tag{2.4}$$

2.3 BM25 Ranking

BM25, or Best-Match 25, is a probabilistic document scoring function, ranking documents by the probability that they are relevant to a given query (Robertson, Zaragoza, et al., 2009). As a bag-of-words model, BM25 does not take into account word order, punctuation, or grammar, instead operating solely on terms and their frequencies. This function improves upon other bag-of-words search approaches, such as TF-IDF, by normalizing term frequency and document length. This thesis uses the implementation from the Java search engine library Lucene[2] (Robertson, Zaragoza, et al., 2009; Pérez-Iglesias et al., 2009), as follows:

[2] `https://lucene.apache.org`

$$\text{BM25}\,(q,d) = \sum_{t \in q} \frac{\text{tf}\,(t,d)}{k_1 \left((1-b) + b \frac{|d|}{\text{AVG}(D)} \right) + \text{tf}\,(t,d)} \cdot \text{idf}\,(t) \tag{2.5}$$

with

$$\text{idf}\,(t) = \log \frac{N - \text{df}\,(t) + 0.5}{\text{df}\,(t) + 0.5} \tag{2.6}$$

where t is a term in q, the query; $tf\,(t,d)$ is the term frequency of t in the document d, where d is a document in the collection D; N is the number of documents in the collection; $df\,(t)$ is the document frequency of the term.

The parameter $k_1 > 0$ defines an asymptotic limit to the contribution of any one term to the ranking score (Robertson, Zaragoza, et al., 2009). This is known as saturation and normalizes the effects of high-frequency terms. A higher k_1 means that subsequent term occurrences will continue to have a high impact on the score. Since the asymptote will be the same for all terms, the asymptote's actual value does not matter so much as the rate at which additional term occurrences increase the score (Robertson, Zaragoza, et al., 2009).

The parameter $b \in [0,1]$ handles document length normalization, as seen in the component $(1-b) + b\frac{|d|}{\text{AVG}(D)}$ in Equation 2.5 (Robertson, Zaragoza, et al., 2009). Setting $b = 0$ will turn off document-length normalization, while $b = 1$ will include full normalization.

2.4 Transformer Models for Natural Language Tasks

2.4.1 Encoder-Decoder Architecture

The encoder-decoder architecture is a framework for solving computing tasks and is often used in computer vision and natural language processing. Given some input data, an encoder is tasked with reading the input and representing it as a fixed length vector, generally of lesser dimension than the input sequence, to represent features (Bahdanau, Kyunghyun Cho, and Bengio, 2014). The decoder, in turn, reads in the encoded feature vector and produces some desired output, depending on the task (Bahdanau, Kyunghyun Cho, and Bengio, 2014).

An example of an encoder-decoder model for machine translation from English to French is presented in KyungHyun Cho et al. (2014). The encoder takes as input a variable-length sentence in English and encodes it as a fixed-length vector. The decoder takes as input the encoder's fixed-length vector and translates it to a variable length output in French autoregressively, one word at a time (KyungHyun Cho et al., 2014; Bahdanau, Kyunghyun Cho, and Bengio, 2014). Since the size and structure of the encoder input and decoder output are isolated by the encoded sequence, the encoder-decoder architecture can handle differences in input and output sequence length, as well as different syntax and grammar rules between languages (KyungHyun Cho et al., 2014).

The example described above uses recurrent neural networks, RNNs, and convolutional neural networks, CNNs, in both the encoder, to learn to identify features from the English text, and the decoder, to learn to translate features into the target language, French (KyungHyun Cho et al., 2014). Note that the details of these networks are out of scope for this thesis since the focus is placed on the improvements that the attention and self-attention mechanisms provide, as described below.

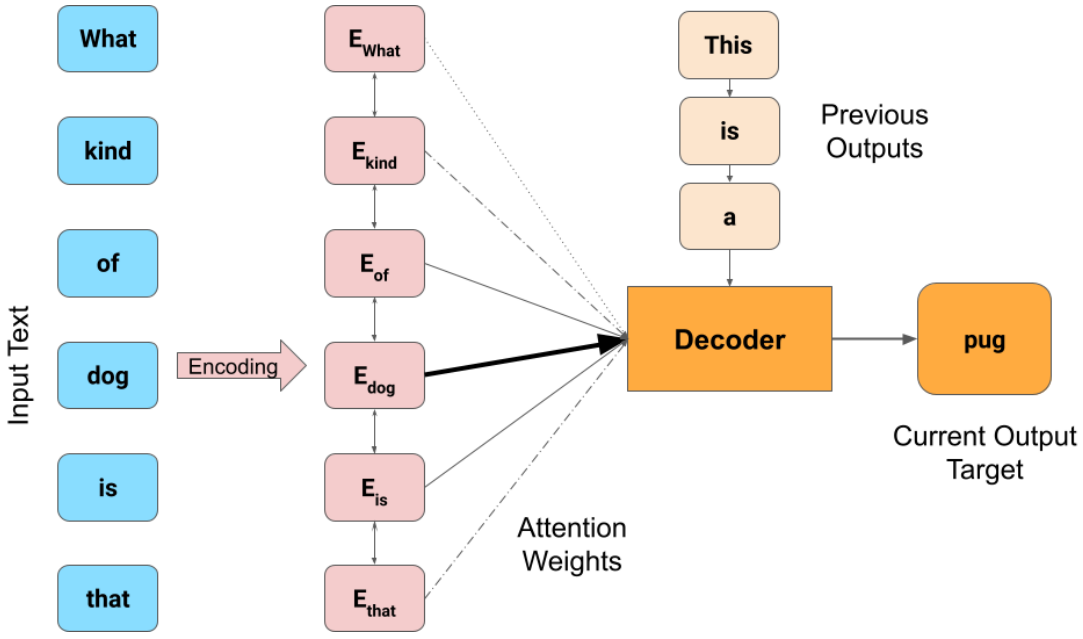

Figure 2.1: Encoder-decoder model using attention. Attention weights are represented by arrow density. Adapted from (Bahdanau, Kyunghyun Cho, and Bengio, 2014).

2.4.2 Attention

The single fixed-length vector output by encoders posed issues for improving performance in language processing tasks. In the machine translation example above, performance rapidly decreased as sequence length increased (KyungHyun Cho et al., 2014). Investigation of this decline suggested that long sequences were difficult for the system since the encoder has to "compress all the necessary information of a source sentence into a fixed-length vector" (Bahdanau, Kyunghyun Cho, and Bengio, 2014).

The attention mechanism was introduced to alleviate the bottleneck of encoding a fixed-length vector in the encoder-decoder architecture (Bahdanau, Kyunghyun Cho, and Bengio, 2014). Encoders and decoders using attention showed significant improvement over existing models in the machine translation task for long sentences (Bahdanau, Kyunghyun Cho, and Bengio, 2014). Figure 2.1 shows an overview of attention in the encoder-decoder architecture.

Rather than using a single vector to represent the entire input, an encoder in a model using attention encodes a sequence of vectors at each token to represent the input (Bahdanau, Kyunghyun Cho, and Bengio, 2014). For example, an encoder operating on an input sentence would encode each word in the sentence into a neural network. Bahdanau, Kyunghyun Cho, and Bengio (2014) use a bidirectional neural network so that each word's encoding includes context from both the previous and following words, with a focus on nearby words.

Decoders for language tasks make predictions token by token, rather than trying to predict an entire output sentence at once. Thus, at each prediction step i, the word being predicted has access to the previously predicted word, as well as the state of the previous hidden layer in the neural network.

Since the decoder's input is a sequence of vectors, the decoder can weigh how much a particular vector contributes to deciding a new network state (Bahdanau, Kyunghyun Cho, and Bengio, 2014). This is the actual attention mechanism. In the translation example, the decoder is determining how related the current word in the target language is to each word in the source language (Bahdanau, Kyunghyun Cho, and Bengio, 2014) and, therefore, how much it should attend to them.

When determining word y_i in the output sequence, there is a probability α_{ij} that y_i is related to word x_j in the input sequence. In Bahdanau, Kyunghyun Cho, and Bengio (2014), the vector of probabilities, the attention vector, is computed by applying the softmax function to a feedforward neural network jointly trained with the rest of the system. This is known as *additive attention* (Vaswani et al., 2017). Note that this is only one way of computing attention and does not represent the only approach.

With the encoded representation of x_j denoted as h_j, the result of the attention mechanism is the sum of the encoded words weighted by the attention vector, producing

a context vector. In Bahdanau, Kyunghyun Cho, and Bengio (2014), the context vector for y_i is computed as follows:

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{2.7}$$

where T_x is the length of the input sequence. Prediction of the hidden state for word y_i is a function of the context vector, the previous hidden state, and the previous word. This state is used to compute a probability distribution over the set of possible words, followed by the system choosing the word with the highest probability (Bahdanau, Kyunghyun Cho, and Bengio, 2014).

In the case of attention, the machine learning term attention aligns well with the concept of human attention. Certain parts of the decoder's output sequence are "paying attention" to specific parts of the input sequence vectors from the encoder (Bahdanau, Kyunghyun Cho, and Bengio, 2014).

2.4.3 Self-attention and the Transformer

While adding the attention mechanism into the encoder-decoder architecture improved performance on long input sequences, the use of RNNs and CNNs held some drawbacks to further improvements. Models using RNNs process neural layers sequentially, where the state at time t is dependent on the state at time $t - 1$. Their sequential computation limits models' abilities to parallelize computation, making them difficult to train without struggling with memory constraints. Additionally, it becomes expensive to map relationships from token dependencies that are distant in the input or output text, which is an issue when trying to model longer sequences (Vaswani et al., 2017).

The arrival of transformer models sparked a paradigm shift in the NLP world. Proposed by Google Brain and Google Research in the paper "Attention Is All You Need"

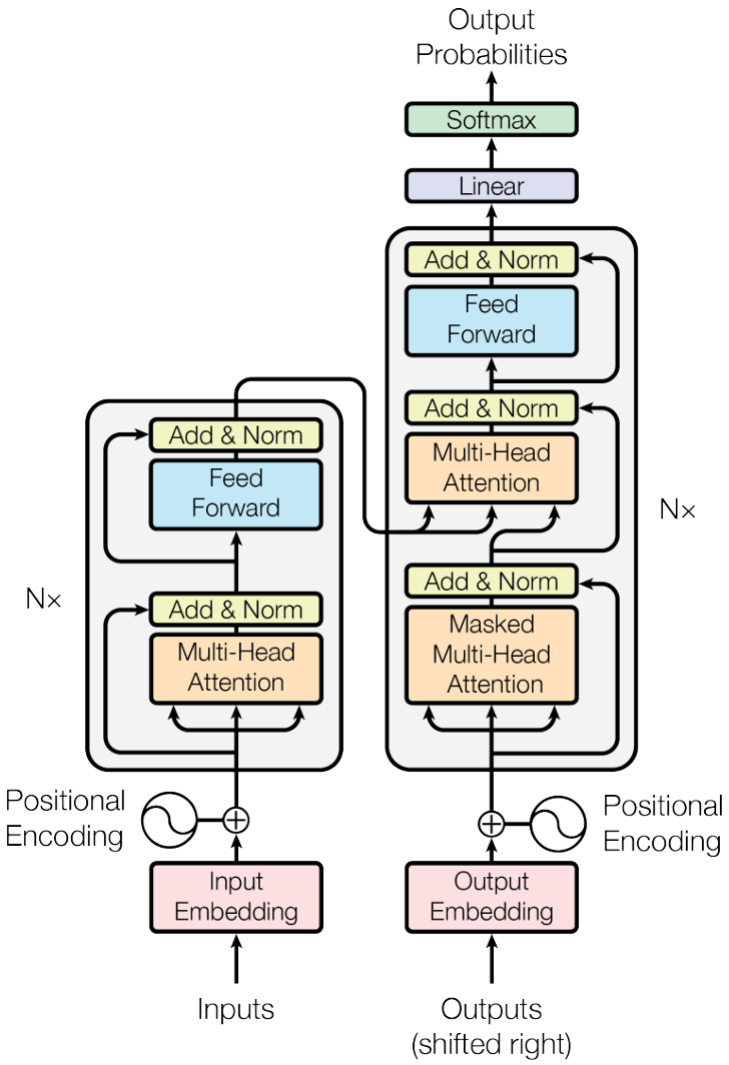

Figure 2.2: The encoder-decoder architecture for the Transformer. Reproduced directly from Vaswani et al. (2017).

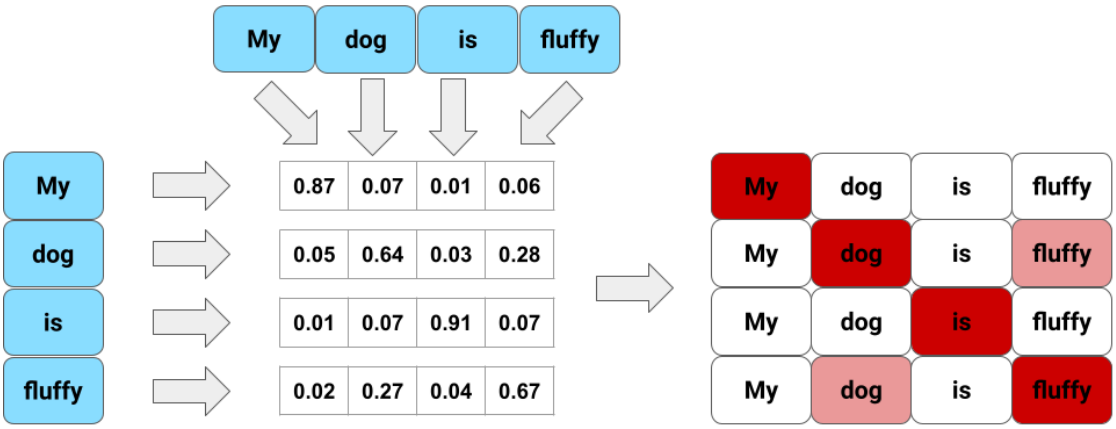

Figure 2.3: Illustration of self-attention. Adapted from (Vaswani et al., 2017)

(Vaswani et al., 2017), the Transformer was the first transduction language model to rely solely on attention mechanisms. Specifically, the Transformer follows the encoder-decoder architecture but adds the mechanism of self-attention to "relat[e] different positions of a single sequence in order to compute a representation of the sequence" (Devlin et al., 2018). Figure 2.2 presents the original Transformer architecture published by Vaswani et al. (2017). The rest of this section will focus on the application of attention in the Transformer.

Where earlier encoder-decoder models with attention mechanisms focused solely on a decoder's ability to pay attention to parts of the encoded representation, the Transformer supplements this with a self-attention mechanism that allows a sequence to attend to itself during the encoding and decoding stages. The vector representation of the sentence itself encodes the relationship between the words in the sentence. Figure 2.3 illustrates the self-attention mechanism at a high level.

2.4.3.1 Computing Attention

An attention function "map[s] a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors" (Vaswani et al., 2017). These queries, keys, and values are vectors derived from the input (Vaswani et al., 2017). To perform computation simultaneously for a set of queries, the query, key, and value vectors are packed into the matrices Q, K, and V, respectively (Vaswani et al., 2017).

The Transformer uses a *scaled dot-product* function, as opposed to the additive attention described by Bahdanau, Kyunghyun Cho, and Bengio (2014), to compute attention (Vaswani et al., 2017):

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.8}$$

where d_k is the dimension of the queries and keys. Taking the dot-product of Q and K finds the compatibility of each key with each query. Since the softmax function struggles with large dot products (Vaswani et al., 2017), this result is scaled down. Then, the softmax function is applied to produce a probability distribution. This probability distribution is multiplied by the value matrix V to produce a set of value vectors weighted by their respective attention coefficients.

2.4.3.2 Multi-Head Attention

The Transformer can create a more powerful language representation by using multiple attention heads. An attention head is a block that performs the attention mechanism as described in Equation 2.8. However, each attention head operates on a different, learned linear projection of the matrices Q, K, and V. Operating on multiple projections allows each attention head to focus on "different representation subspaces at different positions"

(Vaswani et al., 2017). As such, each head attends to more fine-grained linguistic features that may be lost when averaging attention over the entire sequence.

Each attention head, head_i, learns parameter matrices W_I^Q, W_I^K, and W_I^V. Then, the attention for each head is computed as (Vaswani et al., 2017):

$$\text{head}_i = \text{Attention}\left(QW_I^Q, KW_I^K, VW_I^V\right) \tag{2.9}$$

with the full multi-head attention block as follows (Vaswani et al., 2017):

$$\text{MultiHead}\left(Q, K, V\right) = \text{Concat}\left(\text{head}_1, ..., \text{head}_h\right) W^O \tag{2.10}$$

where h is the number of attention heads and W^O is a matrix that re-projects the concatenated attention heads to the output dimensions.

Analysis of attention heads in the Transformer and in the Transformer-based model BERT, described later in this chapter, has shown that different heads attend to different linguistic features (Vaswani et al., 2017; Clark et al., 2019). Most substantially, BERT pays attention to syntax such as punctuation or relative position. However, some individual heads show understanding of deeper language concepts such as noun modifiers, direct objects, and coreferent mentions (Clark et al., 2019). No single head is able to capture an overall representation of language details, further solidifying the usefulness of multiple attention heads in creating a deep understanding of language.

There are three distinct types of attention heads in the Transformer (Vaswani et al., 2017) represented by the three blocks of attention in Figure 2.2. The encoder's attention heads take Q, K, and V from the previous layer of the encoder, or the input sequence. The decoder has two types of attention heads. One set of attention heads attends to all positions in the decoded output sequence to the left of the current position using a masked version of multi-head attention. There is also a traditional encoder-decoder

attention head in the decoder which attends to Q from the previous layer in the decoder, but takes K and V from the encoder's output.

2.4.3.3 Benefits of Self-Attention

There are three main benefits to using self-attention over recurrence or convolution. First, self-attention performs a constant amount of work per layer, whereas recurrent layers operate in linear time (Vaswani et al., 2017). If the sequence length is shorter than its dimensionality, as is the case in WordPiece (KyungHyun Cho et al., 2014) used in BERT (Devlin et al., 2018), self-attention will work faster than a recurrent layer. Second, since recurrent networks are inherently sequential, they are limited in their ability to parallelize within training examples (Vaswani et al., 2017). The Transformer, on the other hand, is able to parallelize work among its multiple attention heads for a single sequence. Third, self-attention keeps the maximum path length of dependencies between positions constant (Vaswani et al., 2017). In contrast, the path lengths in convolutional and recurrent networks grow with the number of layers. Thus, the Transformer is better able to learn long-range dependencies in a sequence.

2.4.4 Bidirectional Encoder Representations from Transformers

The benefits of self-attention and the Transformer allowed researchers to build more powerful models with deeper understanding of language. One of the preeminent language models based on the Transformer is the Bidirectional Encoder Representations from Transformers (BERT) model developed by Google AI Language (Devlin et al., 2018).

2.4.4.1 BERT's Architecture

BERT's representation of input text has two main structures: the sentence and the sequence. Any continuous interval in the input can be treated as a sentence, so while

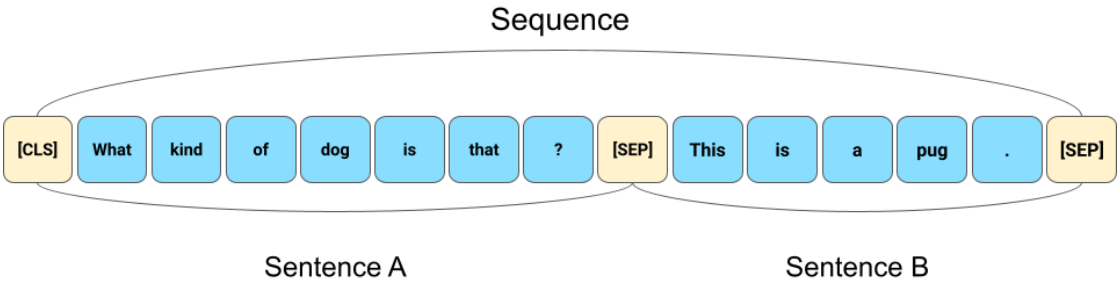

Figure 2.4: A BERT sequence representing a pair of contiguous sentences. Adapted from (Devlin et al., 2018).

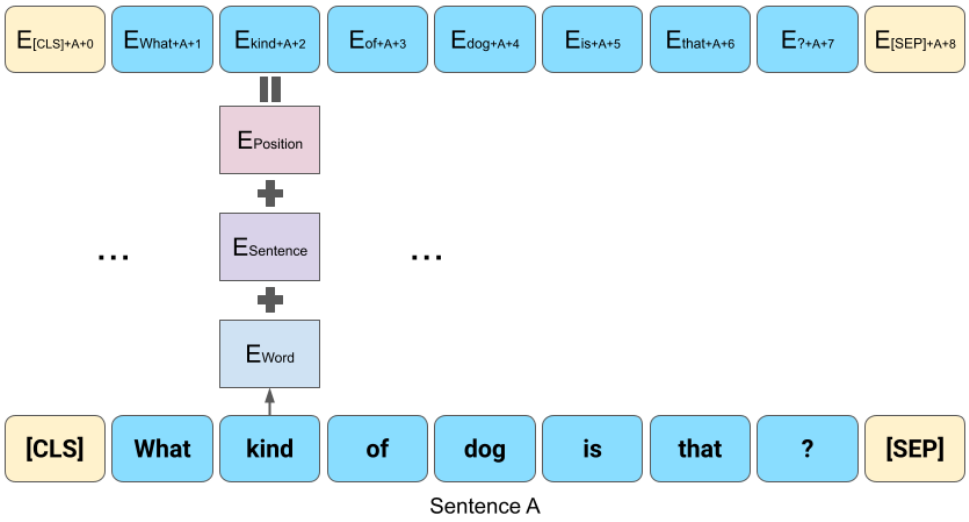

Figure 2.5: The input representation in BERT for a sequence containing one sentence. Adapted from (Devlin et al., 2018).

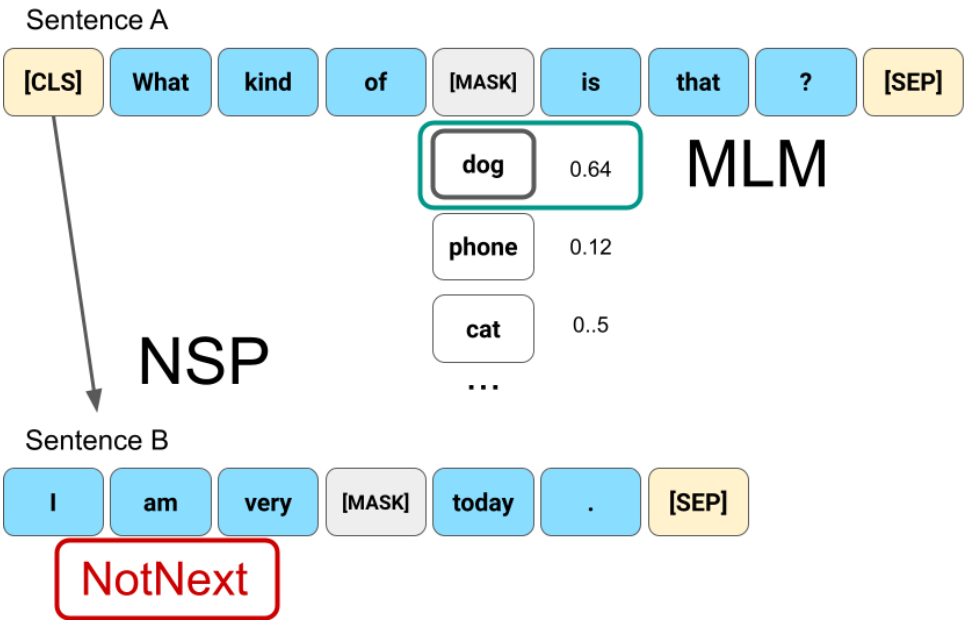

Figure 2.6: The BERT masked language modeling (MLM) and next sentence prediction (NSP) tasks were conducted simultaneously. The MLM task selects the word with the highest predicted probability, which in this case is "dog". The NSP task predicts whether or not Sentence B directly follows Sentence A, which in this case is false as indicated by the NotNext label. Adapted from (Devlin et al., 2018).

sentences may fit the linguistic definition, they do not have to. A sequence is a token sequence made up of either a single sentence or a concatenated pair of sentences. Computing on pairs of sentences is useful for the next sentence prediction pre-training objective, described later. Pairs of sentences are also useful for structuring some downstream tasks, such as pairing a question with its answer. Each sequence is prepended by a [CLS] token, while each sentence is appended with a [SEP] token. Figure 2.4 demonstrates the structure of a sequence.

The structure of a sequence embedding with one sentence is presented in Figure 2.5. Each token in the sequence, including the structural tokens [CLS] and [SEP], is represented by an embedding vector. The token embedding is made up of three parts. First, there is the word embedding, or vector representation of the word itself. BERT uses the word embeddings pre-trained by WordPiece as presented by Wu et al. (2016). Second, there is a learned sentence level embedding to differentiate a pair of sentences in a sequence, referred to as sentence A and sentence B. Third, there is a positional embedding that encodes the token's position in the sequence. The token's encoded representation is the sum of the word, sentence, and position embeddings. Thus, the full sequence's encoded representation is the sequence of these token embeddings.

BERT implements the Transformer in blocks almost identical to the original Transformer from Vaswani et al. (2017). However, it differs from the original Transformer in that BERT removes the decoder and stacks multiple layers of encoder blocks (Devlin et al., 2018). This allows BERT to provide a language model with a "unified architecture across different tasks" (Devlin et al., 2018) with minimal "task-specific architecture modifications" (Devlin et al., 2018).

Devlin et al. (2018) describe their implementation details for two model sizes. BERT Base has 12 Transformer layers, a hidden size of 768, and 12 attention heads per layer. BERT Large has 24 Transformer layers, a hidden size of 1024, and 16 attention heads

per layer. The BERT model described in Section 2.5.2 uses BERT Base as its pre-trained model.

2.4.4.2 Pre-training

Contemporary language models train for word prediction either from left to right or from right to left, since doing both would allow a word to "see itself" (Devlin et al., 2018) in the encoded representation of other words, making the task trivial (Devlin et al., 2018). BERT, however, is able to avoid this issue and pre-train bidirectionally, both from left to right and from right to left, on unlabeled text using the masked language model, or MLM, objective (Devlin et al., 2018).

In the MLM task, some of the tokens in the input are randomly masked, or concealed, and the system must use context to predict the original token. Since the predicted token is masked, the other tokens in the sequence are unable to include it in their representations. By pre-training bidirectionally, BERT builds a more powerful language model than simply concatenating a left-to-right model with a right-to-left model.

Pre-training BERT also involves a next-sentence prediction, or NSP, objective to capture the relationship between sentences (Devlin et al., 2018). This objective shows substantial benefits for downstream tasks such as QA. When creating each training example, two sentences are chosen: sentence A and sentence B. Sentence B has a 50% chance to be the true next sentence after A, and is labeled "IsNext". The rest of the time, sentence B will be a randomly chosen sentence, and be labeled "NotNext". The NSP objective, then, is to predict whether or not sentence B is the true next sentence for sentence A.

The authors of BERT used the BooksCorpus and English Wikipedia corpora, which both provide long, contiguous documents rather than shorter sentence-level texts (Devlin et al., 2018), as the input texts for the pre-training tasks. Both BERT pre-training tasks,

MLM and NSP, are conducted simultaneously. Figure 2.6 shows an overview of the pre-training tasks.

2.4.4.3 Fine-tuning and Task-specific Application

The deep, bidirectional pre-training of BERT makes it a versatile language model, as it only requires one transformer layer of fine-tuning to be suitable for a wide array of NLP tasks. Fine-tuning for a specific downstream task involves loading the pre-trained model and training the parameters in another layer using labeled data (Devlin et al., 2018), essentially acting as the decoder stage. At the time of its publishing, BERT improved the state-of-the-art in eleven such tasks, including question-answering on the dataset SQuAD v1.1 (Devlin et al., 2018). As a result, BERT has become a standard for the development of language models in QA.

2.5 End-to-end Question-Answering

While traditional information retrieval, or IR, techniques such as BM25 are useful, they generally retrieve a ranked set of raw documents (Voorhees and Tice, 2000). The goal of question-answering is to provide answers directly from the input text based on a user's question, which is a common user goal (Voorhees and Tice, 2000). QA systems should be open-domain (Voorhees and Tice, 2000), meaning that their answer extraction capabilities are not restricted to a certain set of documents.

There exist multiple tasks that fit under the umbrella of QA, though they can each be boiled down to the following goal: given a question, find the best answer, or a ranked set of answers. The answer selection task aims to choose the most fitting candidate from a predetermined list of candidate answers (Yang et al., 2019), returning the original candidate document. Reading comprehension, on the other hand, takes as input a single

Paragraph: The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from **Denmark, Iceland and Norway** who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. Through generations of assimilation and mixing with the native Frankish and Roman-Gaulish populations, their descendants would gradually merge with the Carolingian-based cultures of West Francia. The distinct cultural and ethnic identity of the Normans emerged initially in the first half of the 10th century, and it continued to evolve over the succeeding centuries.
Question: From which countries did the Norse originate?
Answer: Denmark, Iceland and Norway

Figure 2.7: Sample question and answer pair for a document in the SQuAD v1.1 development set (id: 56ddde6b9a695914005b962a).

document and aims to identify answer spans within that document (Yang et al., 2019). These two tasks assume that the set of candidate documents, or the single document, is predefined, eschewing the step of document retrieval.

A third QA task, end-to-end QA, brings the IR step back into the picture. In end-to-end QA, the goal is to find the best answer spans from a large document corpus. Due to the impracticality of "apply[ing] inference exhaustively to all documents in a corpus with current models" (Yang et al., 2019), end-to-end QA is generally implemented in two major stages. First, traditional IR is used to identify a subset of candidate documents, "restrict[ing] the input text" (Yang et al., 2019). Second, a neural model reads the candidate documents, extracting and ranking potential answers.

2.5.1 Stanford Question Answering Dataset (SQuAD)

A prominent dataset for use in reading comprehension and question-answering tasks is the Stanford Question Answering Dataset (SQuAD)[3] (Rajpurkar et al., 2016). SQuAD v1.1 consists of 107,785 pairs of questions and answers sourced from a corpus of 536 Wikipedia articles separated into 23,215 paragraphs, with up to 5 QA pairs per paragraph

[3] https://rajpurkar.github.io/SQuAD-explorer/

(Rajpurkar et al., 2016). Wikipedia offers high subject variability on a large collection of documents, making SQuAD a candidate for training and evaluating open-domain QA models.

The structure of SQuAD differs from previous question-answering datasets in that it does not offer a question and a set of predefined candidate answers. Rather, SQuAD's candidates stem from all possible spans in the source article, producing a much larger domain for candidate answers (Rajpurkar et al., 2016). Questions and answer pairs on the set of articles were sourced from human judgment using crowdworkers (Rajpurkar et al., 2016). A sample question-answer pair is presented in Figure 2.7.

SQuAD has been used to evaluate and train reading comprehension models. Reading comprehension differs from full end-to-end QA in that reading comprehension only finds answer spans in one document. However, SQuAD is used to fine-tune pre-trained language models for the second portion of the end-to-end QA task which conducts the reading and extraction of answers (Yang et al., 2019).

2.5.2 Bertserini

Bertserini[4] is an "end-to-end question answering system that integrates BERT with the open-source Anserini information retrieval toolkit" (Yang et al., 2019), specifically the Pyserini[5] (Lin et al., 2021) Python toolkit. This system provides a tool for answering questions in large text corpora end-to-end (Yang et al., 2019), as opposed to ranking shorter, pre-processed segments of input text as does answer selection.

Bertserini, as an end-to-end QA system, is structured as a two-stage pipeline. The first stage uses traditional IR, namely BM25, to identify candidate documents within the corpus. The second stage finds the optimal answer span within each document,

[4] https://github.com/rsvp-ai/bertserini
[5] https://github.com/castorini/pyserini

producing a ranked list of answers. Answers are ranked using a linear weighted sum combining the answer span's score from a BERT reader and the document's context score from BM25 retrieval (Yang et al., 2019), as follows:

$$S = (1 - \mu) \cdot S_{\text{Document}} + \mu \cdot S_{\text{Answer}} \tag{2.11}$$

where the value $\mu \in [0, 1]$ is a hyperparameter. μ has a learned default value of 0.5, meaning that the answer score and the document context score are evenly weighted (Yang et al., 2019).

Bertserini's answer extraction stage uses a pre-trained BERT model, based on the original BERT Base in Google's paper (Devlin et al., 2018). The model was fine-tuned for QA on the SQuAD v1.1 dataset (Yang et al., 2019). The authors of Bertserini provide the BERT model used for their experiments through HuggingFace[6].

Although Bertserini was developed to operate on Wikipedia articles, its open-domain nature makes it applicable to other types of documents, including podcast transcripts. The segment retrieval approach described in Chapter 3 uses Bertserini and the provided fine-tuned BERT model.

[6] https://huggingface.co/rsvp-ai/bertserini-bert-base-squad

Chapter 3

Methods

3.1 Data

This QA segment retrieval system operates on the Spotify Podcast Dataset, which is "orders of magnitude larger than previous speech corpora used for search" (Clifton et al., 2020), and includes over 100,000 podcast episode transcriptions representing nearly 60,000 hours of audio. The dataset was created through a partnership between NIST and Spotify and was distributed to participants in the TREC 2020 Podcast Track. Podcasts were uniformly sampled from Spotify's English podcast catalog and transcribed using Google's Cloud Speech-to-Text API (Clifton et al., 2020).

Each podcast episode's audio recording is transcribed into sequential, 30-second long chunks along with some metadata attributes. An example of a transcript chunk is shown in Figure 3.1. Each chunk provides a transcript of 30-seconds worth of audio, a confidence probability for the transcription's accuracy, and a list of all of the words in the chunk. Along with each word in the list are the word's start and end times. Chunks vary in size between sentence-length and paragraph-length, although they are not necessarily delimited by either. The full transcript for an episode can be obtained by concatenating the sequence of chunks in order.

3.2 Topics

```
{
  "transcript": "Hello, y'all, this is Premier
     from the fifth floor in are you tired of your
      Barber pushing your hairline back worse than
      a bad boy artist album, or maybe you wanted
     them guys to just let anybody walk up in the
     shop and Skip they line up in the seat or did
      you walk up in your Barbershop For A Part in
      Jesus Christ that man had you woke up with a
      Widow's Peak. Well, I'm here to tell you
     over a lifestyle Salon. We ain't having that.
      We ain't condoning that located at 5321.",
  "confidence": 0.8640950322151184,
  "words": [
    {
      "startTime": "3s",
      "endTime": "3.300s",
      "word": "Hello,"
    },
    {
      "startTime": "3.300s",
      "endTime": "3.800s",
      "word": "y'all,"
    },
    ...,
    {
      "startTime": "28.800s",
      "endTime": "29.800s",
      "word": "5321."
    }
}
```


Figure 3.1: Example of a transcript chunk from the Spotify Podcast Dataset.

```
<topic>
  <num>1</num>
  <query>Higgs boson</query>
  <type> topical </type>
  < description >Im looking for news and discussion about the discovery of
      the Higgs boson. When was it discovered? How? Who was involved?
      What are the implications of the discovery for physics?
  </ description >
</ topic >
```

Figure 3.2: Example of a query topic.

The training and test queries provided by TREC in 2020, hereafter referred to as topics (Jones, Carterette, et al., 2021), were used as input for the retriever. An example topic from the training set is provided in Figure 3.2. Each topic is made up of four attributes: the query number, the topic query, the type label, and the detailed description, tagged "num", "query", "type", and "description", respectively. The topic number refers to the index of the topic in the entire set of topics. Topics 1-8 are training topics, while topics 9-58 make up the test set. The keyword query is a short text containing the most important keywords or the main focus of the topic. The type label categorizes the query into one of three types: topical, refinding, or known-item. Topical queries look for "general information about the topic", refinding queries are "searching for a specific episode the user heard before", and known-item queries look to find "something that is known to exist but under an unknown name" (Clifton et al., 2020). The detailed description may provide additional information about the topic, ask specific questions, or discuss preferences regarding the retrieved segment.

shelter something or you may be in some temporary place, but it may not you know, Doug do what you have to do be willing to get dirty too many nights. We get dirty drunk roll on the floor and we don't care who we rolling around where we don't care how dirty we get so don't don't don't don't think you too good for the for detox. Don't think you too good to go through the program. Don't think you're too good to be with her the less fortunate. You're going to have to probably sit in some places for a while, but you're going to have you're going to be you're going to be happy later on. All right, so don't be afraid to talk to your church. Don't be afraid to talk to somebody within the church. Okay, if you if you have teachers or professors or someone that you could trust from your high school or your old school reach out to them reach out to our old guidance counselor you Don't forget you reach out to your local police, you know, if you're female reach out to fax the speech to speak to a female detective. I mean, these are extreme but trust me you can walk into a police station say listen. I'm a recovering addict and I just want help and I don't know where to go to. I don't have no family or may not have anyone that I could trust. Can you help me? Can you point me in the right direction? Could you get me in touch with an addiction Clinic? Could you help me out right alas? Ashley do not undercut the power of Alcoholics Anonymous or Narcotics Anonymous. If you're struggling with addiction and you get old you going to detox the first thing you should be doing I asked or even before once you make that decision is go talk to people at AA you could just you just go on Google and put in your zip code and look

Figure 3.3: Example of a retrieved segment for Topic 13 containing the keyword query "drug addiction recovery".

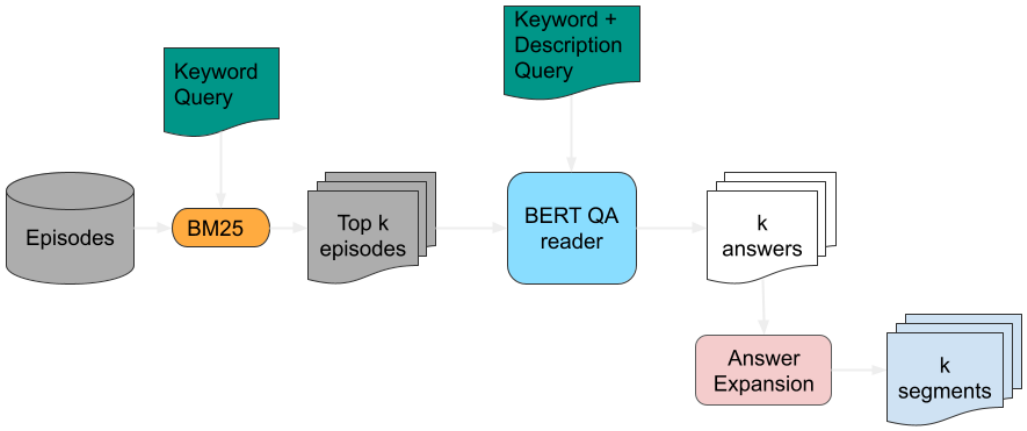

Figure 3.4: Diagram showing the two-stage QA retrieval process.

3.3 QA Segment Retrieval

This QA segment retrieval system follows the two-stage approach for end-to-end QA, as described in Section 2.5, with an additional answer expansion step, shown in Figure 3.4. The first stage uses BM25 to retrieve a set of candidate documents for a topic, while the second stage finds the best answers from that document set. Once retrieved, each answer in the ranked set is expanded by a constant number of words, resulting in a segment containing the answer with its surrounding context.

Selected sections from the QA retriever's main function are presented in Appendix A in Figure A.3. The end-to-end QA pipeline was built using components from the Bertserini library, described in Section 2.5.2. Answer expansion logic was built around the pipeline in Python to create comprehensible segments from raw answers. The QA retriever operates on a single topic, so work on the entire set of test topics can be parallelized across multiple nodes using a scatter and gather approach.

There are two reasons to use BM25 to narrow down the documents of interest before using QA. First, BM25 is much less computationally expensive than span prediction with a BERT model (Yu et al., 2020). It is infeasible to evaluate each episode's transcript using QA, so the system needs to exclude irrelevant documents. Second, BM25 has been shown to slightly outperform another prominent IR technique, query likelihood, for the segment retrieval task on this dataset (Yu et al., 2020).

In the first stage, BM25 ranks potentially relevant episodes using the keyword phrase from the topic as its query. BM25 search operates on an index of the full episode transcripts. This results in a set of the top 500 episode transcripts, ranked by their BM25 score. While other podcast segment retrieval experiments in the literature retrieved 1000 candidate segments (Yu et al., 2020), the use of longer, full episode transcripts necessitates a reduction to 500 candidates.

The second stage takes the results from the bag-of-words stage and uses QA to find a set of answers among the candidate episodes. The retriever uses the Bertserini BERT Base model and tokenizer fine-tuned for QA[1]. The input question for the QA stage is a concatenation of the topic's keyword query and detailed description. This ensures emphasis on the most important keywords while also including the more nuanced information included in the description. The result is a set of the 20 highest ranked answer spans found in the set of candidate episodes. Answers are scored by linearly weighting the BM25 score of the source episode with the answer's relevancy score.

Each answer is then expanded to a full segment to aid in evaluation and better resemble the segments evaluated in the TREC Podcast tracks, as well as providing a result with more context than a shorter answer span. To expand an answer into a segment, the answer was prepended by the previous 170 tokens and appended with the following 170 tokens. This token radius was chosen to resemble the average two-minute segment length found in the Spotify Podcast Dataset of 340 words (Clifton et al., 2020).

To facilitate this approach, two files were edited from the Bertserini library. First, the Answer class was modified to add fields for the start and end indices of the answer in the original document. Second, the BERT reader was modified to add the start and end indices to each answer produced by the reader's prediction. Additionally, each answer's original document ID was added to the answer's existing metadata field, which was previously unused. These modifications helped to accurately relocate each answer in its original document during the segment expansion process. The modified portion of each file can be seen in Appendix A in Figures A.4 and A.5.

[1] `https://huggingface.co/rsvp-ai/bertserini-bert-base-squad`

Chapter 4

Experiments

The segment retrieval process was executed on the 50 test topics provided by TREC [1]. Retrieval for each topic resulted in a set of 20 relevant segments, with the exception of Topic 21, which only produced 13 relevant segments.

In Appendix A, Figure A.1 shows the configuration file for the QA run. In this experiment, two arguments for the BERT reader were changed from their default values to better suit the longer questions resulting from the topics' detailed descriptions. The maximum query length was changed from 64 to 128 and the maximum answer length was changed from 30 to 64.

The jobs for this experiment ran at the OU Supercomputing Center for Education & Research (OSCER) at the University of Oklahoma (OU). Work was parallelized by performing retrieval for each topic individually as an independent job. This parallelization greatly reduced the runtime and memory usage of the retrieval job, as compared to retrieving all topics sequentially. Each job was completed in under 24 hours with 20 GB of memory per job. Performing single-topic retrieval also more accurately mirrors the use case for segment retrieval, as a user is likely to search for information one query at a time. A sample SBATCH file used to define a single-topic job on OSCER is presented in Appendix A in Figure A.2.

The segment retrieval baseline for this experiment consisted of solely BM25 ranking, similar to Spotify's RERANK-QUERY baseline (Yu et al., 2020). A diagram of the

[1] https://trecpodcasts.github.io/resources/podcasts_2020_topics_test.xml

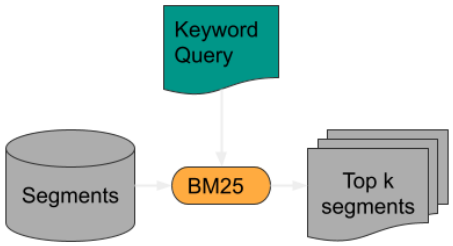

Figure 4.1: Diagram showing the baseline BM25 retrieval process on the topic's keyword query.

BM25 baseline retrieval process is shown in Figure 4.1. Rather than an index of full episode transcripts, the baseline performed retrieval on an index of pre-separated potential segments. A potential segment was defined as "a two-minute chunk starting on the minute" (Jones, Carterette, et al., 2021) in an episode. Thus, each segment included a one-minute overlap with the previous segment and the following segment. The baseline used BM25 on the topic's keyword query to retrieve the 20 most relevant segments from the index of all potential segments in the corpus. The BM25 parameters were set to $k_1 = 0.9$ and $b = 0.4$, the same values as in Spotify's RERANK-QUERY baseline (Yu et al., 2020) and the QA retriever. The execution of the baseline followed the same procedure as QA retrieval.

The evaluation process was designed to resemble the evaluation process for the TREC 2020 podcast track. Each segment was graded using the evaluation rubric from TREC 2020 (Jones, Carterette, et al., 2021), shown in Appendix B. Segments were assessed on a four-point Excellent, Good, Fair, Bad (EGFB) scale, representing scores of 3, 2, 1,

and 0, respectively. For "known-item" or "refinding" topic types, there is also a 4-point "Perfect" score. Segment evaluation was based on human judgment, specifically the assessment of the author of this thesis. Along with the topic and segment pair, the entire transcript of the segment's source episode was provided to add context.

Chapter 5

Results

5.1 Segment Relevance

Figure 5.1 shows the total number of relevant segments, omitting irrelevant segments with a score of 0, along with the topic's distribution of relevance scores for the QA run. This figure demonstrates that retrieval performance varies from topic to topic, even within the same topic type. Additionally, the distribution of relevance levels is not uniform for all topics. These observations hold for the BM25 baseline run on the segments index as well, as seen in Figure 5.3.

For example, in the QA run, the topic with the most relevant segments, Topic 36, shows a relatively even distribution of Fair, Good, and Excellent segments. Some topics, such as Topic 21, do not produce many relevant segments, but those which are retrieved are highly relevant. In contrast, topics such as Topic 28 produce a moderate number of relevant segments, but few are graded as highly relevant.

Topic Type	Total	Relevant	Percent Relevant
Topical	693	314	45.31%
Refinding	160	26	16.25%
Known-item	140	21	15.00%
All	993	361	36.35%

Table 5.1: Percentage of segments of each type from the QA run that were graded as relevant with a relevance score > 0.

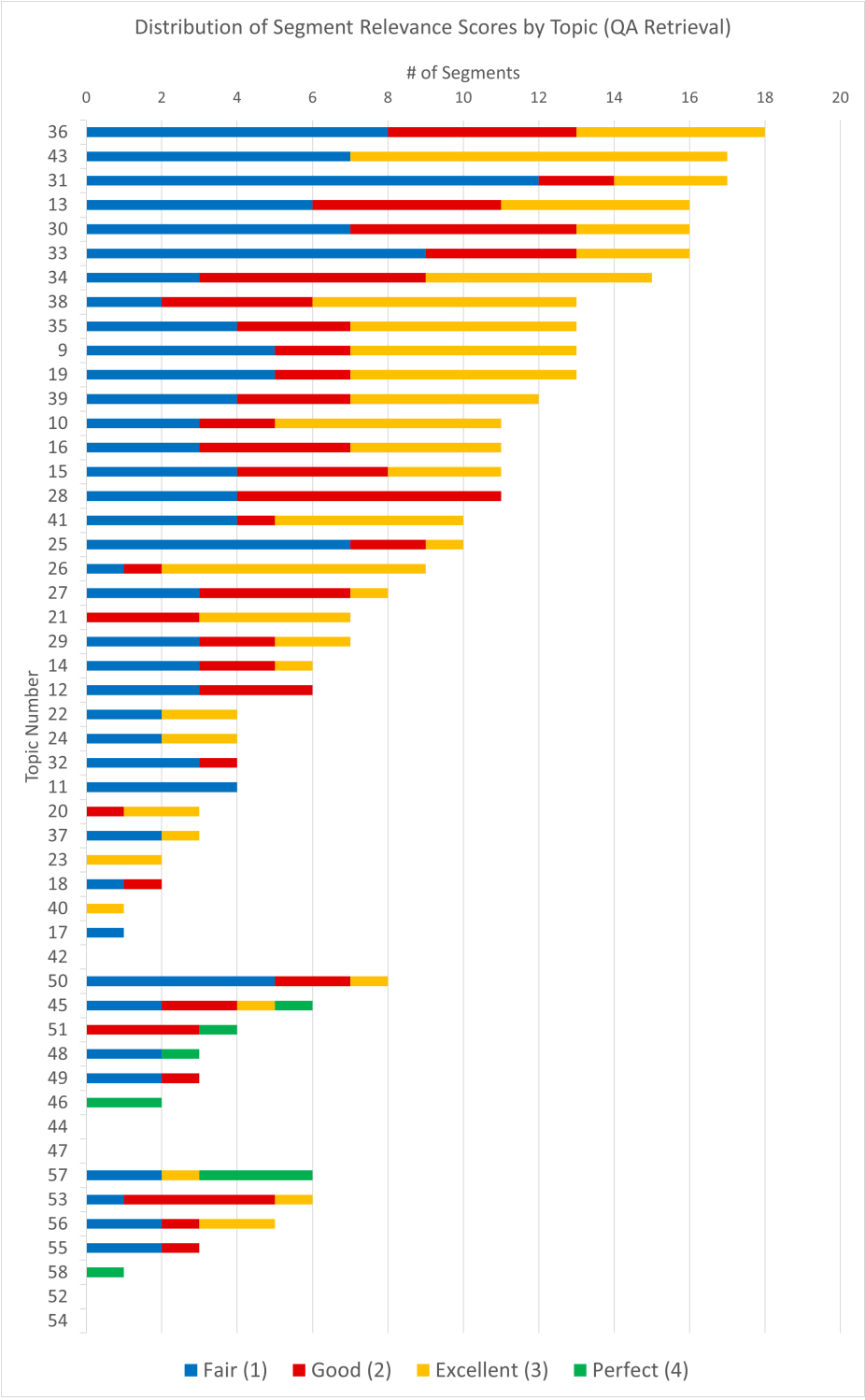

Figure 5.1: Number of relevant segments in the QA run for each topic differentiated by relevance level, with topics grouped by type: topical (36-42), refinding (50-47), and known-item (53-54).

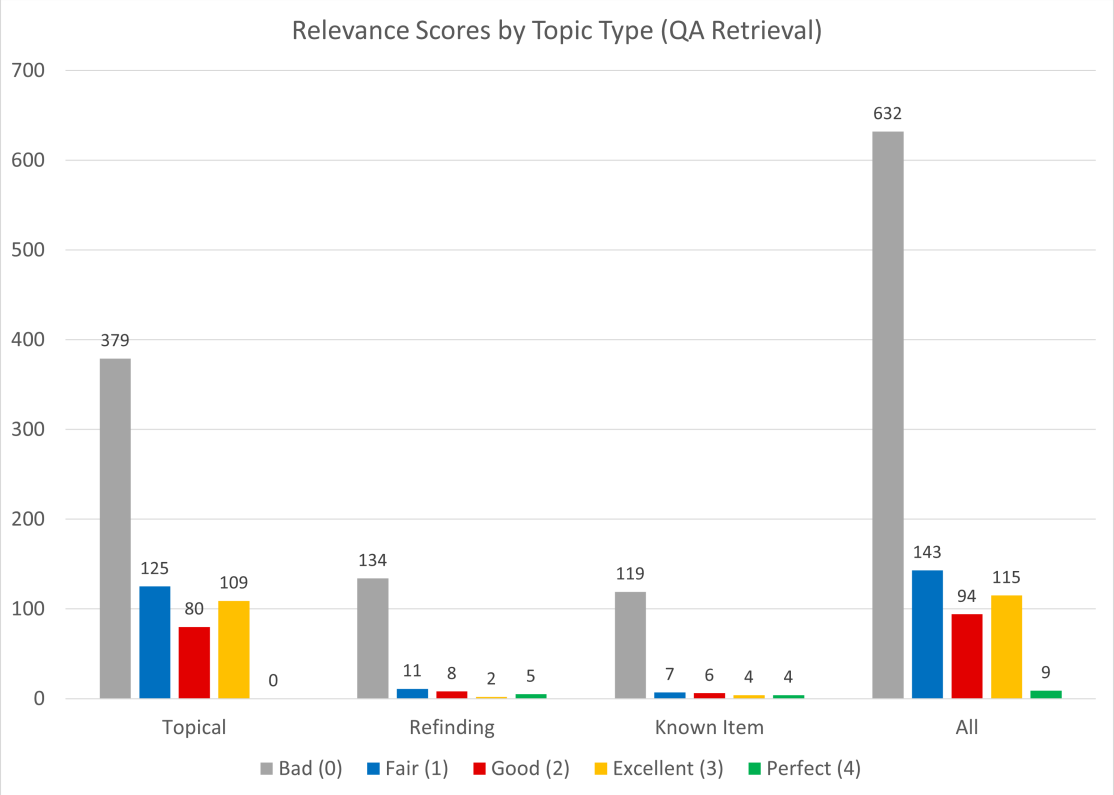

Figure 5.2: Number of segments in the QA run at each relevance level for each topic type (topical, refinding, known-item).

Topic Type	Total	Relevant	Percent Relevant
Topical	700	431	61.57%
Refinding	160	60	37.50%
Known-item	140	77	55.00%
All	1000	568	56.80%

Table 5.2: Percentage of segments of each type from the baseline that were graded as relevant with a relevance score > 0.

Topic Type	Avg. Score	Std. Dev.	Avg. Score (Relevant)	Std. Dev. (Relevant)
Topical	0.8831	1.130	1.949	0.8618
Refinding	0.3313	0.8635	2.038	1.172
Known-item	0.3357	0.9072	2.238	1.109
All	0.7170	1.093	1.972	0.9045

Table 5.3: Average scores and standard deviations for segments of each type retrieved with QA, including and excluding irrelevant segments.

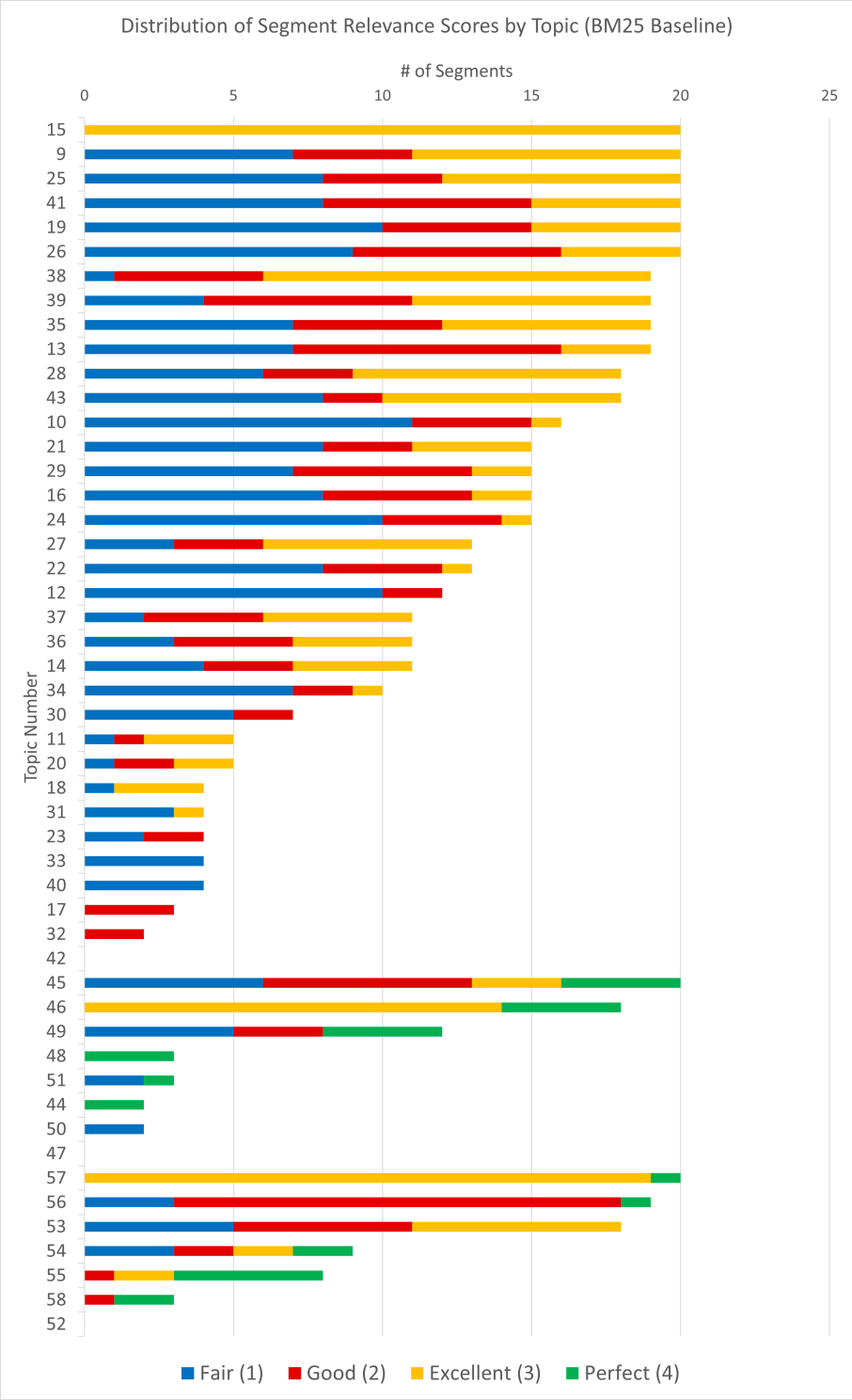

Figure 5.3: Number of relevant segments in the baseline run for each topic differentiated by relevance level, with topics grouped by type: topical (15-42), refinding (45-47), and known-item (57-52).

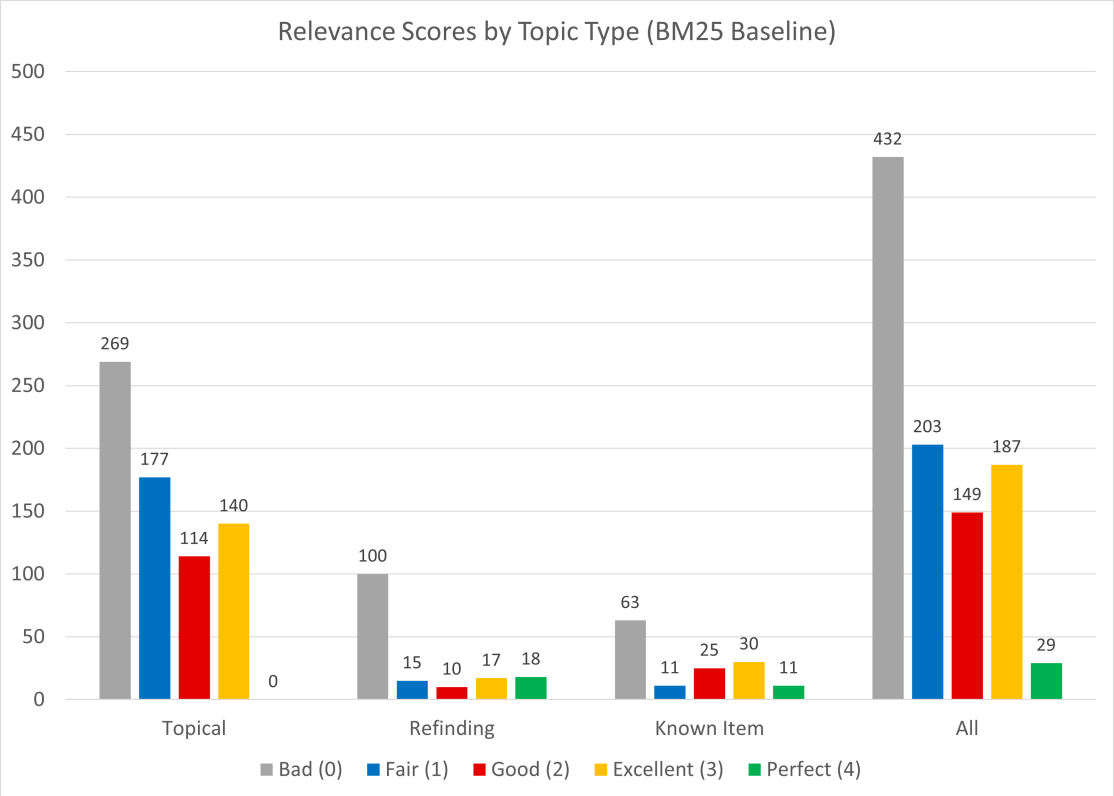

Figure 5.4: Number of segments in the baseline run at each relevance level for each topic type (topical, refinding, known-item).

Topic Type	Avg. Score	Std. Dev.	Avg. Score (Relevant)	Std. Dev. (Relevant)
Topical	1.179	0.5090	1.914	0.8287
Refinding	0.9875	1.458	2.633	1.128
Known-item	1.393	1.428	2.532	1.118
All	1.178	0.5927	2.074	0.9034

Table 5.4: Average scores and standard deviations for segments of each type retrieved with the BM25 baseline, including and excluding irrelevant segments.

Run	nDCG@20	nDCG@10	Prec.@20	Prec.@10	Recall@20	Recall@10
QA	0.619	0.492	0.361	0.426	0.900	0.556
Baseline	0.745	0.620	0.568	0.610	0.940	0.532

Table 5.5: nDCG, precision, and recall results at ranking depths of 20 and 10 for the QA and baseline runs.

For each topic type, there were many irrelevant retrieved segments. Table 5.1 shows that just over a third of graded segments for the QA run were found to be relevant, using a binary relevance judgment, and Table 5.2 shows just over half the segments retrieved by the baseline to be relevant. In both the QA run and the baseline, topical queries showed a higher percentage of their segments to be relevant than refinding and known-item queries.

Table 5.3 shows that the average relevance in the QA run for topical queries is close to a fair score of 1, whereas refinding and known-item queries have an average close to 0, or Bad relevance. Isolating analysis to only the set of relevant segments shows that all topic types have a Good average score of around 2. Table 5.4 displays the same information as Table 5.3 for the baseline run. With baseline retrieval, known-item types boasted the highest average score, with topical and refinding types trailing. Here, topical and refinding queries had an average score around or slightly above 1, or Fair, with known-item types showing a score closer to 1.5. Isolating for only relevant segments, the relevance for topical queries moves closer to 2, or Good, and refinding and known-item relevance jumps to over 2.5, almost an Excellent rating. Differences between topic types are discussed further in Section 6.1.

5.2 Ranking Effectiveness

However, inspecting only the average score for the population, topic type, or a single query ignores information about the internal ranking of segments for a topic, which is

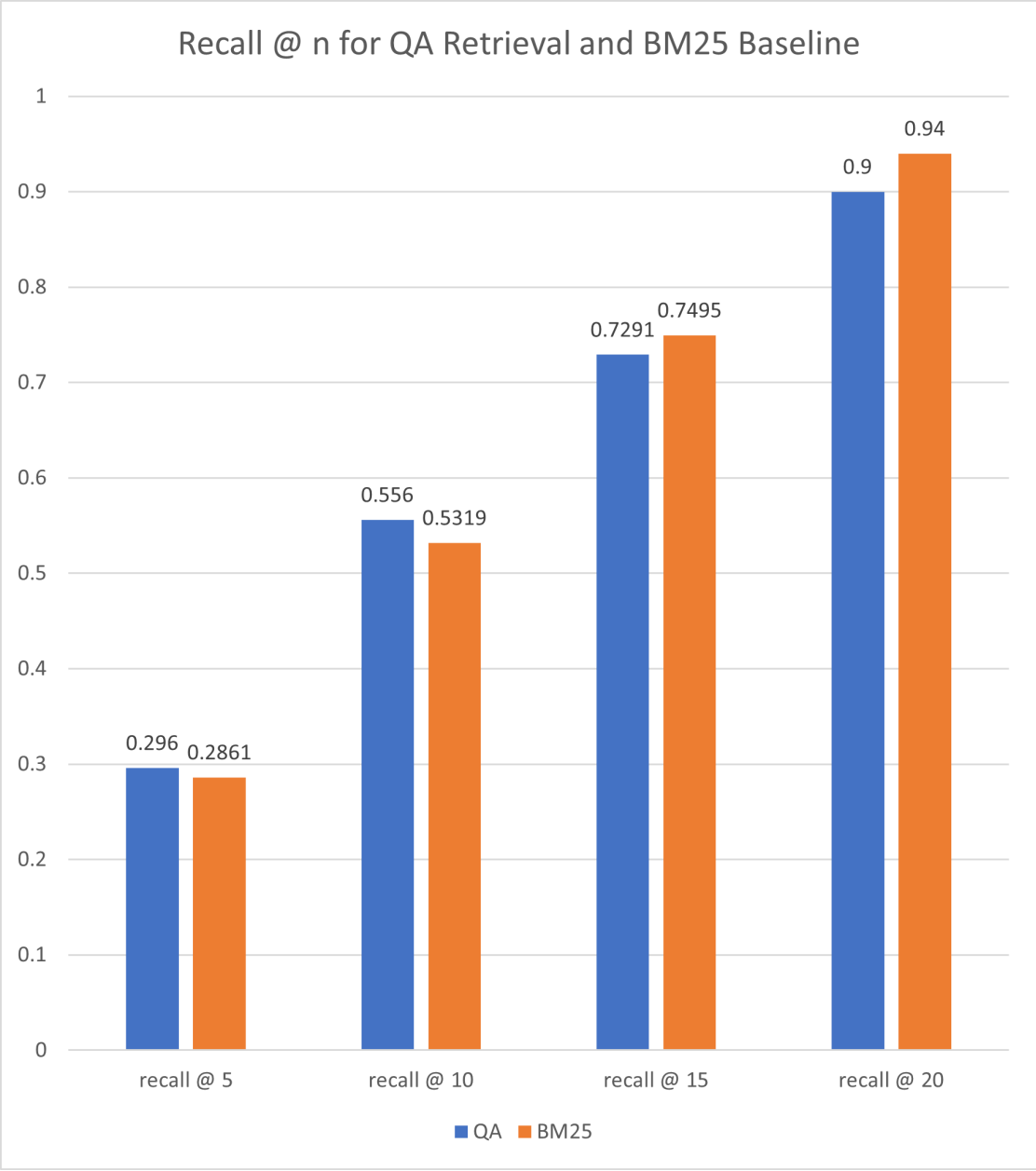

Figure 5.5: Recall@n = 5, 10, 15, 20 for QA retrieval and the BM25 baseline.

important for evaluating a retrieval system's performance. Ranking effectiveness was evaluated by calculating nDCG, precision, and recall at different ranking depths, as described in Section 2.2. These metrics were computed using the trec_eval tool[1]. A full view of the trec_eval results on the entire set of topics is available in Appendix C for QA retrieval and Appendix D for the BM25 baseline. nDCG[2], precision[3], and recall[4] use the implementations provided by the trec_eval tool.

nDCG, precision, and recall results at different rank positions are shown in Table 5.5. The nDCG scores of 0.619 and 0.745 at rank position 20 for the QA run and the baseline, respectively, show that the rankings produced by both runs generally agree with the ideal ranking, although more so for the baseline. The baseline outperforms the QA run in precision at all ranking depths. In recall, the QA run outperforms the baseline up to ranking depth 10, but the baseline overtakes the QA run after that, as seen in Figure 5.5. These differences are discussed further in Section 6.3. Note that the recall@20 does not reach 1.0, as there are topics that retrieved zero relevant segments at all, which contribute a recall value of 0 to the average.

[1] https://github.com/usnistgov/trec_eval
[2] https://github.com/usnistgov/trec_eval/blob/master/m_ndcg.c
[3] https://github.com/usnistgov/trec_eval/blob/master/m_Rprec.c
[4] https://github.com/usnistgov/trec_eval/blob/master/m_recall.c

Chapter 6

Discussion

6.1 Effectiveness Differences Between Topic Types

In both the QA and baseline runs, retrieval found a higher percentage of relevant segments per query topic for topical query types than both refinding and known-item types. The average scores for topical queries exceeded those for refinding and known-item queries, except for known-item topics in the baseline, but trailed their average scores when only looking at relevant segments.

A topical query may have many segments which are somewhat relevant to the query, however none can be considered a perfect match. Refinding and known-item queries, on the other hand, can have a perfect match to their query, since they are looking for a specific item.

Since refinding and known-item queries are looking for a specific piece of information, there may be only one episode that contains any relevant segments. This suggests that refinding and known-item queries may have only a few segments that are relevant at all, but those which are deemed relevant are scored highly. For example, refinding Topic 46 and known-item Topic 58 only have two and one relevant segments in the QA run, respectively. However, each of these segments represents a perfect relevance grade, meaning that the system retrieved the exact item that the query was looking for.

Furthermore, there may actually be no relevant episodes or segments at all for a given topic. For Topics 42, 47, and 52, both the QA and baseline runs failed to retrieve a single relevant segment.

6.2 Hard Queries

Some queries can be considered "harder" for the segment retrieval task (Karlgren et al., 2021), both for the QA retriever and the baseline retriever. A partial explanation for performance loss on some queries lies in the difficulty of handling certain terms at both the transcription and retrieval levels. Topic 42's keyword query was "fyre festival", referring to a specific failed music festival in 2017, but the word "fyre" was often mistranscribed as "fire". As a result, retrieval on Topic 42 produced a mix of segments related to music festivals and segments related to fire, but none related to the actual Fyre Festival.

At the retrieval stage, the two retrievers also struggled with homonyms and phrases with multiple meanings. For example, Topic 23 aimed to find segments related to the 2019 fire at the Notre Dame Cathedral in Paris. However, retrieval mainly returned segments from college football podcasts discussing the University of Notre Dame football team, as well as recent firings of head coaches at other schools. Since both the QA run and the baseline run struggled with Topic 42 and Topic 23, the difficulty may present itself during the BM25 ranking.

Another factor in making certain queries more difficult than others is the addition of negative qualifiers. Many of the topic descriptions contain some sort of discussion on what kind of information is relevant to the query. One description may specify that first-hand accounts are highly relevant, while another may be looking for news reports. Some descriptions explicitly define types of segments that are not relevant. For example,

a topic description may specify that it is specifically not looking for a historical account of an event. Thus, any podcasts which may be otherwise relevant but contain historical accounts are instead graded as irrelevant.

These negative qualifiers only appear in topical queries, in which 9 out of 35 topic descriptions contain a negative qualifier. The average number of relevant segments retrieved from a topical query in the QA run is 8.971. Of the 9 negatively qualified topics, only two of them resulted in more than the average number of relevant segments. Thus, seven topics fall below the average. This suggests that adding negative qualifiers impacts relevancy by restricting the domain of segments, making the query harder than one which relaxes its definition of relevancy to the topic.

6.3 Relevant Documents for Answer Extraction

QA retrieval outperformed the BM25 baseline in recall@5 and recall@10, though the baseline outperformed QA in recall@15 and recall@20, as seen in Figure 5.5. From Equation 2.2, recall is defined as the number of relevant documents retrieved by a search engine divided by the total number of relevant documents in the set of documents. A higher recall value at an earlier ranking position means that more of the relevant documents, proportional to the number of relevant documents in the set, appear earlier in the ranking. This is why the QA run can have a higher recall at earlier ranking positions while retrieving 20.45% less relevant documents than the baseline in total.

Indeed, more of the QA run's relevant documents are clustered towards the top of the rankings. QA retrieval placed 31.30% of its relevant documents in the top 5 rank positions, making up 33.14% of total relevance scores. Contrast this with the baseline placing 26.94% of its relevant documents in the top 5, making up 29.97% of its total relevance scores.

Furthermore, Tables 5.3 and 5.4 show that the baseline's average score for relevant segments is only 0.102 higher than the QA run. In fact, QA retrieval actually outperforms the baseline in topical queries by 0.035 in this metric. This suggests that the relevant documents retrieved by QA and the baseline are comparably relevant. However, QA finds less of these relevant documents than the baseline.

The high recall at early ranking positions, the clustering of relevant documents towards the top of the rankings, and the similar average scores among the QA run and the baseline for relevant documents suggest that QA retrieval is effective at finding and properly ranking relevant segments within input text that contains relevant information. However, the comparatively low number of total relevant documents for the QA run suggests that QA retrieval fails to find an adequate number of relevant candidate documents during first-stage BM25 retrieval on the index of full episode transcripts. Exploring this deficiency allows for the identification of two potential causes.

First, as shown in Lv and Zhai (2011), BM25 can "overly penalize very long documents", leading to inaccurate rankings. This becomes an issue when the length of documents varies, as episode lengths do. In contrast, segment lengths are time-constricted, which makes their transcript lengths more uniform. Additionally, the length of an episode is strictly greater than or equal to the length of each of its segments. Thus, BM25 may struggle to accurately rank long episodes with portions that are very relevant, but that switch between multiple discussion topics throughout the episode.

Second, and perhaps more critically, the QA system only retrieves one segment per episode. In contrast, the BM25 baseline can retrieve multiple segments from the same episode. An average of 7.78, or 38.9%, of segments per topic in the baseline's rankings are not from unique episodes. Thus, there exist multiple relevant segments in some episodes. This makes sense in the context of podcasts, as a conversation that mentions a

topic may last longer than a minute. A podcast may also mention a topic multiple times scattered throughout the episode.

The QA retriever, as currently implemented, extracts one optimal answer per candidate document. Thus, an answer from a document with multiple relevant segments will be ranked highly, but it will only contribute one segment. Instead of adding additional relevant segments to the set of ranked documents, as the baseline does, the QA retriever will look for answers in less relevant documents, depressing its score for some evaluated metrics.

This last point brings up the question of user goals in segment retrieval. If users are interested in finding the single most representative snippet from a corpus, as in the featured snippets used by search engines today, then multiple segments from the same episode are not as useful. Discussion regarding the difference that this makes in addressing user goals takes place in Section 7.3.

Chapter 7

Conclusion

This thesis explored the use of end-to-end question-answering for segment retrieval on a large dataset of podcast transcripts. Experimentation showed that while QA retrieval was able to effectively rank relevant segments, it failed at identifying enough candidate documents containing relevant information to outperform a BM25 baseline in overall ranking evaluation metrics. Some of the limitations affecting experimentation and implementation design are mentioned in Section 7.1. Based on the background explored in Chapter 2, the approach using end-to-end QA has theoretical merit, and further work to improve performance is described in Section 7.2. Finally, Section 7.3 discusses the implications of certain user goals and presents closing remarks.

7.1 Limitations

The evaluation process was limited to a human-judged annotation depth of one for each segment. Segments retrieved by QA were expanded from arbitrary answer spans in the text, so they did not correspond with the predetermined segment index. Thus, each segment needed to be judged by a human annotator. To maintain a consistent grading standard, the human annotator also graded the baseline results rather than using the results from TREC. Adding additional annotation depth through crowdsourcing would increase confidence for segments' scores but at the cost of many hours of labor.

At TREC 2020 and TREC 2021, most other approaches to segment retrieval, much like the retrieval system implemented in this thesis, involved transfer learning by fine-tuning a pre-trained model like BERT without further model training for the podcast domain (Jones, Carterette, et al., 2021; Karlgren et al., 2021). The small number of training and test topics, 108 in total between the two years, along with the scarcity of segment annotations makes it difficult to train models to operate on the salient points of the podcast dataset. The podcast track has been paused for 2022 to reevaluate participant tasks and focus more on the podcast domain. In future years, the track may provide participants with more richly labeled data.

7.2 Future Work

As mentioned in Section 6.3, the QA retriever may struggle to accurately rank transcripts for long episodes in the first stage of retrieval. A possible solution to this would be using a different IR model for the first stage, such as the BM25L model described by Lv and Zhai (2011), that improves performance for long documents. Another issue from Section 6.3 was the selection of only one answer per candidate document. The retriever could be modified to retrieve more than one answer per document. However, extra care would need to be taken to ensure that the answer spans are not overlapping or so close together as to be redundant. An approach that would address both issues could be to operate on the index of predetermined segments directly. However, the aim of this thesis was to investigate the viability of end-to-end question-answering. Simply using the answer span's score to determine a segment's ranking would turn the process into an answer selection task or a segment reranking task.

It is also not guaranteed that the two-minute long segments starting on the minute are always the best segments. They may include irrelevant information, begin in the middle

of a discussion, or cut off relevant information at the end of the segment. The answer expansion method for the QA retriever was chosen for ease of comparison with the predetermined segments, so these segments suffer from the same drawbacks. Building more sophisticated models to handle answer span expansion would improve segment quality, as well as provide a distinct advantage for QA retrieval over answer selection or segment reranking.

The BERT Base reader has a sequence length of 768 (Devlin et al., 2018), meaning that long documents are operated on in chunks to preserve the effectiveness of the attention mechanism. To better handle answer extraction from long episodes, it may be beneficial to explore using a model specialized for long sequences. One such model is Longformer, which can process sequences up to 4,096 tokens long by mixing local attention windows with a global attention mechanism (Beltagy, Peters, and Cohan, 2020).

7.3 Segment Retrieval for Answering Questions

Podcast search has been compared to blog search in the literature (Besser, Hofmann, Larson, et al., 2008; Jones, Zamani, et al., 2021; Jones, Carterette, et al., 2021). Blog search differs from conventional web search in that the user goal mainly consists of informational queries (Besser, Hofmann, Larson, et al., 2008) as opposed to navigational or transactional queries (Broder, 2002). The similarity in user goals between blog search and podcast search is reflected in the topical type query topics in the segment retrieval task that are looking for relevant information, but not necessarily a specific resource.

A fundamental motivation for an informational query is the need to answer a question. In the podcast domain, a user may want to find a podcast segment that answers their question with the option to listen to the rest of the podcast. If the user wishes to answer a question using segment retrieval, it is highly unlikely that they look at the entire list of

results (Croft, Metzler, and Strohman, 2010). In all likelihood, they will only concern themselves with the few highest ranked items (Croft, Metzler, and Strohman, 2010).

The segment retrieval task as described by TREC may not be the ideal use case for a question-answering segment retriever. However, if the user goal is to retrieve a representative segment for a podcast episode's relevance to a query, then retrieving multiple segments from the same episode, as the baseline does, is not desirable. Although research has been conducted on user goals for podcast search, it would be beneficial to further investigate the user goals of segment retrieval on podcasts, specifically. The benefits of the QA segment retriever's architecture may present themselves more clearly for this alternate definition of segment retrieval.

While the QA retriever described in this thesis underperformed the baseline for the defined segment retrieval task, it shows promise in its ability to retrieve and properly rank relevant segments for the most relevant documents. As noted above, this is the most important attribute of a good search system in relation to the actual user goal in informational queries, especially so when the query is a question. When the user's goal is to answer questions, question-answering shows promise as a viable tool for segment retrieval on podcast transcripts.

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Beltagy, Iz, Matthew E Peters, and Arman Cohan (2020). "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150*.

Besser, Jana, Katja Hofmann, Martha A Larson, et al. (2008). "An Exploratory Study of User Goals and Strategies in Podcast Search." In: *LWA*. Citeseer, pp. 27–34.

Broder, Andrei (2002). "A taxonomy of web search". In: *ACM Sigir forum*. Vol. 36. 2. ACM New York, NY, USA, pp. 3–10.

Chelba, Ciprian, Timothy J Hazen, and Murat Saraclar (2008). "Retrieval and browsing of spoken content". In: *IEEE Signal Processing Magazine* 25.3, pp. 39–49.

Cho, KyungHyun et al. (2014). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* abs/1409.1259. arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259.

Clark, Kevin et al. (2019). "What does bert look at? an analysis of bert's attention". In: *arXiv preprint arXiv:1906.04341*.

Clifton, Ann et al. (Dec. 2020). "100,000 Podcasts: A Spoken English Document Corpus". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 5903–5917. DOI: 10.18653/v1/2020.coling-main.519. URL: https://aclanthology.org/2020.coling-main.519.

Croft, W Bruce, Donald Metzler, and Trevor Strohman (2010). *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading.

Devlin, Jacob et al. (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805*.

Edison Research (2020). *The Infinite Dial 2020.* Tech. rep.

Jones, Rosie, Ben Carterette, et al. (2021). “TREC 2020 Podcasts Track Overview”. In: *arXiv preprint arXiv:2103.15953*.

Jones, Rosie, Hamed Zamani, et al. (2021). “Current Challenges and Future Directions in Podcast Information Access”. In: *arXiv preprint arXiv:2106.09227*.

Karlgren, Jussi et al. (2021). “TREC 2021 Podcasts Track Overview”.

Lin, Jimmy et al. (2021). “Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations”. In: *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pp. 2356–2362.

Listen Notes (2022). *Podcast stats: How many podcasts are there?* URL: https://www.listennotes.com/podcast-stats/.

Lv, Yuanhua and ChengXiang Zhai (2011). “When Documents Are Very Long, BM25 Fails!” In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’11. Beijing, China: Association for Computing Machinery, pp. 1103–1104. ISBN: 9781450307574. DOI: 10.1145/2009916.2010070. URL: https://doi.org/10.1145/2009916.2010070.

Pérez-Iglesias, Joaquín et al. (2009). “Integrating the probabilistic models BM25/BM25F into Lucene”. In: *arXiv preprint arXiv:0911.5046*.

Rajpurkar, Pranav et al. (2016). “Squad: 100,000+ questions for machine comprehension of text”. In: *arXiv preprint arXiv:1606.05250*.

Robertson, Stephen, Hugo Zaragoza, et al. (2009). “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* 3.4, pp. 333–389.

Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems*, pp. 5998–6008.

Voorhees, Ellen M. and Dawn M. Tice (May 2000). “The TREC-8 Question Answering Track”. In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC’00)*. Athens, Greece: European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2000/pdf/26.pdf.

Weerkamp, Wouter, Krisztian Balog, and Maarten de Rijke (2009). “A generative blog post retrieval model that uses query expansion based on external collections”. In: *Proceedings of the joint conference of the 47th annual meeting of the ACL and the 4th international joint conference on natural language processing of the AFNLP*, pp. 1057–1065.

Wu, Yonghui et al. (2016). “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144*.

Yang, Wei et al. (2019). “End-to-end open-domain question answering with bertserini”. In: *arXiv preprint arXiv:1902.01718*.

Yu, Yongze et al. (2020). “Spotify at the TREC 2020 Podcasts Track: Segment Retrieval”. In: *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*.

Chapter A

Selected Source Code

The full source code for this thesis can be found at `https://github.com/oudalab/EQUIP`.

```
{
    "topics": "topics/podcasts_2020_topics_test.json",
    "index": "podcast_pyserini_indices/lucene-index.podcasts-
       full-transcript.pos+docvectors+rawdocs",
    "bert_model": "rsvp-ai/bertserini-bert-base-squad",
    "bert_tokenizer": "rsvp-ai/bertserini-bert-base-squad",
    "results": "results/bert/",
    "num_hits": 500,
    "num_answers": 20,
    "score_weight": 0.5,
    "segment_radius": 170,
    "args_to_change": {
        "max_query_length": 128,
        "max_answer_length": 64
    }
}
```

Figure A.1: Configuration file *config.json* containing arguments for the experiment.

```
#!/bin/bash
#
#SBATCH --partition=normal
#SBATCH --ntasks=1
#SBATCH --mem=20G
#SBATCH --output=output/topic_9_output_bert.txt
#SBATCH --error=output/topic_9_error_bert.txt
#SBATCH --time=24:00:00
#SBATCH --job-name=Topic_9_Bert_Retrieval
#SBATCH --mail-user=andrewelaryan@ou.edu
#SBATCH --mail-type=ALL
#SBATCH --chdir=/work/andrewe/EQUIP
#
################################################
module load Java/11.0.2
module load Python/3.7.4-GCCcore-8.3.0
time python src/qa_searcher.py 9
```

Figure A.2: SBATCH job for topic 9 on OSCER *topic_9_bert.sbatch*.

```
    def main():
        # Load configs and command line arguments
        ...

        # Initialize the BERT reader using the model and tokenizer
        bert_reader = BERT(model, tokenizer)
        # Update certain arguments in the BERT Reader
        bert_reader.update_args(args_to_change)
        # Build a Pyserini searcher (defaults to BM25)
        searcher = build_searcher(index)
        # Keyword query
        keyword = Question(topic['query'])
        # Detailed description
        description = topic['description']
        # Concatenate the keyword to the description to create the
                                            question for QA
        question = Question(keyword.text + " " + description)
        # Retrieve the k best candidate episodes using BM25
        candidate_episodes = retriever(keyword, searcher, k)
        # Use the BERT reader to find answers to the question in the
                                            context
        candidate_answers = bert_reader.predict(question,
                                            candidate_episodes)
        # Register the top n best answers using the configured weight
        answers = get_n_best_answers(candidate_answers, n, weight)
        # Retrieve the full episode transcipts
        docs = [searcher.doc(answer.metadata).raw() for answer in
                                            answers]
        # Expand the answers to include the specified radius of text
                                            in the document
        segments = construct_segments(docs, answers, segment_radius)
        # Format the results object
        results = build_results(segments, answers, docs)
        # Write all results to a single file
        write_results(results_folder, topic_num, results)
```

Figure A.3: Main function for the segment retrieval process *qa_searcher.py*.

```
class Answer:
"""
Class representing an answer.
A answer contains the answer text itself and potentially other
                                    metadata.
Parameters
----------
text : str
    The answer text.
metadata : Mapping[str, Any]
    Additional metadata and other annotations.
score : Optional[float]
    The score of the answer.
ctx_score : Optional[float]
    The context score of the answer.
total_score : Optional[float]
    The aggregated score of answer score and ctx_score.
start_idx: Optional[int]
    The answer's start index in the original document.
end_idx: Optional[int]
    The answer's end index in the original document.
"""

def __init__(self,
             text: str,
             language: str = "en",
             metadata: Mapping[str, Any] = None,
             score: Optional[float] = 0,
             ctx_score: Optional[float] = 0,
             total_score: Optional[float] = 0,
             start_idx: Optional[int] = 0,
             end_idx: Optional[int] = 0):
    self.text = text
    self.language = language
    if metadata is None:
        metadata = dict()
    self.metadata = metadata
    self.score = score
    self.ctx_score = ctx_score
    self.total_score = total_score
    # Added fields to store answer's start and end indices from
                                        original document
    self.start_idx = start_idx
    self.end_idx = end_idx
```

Figure A.4: Modified Answer class in *bertserini_modified/base.py*.

```
...

answers, _ = compute_predictions_logits(
    all_examples=examples,
    all_features=features,
    all_results=all_results,
    n_best_size=self.args["n_best_size"],
    max_answer_length=self.args["max_answer_length"],
    do_lower_case=self.args["do_lower_case"],
    output_prediction_file=self.args["output_prediction_file"],
    output_nbest_file=self.args["output_nbest_file"],
    output_null_log_odds_file=self.args["
                                        output_null_log_odds_file"
                                        ],
    verbose_logging=self.args["verbose_logging"],
    version_2_with_negative=self.args["version_2_with_negative"],
    null_score_diff_threshold=self.args["
                                        null_score_diff_threshold"
                                        ],
    tokenizer=self.tokenizer,
    language=question.language
)

all_answers = []
for idx, ans in enumerate(answers):
    all_answers.append(Answer(
        text=answers[ans][0],
        score=answers[ans][1],
        ctx_score=contexts[idx].score,
        # Add the docid to the answer's metadata
        metadata=contexts[idx].metadata['docid'],
        language=question.language,
        # Add the answer's start index
        start_idx=answers[ans][2],
        # Add the answer's end index
        end_idx=answers[ans][3]
    ))
return all_answers
```

Figure A.5: Snippet of modified BERT reader in *bertserini_modified/bert_reader.py*.

Chapter B

Segment Grading Rubric

Perfect (4): this grade is used only for "known-item" and "refinding" topic types. It reflects the segment that is the earliest entry point into the one episode that the user is seeking.

Excellent (3): the segment conveys highly relevant information, is an ideal entry point for a human listener, and is fully on topic. An example would be a segment that begins at or very close to the start of a discussion on the topic, immediately signaling relevance and context to the user.

Good (2): the segment conveys highly-to-somewhat relevant information, is a good entry point for a human listener, and is fully to mostly on topic. An example would be a segment that is a few minutes "off" in terms of position, so that while it is relevant to the user's information need, they might have preferred to start two minutes earlier or later.

Fair (1): the segment conveys somewhat relevant information, but is a sub-par entry point for a human listener and may not be fully on topic. Examples would be segments that switch from non-relevant to relevant (so that the listener is not able to immediately understand the relevance of the segment), segments that start well into a discussion without providing enough context for understanding, etc.

Bad (1): the segment is not relevant.

Chapter C

Output of trec_eval for all Test Topics, QA Retrieval

```
runid                   all oudalab_bert
num_q                   all 50
num_ret                 all 993
num_rel                 all 361
num_rel_ret             all 361
map                     all 0.5262
gm_map                  all 0.1655
Rprec                   all 0.4451
bpref                   all 0.3936
recip_rank              all 0.6228
iprec_at_recall_0.00    all 0.6770
iprec_at_recall_0.10    all 0.6711
iprec_at_recall_0.20    all 0.6538
iprec_at_recall_0.30    all 0.5999
iprec_at_recall_0.40    all 0.5633
iprec_at_recall_0.50    all 0.5531
iprec_at_recall_0.60    all 0.5289
iprec_at_recall_0.70    all 0.5054
iprec_at_recall_0.80    all 0.4752
iprec_at_recall_0.90    all 0.4527
```

```
iprec_at_recall_1.00    all 0.4482
P_5                     all 0.4520
P_10                    all 0.4260
P_15                    all 0.3880
P_20                    all 0.3610
P_30                    all 0.2407
P_100                   all 0.0722
P_200                   all 0.0361
P_500                   all 0.0144
P_1000                  all 0.0072
recall_5                all 0.2960
recall_10               all 0.5560
recall_15               all 0.7291
recall_20               all 0.9000
recall_30               all 0.9000
recall_100              all 0.9000
recall_200              all 0.9000
recall_500              all 0.9000
recall_1000             all 0.9000
infAP                   all 0.5262
gm_bpref                all 0.0242
Rprec_mult_0.20         all 0.5233
Rprec_mult_0.40         all 0.5032
Rprec_mult_0.60         all 0.4759
Rprec_mult_0.80         all 0.4490
Rprec_mult_1.00         all 0.4451
```

```
Rprec_mult_1.20         all 0.4218
Rprec_mult_1.40         all 0.3953
Rprec_mult_1.60         all 0.3705
Rprec_mult_1.80         all 0.3446
Rprec_mult_2.00         all 0.3290
utility                 all -5.4200
11pt_avg                all 0.5571
binG                    all 0.4723
G                       all 0.3869
ndcg                    all 0.6193
ndcg_rel                all 0.5220
Rndcg                   all 0.4735
ndcg_cut_5              all 0.4057
ndcg_cut_10             all 0.4919
ndcg_cut_15             all 0.5560
ndcg_cut_20             all 0.6193
ndcg_cut_30             all 0.6193
ndcg_cut_100            all 0.6193
ndcg_cut_200            all 0.6193
ndcg_cut_500            all 0.6193
ndcg_cut_1000           all 0.6193
map_cut_5               all 0.2315
map_cut_10              all 0.3675
map_cut_15              all 0.4537
map_cut_20              all 0.5262
map_cut_30              all 0.5262
```

```
map_cut_100             all 0.5262
map_cut_200             all 0.5262
map_cut_500             all 0.5262
map_cut_1000            all 0.5262
relative_P_5            all 0.4920
relative_P_10           all 0.6226
relative_P_15           all 0.7394
relative_P_20           all 0.9000
relative_P_30           all 0.9000
relative_P_100          all 0.9000
relative_P_200          all 0.9000
relative_P_500          all 0.9000
relative_P_1000         all 0.9000
success_1               all 0.5000
success_5               all 0.7800
success_10              all 0.8600
set_P                   all 0.3648
set_relative_P          all 0.9000
set_recall              all 0.9000
set_map                 all 0.3648
set_F                   all 0.4775
num_nonrel_judged_ret   all 632
```

Chapter D

Output of trec_eval for all Test Topics, BM25 Retrieval Baseline

```
runid                   all oudalab_bm25
num_q                   all 50
num_ret                 all 1000
num_rel                 all 568
num_rel_ret             all 568
map                     all 0.6976
gm_map                  all 0.3416
Rprec                   all 0.6390
bpref                   all 0.5553
recip_rank              all 0.7850
iprec_at_recall_0.00    all 0.8320
iprec_at_recall_0.10    all 0.8099
iprec_at_recall_0.20    all 0.7891
iprec_at_recall_0.30    all 0.7499
iprec_at_recall_0.40    all 0.7187
iprec_at_recall_0.50    all 0.7177
iprec_at_recall_0.60    all 0.6959
iprec_at_recall_0.70    all 0.6844
```

```
iprec_at_recall_0.80    all 0.6772
iprec_at_recall_0.90    all 0.6566
iprec_at_recall_1.00    all 0.6465
P_5                     all 0.6120
P_10                    all 0.6100
P_15                    all 0.5853
P_20                    all 0.5680
P_30                    all 0.3787
P_100                   all 0.1136
P_200                   all 0.0568
P_500                   all 0.0227
P_1000                  all 0.0114
recall_5                all 0.2861
recall_10               all 0.5319
recall_15               all 0.7495
recall_20               all 0.9400
recall_30               all 0.9400
recall_100              all 0.9400
recall_200              all 0.9400
recall_500              all 0.9400
recall_1000             all 0.9400
infAP                   all 0.6976
gm_bpref                all 0.1627
Rprec_mult_0.20         all 0.7100
Rprec_mult_0.40         all 0.6623
Rprec_mult_0.60         all 0.6488
```

```
Rprec_mult_0.80         all 0.6410
Rprec_mult_1.00         all 0.6390
Rprec_mult_1.20         all 0.5668
Rprec_mult_1.40         all 0.5099
Rprec_mult_1.60         all 0.4622
Rprec_mult_1.80         all 0.4267
Rprec_mult_2.00         all 0.4021
utility                 all 2.7200
11pt_avg                all 0.7253
binG                    all 0.6405
G                       all 0.4842
ndcg                    all 0.7450
ndcg_rel                all 0.6532
Rndcg                   all 0.6123
ndcg_cut_5              all 0.5570
ndcg_cut_10             all 0.6200
ndcg_cut_15             all 0.6831
ndcg_cut_20             all 0.7450
ndcg_cut_30             all 0.7450
ndcg_cut_100            all 0.7450
ndcg_cut_200            all 0.7450
ndcg_cut_500            all 0.7450
ndcg_cut_1000           all 0.7450
map_cut_5               all 0.2543
map_cut_10              all 0.4238
map_cut_15              all 0.5673
```

```
map_cut_20              all 0.6976
map_cut_30              all 0.6976
map_cut_100             all 0.6976
map_cut_200             all 0.6976
map_cut_500             all 0.6976
map_cut_1000            all 0.6976
relative_P_5            all 0.6540
relative_P_10           all 0.7353
relative_P_15           all 0.8231
relative_P_20           all 0.9400
relative_P_30           all 0.9400
relative_P_100          all 0.9400
relative_P_200          all 0.9400
relative_P_500          all 0.9400
relative_P_1000         all 0.9400
success_1               all 0.7200
success_5               all 0.8800
success_10              all 0.9200
set_P                   all 0.5680
set_relative_P          all 0.9400
set_recall              all 0.9400
set_map                 all 0.5680
set_F                   all 0.6523
num_nonrel_judged_ret   all 432
```