UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

SCALING UP LABELING, MINING, AND INFERENCING ON EVENT EXTRACTION

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR of PHILOSOPHY

By

Yan Liang

Norman, Oklahoma

2022

SCALING UP LABELING, MINING, AND INFERENCING ON EVENT EXTRACTION

A DISSERTATION APPROVED FOR THE

SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Dr. Christan Grant, Chair

Dr. Andrew Fagg

Dr. Kun Lu

Dr. Dean Hougen

Dr. Qi Cheng

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Christan Grant, who offered me his mentorship, patience, and care. This work would not have been done without his excellent guidance and unconditional support. Under his guidance, I overcame a lot of difficulties and learned a lot from him and my awesome labmates. I am grateful to have the opportunity working with him.

I would like to specially thank Dr. Jill Irvine, who opens the door for me to conduct research in event extraction, for all her support and guidance. I would like to thank Dr. Andy Halterman for his cooperation and feedback in my work. I would like to extend my thanks to my dissertation committee Dr. Andrew Fagg, Dr. Kun Lu, Dr. Dean Hougen, and Dr. Qi Cheng for their valuable time to guide me, and invaluable feedback.

The School of Computer Science at OU has provided me with great learning experiences. I would like to thank Department Chair Dr. Sridhar Radhakrishnan, all the faculty and wonderful staff, specially Philip Johnson and Virginie Perez, for their guidance and help during all these years of graduate work.

I would wholeheartedly like to acknowledge the people who mean a lot to me, especially my parents, parents-in-law, uncles, aunts, and cousins, for showing faith in me and giving me freedom to choose what I desired. I salute you all for selfless love, care and sacrifice you did to support me.

Finally, and most importantly, I owe thanks to a very special person, my husband Shaohui Wang for his continued and unfailing love, support and understanding during my pursuit of Ph.D. degree which made the completion of this dissertation possible.

Table of Contents

List of Tables

List of Figures

Abstract

Numerous important events happen every day and are reported in different media sources with varying narrative styles across different knowledge domains and languages. Detecting the real-world events that have been reported from online articles and posts is one of the main tasks in event extraction. Other tasks include identifying event triggers and trigger types, identifying event arguments and argument types, clustering and tracking similar events from different texts, event prediction, and event evolution. As one of the most important research themes in natural language processing and understanding, event extraction has wide applications in diverse domains and has been intensively researched for decades.

This work targets a scaling-up of End-to-End event extraction task through three ways. First, scaling up the event labeling process to different languages and domains. We designed and implemented four approaches to accurately and efficiently produce multi-lingual labels for events. Using the approaches we developed, we were able to complete Arabic actor and verb dictionaries with coverage equivalent to English in less than two years of work, compared to two decades for English dictionary development. Second, scaling up event extraction by using the document topics information in a topic-aware deep learning framework. We propose a domain-aware event extraction method by using the topic name embeddings to enrich the sentences' contextual representations and multi-task setup of event extraction and topic classification task. With the topic-aware model we developed, we were able to improve F_1 by 1.8% on all event types, and F_1 by 13.34% on few-shot event types. Third, scaling up event extraction by designing containerized and efficient pipelines, which researchers can comfortably adopt. The pipeline has a container-based architecture that adapts to the available systems and load to process text. With the Kalman filter based batch size optimization, we were able to achieve 20.33% improvement on processing time compared to

static batch size. Using the pipeline we developed, we were able to publish largest machine-coded political event dataset covering 1979 to 2016 (2TB, 300 million documents).

Chapter 1

Introduction

1.1 Overview

Numerous important events happen every day but are reported in different media sources with different narrative styles across different knowledge domains. Detecting whether real-world events have been reported from online articles and posts is one of the main tasks in event extraction. Other tasks include identifying event triggers and trigger types, identifying event arguments and argument types, clustering and tracking similar events from different texts, and event prediction and event evolution. As one of the most important research themes in natural language processing and understanding, event extraction has wide applications in diverse domains and has been intensively researched for decades. For example, structured events can be directly used in constructing or expanding the knowledge base [Wu et al., 2019; Bosselut et al., 2021], a centralized repository where information is stored, organized, and then shared, upon which further logical reasoning and inference can be made [Liu et al., 2018; Rospocher et al., 2016]. Event Detection and monitoring have long been the focus of public affair management for governments, as timely knowing the outbursts and evolution of popular social events helps the authorities to respond promptly [Conlon et al., 2015; Atkinson et al., 2009]. In the business and financial domain, event extraction can also help

companies quickly discover market responses to their products and influential signals for risks analysis and suggestions [Nuij et al., 2013; Capet et al., 2008]. In the biomedical domain, event extraction can be used to identify the alterations in the state of a biomolecule (e.g., a gene or a protein) or interactions between two or more biomolecues, which is explained in the literature on understanding physiological and pathogenesis mechanisms [Vanegas et al., 2015]. To sum up, many domains can benefit from the advancements in event extraction techniques and systems.

1.2 Motivation

In the machine learning domain, specifically in supervised methods, large amounts of annotated data are needed to build a robust model and improve the model's prediction accuracy. Researchers have gathered datasets containing labelled data from various domains, such as ImageNet [Deng et al., 2009], the Lung Image Database Consortium and the Image Database Resource Initiative (LIDC-IDRI) [Armato III et al., 2011], and the Modified National Institute of Standards and Technology dataset (MNIST) [LeCun et al., 1998]. Previous work has obtained fascinating results by implementing machine learning techniques with labelled datasets [You et al., 2018; He et al., 2019; Kussul and Baidyk, 2004; Tabik et al., 2017; Schott et al., 2019; Pedamonti, 2018]. It is well known that the majority of the time spent conducting experiments in machine learning is spent on data preparation [Munson, 2012]. This includes data label collecting, cleaning, analyzing, visualizing, and feature engineering. While all of these steps are time consuming, data label collection has recently become a challenge for the following reasons.

First, as machine learning is used in new applications, it is usually the case that there is not enough training data. Traditional applications like machine translation or object detection enjoy massive amounts of training data that have been accumulated for decades. Similarly, in the political event coding domain, Dr. Philip Schrodt and colleagues at the

University of Kansas spent two decades creating the original English language event coding dictionaries: Texual analysis by augmented replacement instructions (TABARI).[1] On the other hand, more recent applications have little or no training data. As an illustration, smart factories are increasingly becoming automated where product quality control is performed with machine learning. There is little or no training data to start with whenever there is a new product or a new defect to detect. The naïve approach of manual labelling may not be feasible because it is expensive and requires domain expertise. This problem applies to any novel application that benefits from machine learning.

Moreover, as deep learning becomes popular, there is even more need for training data. In traditional machine learning, feature engineering is one of the most challenging steps because the user needs to understand the application and provide features used for training models. On the other hand, deep learning can automatically generate features, which saves us time on feature engineering, and is a significant part of data preparation. However, deep learning may require much more training data to perform well [Bach et al., 2017].

Data produced from news stories is one of the most important new sources of information for quantitative political science research. Machine-coded data of geo-referenced political and social activity has opened up new political phenomena ranging from social movements to violent conflict, unrest, and government responses, with news reporting in English. However, one limitation of event data is its restriction to English language sources only. To study the political event extraction in other language sources, we need to scale up the event labelling process from English to other languages. Several issues make automated event labelling in multiple languages an unsolved and challenging task, including the unwieldy size of text data which can reach terabytes, extracting the relevant actors to the events being studied, and guaranteeing the accuracy and abundance of the coded results.

Further, event extraction is not only limited to the political science domain [Nuij et al., 2013; Capet et al., 2008; Vanegas et al., 2015]. Identifying events across event type domains

[1] `https://parusanalytics.com/eventdata/software.dir/tabari.html`

is also an essential task to work on. A domain-adaptive event extraction system, where an extraction model with domain specific knowledge is needed to improve extraction performance. The topic/domain information of the document where the events are extracted from are rarely explored. Examples of the topics of the documents can be military conflict, earthquake, concert tour, or wrestling. Topic information is vital for event extraction, since semantically similar topics share similar event types, while event types are quite different between distinguishable topics. In addition, the event types we try to detect might have data imbalance issues: some of the event types frequently seen in the training data are going to have higher performance, while the event types rarely seen in the training data have lower performance. Using the extra knowledge from the topic or domain could bring in extra "bridge" knowledge to make the information transfer from "high resource" to "low resource" event types. A "high resource" event type is one that has been seen quite a few times in the training data, while a "low resource" event type is one that has been rarely seen in the training data. This further boosts the performance of the event types with a limited number of training labels.

Moreover, the availability of large corpora of online news documents has made it possible for computer and social scientists to study human political behavior at previously impossible scales. One of the primary bottlenecks in deriving meaning from text documents is the resource demands of the natural language processing and information extraction that need to be performed. Current document processing pipelines suffer from a range of limitations when processing hundreds of millions of news articles for social science applications, either in terms of poor performance or ease of use. Document processing pipelines written by social scientists are easily installed and customized, but tend to be single-threaded and slow for corpora larger than a few hundred thousand documents. Frameworks for distributed pipelines that are fully flexible (Apache Nifi, Kubernetes, Spark) require sophisticated infrastructure and significant technical expertise both to set up and to customize for document processing tasks, which creates a high hurdle for applied use. The need to develop a simple, adaptive

pipeline for extracting political events from many news documents is urgent.

1.3 Contribution

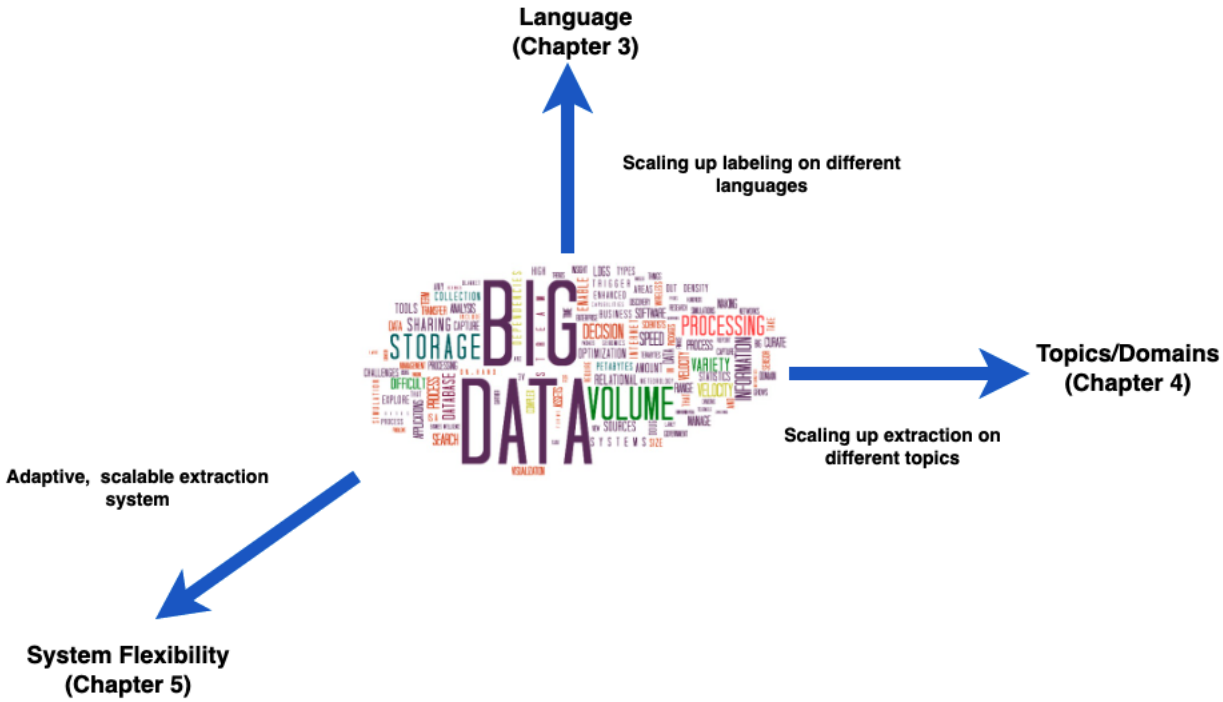

Figure 1.1: Scaling up event extraction from three dimensions.

We are targeting the scaling up of event extraction task end-to-end from three dimensions:

- Scaling up event labelling process that can be adapted to different languages and domains.
- Scaling up event extraction by using the document topics information in a topic-aware learning framework.
- Scaling up event extraction by designing containerized and efficient pipeline which researchers can easily adopt.

The general machine learning pipeline involves three components: data, model, and model deployment [DeHart et al., 2021]. Event extraction follows a similar workflow. We summarize the event extraction process in Figure 1.2 and highlight the components that we contributed to.

- We designed and implemented four different approaches to producing labels for events more accurately and efficiently, which can be generalized to different languages and different event domains. Using the above approaches for developing dictionaries, we were able to complete Arabic actor and verb dictionaries with coverage equivalent to the English language dictionaries in less than two years of work, compared to the two decades that the English language dictionaries took to produce (Chapter 3).

- We proposed a domain-aware event extraction method by utilizing the topic name embeddings to enrich the sentences' contextual representations and multi-task set-up of event extraction and topic classification task, TAED. In detail: (1) We performed detailed analyses explaining why topic information helps the event detection task. (2) We introduce topic name enhanced sentence representation for event detection and explore different ways to embed the topic name information, including using attention-based versus concatenation-based interaction, `[CLS]` versus token average based attribute embedding, and topic keywords to generate topic embedding versus using topic names. (3) We introduce topic classification and event detection as a multi-task learning set-up, which further improves the performance and conducted experiments with two event detection datasets with various event types. We achieved up to +1.8% on the $F1$ score compared to the baseline. (4) Furthermore, we show that the topic-aware model we propose can improve the few-shot event types scenario with a large margin of +13.34% on the $F1$ score and provide heuristic explanations in the case study (Chapter 4).

- We designed and implemented a pipeline for efficiently extracting event data from web documents. The pipeline has a container based architecture that adapts to the available

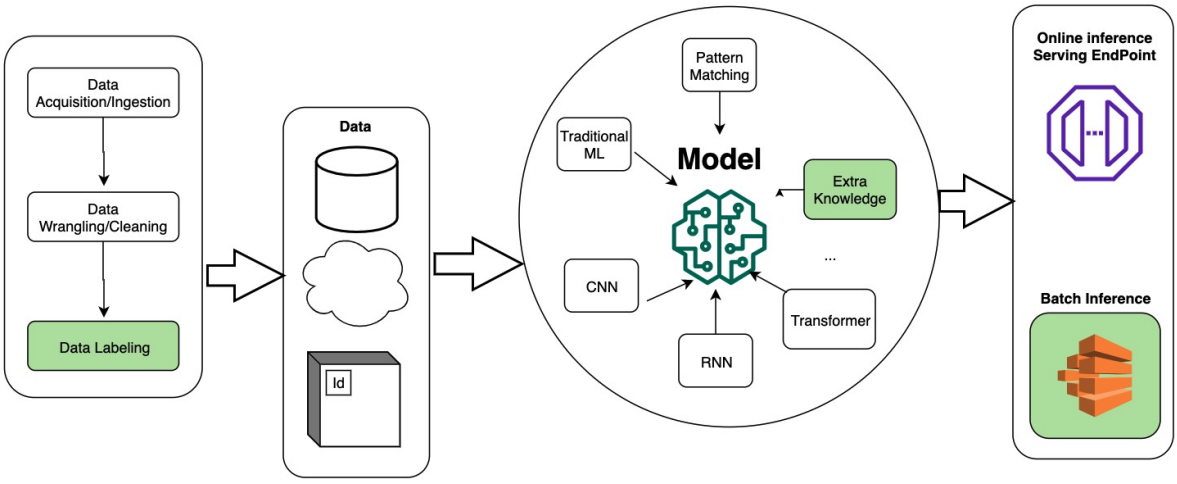

Figure 1.2: The event extraction workflow. Our contributions are highlighted.

systems and load to process text. This architecture allows researchers to use whatever resources they have to process a large data set. Furthermore, with the adaptive batch size learned from Kalman filter [Kalman, 1960], we see up to 20.33% improvement on processing time compared to static batch size (Chapter 5).

Chapter 2

Literature Review

2.1 Concepts

An event indicates the occurrence of an action or state change, often driven by verbs or gerunds. Event extraction technology extracts events that users are interested in from unstructured information and presents to users in a structured form [Chen et al., 2015]. An event is consisted of the primary components involved in the action, such as the time, the place, and the character. Event extraction needs to detect not only the event type but also the corresponding words/phrases to fill in the predefined event schema as defined in ACE05[1] shown in Table 2.1, in order to output a structured format event. This is also known as closed-domain event extraction, since the event schemas vary for different event domains. There is also another type of event extraction task: open-domain event extraction, in which the predefined event schema is not required. Open-domain event extraction focuses on detecting the existence of an event and further targets to cluster the events based on their semantic meaning.

As an example, given the sentence

At daybreak on the 9^{th}, the terrorists set off a truck bomb attack in Nazareth.

[1] `https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-events-guidelines-v5.4.3.pdf`

Event Type: Conflict-Attack		Event Type: Justice-Execute	
Argument Role	Attacker	Argument Role	Person
	Target		Agent
	Instrument		Crime
	Time		Time
	Place		Place
Event Type: Transfer-Money		Event Type: Life-Die	
Argument Role	Giver	Argument Role	Agent
	Recipient		Victim
	Beneficiary		Instrument
	Money		Time
	Time		Place
	Place		

Table 2.1: Examples of predefined event type to arguments schema in ACE05 for four event types: Conflict-Attack, Justice-Execute, Transfer-Money, and Life-Die.

we want to identify the event components as follows:

- Type: Conflict/Attack
- Trigger: attack
- Arg-Attacker: the terrorists
- Arg-Target: [NULL]
- Arg-Instrument: bomb
- Arg-Time: At daybreak on the 9^{th}
- Arg-place: Nazareth

2.1.1 Open-Domain Event Extraction

Unlike closed-domain event extraction, open-domain event extraction does not need predefined event schemas. Open-domain event extraction aims at detecting events and, in some cases, clustering the detected events based on extracted event keywords. The event keywords refer to the words/phrases mostly describing the event.

Sentence	Event Components	Values
At daybreak on the 9^{th}, the terrorists set off a truck bomb attack in Nazareth.	Event Type	Conflict-Attack
	Trigger	attack
	Arg-Attacker	the terrorists
	Arg-Target	NULL
	Arg-Instrument	bomb
	Arg-Time	at daybreak on the 9^{th}
	Arg-Place	Nazareth
Baghdad, a cameraman died when an American tank fired on the Palestine Hotel.	Event Type	Life-Die
	Trigger	die
	Arg-Victim	cameraman
	Arg-Instrument	American tank
	Arg-Place	Baghdad
	Event Type	Conflict-Attack
	Trigger	fired
	Arg-Target	Palestine hotel, cameraman
	Arg-Instrument	American tank
	Arg-Place	Baghdad

Table 2.2: Event extraction examples with their event types and argument roles.

The Topic Detection and Tracking (TDT) [Allan, 2012] public evaluation program aims at automatically detecting unreported events or tracking the progress of the previously detected events from the news texts. Besides events, the TDT also defines the story as some segment of the news article that describes specific events. Furthermore, it defines a topic as a set of events in articles that are strongly related to some real-world topic. Based on the above definition, it defines the following tasks:

- **Story Segmentation**: Detecting the boundaries of a story from news articles.
- **First Story Detection**: Detecting the story that discusses a new topic in the stream of news.
- **Topic Detection**: Grouping the stories based on the topics they discuss.
- **Topic Tracking**: Detecting stories that discuss a previously detected topic.
- **Story Link Detection**: Deciding whether a pair of stories discuss the same topic.

The first two tasks mainly focus on event detection, while the remaining three focus on event clustering. The relations between the five tasks are clear. Each of them needs a specific evaluation process and each task needs a specific method to address.

Linguistic Data Consortium (LDC)[2] provides a series of corpora to support the research in open-domain event extraction for the above TDT tasks, including TDT-1 to TDT-5. These datasets include text and speech in English and Chinese [Allan et al., 1998; Allan, 2012]. Each TDT corpus contains millions of news stories with annotated topics collected from various news sources like newswire and broadcast articles. The datasets present the story-topic labels in a way that, if the story is exclusively discussing the topic, then it is assigned an YES for the given topic. If the discussion of the given topic in the story is less than 10%, then a BRIEF label is assigned. Otherwise, the default value NO is assigned for the story for the given topic.

Besides using the TDT data provided and also TDT tasks defined above, many other types of research have been conducted for detecting and clustering open-domain events from news articles [Tanev et al., 2008; Piskorski et al., 2011; Ribeiro et al., 2017; Yu and Wu, 2018; Liu et al., 2008]. For example, for global crisis monitoring by using online news articles, the Joint Research Centre of the European Commission conducted research on extracting violent events with violence-related keywords such as killed or kidnapped, with their designed monitoring pipeline system [Tanev et al., 2008; Piskorski et al., 2011]. Liu et al. [2008] conducted research to extract key entities on a specific day and track their trends within a specific time window. Further, the news articles are clustered to generate significant daily events about various topics such as entertainment, societies, sports, economics, and politics.

2.1.2 Closed-Domain Event Extraction

Closed-domain event extraction uses predefined event schemas in ACE05, such as exemplified in Table 2.1, to discover and extract events of a particular type from texts. An event schema

[2] https://www.ldc.upenn.edu/

contains several event types and their corresponding event structures, i.e., event type to arguments templates.

For the remaining discussion, we will focus on the closed-domain event extraction task.

2.2 Terminology

Many frontier technologies have been applied to the event extraction task such as: machine learning, pattern matching, NLP techniques, and deep learning disciplines. Furthermore, event extraction can help identify/extract structure information from massive text data, which improves word/research timeliness and provides a way for quantitative analysis in different domains. Hence, event extraction has broad applications in a variety of domains. An event is a specific occurrence involving participants. When performing closed-domain event extraction we follow the terminology defined in *ACE (Automatic Content Extraction) English Annotation Guidelines for Events*.[3]

- **Entity**: The entity is an object or group of objects in a semantic category. Entities mainly include people, organizations, places, times, and things. In Table 2.2, the words the terrorists, bomb, Nazareth, Baghdad, cameraman, American tank, and Palestine Hotel, are entities.

- **Event mentions**: The phrases or sentences describing the event contains a trigger and corresponding arguments.

- **Event Type**: The event type describes the nature of the event and refers to the category to which the event corresponds, usually represented by the type of the event trigger. For example, in Table 2.2, attack triggers a Conflict-Attack event, die triggers a Life-Die event, and fired triggers a Conflict-Attack event.

[3] https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-events-guidelines-v5.4.3.pdf

- **Event Trigger**: Event trigger refers to the core unit in event extraction, a verb or a noun. Trigger identification is a key step in event extraction.

- **Event Arguments**: Event argument is the main attribute of events. The arguments always belong to the corresponding event type. For example, in Table 2.2, the terrorists, bomb, at daybreak on the 9^{th}, and Nazareth are the event arguments belonging to the event type Conflict-Attack.

- **Argument Role**: An argument role is a role played by an argument in an event, that is, the relationship between the event arguments and the event triggers. Moreover, the argument role that an event type can have is predefined in closed-domain event extraction. For example, in Table 2.1, the event type Conflict-Attack has argument roles like Attacker, Target, Instrument, Time, and Place. In a concrete example in Table 2.2, Palestine hotel and cameraman serve as the targets, American tank serves as the instrument, and Baghdad serves as the place for the Conflict-Attack event type.

2.3 Sub-tasks

Generally, the event extraction task includes four sub-tasks: trigger identification, trigger classification, argument identification, and argument role classification. Trigger identification and trigger classification combined can be classified as the event detection task [Liao et al., 2021; Li et al., 2020b; Cao et al., 2021; Lin et al., 2019].

- **Trigger Identification**: Generally, the trigger is considered the core component in event extraction that can represent an event's occurrence. The trigger identification sub-task is to identify the trigger words from the texts. For example, for the first sentence in Table 2.2, attack is identified as the trigger for the first sentence. For the second sentence, there are two event occurrences: die is identified as the first trigger, and fired is identified as the second trigger.

- **Trigger Classification**: Trigger classification first needs to determine if an event occurs according to the identified triggers. Moreover, if the sentence has events, we must classify the event triggers to their corresponding event types. For example, in Table 2.2, the sub-task targets to identify that attack and fired trigger a Conflict-Attack event and that die triggers a Life-Die event. Thus, the trigger classification sub-task can be modeled as a multi-class classification task.

- **Argument Identification**: Argument identification is to identify all the arguments contained in the event text. Argument identification usually depends on the output of the trigger identification and trigger classification tasks since the arguments are attached to an event type identified by the previous two tasks. For example, in Table 2.2, for Life-Die event, we want to identify the arguments cameraman, American Tank and Baghdad.

- **Argument Role Classification**: Argument role classification is based on the argument categories in the predefined schema for a specific event type, and the category of each argument is classified for each argument identified in the argument identification step. For the extracted arguments in Table 2.2, this sub-task is to classify cameraman as the victim, American tank as the instrument, and Baghdad as the place for the Life-Die event type.

2.4 Event Extraction Corpora

The availability of labelled datasets for event extraction has become the driving force behind the advancement of new technologies developed. In this section, we summarize the datasets generated for event extraction tasks.

2.4.1 ACE: The Automatic Content Extraction

The ACE (Automatic Content Extraction) Multilingual Training Corpus[4] provides annotated data for different extraction tasks, including event, entity, time, value, and relation extractions. Two individual annotators labelled every text sample, and a senior annotator monitored their process and adjusted the discrepancies between them.

Events in the ACE dataset have complex event types and argument structures. The ACE05 event corpus defined eight event types and thirty-three sub-types. Each event sub-type corresponds to a set of predefined argument roles. There are thirty-six argument roles for all event sub-types. Most of the research based on the ACE dataset only targets on extracting the event sub-type independently, without considering extracting the event type and sub-type in a hierarchical way. Table 2.10 provides all the eight event types along with their corresponding sub-types. The ACE05 corpus provides a total of 599 annotated documents and about 6000 labelled events across different languages, including English, Arabic and Chinese, from different media sources such as newswire articles, broadcast news, and broadcast conversation. Table 2.4 provides their source statistics.

Event type index	**Event types**	**Sub-type index**	**Sub-types**
1	Life	1-5	Be-Born, Marry, Divorce, Injury, Die
2	Movement	6	Transport
3	Contact	7-8	Meet, Phone-write
4	Conflict	9-10	Attack, Demonstrate
5	Business	11-14	Merge-org, Declare-bankruptcy, Start-org, End-org
6	Transaction	15-16	Transfer-money, Transfer-ownership
7	Personnel	17-20	Elect, Start-position, End-position, Nominate
8	Justice	21-33	Arrest-jail, Execute, Pardon, Release-parole, Fine, Convict, Charge-indict, Trial-hearing, Acquit, Sentence, Sue, Extradite, Appeal

Table 2.3: Event types and sub-types of the ACE05 dataset.

[4] https://catalog.ldc.upenn.edu/LDC2006T06

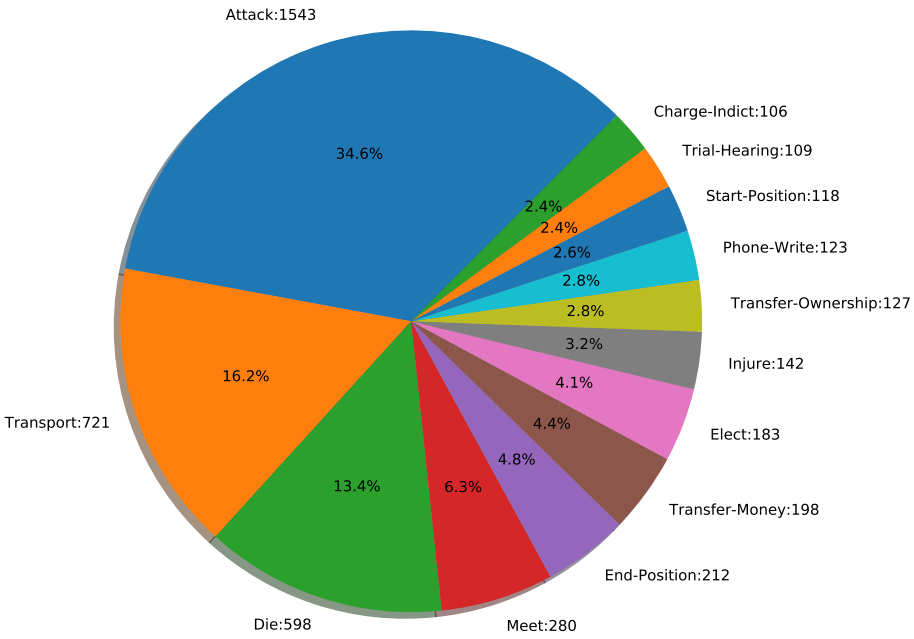

Figure 2.1: Data distribution of the ACE05 dataset. It contains 33 event types, 13 of which with an instance count (event mentions) more than 100, and are shown in the pie chart. The other 20 event types, not shown in the pie chart, have less than 100 instances.

2.4.2 TAC KBP: Text Analysis Conference Knowledge Based Filling

Text Analysis Conference Knowledge Based Filling is known as TAC KBP. The goal of the TAC KBP event tracking from 2015 to 2017 is to extract related information about an event and fill it into a knowledge base. TAC KBP 2015[5] defines nine different event types and thirty-eight event sub-types in English. TAC KBP 2016[6] and TAC KBP 2017[7] have the annotation data in three different languages: English, Chinese, and Spanish. It contains eight event types and eighteen event sub-types.

[5] `https://tac.nist.gov/2015/KBP/data.html`
[6] `https://tac.nist.gov/2016/KBP/data.html`
[7] `https://tac.nist.gov/2017/KBP/data.html`

Data Source	Language		
	English	Chinese	Arabic
Newswire (NW)	20%	40%	40%
Broadcast News (BN)	20%	40%	40%
Broadcast Conversation (BC)	15%	0%	0%
Weblog (WL)	15%	20%	20%
Usenet Newsgroups (UN)	15%	0%	0%
Conversational Telephone Speech (CTS)	15%	0%	0%

Table 2.4: Data sources with their statistics in the ACE05 dataset.

2.4.3 MAVEN: A Massive General Domain Event Detection Dataset

The MAVEN dataset [Wang et al., 2020] is designed to solve two problems of the existing event extraction datasets.

- **Data Scarcity**: Existing small-sized datasets are insufficient for training and stably bench-marking increasingly sophisticated deep neural networks.
- **Low Coverage of Event Types**: Limited event types in the existing datasets hinder the development of event extraction techniques for general domains.

Event detection (ED), which includes trigger identification and trigger classification defined in Section 2.3, is an essential information extraction task. Its goal is to identify event triggers (the words or phrases that evoke events) and classify the triggers into event types. The MAVEN dataset focuses on the data used for event detection, and it does not include the arguments or argument role labels.

An event is a specific occurrence involving participants. In MAVEN each event will be labelled with an event type. An event mention is a phrase or sentence describing the event containing a trigger and its corresponding arguments. Since the same event may be mentioned several times in the same sentence, there are generally more event mentions than events. An event trigger is the key word or phrase in an event mention that most clearly explains the event occurrence. MAVEN contains 111,611 events and 118,732 event mentions

Dataset	**#Documents**	**#Tokens**	**#Sentences**	**#Event Types**	**#Events**	**#Event Mentions**
ACE05	599	303K	15,789	33	4,090	5,349
TAC KBP 2015	360	238K	11,535	38	7,460	12,976
TAC KBP 2016	169	109K	5,295	18	3,191	4,155
TAC KBP 2017	167	99K	4,839	18	2,963	4,375
Total TAC KBP	696	446K	21,669	38	13,614	21,506
MAVEN	**4,480**	**1,276K**	**49,873**	**168**	**111,611**	**118,732**

Table 2.5: Statistics of documents, tokens, sentences, event types, events, and event mentions in the MAVEN dataset, compared to other widely-used event detection datasets. The number of event types shows the most fine-grained event types, i.e., the sub-types used in the ACE05 and TAC KBP datasets. For the multi-lingual datasets like TAC KBP 2016 and TAC KBP 2017, we only report the statistics of the English subset for direct comparisons with MAVEN.

compared to the widely used ACE05 dataset, which has 4090 events and 5349 event mentions. MAVEN is 20 times larger than ACE05 in this sense. Furthermore, MAVEN covers 168 event types, five times larger than ACE05's 33 event types. The details of the data size comparison between MAVEN and other datasets is in Table 2.5.

MAVEN has a large amount of event types, i.e., 168. MAVEN has 41% and 82% event types having more than 500 and 100 instances, respectively, as seen in Figure 2.2. The ACE05 dataset, in comparison, only has 39% of the event types with more than 100 instances, as shown in Figure 2.1. MAVEN significantly alleviates the data scarcity problem, which will help develop more robust event detection models. The event types mainly use FrameNet [Baker et al., 1998], primarily a linguistic resource constructed by linguistic experts, as an event type source. FrameNet prioritizes lexicographic and linguistic completeness over the easiness of annotation [Aguilar et al., 2014]. A frame in FrameNet is defined as a composition of frame name, a set of Frame Elements (FEs) and a list of Lexical Units (LUs). A LU is a word or phrase that evokes the corresponding frame. FEs indicate the set of semantic roles associated with the frame. In order to facilitate a large number of annotators in

the crowd-sourcing environment, MAVEN ends up getting the 168 event types by performing the following steps. First, it recursively selects the frames having inheritance, subframe or using relations with the event frame as in Li et al. [2019]. Secondly, a manual step is conducted to further filter out abstracted frames like process resume. Furthermore, a merge of similar frames and assemble too fine-grained frames into more generalized frames step is conducted. As an example, choosing and adopt selection can be merged, and visitor arrival and drop in on can be assembled into a more general one as arriving. Based on the FrameNet inheritance relation, the final 168 event types can be organized into a tree-structured event type hierarchy. During the annotations, the annotator are asked to label with the most fine-grained event types like theft and robbery. The coarse-grained types like committing crime are only used if the events have no fine-grained event types. In Figure 2.3, a hierarchical event type tree rooted from the coarse-grained change is shown. The extensive coverage of event types and complicated structured information from the event types tree can be leveraged by advanced event detection method development.

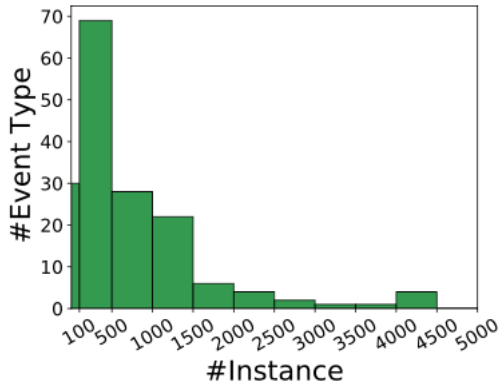

Figure 2.2: Distribution of MAVEN event types by their corresponding instance numbers. [Wang et al., 2020]

MAVEN follows a simple intuition that the articles describing grand "topic events" may contain much more basic events than the articles about specific entity definitions. MAVEN adopts EventWiki [Ge et al., 2018], a knowledge base for major events, where each major event is described in a Wikipedia article. MAVEN uses the EventWiki articles as the base

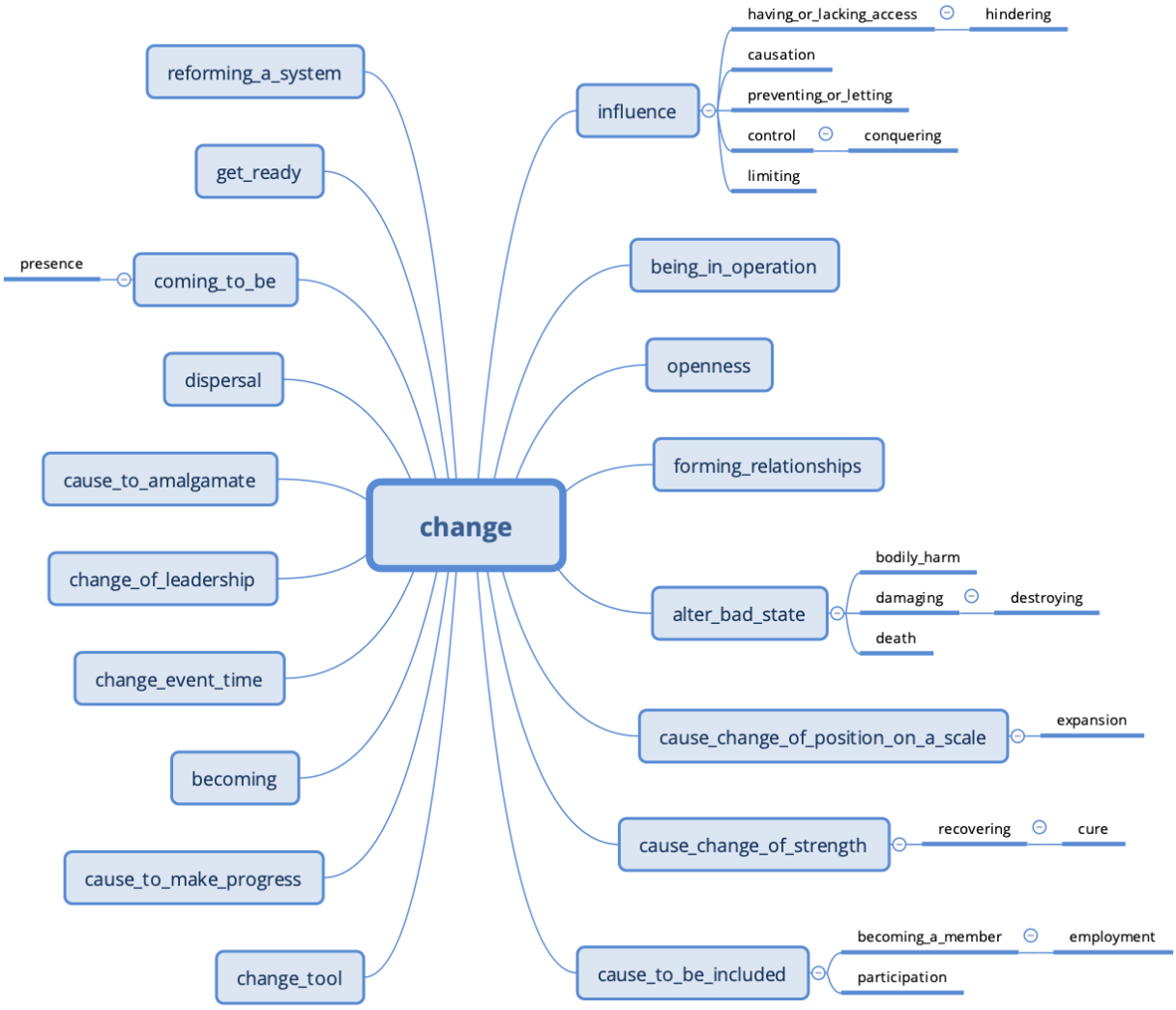

Figure 2.3: The hierarchical event types used in MAVEN, rooted from the coarse-grained event type change.

and manually selects some articles to be labelled with the 168 event types described above. To choose the high quality articles, MAVEN performs pre-processing, such as (1) filtering out the articles with fewer than five sentences, and (2) only using the introductory sections to be annotated, following the same settings used in Yao et al. [2019]. EventWiki comes with the topic information of the documents, so the MAVEN dataset also comes with the document topics. Table 2.6 shows the statistics of MAVEN documents for top EventWiki topics. In this way, MAVEN can facilitate the advances in developing methods for event detection in a more general domain by leveraging external information like topics of the

documents where the events reside. For example, we use the topic information to conduct research in topic-aware event detection in Chapter 4, where we use the topic information as prior knowledge to enrich sentences with contextual representations in a multi-task set-up of deep neural network.

Topic	**#Documents**	**Percentage**
Military conflict	1,458	32.5%
Hurricane	480	10.7%
Civilian attack	287	6.4%
Concert tour	255	5.7%
Music festival	170	3.8%
Total	2,650	59.2%

Table 2.6: Document counts and percentages of the top five EventWiki topics in MAVEN.

2.4.4 RAMS: Roles Across Multiple Sentences

The argument linking task is defined as linking all the argument role mentions to the argument role. The Roles Across Multiple Sentences (RAMS) dataset [Ebner et al., 2020] is mainly created for the argument linking task since the existing datasets for cross-sentence role argument linking are small and cannot support developing complicated neural models. Argument linking aims at identifying event arguments of a given event type across multiple sentences. RAMS contains 3,194 documents, 9,124 annotated events across 139 event types and 65 argument roles, shown in Table 2.7. The broad coverage and diversity of the event types is the reason we choose RAMS as our experiment dataset in topic-aware event detection in Chapter 4.

	Train	**Dev**	**Test**	**Total**
Documents	3,194	399	400	3,993
Event Instances	7,329	924	871	9,124
Event Types	139	131	–	139
Roles	65	62	–	65
Arguments	17,026	2,188	2,023	21,237

Table 2.7: RAMS statistics of documents, examples, event types, roles, and arguments.

2.4.5 Other Domain-Specific Corpora

In addition to the aforementioned well-known corpora, some domain-specific event corpora are also released. The BioNLP Shared Task[8] is an initiative that aims at extracting fine-grained biomolecular event from biomedical related scientific documents [Bossy et al., 2013]. Teams participating in the BioNLP Shared Task initiative compiled various manually annotated biological corpora, including the GENIA event corpus, the BioInfer event corpus, the Gene regulation event corpus, and the GeneReg event corpus [Nédellec et al., 2013; Vanegas et al., 2015]. GNBusiness [Liu et al., 2019] collects news reports from Google Business News describing each event type from different sources. The GNBusiness dataset contains 55,618 articles collected from Oct 2018 to Jan 2019. Time and Event Recognition for Question Answering Systems (TERQAS) workshop[9] built a corpus called TimeBANK. TimeBANK is annotated with events, times, and temporal relations from various of media sources for breaking news event extraction. The Chinese Event Corpus (CEC) also annotates breaking news events in Chinese [Meng, 2015]. The MUC series of corpora contain the domain-specific events in military intelligence, terrorist attack, chip technology, and finance [Sundheim and Chinchor, 1993; Hirschman, 1998]. There are also event extraction data for the music domain, introduced in Ding et al. [2011].

2.5 Formulating Event Extraction Sub-tasks as Different Machine Learning Tasks

2.5.1 Formulating EE as a Classification Based Task

The event extraction task can be modeled as a token-level classification task that directly maps a token to an event type or argument role corresponding to the detected event types as shown in Figure 2.4. Given n predefined event types e_i, $i \in \{1, \ldots, n\}$, and an event mention

[8] `https://sites.google.com/site/bionlpst/`
[9] `https://slideplayer.com/slide/6426164/`

m, the model needs to output a result vector T, where T_i represents the probability that m belongs to event type i. After obtaining the final event type e_k of m, the model outputs a matrix R, where R_{ij} is the probability that the token a_i belongs to the jth argument role for event type e_k.

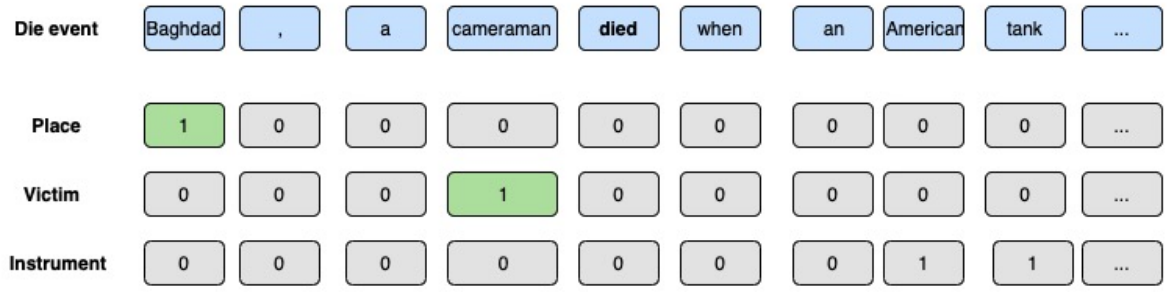

Figure 2.4: Classification based task.

2.5.2 Formulating EE as a Sequence Labelling Based Task

The simple classifier layer from hidden representations to tags sequence doesn't consider the sequence nature of the tags. As a result, only using a linear layer to output the tags does not consider the coherency of tags during prediction. The BIOE schema is adopted for sequence labeling based event extraction solutions. Here, the tags "B" or "E" indicate the corresponding word is the beginning or ending of an entity, respectively; "I" indicates the word is inside an entity, and "O" indicates the word is outside any entities. For example, given the tags B, I, O, E, the model may predict a mis-aligned tag sequence like $BOIE$, leading to an incoherently extracted entity. In order to alleviate this problem, we use the Conditional Random Fields (CRF) to consider the sequence nature of the tags and capture the dependency between the output tags.

The CRF is described as the following. Given an input sequence $x = x_1, x_2, \ldots, x_n$ and its corresponding label sequence $y = y_1, y_2, \ldots, y_n$, the conditional probability of y given x

for general CRF can be written as

$$\Pr(y|x;\Phi) \propto \exp\left(\sum_{k=1}^{K} \varphi_k f_k(y,x)\right), \tag{2.1}$$

where $f_k(y,x)$ is the feature function, φ_k is the corresponding weights to be learned, and K is the number of features. The graphical structure of Equation (2.1) is shown in Figure 2.5. In our sequence tagging task, a linear-chain CRF is usually adopted, which means the feature functions only depend on the neighboring tags y_t, y_{t-1}, x_t at time step t, as shown in Figure 2.6. In this way, we can re-write the above equation as:

$$\Pr(y|x;\Phi) \propto \sum_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \varphi_k f_k(y_{t-1}, y_t, x_t)\right), \tag{2.2}$$

where T is the length of the tag sequence. The graphical structure of Equation (2.2) is shown in Figure 2.6.

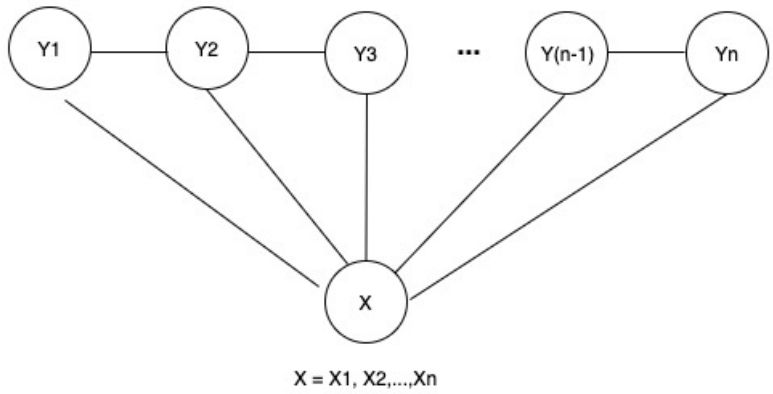

Figure 2.5: Graphical structure of the general CRF.

Event extraction sub-tasks trigger classification and argument role classification can be modeled as token-level multi-classification task [Chen et al., 2020a; Gui et al., 2020; Ramponi et al., 2020] using BIOE or BIE schema as its output. Using the output tag like BIOE is different from directly predicting each token on the event types, since, generally, the decoder process of generating BIOE uses the benefit of CRF [Lafferty et al., 2001]. CRF takes

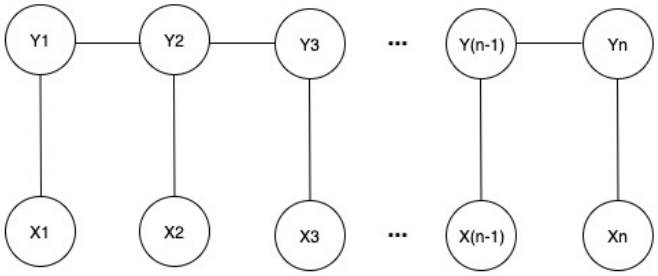

Figure 2.6: Graphical structure of the linear-chain CRF.

advantage of the label dependencies. For example, the probability of I following B will be larger than the probability of I following E. In this way, the sequence labelling method marks out the target value from the texts, which is suitable for event extraction. As shown in Figure 2.7, for the die event, given a sequence of tokens in the text $(x_1, x_2, \ldots, x_n)$, the tags output $(y_1, y_2, \ldots, y_n)$ of the sequence labelling model tag all the tokens in the text. As shown in Figure 2.7, from the tags output "B-Place O O B-Victim O O O B-Instrument E-Instrument," we can decode that, the place for the event is Baghdad, the victim is cameraman, and the instrument is American tank for the die event. The example is shown for argument role classification with a given event trigger. For the event trigger classification task, a similar sequence labelling method can be applied.

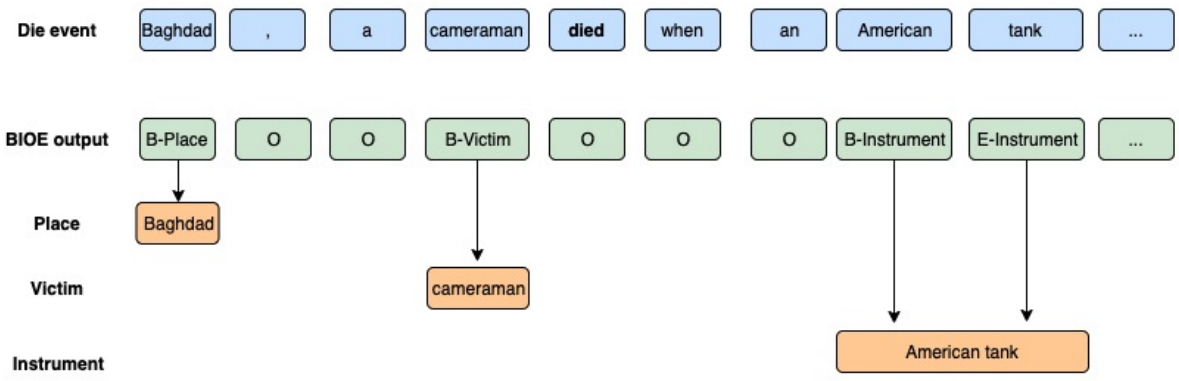

Figure 2.7: Sequence labelling based task.

2.5.3 Formulating EE as a Question Answering Based Task

The machine reading comprehension model [Guo et al., 2020; Chen et al., 2021] can understand text in natural language and answer questions about it [Yuan et al., 2021; Du and Cardie, 2020]. For the trigger detection and classification task, we can use the question templates like what is the trigger, trigger, action, and verb. After obtaining the trigger phrases, we embed the information in the argument roles detection and classification tasks corresponding to the event trigger by using the trigger information in the argument role question. This can leverage the dependencies information between event types and argument roles. The argument roles question is defined by considering the predefined event type and its argument roles templates. For example, the Declare-Bankruptcy event has two argument roles: Org and Place. For the Org role, the question can be like " What declare bankruptcy?" For the Place role, the question can be like "Where the event takes place?" To summarize, we have a fixed set of questions Q_e for the triggers, and for each event type, we have a set of questions defined for its role arguments like Q_{er}. The designed questions are applied to the QA model [Du and Cardie, 2020]. Figure 2.8 shows an example of extracting argument roles for a die event using question answering model. In Du and Cardie [2020], the output layers of event trigger and argument role classification defers. For trigger classification, the output layer predicts the event type for each token in the sentence (or None if there is no event trigger). For argument role classification, the output layer predicts the start and end offsets for the argument span.

2.5.4 Formulating EE as a Sequence-to-Structure Generation Based Task

The sequence-to-structure generation based event extraction method [Lu et al., 2021] extracts triggers and arguments in an end-to-end fashion in Figure 2.9. It uniformly models all trigger detection, trigger classification, argument detection, and argument classification in a single

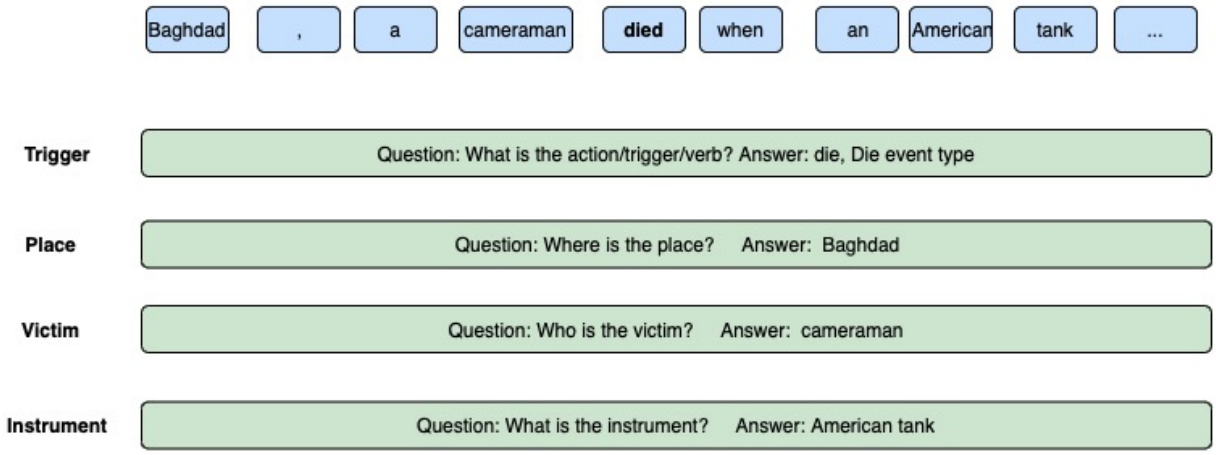

Figure 2.8: Question answering based task.

model by adopting encoder-decoder models [Vaswani et al., 2017]. The encoder-decoder structure is an easy way to convert text to a structured form. The encoder maps an input sequence of symbol representation $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Given z, the decoder then generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time. At each step, the model is auto-regressive, where it consumes previously generated symbols as additional input when generating the next one. The straight-forward solution is to use greedy decoding, which selects the token with the highest probability at the decoding step. Nevertheless, greedy decoding can not guarantee the generation of valid event structures. On the other hand, greedy decoding ignores the useful event schema knowledge. To leverage the event schema knowledge, Lu et al. [2021] propose a trie-based constrained decoding algorithm for event extraction [Chen et al., 2020b; De Cao et al., 2021].

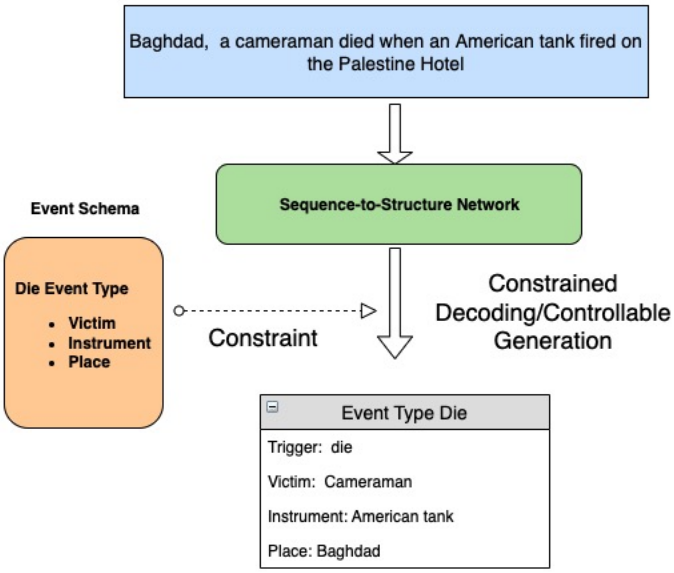

Figure 2.9: Sequence-to-structure generation based task.

2.6 Event Extraction Paradigms

2.6.1 Pipeline-based Versus Joint-based Event Extraction

Pipeline-based Event Extraction Paradigm

The pipeline-based method treats all sub-tasks as independent classification problems. The pipeline-based approach is widely used since it simplifies the entire event extraction task. The pipeline-based method converts event extraction into a multi-stage classification process. Two classifiers are required for the process: the trigger classifier and the the argument role classifier. The trigger classifier identifies the event trigger and classifies its event type. The argument role classifier identifies the argument for a specific event type and classifies the argument into a specific argument role.

The pipeline-based method will first perform trigger classification, and then use the detected triggers as features for the argument role classification. The most significant defect of the pipeline-based method is error propagation through trigger classification to argument role classification. The pipeline-based method uses the trigger classification output as the input for argument role classification. Thus, if there is an error in trigger classification in the

first step, the accuracy of argument role classification will be lowered. Therefore, the pipeline-based event extraction method considers trigger classification as its core task. Furthermore, the trigger classification task always precedes the argument role classification task in the pipeline-based setting. The information learned from the argument role classification can't effectively help the trigger classification task. Thus, there is a missing mutual interaction between trigger classification and argument role classification in the pipeline-based methods.

Joint-based Event Extraction Paradigm

In order to overcome the shortcomings of the pipeline-based method, the joint extraction of triggers and arguments role method has been explored. With the joint-based event extraction method, event triggers and arguments can be extracted simultaneously, compared to the pipeline-based method where trigger classification must be conducted before the argument role classification. In this way, the trigger classification and argument role classification can mutually promote each other's extraction effect. Some of the joint extraction researches have proven that the joint learning method can be better than the pipeline-based method. A classic example is joint event extraction via recurrent neural networks [Nguyen et al., 2016].

2.6.2 Sentence-Level Versus Document-Level Event Extraction

Sentence-Level Event Extraction

Sentence-level event extraction aims at extracting events and their corresponding arguments within single sentences. Most of the research work are sentence-level event extraction based.

Document-Level Event Extraction

Document-level event extraction aims at extracting events across multiple sentences in an article. Compared to sentence-level event extraction, two extra challenges are present. The first challenge is arguments scattering, where arguments of one event might be scattered into different sentences in the document, which means an single event cannot be extracted within

one sentence. Thus, document global information needs to be learned. Furthermore, at the document level, the locations of event triggers and their corresponding event arguments tend to be further apart, which increases the difficulty in modeling their dependencies. The second challenge involves multiple events, where the chance of one document simultaneously containing multiple events is bigger than that of the one-sentence scenario. This requires holistic modeling of the inter-dependencies among multiple events.

2.7 Event Extraction Models

The development of event extraction methods went through three revolutions: (1) event extraction based on pattern matching; (2) event extraction based on traditional machine learning methods; and (3) event extraction based on deep learning methods. Pattern matching methods need acquisition of extraction patterns for different domains. high performance can be achieved in a specific field with pattern matching methods. With traditional machine learning methods, using the labelled corpus in specific domains, the model can be portable and domain agnostic, compared to the pattern matching methods. For pattern matching, specific event patterns need to be constructed for each new domain. However, it still requires us to manually select features and design the classifiers. With deep learning models, it can automatically extract useful features from the texts without the need for in-depth features design required by complex semantic relations in event extraction tasks like traditional machine learning methods do. We summarize the advantages and disadvantages of each type of methods in Table 2.8.

2.7.1 Event Extraction Based on Pattern Matching

The earlier approaches for event extraction are pattern matching techniques. These approaches first need to construct event pattern templates, and then perform pattern matching to extract events and arguments. Event pattern templates can be generated from lin-

	Pattern Matching	Traditional Machine Learning	Deep Learning
Concept	Based on a certain pattern, the text to be extracted needs to match the event patterns	The event extraction problem is transferred to a classification task	The event extraction problem is transferred to a classification task
Advantage	High performance can be achieved in a specific domain by using event patterns	Domain-agnostic, portable	Automatic feature selection, domain-agnostic, portable, can leverage the knowledge in large scale pre-trained model
Disadvantage	Poor portability and flexibility, new event patterns need to be designed for new domain	Needs labelled corpus	Needs labelled corpus on a larger scale
Main Idea	Acquisition of event patterns and matched by the text that events are extracted from	Select features and design classifier	Automatic feature selections from the neural networks with corresponding classifier on top

Table 2.8: Comparison of advantages and disadvantages for pattern matching, traditional machine learning, and deep learning methods.

guistic patterns compiled by the linguistic/domain experts along with manually annotated events [Riloff, 1993].

The first pattern-based extraction system dates back to AutoSlog [Riloff, 1993], a domain-specific event extraction system for terrorist events. AutoSlog exploited a small set of linguistic patterns, shown in Table 2.9, and manually labelled the corpora to obtain event patterns. The linguistic patterns and event patterns are different concepts in AutoSlog. Linguistic patterns, along with the annotated events, are used to generate event patterns. Event patterns are used to perform pattern matching/event extraction. By the design of AutoSlog, it can only extract one argument for one event, so the event corpus is also annotated in that way.

Event Pattern Construction

The event pattern is constructed by using linguistic patterns and annotated event. As an example, given the following sentence:

Index	Linguistic Pattern	Event Pattern Example
1	⟨*subject*⟩ passive-verb	⟨*victim*⟩ was murdered
2	⟨*subject*⟩ active-verb	⟨*perpetrator*⟩ was bombed
3	⟨*subject*⟩ verb infinitive	⟨*perpetrator*⟩ attempted to kill
4	⟨*subject*⟩ auxiliary noun	⟨*victim*⟩ was victim
5	passive-verb ⟨*dobj*⟩	killed ⟨*victim*⟩
6	active-verb ⟨*dobj*⟩	bombed ⟨*target*⟩
7	infinitive ⟨*dobj*⟩	to kill ⟨*victim*⟩
8	verb infinitive ⟨*dobj*⟩	threatened to attack ⟨*target*⟩
9	gerund ⟨*dobj*⟩	killing ⟨*victim*⟩
10	noun auxiliary ⟨*dobj*⟩	fatality was ⟨*victim*⟩
11	noun prep ⟨*np*⟩	bomb against ⟨*target*⟩
12	active-verb ⟨*np*⟩	killed with ⟨*instrument*⟩
13	passive-verb ⟨*np*⟩	was aimed at ⟨*target*⟩

Table 2.9: Thirteen linguistic patterns with examples.

In La Oroya, Junin department, in the central Peruvian mountain range, public buildings were bombed and a car-bomb was detonated.

AutoSlog first employs a syntactic analyzer to identify the part of speech (POS) (such as what tokens in the sentence are subjects, predicates, objects, prepositions) of each sentence in the manually labelled corpora. In the sentence, the phrase public buildings is identified as the subject, and bombed is identified as the passive verb, a case that matches Linguistic Pattern 1 in Table 2.9. As a result, AutoSlog will add bombed into the trigger word dictionary and generate a pattern "⟨*target*⟩ was bombed" with bombing being the event type and public buildings being its target.

Event Pattern Matching

In the process of pattern matching for event extraction, AutoSlog first uses the trigger word dictionary to locate candidate event sentences, and then performs POS tagging on the candidate event sentence. Further, AutoSlog associates the POS tagging features surrounding the detected trigger from the vocabulary to extract the argument and the role of the argument. As an example, given the sentence:

They took 2-year-old Gilberto Molasco, son of Patricio Rodriguez.

The pre-constructed pattern, "took ⟨*victim*⟩" with the kidnapping event type, locates the trigger word took in the above candidate sentence. From the POS tagging, AutoSlog identifies that Gilberto Molasco is a *dobj* (direct object). By combining the predefined event pattern and the POS tagging results, AutoSlog can extract an event of the kidnapping event type with its trigger word being took and its victim argument being Gilberto Molasco.

It could be pretty expensive for event patterns learned from annotated data to label all the patterns with expert knowledge. In an extension to AutoSlog, Riloff and Shoen [1995] proposed to obtain new event patterns from un-annotated corpora along with the thirteen linguistic patterns.

2.7.2 Event Extraction Based on Traditional Machine Learning Models

Event extraction can be modeled as a classification task and traditional classification methods like Support Vector Machines (SVM) and Naïve Bayes can be used [Saha et al., 2011; Wu et al., 2013]. The basic ideas behind machine learning approaches are mostly the same, that is to learn a classifier model from the training data and apply the learned classifiers to event extraction tasks on the new text. The difference between traditional machine learning models or neural network models (so-called deep learning) is that deep learning can automatically learn the features, while traditional machine learning algorithms need feature engineering.

Feature engineering includes extracting features from texts as input and choosing what features are important for specific tasks. The widely used text features can be generally divided into three types: lexical features, syntactic features, and semantic features. All of those features can be obtained by applying some open-source NLP tools.

- **Lexical features** refer to information about the words, meaning and structure. Commonly used lexical features include: (1) upper case, lower case, or capitalization of

words; (2) lemmatized words, for which their root forms are used (e.g., the word desks is an inflected form of desk); and (3) POS tags: categorizing words in a text in correspondence with a particular part of speech, depending on the word's definition and its context, such as nouns, verbs, adjectives, or adverbs.

- **Syntactic features** refer to the structure of the text and the syntactic rules such as how words make up phrases and how phrases make up sentences.

  The syntactic features are usually obtained from dependency parsing, which analyzes the grammatical structure of a sentence based on the dependencies between the words in a sentence. Dependency parsing creates edges between words representing different types of relations and organizes them as a tree structure. Commonly used syntactic features include: (1) the label of the dependency path; (2) the dependency word and its lexical features; and (3) the depth of the word in the dependency parsing tree.

- **Semantic features** refer to the meaning of the text, as it is constructed from the meaning of the words and their compositions.

  Some commonly used semantic features for event extraction include: (1) synonyms of the word; and (2) event type and event trigger features that can be used in argument classification task.

After the features have been extracted from the text, feature engineering needs to be applied in order to choose the most important features. Further, the feature engineering needs to figure out how to combine the selected features in a high dimensional vector space to represent each word in the sentence. Finally, the chosen text features along with the event related labels will be used to train a traditional machine learning models.

2.7.3 Event Extraction Based on Deep Learning Models

CNN-based Models

We will use one of the CNN based event extraction works, DMCNN [Chen et al., 2015], as a representative example. Traditional approaches for ACE event extraction tasks primary rely on manually designed elaborated natural language processing (NLP) features and complicated NLP tools to extract the features. However, these traditional approaches lack the generalization ability, need much human effort to design the features, and are not robust against data sparsity problems. DMCNN aims at automatically extracting lexical-level and sentence-level features without using manually defined NLP features or complicated NLP rules. DMCNN introduces a word representation model to capture semantic similarities between words. In order to capture sentence-level features, DMCNN adopts a framework based on Convolutional Neural Networks (CNN).

In most NLP tasks, deep learning models operate on a word embedding, which is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. Convolutions of kernel size n on word embeddings will learn to emphasize or disregard n-grams in the input. The CNN, with max-pooling layers, is an excellent way to capture the semantics between long-distance words in a sentence [Collobert et al., 2011].

The highlights of DMCNN are twofold:

- DMCNN automatically induces lexical-level features by using a word embedding representation learning, and sentence-level features by using a CNN based feature extractor without using human manually defined NLP features and complicated NLP tools to extract those features.

- However, the CNN can only capture sentence-level clues. DMCNN uses a dynamic multi-pooling layer determined by the triggers and arguments in order to store more

critical information for triggers and arguments.

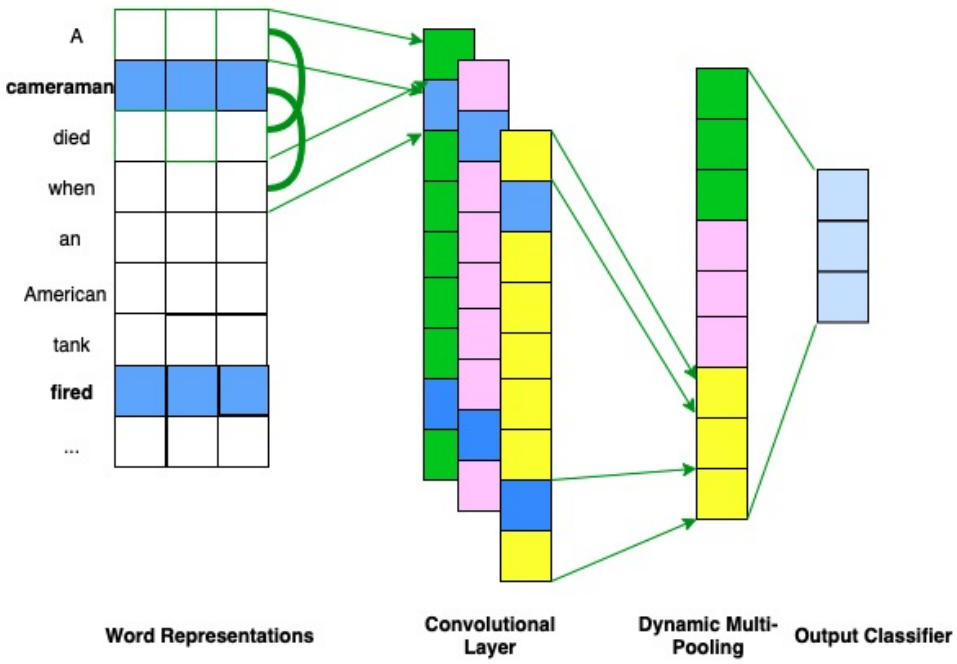

Figure 2.10: A simplified architecture of CNN-based event extraction. The process of feature extraction with event trigger fired and the corresponding argument cameraman is illustrated.

Figure 2.10 shows a simplified version of using word embedding representations and a dynamic multi-pooling CNN as a feature extractor for event extraction. The word representation layers can be obtained using a pre-trained embedding from the Skip-gram model [Tomas et al., 2013]. The convolutional layer convolves semantic relationships between h-grams by using a kernel of size $h \times d$ where d is the embedding dimension for the word, and compresses this valuable semantic information into feature maps. A kernel filter is a vector $w \in \mathbb{R}^{h \times d}$ that can be applied to a sequence of words with window size h to generate a new feature c_i as:

$$c_i = \phi(w \cdot x_{i:i+h-1} + b), \tag{2.3}$$

where $b \in \mathbb{R}$ is a bias term, ϕ is a non-linear function, and the notion $x_{i:i+h}$ refers to the concatenation of words $x_i, x_{i+1}, \ldots, x_{i+h}$. This kernel filter is applied to each possible window of words in the sentence $x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}$, producing a feature map c_i where

i ranges from 1 to $n-h+1$. This feature map calculation for one feature can be generalized to multiple features by using multiple kernel filters. Assuming we have m kernel filters, $W = (w_1, w_2, .., w_m)$, then the convolutional operations can be applied for each kernel filter w_j, where j ranges from 1 to m. Formally, the equation for computing the feature map for feature c_{ji} is:

$$c_{ji} = \phi(w_j \cdot x_{i:i+h-1} + b_j). \tag{2.4}$$

The result of the convolutional computations is a matrix $C \in \mathbb{R}^{m\times(n-h+1)}$.

Taking Figure 2.10 as an example for illustration purposes, the size of the embedding is three. The feature map calculation uses 3 kernels with kernel size 3×3 to convolve (an illustration of the process is shown in Figure 2.11), which can gather the 3-grams information (taking $h = 3$ in the Skip-gram model [Tomas et al., 2013]) from the text input. Then the dynamic multi-pooling layer is applied based on the separation boundary of the event trigger and its corresponding argument. The feature extracted from each kernel is then concatenated to form the last hidden features presented to the classification layer.

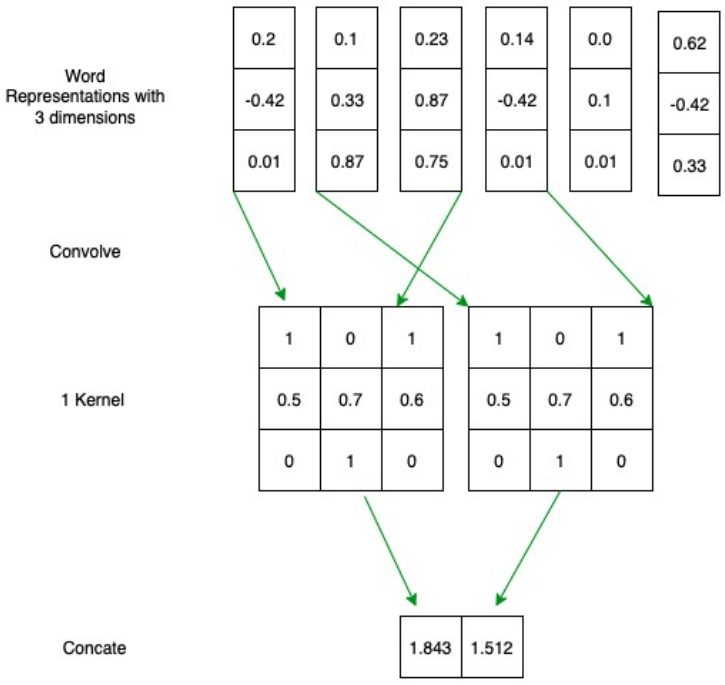

Figure 2.11: An illustration example of using CNN kernels to convolve n-grams ($n = 3$).

RNN-based Models

We will use one of the Recurrent Neural Networks (RNNs) based event extraction works joint event extraction via recurrent neural networks [Nguyen et al., 2016] as a representative example. Deep Neural Networks (DNNs) have revolutionized different tasks in Natural Language Processing. CNNs and RNNs, the most used two main types of DNNs, are widely explored to tackle different NLP tasks. RNNs are good at modeling units in sequence, while CNNs are good at extracting position-invariant features. The state-of-the-art solutions for many NLP tasks often switch between CNNs and RNNs [Yin et al., 2017]. The state-of-the-art models for the event extraction problem either followed the joint architecture via structured predictions with rich global and local features [Li et al., 2013], or used a CNN in a pipeline paradigm to extract event triggers first, then argument roles for each detected event trigger. The former can mitigate the error propagation problem that comes along with the pipeline-based approach and explore the inter-dependencies of event trigger and argument roles via joint training paradigm. The latter, on the other hand, can automatically extract lexical-level and sentence-level features through word representation layers along with the CNN feature extractor.

Unlike the above work, JRNN proposes a way to jointly extract event triggers and argument roles with automatically extracted features, benefiting from the advantages of the two models introduced above.

The highlights of JRNN are twofold:

- JRNN proposes a joint model to do event extraction based on bidirectional RNN to overcome the limitation of previous work.
- JRNN introduces the memory matrix that can effectively help capture the dependency between event triggers and arguments roles.

Figure 2.12 shows a simplified version of using word representations and a bidirectional RNN architecture to do event trigger and event argument role classification jointly. Let

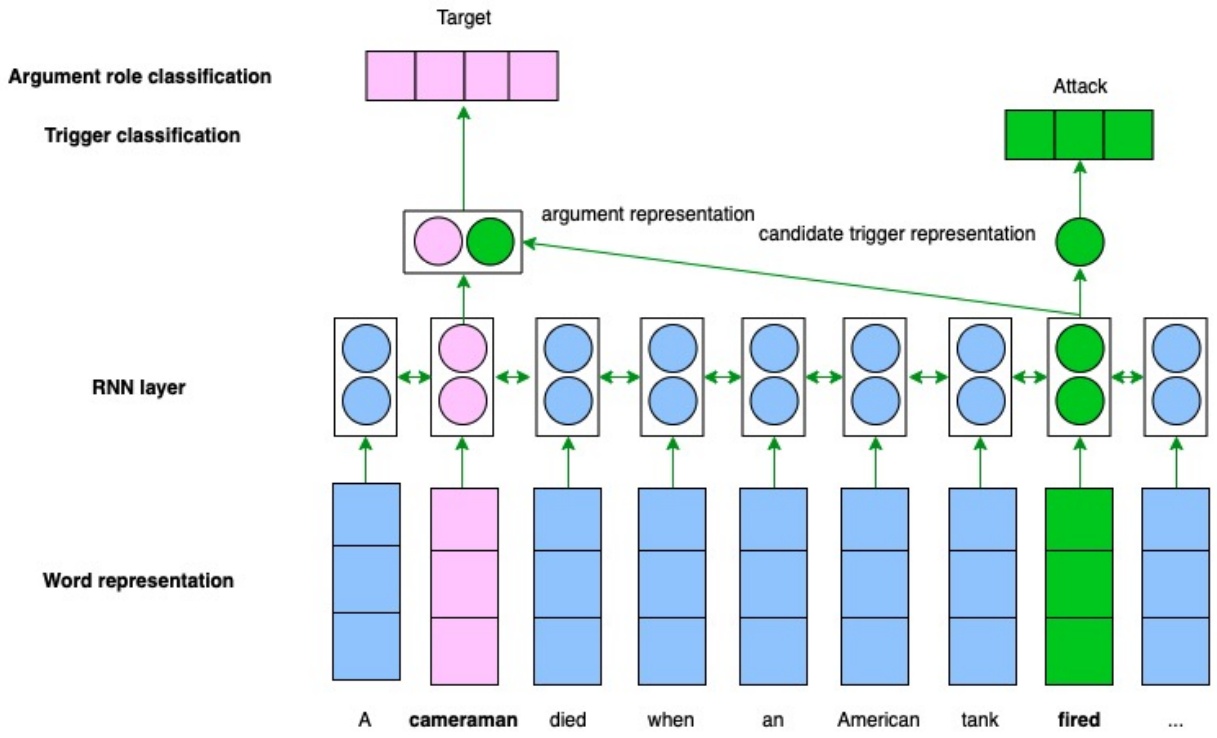

Figure 2.12: A simplified architecture of RNN based event extraction for the input sentence *A cameraman died when an American tank fired on the Palestine Hotel.*

$X = (x_1, x_2, \ldots, x_n)$ be the input sequence. At each time step i, we compute the hidden vector y_i based on the input vector x_i and the previous time step $(i-1)$'s hidden vector, using a non-linear transformation function ϕ: $y_i = \phi(x_i, y_{i-1})$. The above recurrent computation is done for each time step i in the input X. A sequence of hidden vectors $(y_1, y_2, \ldots, y_n)$ is generated as the output. From the above explanation, we can see that RNN inductively accumulates the context information of the input text from time step 1 to time step i into the hidden vector y_i. This makes y_i a rich representation carrying all information till time step i. However, y_i might not be rich enough to do trigger and argument role classification for a token at time step i, since such predictions might also rely on information from the future (time step $(i+1)$ to time step n). In order to solve this issue, a bidirectional RNN is introduced. We can run another sequence of recurrent computation in a reverse direction from x_n to x_1 to generate a second sequence of hidden vectors as: $(y'_1, y'_2, \ldots, y'_n)$. Eventually, we can obtain a complete sequence of hidden representations as $(h_1, h_2, \ldots, h_n)$, where $h_i = [y_i, y'_i]$, and

the $[a, b]$ notation here means the concatenation of a and b. In this way, h_i encapsulates the context information from token 1 to token n over the whole sentence with a focus on time step i. As shown in Figure 2.12, the hidden representation of token i is fed into the classifier layer to do trigger classification. A concatenation of token i's representation and event trigger's representation is fed into a classifier layer for argument role classification. Since the argument role classification has a dependency on event trigger classification results.

Introduction to Transformer

A Transformer is a deep learning model architecture introduced in attention is all you need [Vaswani et al., 2017]. The Transformer is the first model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNN or convolution.

An attention can be viewed as function that maps a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is calculated as a weighted sum of the values, where the weight assigned to each value is computed by a function operated on query with the corresponding key.

A particular attention type "Scaled Dot-Product Attention", is shown in Figure 2.13. The input consists of queries and keys of dimension d_v, and values of dimension d_v. A dot products of the query with all keys, divided by $\sqrt{d_v}$, followed by softmax operation is applied to obtain the weights for the values.

In practice, a set of queries can be packed together into a matrix Q with dimension $w \times d_v$, where w is the number of the words in the sequence and d_v is the dimension of the hidden vector for each word. In this way, we can compute the attention function on a set of queries simultaneously. The keys and values are also packed together into matrices K with dimension $w \times d_v$ and V with dimension $w \times d_v$. The output with dimension $w \times d_v$ can be computed as,

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_v}})V, \tag{2.5}$$

where QK^T has dimension $w \times w$, and softmax is applied on each row of QK^T.

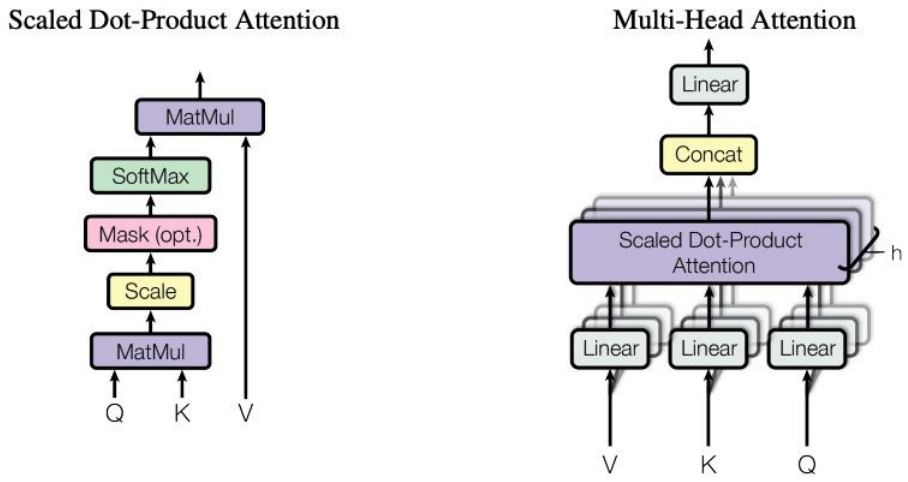

Figure 2.13: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [Vaswani et al., 2017]

Instead of performing a single function with keys, values and queries. It is found beneficial to linearly project the queries, keys and values h times with different learned linear projections. Each linear projection might represent a particular feature. For example, one projection can learn longer-distance tokens dependencies while another projection can learn shorter-distance tokens dependencies. On each of these projected versions of queries, keys and values, the attention can be applied in parallel, yielding a $w \times d_v$ dimensional output value. These output values are concatenated yielding a $w \times d_v \times h$ dimensional output value, where h is the number of heads. The concatenated output value is once again projected, resulting in the final values, again yielding a $w \times d_v$ dimensional output value as depicted in Figure 2.13.

The encoder is composed of a stack of six identical layers as shown in 2.14. Each layer has two sub-layers. the first is a multi-head self-attention layer, and the second is a fully connected feed-forward network. A residual connection [He et al., 2016] is employed on each of the two sub-layers, followed by a layer normalization [Ba et al., 2016].

The decoder is composed of a stack of six identical layers as shown in 2.14. In addition to the two sub-layers in each encoder layer, the encoder introduces a third sub-layer. The third

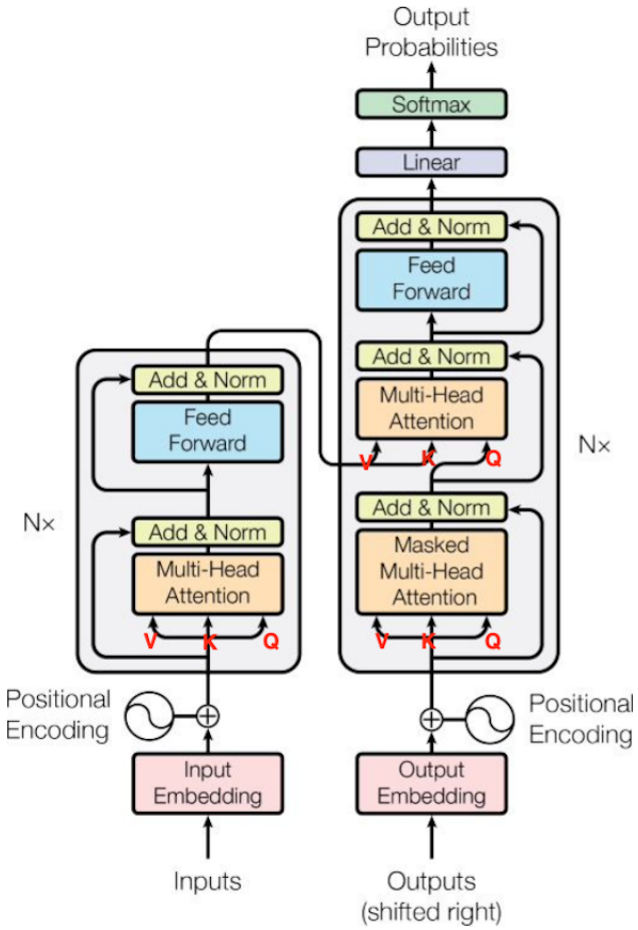

Figure 2.14: The Transformer model architecture [Vaswani et al., 2017].

sub-layer performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections are applied on each sub-layer, followed by a layer normalization. The masked self-attention sub-layer in the decoder stack is modified to prevent positions from attending to the subsequent positions.

Introduction to BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is designed to pre-train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers [Devlin et al., 2019]. There are two steps in the BERT framework, pre-training and fine-tuning, as shown in Figure 2.15. During the process of pre-training, the model is being trained in an unsupervised way with different pre-

training tasks. During fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are tuned with the labelled data for the targeted tasks.

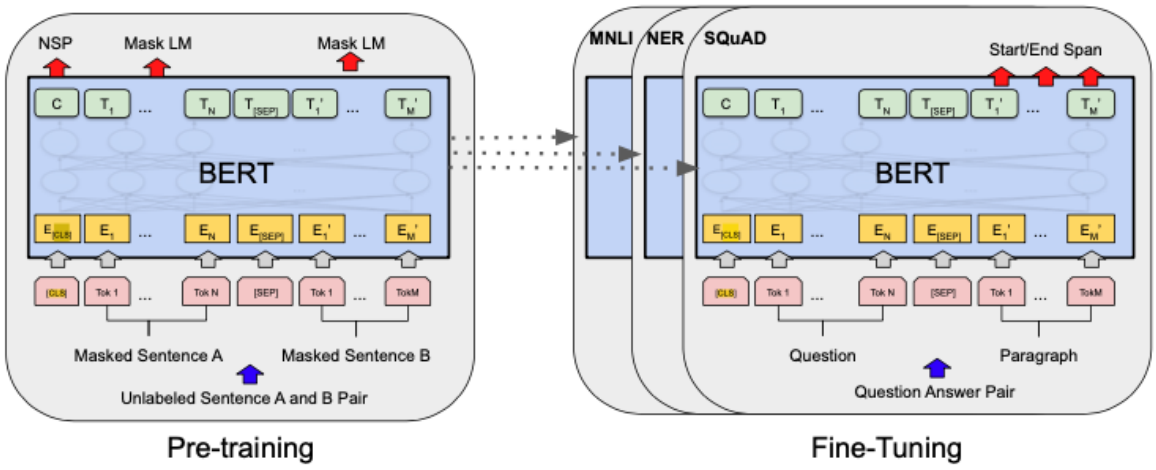

Figure 2.15: Illustration of the BERT training framework which includes two steps: pre-training and fine-tuning [Devlin et al., 2019].

BERT's model architecture is a multi-layer bidirectional transformer encoder based on the original implementation in Vaswani et al. [2017]. Using L to denote the number of layers (i.e., transformer block), H to denote the size of the layers, A to denote the number of self-attention heads. For the BERT base model we used in our work, $L = 12$, $H = 768$, $A = 12$ and the number of total parameters is 110M.

BERT is pre-trained on two different tasks. The first pre-trained task is a Masked Language Model (MLM). It is different from the standard conditional language models, which can only be trained from left-to-right or right-to-left. In order to train a deep bidirectional representation, the model simply mask some percentage of the input token at random and then predicts the masked tokens. The final hidden presentation of the masked token is fed into an output softmax over the vocab, which is the same as the standard language model.

The second pre-trained task is Next Sentence Prediction (NSP). This pre-training task is needed since the idea of BERT is to use the pre-trained language model to be fine-tuned on all kinds of different downstream tasks. Furthermore, several very critical downstream

tasks like Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship of two sentences. This relationship information is not directly captured by the language model. The pre-trained task is given two sentences A and B, we need to determine if B is the following sentence of A. In the pre-training dataset, 50% of the time, B is actually the next sentence of A. A special token `[CLS]` is introduced. The first token of every sequence is always a special classification token `[CLS]`. And the final hidden state corresponding to the token is used as the aggregate sequence representation for the classification task. The Next Sentence Prediction task fed the `[CLS]` hidden state to a classification layer. In this way, the `[CLS]` token is pre-traiend in a way that hidden representation of `[CLS]` token can capture the semantics of the entire sequence. As in our work, we used the `[CLS]` token representation of the topic sequence as our topic representation.

Transformer-based Models

We will use one of the transformer-based event extraction work, PLMEE (Pre-trained Language Models for Event Extraction) [Yang et al., 2019], as a representative example. PLMEE uses BERT [Devlin et al., 2019] as the features encoder. Starting from BERT, in the NLP domain, the pre-trained model on large scale dataset then fine-tuned on smaller downstream task dataset with extra layer is widely adopted. While the pre-training process is capable of capturing the meaning of words in consideration of their context, the fine-tuning process with extra layers can further adapt the network to the specific downstream tasks requirement.

The highlights of PLMEE on event extraction are twofold:

- PLMEE used pre-trained language model BERT as the text encoder that captures the contextual information of the tokens.
- PLMEE fine-tuned the pre-trained BERT on two downstream tasks: trigger classification and argument role classification, with special consideration of the role overlapping problem.

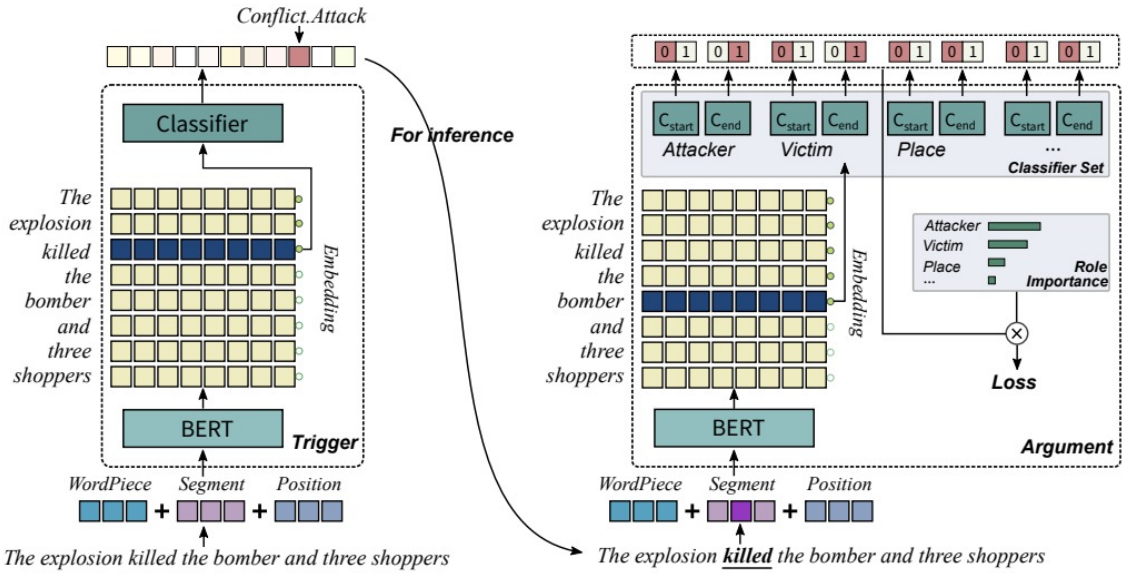

Figure 2.16: Illustration of the PLMEE architecture, including trigger classification and argument role extraction. The trigger classification and argument role extraction tasks are trained separately with the trigger tokens as the input to the argument classification [Yang et al., 2019].

PLMEE models the trigger classification and argument role classification separately. For trigger classification, the input texts are embedded by BERT [Devlin et al., 2019]. The output embeddings are constructed from WordPiece token embeddings, segment emebddings and position embeddings as in Figure 2.17. Then a classifier layer is applied onto the hidden representation of texts.

For the argument role classification, PLMEE also uses the BERT as the encoder with the special segment embeddings on the trigger tokens to represent the dependencies between trigger and argument. Multiple sets of binary classifiers are applied on top of BERT for each argument role to tackle the roles overlapping problem for argument role classification. The roles overlapping problem means that the same token can serve as different argument roles for the same event. For example, in sentence:

The explosion killed the bomber and three shoppers.

The word killed triggers an attack event, while the phrase the bomber plays both the attacker and the victim argument roles for the attack event. As illustrated in Figure 2.16, a token t is

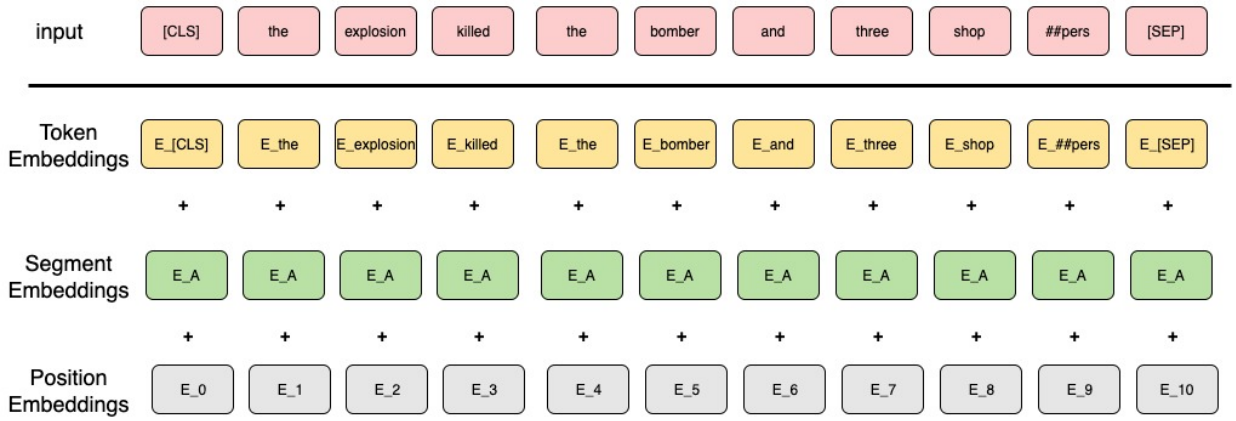

Figure 2.17: BERT input representation. The input embeddings are the sum of the token embeddings, segment embeddings, and the position embeddings for trigger classification.

predicted as the start of an argument that plays role r with probability $p_s^r(t) = \text{softmax}(W_s^r \cdot B(t))$, while as the end of an argument that plays role r with probability $p_e^r(t) = \text{softmax}(W_e^r \cdot B(t))$. Here, the subscript s represents "start," while e represents "end." W_s^r is the weight of the binary classifier detects starts of arguments playing role r, while W_e^r is the weight of another binary classifier that detects ends of arguments playing role r. B is the BERT embedding. For each role r, we obtain two sequences P_s^r and P_e^r. And based on the two sequences, we can determine if the detected tokens belong to the role. Furthermore, PLMEE uses weighted loss in the argument role classification by following the intuition that some argument roles are more important in certain event types, and should be penalized more.

2.8 Event Extraction Evaluation and Metrics

The four sub-tasks defined in Section 2.3 include trigger identification, trigger classification, argument identification, and argument role classification. Three metrics, precision, recall, and $F1$, are used to measure the performance.

Precision can be seen as a measure of quality, and recall as a measure of quantity. Higher

precision means that the algorithm returns higher number of correct results than incorrect ones. Higher recall means an algorithm returns more correct results (it does not care if more incorrect results are also returned). $F1$ is the harmonic mean of precision and recall, which can be used as a measure that considers both precision and recall. Precision, recall and $F1$ can be defined as:

$$\text{precision} = \frac{TP}{(TP + FP)}, \tag{2.6}$$

$$\text{recall} = \frac{TP}{(TP + FN)}, \text{ and} \tag{2.7}$$

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{2.8}$$

Total Population = P + N	Positive Prediction	Negative Prediction
Actual Positive (P)	True Positive (TP)	False Negative (FN)
Actual Negative (N)	False Positive (FP)	True Negative (TN)

Table 2.10: Definitions of True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN)

Using the notation of indicator function I(boolean), where $I(True) = 1$ and $I(False) = 0$, we can tailor the definitions of the precision, recall and $F1$ metrics to the event extraction tasks defined in Section 2.3: trigger identification, trigger classification, argument identification and argument classification tasks.

- **Trigger Identification**: a trigger is correctly identified, if its span offsets exactly match the gold/labeled trigger. The corresponding metrics include:

$$P_{TI} = \frac{\sum I(TD_L = T_L \wedge TD_R = T_R)}{N_{TD}}, \tag{2.9}$$

$$R_{TI} = \frac{\sum I(TD_L = T_L \wedge TD_R = T_R)}{N_T}, \text{ and} \tag{2.10}$$

$$F1_{TI} = \frac{2 \cdot P_{TI} \cdot R_{TI}}{P_{TI} + R_{TI}}. \tag{2.11}$$

where TD is the detected trigger, and T is the gold trigger. TD_L, TD_R are the left

and right offset of the detected trigger, and T_L, T_R are the left and right offset of the gold trigger.

- **Trigger Classification**: a trigger is correctly classified, if its span offsets and detected event type both exactly match the gold trigger and its event type. The corresponding metrics include:

$$P_{TC} = \frac{\sum I(TD_L = T_L \wedge TD_R = T_R \wedge TD_t = T_t)}{N_{TD}}, \tag{2.12}$$

$$R_{TC} = \frac{\sum I(TD_L = T_L \wedge TD_R = T_R \wedge TD_t = T_t)}{N_T}, \text{ and} \tag{2.13}$$

$$F1_{TC} = \frac{2 \cdot P_{TC} \cdot R_{TC}}{P_{TC} + R_{TC}}. \tag{2.14}$$

  where TD_t is the detected event type of the trigger, and T_t is the event type of the gold trigger.

- **Argument Identification**: an argument is correctly identified if its offset span and corresponding event type exactly match the gold argument. The corresponding metrics include:

$$P_{AI} = \frac{\sum I(AD_L = A_L \wedge AD_R = A_R \wedge TD_t = T_t)}{N_{AD}}, \tag{2.15}$$

$$R_{AI} = \frac{\sum I(AD_L = A_L \wedge AD_R = A_R \wedge TD_t = T_t)}{N_A}, \text{ and} \tag{2.16}$$

$$F1_{AI} = \frac{2 \cdot P_{AI} \cdot R_{AI}}{P_{AI} + R_{AI}}. \tag{2.17}$$

  where AD is the detected argument, and A is the gold argument. AD_L is the left offset of the detected argument and AD_R is the right offset of the detected argument. A_L is the left offset of the gold argument and A_R is the right offset of the gold argument.

- **Argument Classification**: an argument is correctly classified if its offset span, corresponding event type, and argument role all exactly matches the gold argument. The

corresponding metrics include:

$$P_{AC} = \frac{\sum I(AD_L = A_L \wedge AD_R = A_R \wedge TD_t = T_t \wedge AD_r = A_r)}{N_{AD}}, \tag{2.18}$$

$$R_{AC} = \frac{\sum I(AD_L = A_L \wedge AD_R = A_R \wedge TD_t = T_t \wedge AD_r = A_r)}{N_A}, \text{ and} \tag{2.19}$$

$$F1_{AC} = \frac{2 \cdot P_{AC} \cdot R_{AC}}{P_{AC} + R_{AC}}. \tag{2.20}$$

where AD_r is the detected argument role and A_r is the gold argument role.

2.9 Research Trends

Event extraction is a critical and challenging task in information extraction, which focuses on extracting the structured representation of events from the text that describes the events. Event extraction can be categorized into two tasks: event detection and argument extraction. The essence of the event extraction task is to identify the event-related words, and then recognize their event type or argument role of an event type. The traditional machine learning models need to extract lexical, syntactic and semantic features by using NLP tools and carefully design the features used in event extraction. The deep learning models introduced for event extraction can avoid the tedious work of choosing the specific features. The deep learning models use the word embeddings with rich language features as input, which further reduces the errors propagated by the NLP tools when designing features manually. However, we are still facing a lot of challenges from both the dataset corpora level and the extraction model level to tackle event extraction from different domains. We summarize the challenges that the event extraction task faces in the two categories below, from the data and model perspectives.

2.9.1 Challenges from Event Extraction Corpora

Event Extraction Dataset Construction

The event extraction task is complicated and has its characteristics. In the NLP domain, the pre-trained language model is used widely [Devlin et al., 2019; Dai and Le, 2015; Radford et al., 2018] to capture the contextual information embedded in the text and learn general language features. The existing pre-trained language models lack specific designs for event extraction tasks. In order to efficiently use pre-trained language model, new specific pre-training tasks need to be defined, along with some dataset suitable for language pre-training for event extraction tasks. Deep learning methods require a lot of training data to be effectively trained. The current existing event extraction dataset has a limitation on both the data size and diversity of covered domains. This hinders the high performance deep learning model of event extraction being developed. Furthermore, manual annotations of event data is quite expensive and the task could be hard, thus introducing annotation errors if we blindly work with crowd-sourcing annotators who are lack of domain knowledge. Thus, it becomes an urgent task to construct a large-scale event extraction dataset or design automatic/semi-automatic methods to facilitate the creation of large-scale event extraction dataset that can cover a fair amount of domains and different languages.

Event Extraction Schema Construction

Closed-domain event extraction tasks require a predefined event trigger to argument role template/schema. For different domains, the event types could be very different along with their argument role schema. Without the schema, it will be hard to evaluate the event extraction task performance. Furthermore, without the event schema, the development of the event extraction labelling dataset is hindered. Therefore, designing a general event extraction schema based on event characteristics, or automatically constructing event schema within different domains, become a future trend of developing advances in technologies for

event extraction.

2.9.2 Challenges from Event Extraction Models

Event Extraction Across Domains

Different domains have different event types, and also the training data is not evenly distributed across all domains. How to effectively use the domain knowledge explicitly/implicitly presented in the domain texts or documents becomes an important task. This will leverage the external knowledge from the domain, which can further boost the performance of cross-domain event extraction. At the same time, to alleviate the data scarcity issues existing in some domains, information from the domain itself or knowledge learned from similar domains with a large amount of training data can be leveraged.

Dependency Learning

Using BERT [Devlin et al., 2019] to encode the textual information gain an ever-increasing popularity in event extraction tasks. However, the pre-training task of BERT includes Masked Language Model (MLM) and Next Sentence Prediction (NSP), neither of which is designed for event extraction. In the MLM task, 15% of the tokens from each sequence are randomly masked (replaced with the token [**MASK**]). The model is trained to predict these tokens using all the other tokens of the sequence. The NSP task is a binary classification task to determine if the second sentence actually succeeds the first sentence in the corpus. For the NSP task, 50% of the time, the next sentence is indeed used as the next sentence in the training data, and 50% of the time, a random sentence is taken from the corpus for training. We can see that the two pre-training tasks in BERT is not specifically designed for event extraction. For example, the argument extraction task needs to learn the relationship between event triggers and other argument roles. This requires the pre-trained embedding to learn the syntactic dependencies of the text. Therefore designing the pre-trained task for event extraction to specifically consider the dependency learning is a direction worth

exploring.

Multi-event Extraction

Based on the different granularity of event extraction, the event extraction task can be divided into sentence-level event extraction and document-level event extraction. There has been a lot of research conducted on sentence-level event extractions. However, the document-level event extraction techniques are mainly in the exploring stage. Document-level event extraction is closer to our practical daily usage of the event extraction system. There are several characteristics that make document-level event extraction different from sentence-level extraction. In a document-level event extraction task, the argument role of an event trigger can spread across different sentences. Further, in a document-level event extraction task, the event type triggers might have correlations across different sentences within the same document. How to efficiently leverage the cross-sentence multi-event correlations and their argument correlations becomes a critical path to the success of document-level multi-event extraction.

Chapter 3

New Techniques for Scaling up Political Events Coding across Languages

3.1 Introduction

Data produced from news stories is one of the most important new sources of information for quantitative political science research. Drawn from English language news reporting worldwide, machine coded data of geo-referenced political and social activity has opened up new possibilities for the study of numerous political phenomena ranging from social movements to violent conflict and unrest and government responses. Data is automatically coded from the text by comparing phrases in the text to hand-coded dictionaries to identify actors and events. However, one limitation of event data is its restriction to English language sources only. We aim to build a new machine-coded event data set of Arabic news corpora and detect events such as protest or attack from the news corpus using the event coder UniversalPetrarch.[1] While UniversalPetrarch's code only requires minor changes to accom-

[1] `https://github.com/openeventdata/UniversalPetrarch`

modate new languages, the dictionaries used to map phrases to codes need to be completely re-written for each new language. Each dictionary encodes several specific cases used to translate sentences to events. Several issues make automated event coding in multiple languages a difficult and unsolved task, including the unwieldy size of text data that can reach terabytes, extracting the relevant actors to the events being studied, and guaranteeing both the accuracy and abundance of the coded results.

3.2 Related Work

Philip Schrodt and colleagues at the University of Kansas created the original English language event coding dictionaries. To develop the dictionaries, TABARI,[2] Schrodt's coding program, displayed sentences to coders if the system recognized an event in a sentence, but the actor was not in the dictionary. Coders were then responsible for adding these new phrases to the dictionaries. Our process improves TABARI's approach in several ways. First, it is a web-based system, allowing many coders to work together without interfering with each others' work. Second, we provide a useful structure to the task by suggesting the CAMEO [Schrodt, 2012] ontology in a dropdown list instead of asking coders to either memorize or refer to the codebook for the list of possible codes. We used other technologies to facilitate our process, for example, using word2vec [Tomas et al., 2013] to suggest synonyms of relevant actors to be added to the dictionary. We also developed a validation system based on peer review allowing coders to flag the dictionary entries about which they are not confident. All of these tools improve our speed and accuracy in generating dictionaries.

Javier Osorio and colleagues worked on dictionary development for Spanish [Osorio and Reyes, 2017], but because their text was drawn from politically relevant sources, they did not need to develop mechanics to filter out less useful text from a large text corpus as we needed to do. Because we used a large corpus of data, a challenge for our coders was the number of stories they received that had no relevance to coding political events. In order

[2] `https://parusanalytics.com/eventdata/software.dir/tabari.html`

to address this problem, we developed a way to filter out stories of no apparent relevance, delivering stories to coders most likely to contain political events. Another aspect of coding we considered in Arabic dictionary development is an actor's role during different periods since each actor might serve different roles at different times. Different actor's role during different time periods information is essential when detecting new political events, and we built a system to facilitate recording that piece of information. We developed an additional innovation to automatically find Arabic transliterations for existing actors in the English dictionary using Wikipedia. Our "wiki-bio" coding approach pre-populates all the possible roles an actor might occupy in different time ranges with a Wikipedia link attached and is much more convenient for coders to use.

Martin Atkinson worked on extracting security-related events from a multilingual corpus but did not build a system for a specific language like our system does for Arabic [Atkinson et al., 2017]. The way the authors cluster events is SVM-based [Ben-Hur et al., 2001], while we use a dictionary-based system. UniversalPetrarch uses dictionary entries to a specific CAMEO code. This dictionary-based system should be able to capture more specific events of relevance than a machine-learning system and achieve higher accuracy, but it requires much work to build a language-specific dictionary. Therefore, rapid development of dictionaries is needed to make dictionary-based event coding feasible for new languages and domains.

3.3 Our Contribution

We summarize our contributions for rapid development of dictionaries across different languages as the following:

- We designed four different approaches to speed up the dictionary development. The four approaches include: CoreNLP based approach, NER based approach, wiki-bio based approach and directed translation based approach.

- For the approaches we developed, we designed and implemented four different interfaces

to produce labels more accurately and efficiently.

- Using the approaches and interfaces we developed, we were able to complete Arabic actor and verb dictionaries with coverage equivalent to English in less than two years of work, compared to two decades for English dictionary development.

3.4 Coding Approaches and Interfaces

Arabic-language actor dictionary development is crucial for the event detection step performed by UniversalPetrarch. We used different techniques ranging from manual to fully automatic to help our team of human annotators create the Arabic-language actor dictionaries. This section outlines each approach, discusses their advantages and limitations, and describe the interface we built for each technique. Finally, we make recommendations to other researchers creating non-English language actor dictionaries or researchers developing dictionaries using new ontologies in new domains.

3.4.1 Main Coding Interface With Keywords Search, LDA Topic Filtering, and Synonyms Facilitator

Interface Walk Through

We are going to walk through this interface, shown in Figure 3.1, from left to right. On the left hand side, the Arabic sentence along with the parsed nouns and verbs are presented. Each of the nouns and verbs is a link that coders can click on in the interface. Clicking on the link will populate the noun or the verb to the right hand side which is the coding area. On the bottom left, the coders can either choose to select a random sentence to code next or provide some keywords for the sentence searching process. The motivation behind is: coders can always search the sentence that contains bomb as an example. The sentences contain bomb have a bigger chance to have more correlated actors and verbs presented. In

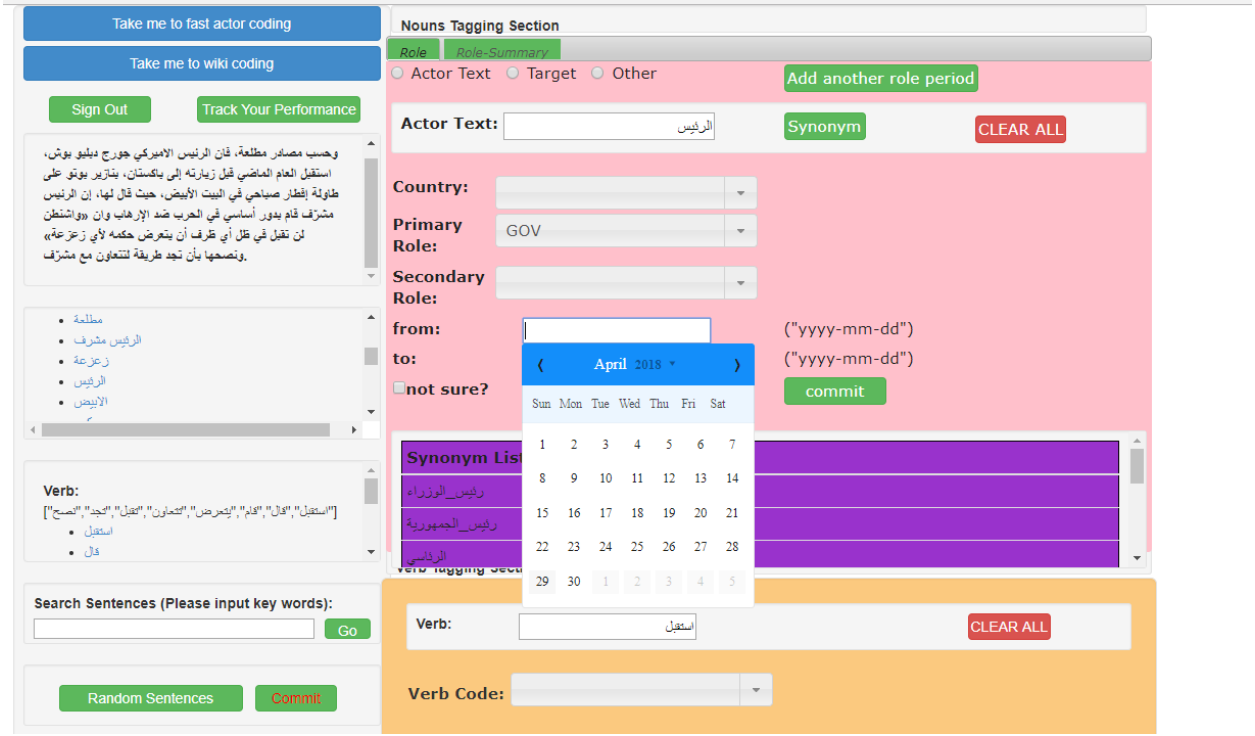

Figure 3.1: The Regular coding interface. Nouns and verbs in the sentence are presented on the left hand side. A search keyword feature is presented in the bottom left. The actor and verb coding area is on the right hand size. Given an actor, the interface supports the corresponding country, primary role, secondary role, from time period and to time period coding. The interface also supports adding a different role from different time period. This feature can be triggered by clicking "add another role period" button on the top right. The interface supports Word2Vec based synonyms for an actor word. The verb coding area is on the bottom right. If the actor to be coded is *Angela Merkel*, the country should be coded as DEU and primary role should be coded as GOV, while both DEU and GOV come from CAMEO taxonomy. If the verb to be coded is *demonstrated* or *rallied in the streets*, it should be coded as 145: Protest violently, riot, not specified below based on CAMEO taxonomy.

this way, keywords query feature can speed up the coding process. However, we might also face the biased selection issue for certain types of actors and coders. Once the coder click on the noun, the noun will populate in the right hand side "Actor Text" text box. Then, the coder can click on the synonyms button, which will trigger the generation of the Word2Vec based synonyms of the current actor. If the candidate synonyms are correct, the coder will add that directly into dictionary with the same codes. This can improve the yield of the dictionaries. People might serve different roles across different time period. As an example, Donald Trump's primary role is GOV ("Government: the executive, governing parties")

from 2017 to 2021. And before 2017, his primary role is BUS ("Business: businesspeople, companies, and enterprises, not including MNCs"). This interface supports adding another role period for an actor by clicking the "Add another role period" button on the top right. Then there are five required coding features. Including country, primary role, secondary role, from time period and end time period. For each of the country, primary role and secondary role features, a drop down list is provided. The elements of the drop down are from the CAMEO taxonomy. For the from and to time period features, when the text box is clicked, a calendar will populate to support quicker entry. There is also a toggle box provided in the middle of right hand side. The toggle box is used to flag the coded actors that the coder has lower confidence in. With the commit button, the coder can post their coded results for the actor to database. Verbs are coded in a similar way as actors from the bottom right area.

Interface Features

The first approach we developed was to sample the data from our Arabic Gigaword text corpus and use CoreNLP [Manning et al., 2014] to parse the data into a grammatical format with nouns and verbs and feed the parsed results to UniversalPetrarch. The next step was for our coders to code an entry for each extracted actor produced by UniversalPetrarch. To do so, we developed our main coding interface named Regular coding interface, as shown in Figure 3.1, where we used several new techniques for this step to improve accuracy and coding speed.

First, we exposed a search option for the coders with a "text" index on our data in MongoDB,[3] so that coders can work on related topics by querying similar keywords. It also allowed them to focus on coding similar topics, speeding up the process.

Second, when an actor entry is already coded, and the same word appears in the current sentence, its coded record from the database will pop up on the interface. Other coders can then confirm the previous coder's work, rather than entering a new entry. This set-up

[3] `https://en.wikipedia.org/wiki/MongoDB`

ensures that the most common entries are reviewed most frequently.

Third, the interface will also suggest alternative spellings of the names with its "synonym" based on word2vec. This greatly increases the system's yield, since many versions of the same name can be added once to the dictionary. Finally, the interface also includes an "unsure" button which flags the entry for peer review or review by the supervisor of the coding process.

Using these techniques and the interface built for it, the coders added 6,387 actor entries and 1,628 verb entries.

Interface Advantages and Disadvantages Discussion

The advantage of this interface and its techniques is it allows for broad coverage of the text by pulling out all the possible actors in the text and ensures that the dictionaries will have entries for them.

The disadvantage is clear. First, we are randomly sampling sentences out of a corpus of millions of sentences, with the result that the coders might code actors who are not, in fact, a high priority for adding. In order to address this problem, we applied a topic modeling strategy, Latent Dirichlet Allocation (LDA) [Blei et al., 2003], to our sentences so we could choose the politically relevant topics to code. We clustered the sentences into a different number of topics ($N \in \{5, 10, 20, 40\}$). We sampled sentences from each cluster to show different native Arabic-speaking coders and asked them to summarize which topic each clustering is mainly about. We then used their assessment to conclude that clustering the corpus into 20 groups gives us the best results. Finally, we filtered out unrelated topics like sports from the corpus, only showing the sentences from more political related topics.

Second, CoreNLP parsing only considers grammar structure, so a lot of nouns and verbs extracted might not be political event related. In order to address this issue, we proposed the NER based approach and interface in Section 3.4.3.

Third, each actor might serve different roles at different time periods. The information is important for new political event detection. Coders spend a lot of time on finding different

roles and putting entries for an actor. In order to address this issue, we proposed the Wikipedia based approach and interface in Section 3.4.4.

3.4.2 Wikipedia Facilitated Translating Existing English Actor Dictionary

The second approach we used in actor coding was to directly translate the existing English actor dictionary to Arabic using Wikipedia. Because each record is costly to add, being able to translate existing English dictionary entries into Arabic would significantly increase our efficiency. For each actor in the English dictionaries, we attempted to find its Wikipedia page by an exact name match. Once we located an English article, we checked to see if a corresponding Arabic article existed. If it did, we took the Arabic name for this actor and the current role information in the English language dictionary and added them to the Arabic dictionaries.

The major advantage of this approach is its efficiency and speed, as no human effort is needed. The limitation of this approach is that we can only add actors that are already in the existing English dictionaries. Additionally, entries may not include each role of the relevant actors in Arabic language news sources. With this approach, we were able to achieve 5,696 actor entries. We did not develop a special interface for this approach.

3.4.3 Frequency Ranked NER Based Coding Interface

Interface Walk Through

Figure 3.2 shows the NER based coding interface. When a coder clicks the "start" button, a frequency and NER filtered entity will populate. And the timing starts to track how long it takes the coder to label the entity. The coder then needs to decide if the entity is a political related entity. If not, the coder will click on the "skip" button on the top right to skip the annotations for the entity. If the entity is indeed a political related entity, the coder will

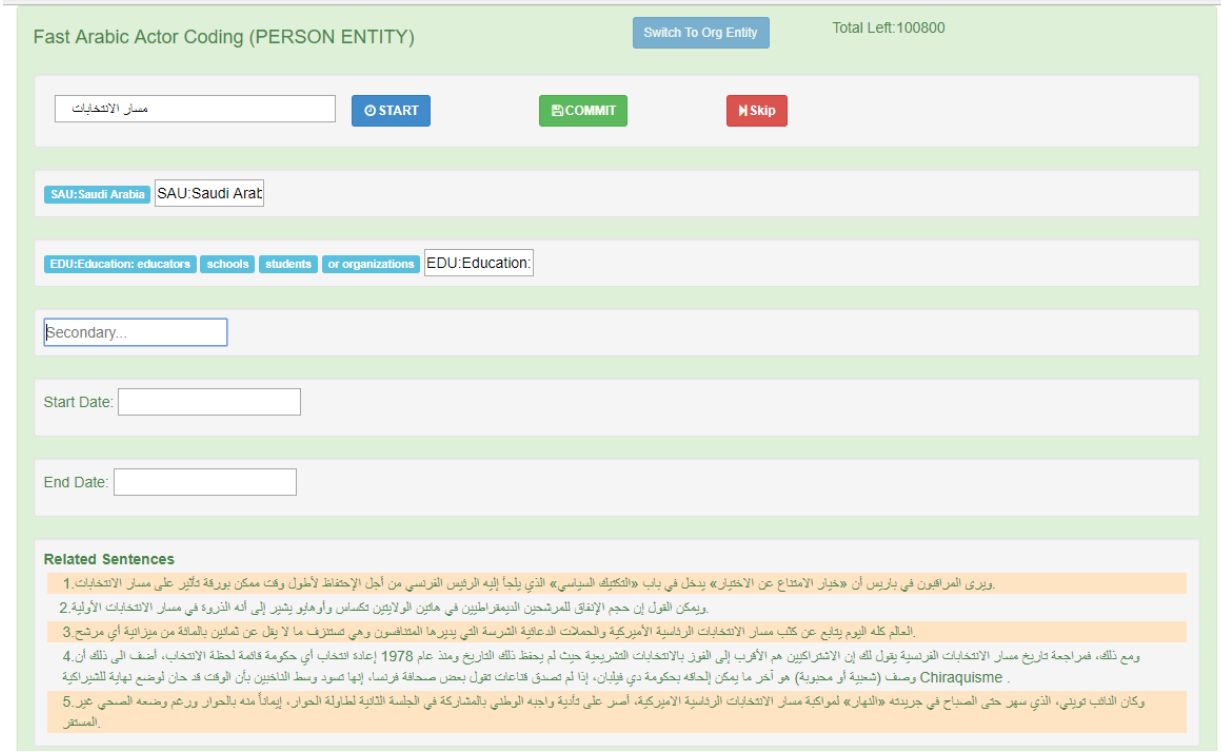

Figure 3.2: The Fast coding interface. Frequency and NER based PERSON or ORGANIZATION entity is presented to coders. In addition, five sentences which contain the entity are presented to the coders on the bottom to provide contextual information of the entity.

code the features including: country, primary role, secondary role, from time period and end time period, similarly as the Regular coding interface introduced in Section 3.4.1. Instead of finding the entity information by searching the web, at the bottom of the interface, five sentences which contain the entity are also presented. The sentences can provide extra contextual information to speed up the coding process. When the coder click on the "commit" button on the top right. The time tracking ends.

Interface Features

The third approach we used in actor coding was to automatically find high-frequency actors in the corpus and have coders create entries for them. We first use LDA to filter out unrelated topics, and then pass the remaining documents through CoreNLP to parse the sentences into noun and verb phrases. We then used a MapReduce [Dean and Ghemawat, 2008] strategy

to count the frequency of each actor in the corpus and rank the count in decreasing order. Next, we filtered out all nouns that are not named entities using a multilingual named entity recognition (NER) model from the spaCy tool.[4] We then supply the actors to the coders ranked by decreasing order of appearance. Along with the PERSON or ORGANIZATION entity showing on the interface, we also presented five sentences in the corpus where the noun appears as background content to our coders, so that the coders could have more content-related information when they made a coding decision. A screenshot of the interface is shown in Figure 3.2.

Interface Advantages and Disadvantages Discussion

The advantage of this method is that it recommends the most important actors (measured by frequency) in the corpus and directs the coders to work on them.

Unfortunately, several off-the-shelf programs exhibited poor performance when performing Arabic NER extraction. The NER model trained with insufficient data in spaCy, so its performance is inadequate in recognizing PERSON and ORGANIZATION names. It also cannot distinguish between politically relevant and irrelevant people. Out of the most frequent 7,180 people or organizations recognized by the spaCy multilingual model, only 204 were political actors that could be added to the dictionary. This low yield is attributable to the system returning text that is not a named entity, is politically irrelevant, or names that are too vague (i.e., common first or last names). The disadvantage of this approach is substantial: coders spend a significant amount of time skipping irrelevant actors in order to find one to add to the dictionary, which is frustrating and time-consuming. In order to enhance the performance of this approach, we need to build a customized Arabic-specific NER model.

[4] `https://spacy.io`

3.4.4 Wikipedia Based Coding Interface

Interface Walk Through

Figure 3.3 shows the Wiki-bio coding interface. When a coder clicks the "start" button, a

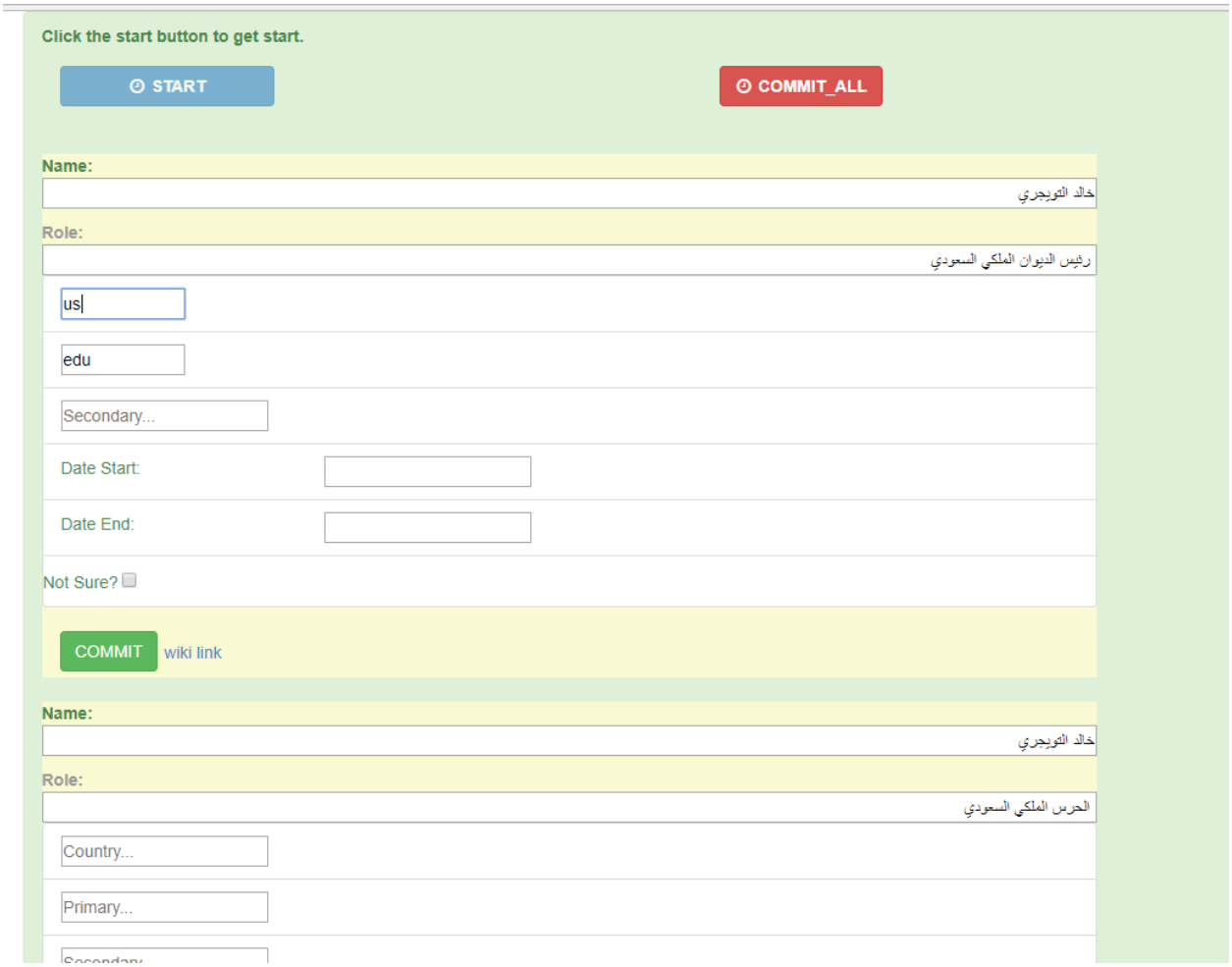

Figure 3.3: The Wiki-bio coding interface. The candidates to code with this interface are scraped from Wikipedia. A list of Wikipedia links such as government ministers and prominent politicians in the 22 Arab states are compiled. And the corresponding Wikipedia data is extracted. The actor roles across different time periods are pre-filled in the interface. And a Wikipedia link of the related role is also presented to the coders for validation purpose.

Wikipedia scraped actor will populate. Along with the actor name, the role and time period of the role are pre-filled. Based on the role information, the coder needs to find corresponding features, country, primary role and secondary role for the actor. And the timing starts to track how long it takes the coder to label the entity for all actors. When the coder finishes

all the role codings for the actor, they will need to click on the "commit all" button on the top right. The time tracking ends. The time taken and number of roles are recorded by the interface. In this way, we can calculate how long it takes to code on both actor and role levels.

Interface Features

The fourth approach we used was to leverage the information available on Wikipedia. Our target Wikipedia data was the data in the info box of politically relevant actors. To scrape the data, we compiled a list of links of Wikipedia categories, such as categories of government ministers and prominent politicians in the 22 Arab states, and wrote a specialized scraper to extract this Wikipedia data. The data contained the name of the actor, any roles he/she occupied, and any dates associated with these roles. Once we scraped the data, we developed our third web-based interface, "Wiki-Interface" (Figure 3.3), in which the actors' data was repopulated into the interface, and presented to the coders. The coders ensured the data's consistency by removing any extra title information from the actor's name and translating names and roles from English to Arabic for actors with no Arabic Wikipedia pages. Coders also discarded actor entries that had no role data associated with them. Each actor role was presented to the coders as a card that they could commit or discard. A link to the Wikipedia page that the current actor data was scraped from was also provided to help coders disambiguate any uncertainties they had.

Using this interface, we generated entries for 2,327 actors, totaling 4,286 role period ranges.

Interface Advantages and Disadvantages Discussion

The advantage of this approach is that coders are only working on the actors that are politically relevant since we only scraped entries from politically relevant Wikipedia category pages. Coding is also extremely fast, as the highly structured information is presented to

the coders with the date ranges pre-populated.

The disadvantage is that not all politically relevant actors have Wikipedia pages, nor do these pages always have biographical sidebars. Organizations also do not have biographical sidebars as people do, making this interface useful only for coding people.

3.4.5 Peer Review Interface for Unsure Records

Interface Walk Through

On the peer review interface, shown in Figure 3.4, the coder can acquire unsure records by searching coder name. On the top, there are two buttons "Show Flagged Nouns" and "Show Flagged Verbs". The coders can click either of them to choose the type of the records they want to work on. The results are paginated on the interface. For each unsure record, the coded results along with the coder name, added time, and updated time are shown. If the coders want to fix an unsure record, they need to click on the "Edit" button towards the end of each record row. When they finish the editing, the "EditBy" and "EditTime" columns will be updated along with the updated coding results. The coder can also click the "sentenceId" button at the beginning of each record row. This will trigger the population of the sentence which the record is coded from to the sentence text box in top left.

Interface Features

With these aforementioned interfaces, and to get more accurate records, we implemented an *unsure* strategy, allowing coders to flag a record as *unsure* if coders were not confident about what they coded. We then implemented a peer review interface (Figure 3.4) for the coders to check each others' unsure records and make corrections, and had a supervisor track each coder's performance. During the development of all our approaches, we kept our coders in the loop by taking their feedback as an input to our design process.

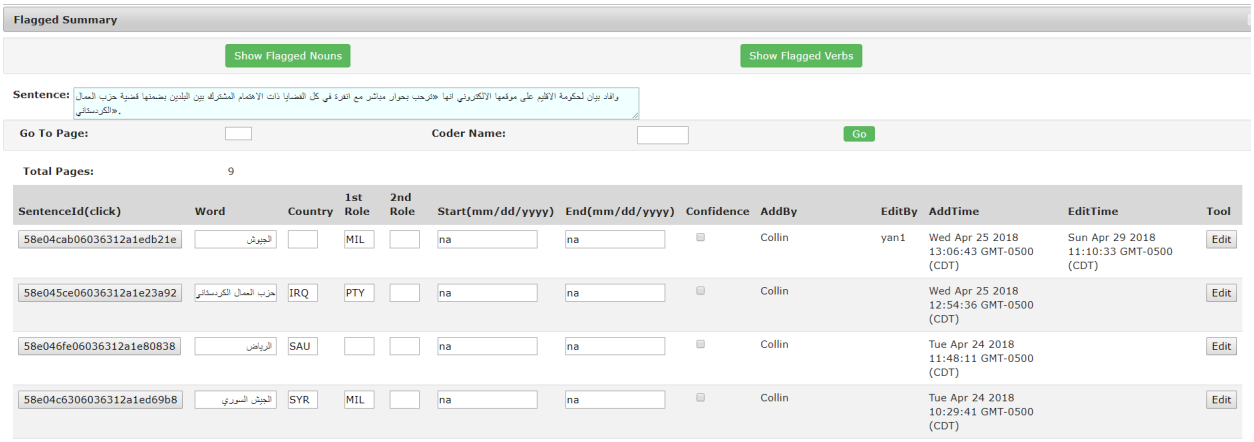

Figure 3.4: Peer review interface. The interface tracks the unsure records and allow multiple people to review and verify. The sentence id is also provided for the unsure records to provide context.

3.5 Coding Teams

In order to assist with our dictionary development, we hired 8-10 Arabic coders. The coders were primarily undergraduate students and native Arabic speakers with direct experience teaching the language. We split the coders into two teams: Team 1 and Team 2. Within each team, coders were paired into groups of two to perform the task and to verify it, with one performing and the second verifying. If both coders in the pair were unsure, the interfaces allowed the coders to flag the task. Other coders may then contribute to completing the task. Having such a team structure helped the coders get accustomed to the task faster and develop shared norms for approaching coding issues.

This team set-up, as well as our interfaces, has many advantages. First, the division of the tasks and the verification mechanism ensured better results and higher accuracy. Second, our web-based interfaces allowed for much greater work flexibility for our coders, especially those who were abroad or working remotely. The ease, clarity, and segmentation of tasks allowed for very cheap training costs and fast adaptation of tasks by the coders. Third, having various interfaces allowed much more feedback and tweaking on the developmental side, making it easier to accommodate coders' needs.

The main disadvantage of our set-up is decentralization. Having various interfaces for

related tasks can lead to unorganized and sparse results. To avoid that, we had a strict timeline for how we used those interfaces, which team used it, and when to transition to another interface, which seemed to help greatly in this regard.

Interface	**Teams**	**Description**
Regular Interface (Figure 3.1)	Team1	Extracted nouns and verbs from raw sentence with auto complete and synonym-aided feature.
NER based (Figure 3.2)	Team1	Frequency ordered and NER based PER and ORG coding.
Wiki-Bio (Figure 3.3)	Team1, Team 2	Provided pre-filled information from Wikipedia name cards.
NER Annotation with Prodigy (Figure 3.8)	Team1	Interactive NER model updating interface provided by Prodigy.

Table 3.1: Description of each interface used by coders from Team 1 and Team 2.

Aside from using the interfaces to code, the coders attend weekly meetings with the interface development team to provide feedback about the interfaces and discuss questions about the coding process. Holding these meetings is a vital aspect of eliciting feedback, understanding the language requirements, and assessing the coders' needs, which is a driving factor in developing the most effective interfaces.

3.6 Evaluation

We conduct the analysis to answer the following two research questions:

1. For the interface we developed, which one works better for a given time budget?

2. Will the coding speed improve if the coder has coded more actors?

To answer the first question, we track how much time each coder takes when working on the Wiki-Bio based interface and the NER based interface with the intention of understanding which interface works better for a given time budget. Our assumption is that the Wiki-Bio based interface should be more efficient. The efficiency is measured by how long it takes an coder to code an actor.

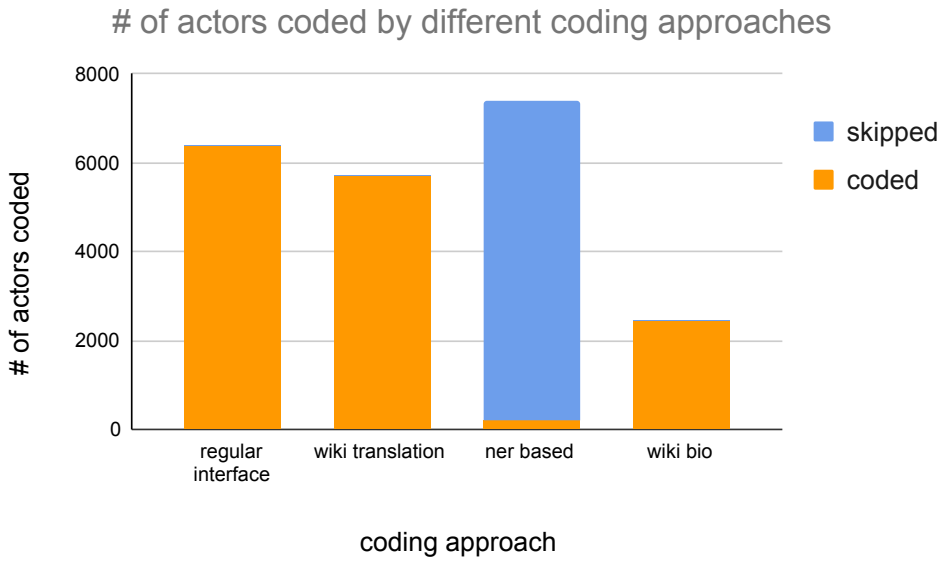

Figure 3.5: Total number of actors coded for each approach.

The coders' performance is presented in Table 3.2 and Figure 3.5. We did not track time per task in the original interface because we bundled many tasks into one: coders added between 0 and a half-dozen dictionary entries per sentence, with several roles for each actor, and identifying the source and target actors in the sentence. We therefore cannot unbundle the task timing to identify how long each discrete task took. Also note that there is no outlier time taken (extremely large time produced in the scenario like the coder might take lunch during one actor coding. The interface simply will timeout.) during the labelling process.

Approach	**Actors Coded**	**Actors Skipped**	**Total Time (seconds)**	**Seconds per Actor**	**Seconds per Role**
Regular Interface	6,387	-	-	-	-
NER based	204	7,180	11,343	55.6	55.6
Wiki-Bio	2,459	-	926,289	377	202
Wiki Translations (no interface)	5,696	-	-	-	-

Table 3.2: Performance of coders with different coding approaches.

First, we were surprised at how poor the performance was for the multilingual trained Arabic NER model. within 7,384 records extracted from the NER model, we were only able

to code 204 politically relevant ones. The poor performance of the NER model is caused by training on poor multilingual data.

Once we find a politically relevant entity with this interface, coding it only requires an average of one minute. One possible explanation for this relative high coding speed per actor (Data is shown in seconds per actor column in Table 3.2) is that we present five related sentences in which the actor appears so that the coders get a richer understanding of the actor and they can create the entry faster. This suggests that if we can get a better working version of Arabic-specific NER model, it will significantly enhance our yield and overall performance.

We found the performance of the Wiki-bio approach to be unexpectedly slow. As our assumption, we expected Wiki-bio interface to be faster than the NER based interface since we had already pre-populated the time range for each entity and provided the URL to link the actor back to the Wikipedia page. For each actor entity, it still took six minutes to code on average, and each role took roughly three minutes compared to 1 min for NER based interface. A possible reason for the slow speed is that we do not provide the same background content in Wiki-bio based interface as the NER based interface does. Thus, the coder still needs to go to the related Wikipedia pages to verify. Validating at Wikipedia pages takes time.

Note that we only have 204 NER based actors coded. The sample size is small, and our time estimates may therefore be imprecise.

To answer the second research question: Will the coding speed improve if the coder has coded more actors? Our assumption is the coding speed will be faster for an coder who has coded more actors. The assumption is based on the more experienced coder will understand CAMEO taxonomy better. The better memorization and understanding of the CAMEO taxonomy will lead to higher coding speed.

We gather the statistics of number of actors each coder worked on along with average time taken of each actor for each coder in Figure 3.6.

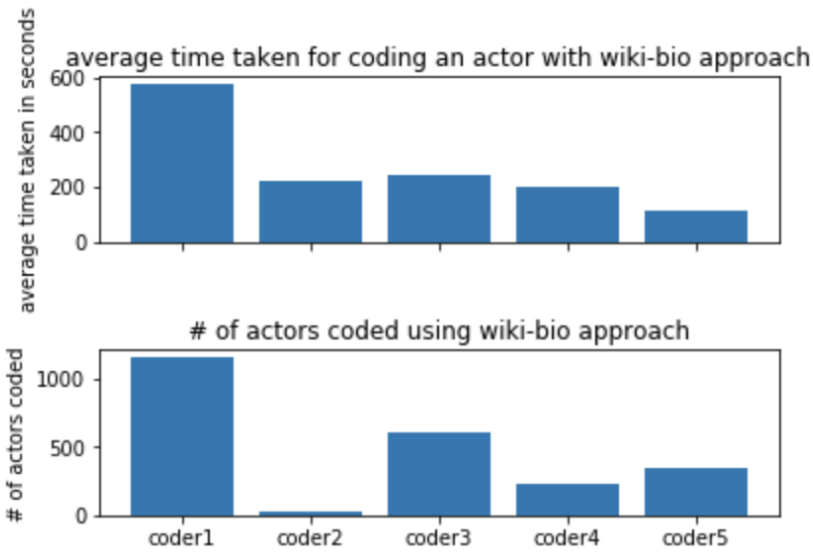

Figure 3.6: Performance of wiki-based approach of five coders.

We find a result related to coder efficiency that is different from our assumption. The more actors a coder has been coding over time, and presumably the more experienced a coder becomes, it takes the coder more time on average to code an actor, not less.

This may be because the more actors a coder encounters, the higher the probability that the coder will encounter ones that are more difficult to decide how to code. It could also be that as coders become more experienced, they are more likely to consider various possible roles or more complicated coding issues. From Figure 3.7 we can see that the variance of time per task increases along with the number of actors coded, which appears to support our explanation.

3.7 Summary

Using the above approaches for developing dictionaries, we were able to complete Arabic actor and verb dictionaries with coverage equivalent to the English language dictionaries in less than two years of work, compared to the two decades that the English language dictionaries took to produce.

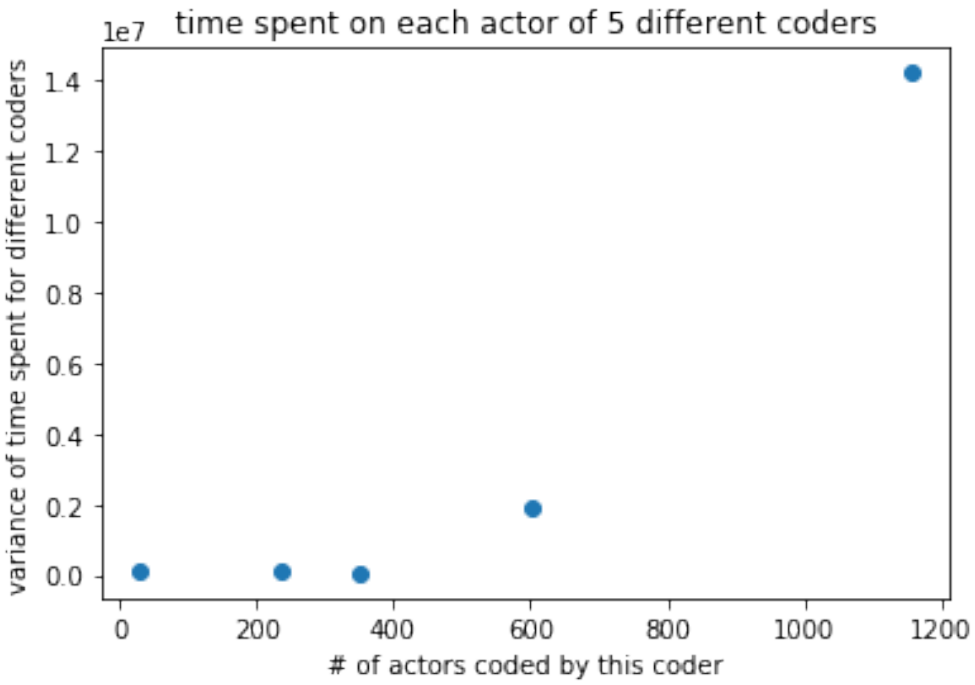

Figure 3.7: Variance of time spent on each actor of different coders versus the number of actors coded by each coder.

We have used UniversalPetrach to generate events from our corpus of millions of Arabic sources using the dictionary we developed, and we expect to make comparisons between it and the English corpus after final debugging and quality checking.

It is difficult to determine how many actor dictionary entries are sufficient to generate accurate event data. Within our budget on hiring coders, we aim to generate as many politically relevant actor and verb dictionary entries as possible, so we need the fastest possible coding framework to achieve this.

3.8 Discussion and Future Work

We could potentially do better than one minute on NER coding and three minutes on Wikibio coding by applying crowd sourcing strategies, e.g., we could make recommendations to our coders and simply ask them to verify the recommendations. In that way they would just need to choose yes or no instead of entering detailed information. Prodigy is a promis-

ing framework that can provide us with that functionality [Honnibal and Montani, 2017], Figure 3.8.

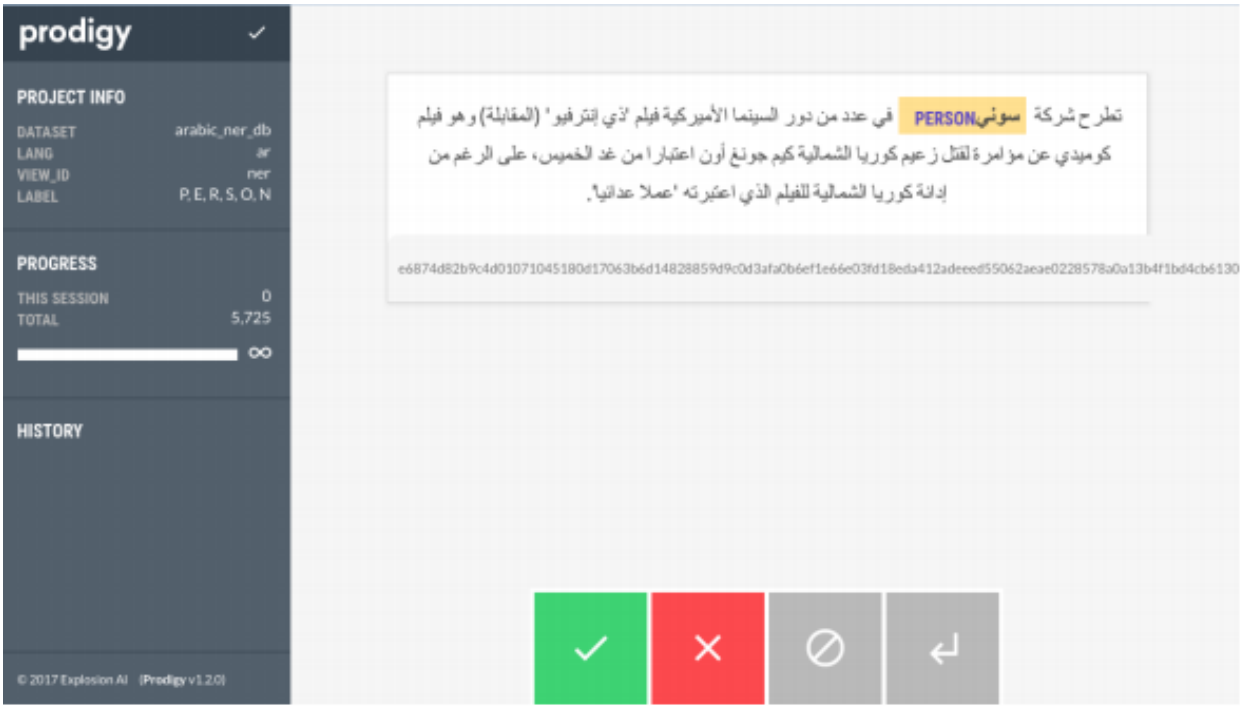

Figure 3.8: The Prodigy interface used for training a customized named entity recognition model for Arabic.

Chapter 4

Scaling up Event Detection by Utilizing Document Topic Information

4.1 Introduction

Event detection is an essential task of information retrieval in natural language processing, which has lots of applications in different domains. For example, event detection and monitoring have long been the focus of public affairs management for governments, as timely knowing the outbursts and evolution of popular social events helps the authorities respond promptly [Conlon et al., 2015; Atkinson et al., 2009]. Structured events can be directly used in constructing or expanding knowledge bases [Rospocher et al., 2016; Li et al., 2018]. In the business and financial domain, event detection can also help companies quickly discover market responses to their products and influential signals for risks analysis and suggestions [Nuij et al., 2013; Capet et al., 2008]. Despite its promising applications, event detection is still a rather challenging task. As events come with different structures and components. Further, natural languages are often with semantic ambiguities and discourse styles.

Event detection aims at finding the *event triggers*—the main word that most clearly expresses an event occurrence, typically a verb or a noun. Event detection techniques then

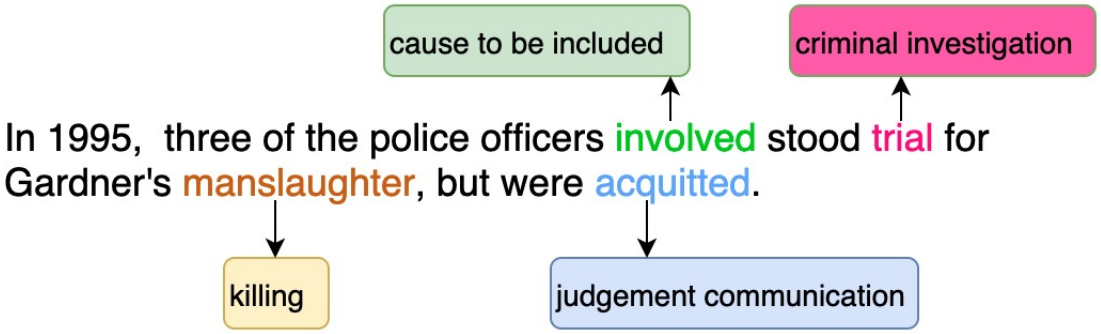

Figure 4.1: An example of event triggers and event detection. Specifically, this sentence contains multiple event-trigger words belonging to different event types.

use the triggers to classify the *event type* into a predefined set. For example, in the sentence in Figure 4.1,

> *In 1995, three of the police officers involved stood trial for Gardner's manslaughter, but were acquitted.*

involved triggers a cause to be included event, trial triggers a criminal investigation event, manslaughter triggers a killing event, and acquitted triggers a judgment communication event.

4.2 Related Work

Early approaches for event detection use pattern matching techniques [Riloff, 1993]. Later on, people used traditional machine learning algorithms other than neural networks to address the event detection problem. Techniques include using support vector machine (SVM) [Li et al., 2012]. More recently, deep learning has been successfully applied to different NLP tasks including event detection. The general process to build a neural network is to take word embeddings [Tomas et al., 2013] as input and output a classification result for each word. Convolutional Neural Networks (CNN) [Kim, 2014], Recurrent Neural Networks (RNN) [Nguyen et al., 2016; Ghaeini et al., 2016], and Graph Neural Networks (GNN) [Liu et al., 2018] have been all explored and applied to the event detection problem.

Ji and Grishman [2008] employs an approach to propagate consistent event arguments across sentences and documents. By combining global evidence from related documents and local decisions, a cross-document method is created to improve event detection. Li et al. [2013] proposes a joint framework that extracts triggers and arguments together to alleviate the problem of error propagation caused by event triggers and arguments are predicted in isolation. Chen et al. [2015] proposes a dynamic multi-pooling convolutional neural network according to event triggers and arguments to reserve more crucial information for event detection. Zhao et al. [2018] first learns event detection oriented document embedding through a hierarchical and supervised attention based RNN, then further uses the learned document embedding to identify event triggers. Yan et al. [2019] uses a dependency tree based on a graph convolutional network with aggregative attention to explicitly model and aggregate multi-order syntactic representations in sentences. Du and Cardie [2020] formulates the event extraction task as a question answering task that extracts the event arguments end-to-end. Li et al. [2020a] casts the event extraction task into a series of reading comprehension problems, by which it extracts triggers and arguments successively from a given sentence. Luan et al. [2019] introduces a general framework for several event extraction tasks that share span representations using a dynamically constructed span graph. The dynamic span graph refines the span representations by allowing the co-reference and relation type confidences to propagate through the graph. Unlike their work, we used the topic information to enhance the sentence representation and further utilized the topic classification task as a facilitator for event detection by having a multitask set-up.

4.3 Motivation and Our Contribution

None of the previous efforts [Kim, 2014; Nguyen et al., 2016; Ghaeini et al., 2016; Liu et al., 2018] consider event topic information for the event detection task. Examples of topics that the documents belong to include a terrorist attack, a horse race, and an earthquake, as

shown in Figure 4.1. Topic information is vital for event detection tasks, since, intuitively, documents belonging to different topics have different event type distributions. There are several existing event detection datasets, for example, ACE05 [Walker et al., 2006], ERE series [Ellis et al., 2014, 2015], and TERRIER [Liang et al., 2018]. In order to validate our assumption, we analyzed the MAVEN dataset [Wang et al., 2020]. We chose MAVEN because:

- MAVEN has an extensive range of event types compared to others. For example, Maven has 168 event types, while ACE05 contains 8 event types and 33 specific subtypes.
- MAVEN comes with human-annotated document topics, which is needed in the TAED model. However, the other datasets do not have the topic annotations.

We gather all the 168 event types in MAVEN. Then, for each topic, we normalized the event type occurrence count to a 168 dimension vector, of which all the 168 elements in the vector summarize to 1. We use this vector to represent the event distribution of the current topic. For each pair of the topics, we performed a two sample Kolmogorov–Smirnov test and report the P-Value as a heat map in Figure 4.2. When the P-Value is larger than 0.05, we cannot reject the null hypothesis that the two topics follow the same event type distribution. When the P-Value is less than or equal to 0.05, we can reject the null hypothesis.

Based on Figure 4.2 we can see that when talking about event type distribution, the topic terrorist attack is most similar to the military conflict and civil conflict topics with large corresponding P-Values of 0.61 and 0.52, respectively. The topic horse race is most similar to the international ice hockey competition and individual golf tournament topics with corresponding P-Values of 0.43 and 0.29, respectively. In Table 4.1, we show the top five event types for the three topics earthquake, horse race and terrorist attack, from which we can see that event type distributions are affected by different topics.

Semantically similar topics share similar event type distributions, while semantically different topics have heterogeneous distributions of event types. This inspires us to explore

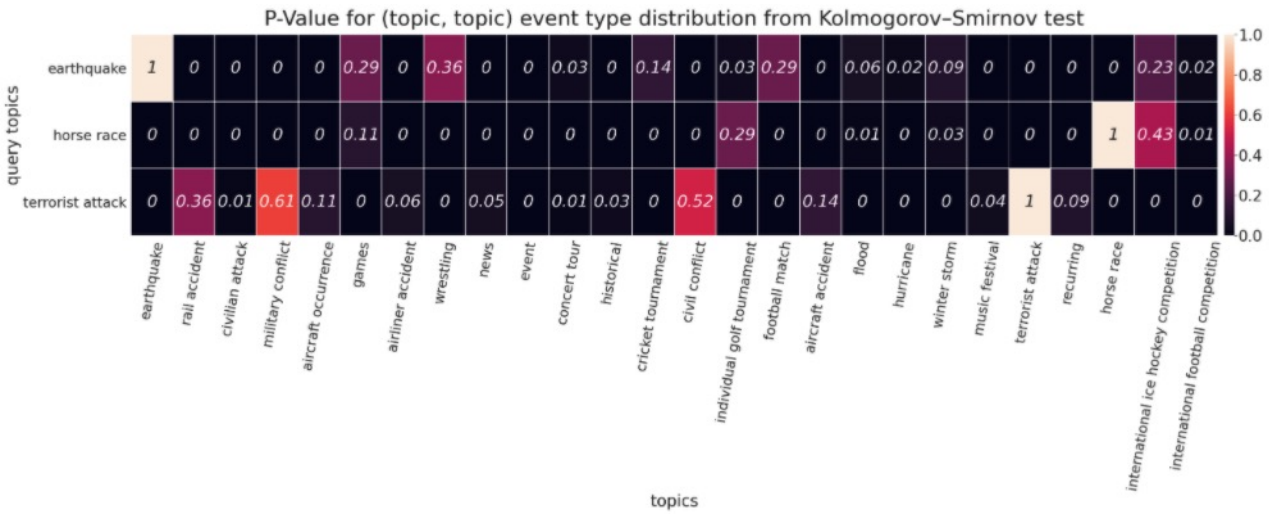

Figure 4.2: P-Value from Kolmogorov–Smirnov test on distribution of event types across topics. The P-value is calculated based on two topics' event type distributions. The null hypothesis is the two topics' event types follow the same distribution. (Partial version. Full version in Figure 4.3.)

earthquake	event types	catastrophe	causation	damaging	coming to be	destroying
	event types distribution	0.255	0.076	0.072	0.043	0.033
horse race	event types	competition	process start	process end	causation	hold
	event types distribution	0.201	0.104	0.058	0.047	0.036
terrorist attack	event types	attack	killing	terrorism	bodily harm	causation
	event types distribution	0.145	0.074	0.058	0.049	0.035
civilian attack	event types	killing	attack	statement	causation	bodily harm
	event types distribution	0.094	0.068	0.041	0.035	0.032

Table 4.1: An example of the top five event type distributions for each of the topics: earthquake, horse race, terrorist attack and civilian attack.

effective ways of using topic information in event detection tasks to improve performance. We summarize our contributions as the following:

- We perform a detailed analysis explaining why topic information helps on event detection task.
- We introduce topic name enhanced sentence representation for event detection and explore different ways to embed the topic name information including: using attention-based versus concatenation-based interaction, `[CLS]` versus token average based attribute embedding and using topic keywords to generate topic embedding versus using topic names.

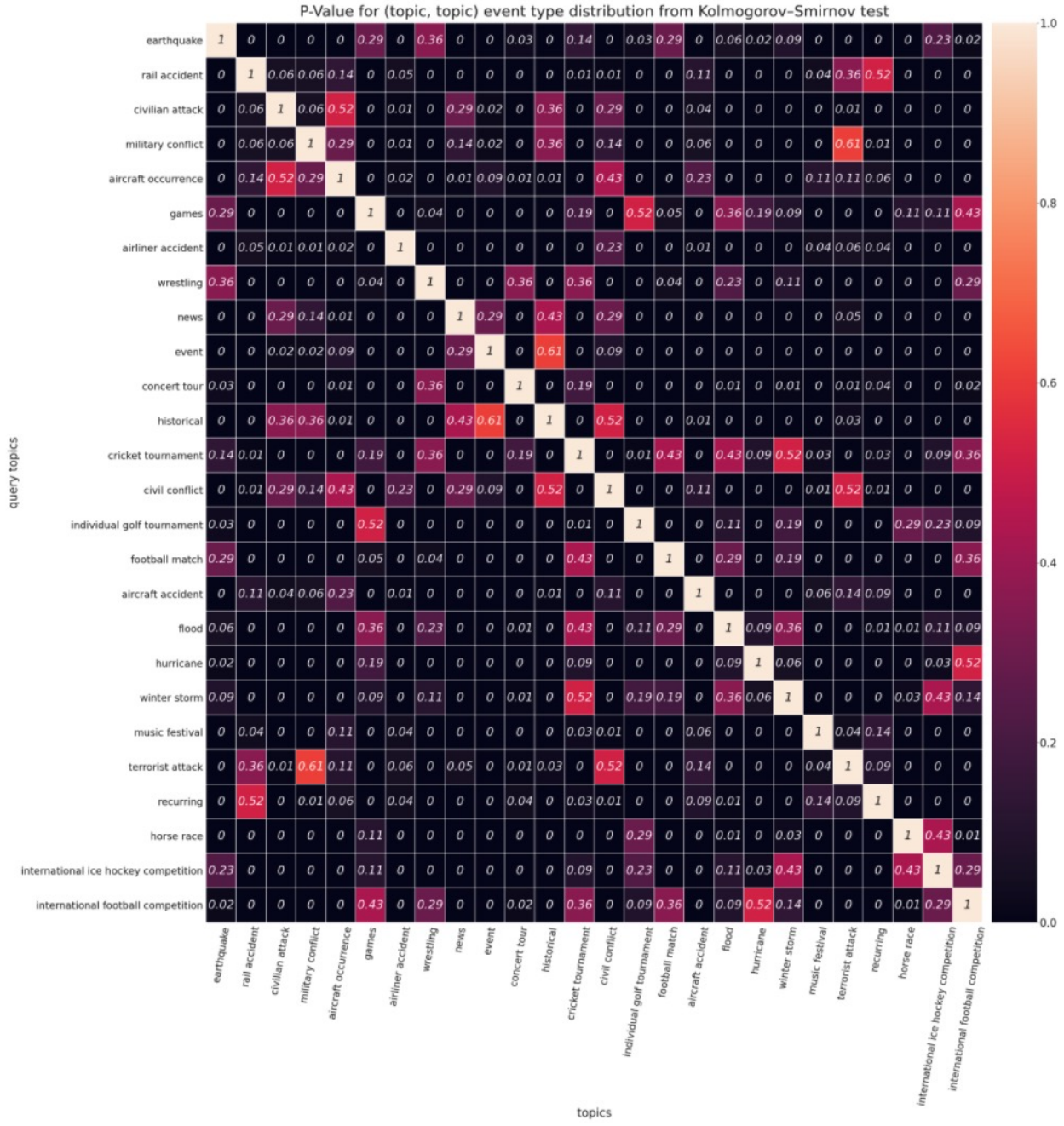

Figure 4.3: P-Value from Kolmogorov–Smirnov test on distribution of event types across topics. The smaller the P-Value in the cell is, the bigger the difference of event type distributions between the two topics. (Full version)

- We introduce topic classification and event detection as a multi-task learning set-up, which further improves the performance and conducted experiments with two event detection datasets with various event types. We achieved up to +1.8% improvement on the $F1$ score compared to the BERT based non-topic aware baseline.

- Furthermore, we show that the topic-aware model we propose can improve the few-shot event types scenario with +13.34% improvement on the $F1$ score and provide heuristic explanations in the case study.

4.4 Event Detection Definition

An event is a specific occurrence of something that happens at a particular time and place, which can frequently be described as a change of state. An *event structure* is defined as follows in ACE05 terminology:

- Event Mention: a phrase or sentence describing an event, including a trigger and several arguments.

- Event Trigger: the main word that most clearly expresses an event occurrence, typically a verb or a noun.

The event detection tasks are defined as follows:

- Trigger Identification: aims at identifying the most important word that characterizes an event.

- Trigger Classification: aims at classifying the event trigger into predefined, fine-grained categories.

Recent neural network methods typically formulate event detection task as a token-level multi-class classification task [Chen et al., 2015; Nguyen et al., 2016] or a sequence labelling task [Chen et al., 2018], and only report the trigger classification results [Wang et al., 2020;

Zeng et al., 2018]. An additional type N/A is introduced and classified at the same time to indicate that the candidate is not a trigger. We adopt the above settings and evaluate the performance with precision, recall and $F1$ on a micro level.

4.5 Methodology

TAED leverages document topics for event detection. The underlying intuition is that event type distributions are different across the topics. Our model uses a topic name embedding to enhance the sentence representation. Furthermore, we have modeled topic classification and event detection as a multi-task learning set-up, which means topic classification and event detection are jointly trained.

4.5.1 Architecture Overview

In this section, we present a holistic description of the topic-aware event detection model we proposed as presented in Figure 4.4.

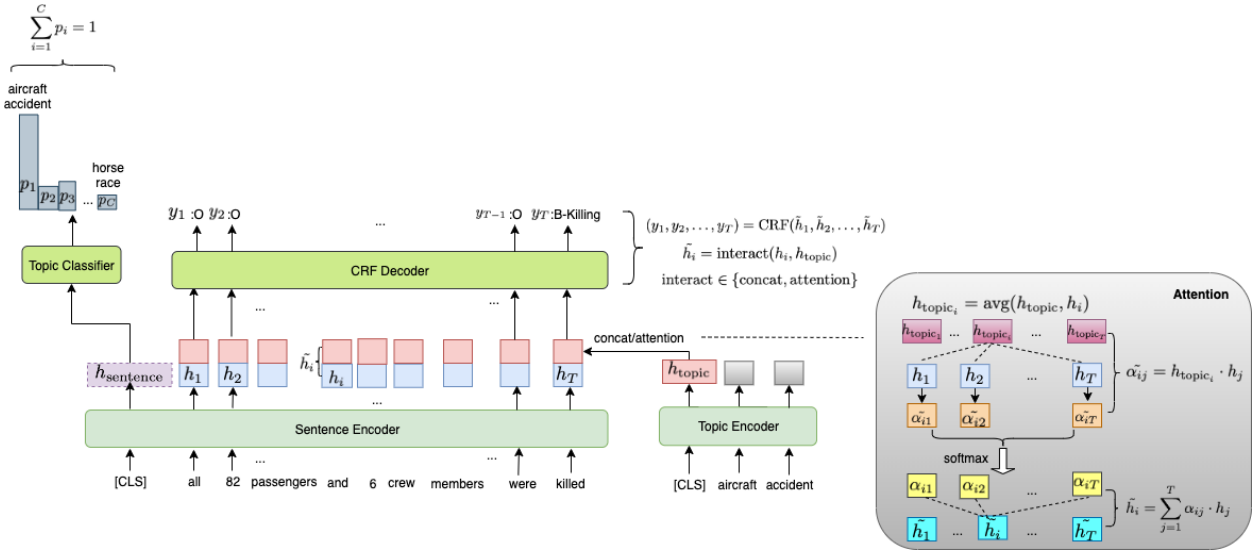

Figure 4.4: TAED architecture. TAED uses the topic name embedding and the sentence tokens embedding as the enhanced representation of the sentence. Multi-task learning of event detection and topic classification is utilized to further improve the performance.

The architecture workflow can be read from the bottom to top. The inputs of the model

are: (1) text tokens $(x_1, x_2, \ldots, x_T)$ of the sentence, and (2) the text tokens (topicword$_1$, topicword$_2$, $\ldots$, topicword$_N$) of the topics that the sentence belongs to. The length of the sentence tokens is denoted by T. The length of the topic tokens is denoted by N. The sentence tokens are fed into a sentence encoder, which maps each token to a d-dimensional vector. The output of the sentence layer is a sequence of hidden vectors that captures the semantic information of the each token along with dependencies between the tokens. The output from the encoder can be denoted as $(h_1, h_2, \ldots, h_T)$, where h_i is a d-dimensional vector. Similar to the sentence tokens, the topic tokens are also fed into a topic encoder. Different from the sentence encoder that generates a sequence of hidden vectors, one for each token, we want to obtain a single d-dimensional vector that summarizes the information of all topic name tokens. In order to generate the summarizing vector for the topic name tokens, two approaches are proposed. The first approach is to adopt the special `[CLS]` token introduced in BERT in the next sentence prediction pre-training step, which is described in detail in Section 2.7.3. The special `[CLS]` token is pre-trained in a way that the hidden vector out of the encoder is a representative hidden vector of the entire sentence. Thus, it is appropriate to use the `[CLS]` token's hidden vector as a representation of the topic. The second approach is to generate hidden vectors for each token in the topic name, and then to take the average of all hidden vectors as one representative hidden vector for all topic name tokens. In either of the two cases, the output of topic encoder is a single hidden vector h_{topic} in the d-dimensional space. At the same time, we also obtain a summarizing vector for the entire sentence tokens denoted as h_{sentence}. It is the hidden vector for the sentence `[CLS]` token which is highlighted in purple in Figure 4.4.

In the next step, we focus on how to generate topic-aware sentence representation vectors. So far, we have obtained a sequence of hidden vectors $(h_1, h_2, \ldots, h_T)$ in d-dimensional space representing the sentence and a single vector h_{topic} representing the topic name. We proposed two approaches for the topic and sentence interaction. The first approach is to concatenate the topic vector h_{topic} to each of the sentence tokens h_i, where i ranges from 1

to T. The output from the concatenation operation is a sequence of $2d$-dimensional vector $(h_1; h_{\text{topic}}, h_2; h_{\text{topic}}, \ldots, h_T; h_{\text{topic}})$, where the "$a; b$" notation here represents the concatenation of two vectors a and b. This sequence of $2d$-dimensional vectors are topic-aware, since each of the hidden vector $\tilde{h_i} = (h_i; h_{\text{topic}})$ carries the topic information. However, with this approach, the topic representation interacts with each sentence token evenly. In order to address this issue, we propose the second approach: attention based interaction. Before we start to describe the attention based interaction approach, note that the concepts of query, key and value used in the attention calculation are described in Section 2.7.3. In the attention based interaction approach, we use the $(h_{\text{topic}_1}, h_{\text{topic}_2}, \ldots, h_{\text{topic}_T})$ as the queries for the attention calculation. h_{topic_i} is defined as the average of h_{topic} and h_i. We use the average operation instead of the concatenation operation because we need to maintain the vectors in the queries as d-dimensional for the subsequent attention calculations. The keys and values are both the sequence of sentence vectors $(h_1, h_2, \ldots, h_T)$. For each query h_{topic_i}, we obtain a sequence of similarity scores $(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iT})$ by conducting dot product with each key $\alpha_{ij} = h_{\text{topic}_i} \cdot h_j$ and followed by softmax to normalize. The similarity scores are used as the weights to highlight which tokens in the sequence is more important to the topic. A bigger α score indicates a relative higher similarity. The details can be found in Section 4.5.4. In this way, for each query h_{topic_i}, we obtain a topic-weighted comprehensive representation $\tilde{h_i}$, which is a d-dimensional vector. To summarize the output from the topic and sentence interaction, we obtain a sequence of hidden vectors $(\tilde{h_1}, \tilde{h_2}, \ldots, \tilde{h_T})$. For the concatenation approach, $\tilde{h_i}$ is a $2d$-dimensional vector. For the interaction approach, $\tilde{h_i}$ is a d-dimensional vector.

With the output from the previous encoding and topic-sentence interaction layers, we can proceed with the event detection and topic classification process. We train event detection and topic classification jointly. The idea is to use the topic classification to further help embed the topic information in the sentence representation implicitly, instead of explicitly embedding the topic information as the interaction layers do. We utilize h_{sentence} as the input

to the topic classification. We choose to not use the topic comprehensive hidden vectors as the input because if the hidden vector already has the topic information, it will be trivial for the classifier to predict the topic of the sentence. We adopt the softmax layer for the topic classification, since it is a multi-class classification problem. The details are described in Section 4.5.7. The output of the softmax layer is a C-dimensional vector $(p_1, p_2, \ldots, p_C)$, where C is the number of topics in the dataset, and all elements in $(p_1, p_2, \ldots, p_C)$ sum to 1. Each p_i represents the probability that the sentence belongs to topic_i. With the output for the topic classifier model and the topic label of the sentence in the training dataset, we were able to calculate the negative log likelihood loss defined in Equation (4.12). The input for the event detection is the topic comprehensive vectors from the interaction layers. In order to enforce the dependency between BIOE tagging schema, we adopt a CRF layer. The details of CRF layer can be found in Section 2.5.2, and the training details as well as the loss function definition (Equation (4.10)) can be found in Section 4.5.6. The idea behind the loss function design is to minimize the loss on the labelled tag sequences in order to find the best transition and emission metrics that fit the training data well. We further train the model by combining the loss of topic classification and event detection in a weighted fashion.

4.5.2 Sentence Encoder

The sentence encoder represents the text tokens $(x_1, x_2, \ldots, x_T)$ of the sentence as low-dimensional, real-valued vectors. To effectively capture the long-range dependencies between the input tokens, we use BERT [Devlin et al., 2019], whose underlying layers use the self-attention mechanism to mitigate the long-range dependencies issue. The input and output relations for a sentence encoder is represented as:

$$h_1, h_2, \ldots, h_T = \text{SentenceEncoder}(x_1, x_2, \ldots, x_T), \tag{4.1}$$

where $h_i \in \mathbb{R}^d$. Equation (4.1) means that, for each input text token x_i in a sentence, the sentence encoder will output a d-dimensional real-valued vector as its representation.

4.5.3 Topic Encoder

Our topic encoder encodes the topic information by using the topic name or topic representative vocabulary that is mined by using the ranked TF-IDF features from each topic. For example, the top five representative vocabularies for the civilian attack topic are: massacre, attack, kill, police, and people. Similarly, we use BERT to encode the topic information. Unlike the sentence encoder where each token is encoded as a vector, we use a hidden vector h_{topic} (red, solid-border token in Figure 4.4) returned from the BERT encoder for the `[CLS]` token to represent the entire information carried by the topic keywords. This is represented as:

$$h_{\text{topic}} = \text{TopicEncoder}(\text{topicword}_1, \ldots, \text{topicword}_N), \tag{4.2}$$

where $h_{\text{topic}} \in \mathbb{R}^d$.

4.5.4 Topic-Aware Sentence Representation

To associate the sentence representation with its document's topic, we append the topic vector representation h_{topic} to each token vector representation $h = (h_1, h_2, \ldots, h_T)$ in the sentence, shown in Figure 4.4. Then we obtain the topic-aware contextualized vector representation of the sentence tokens as:

$$\tilde{h} = (\tilde{h}_1, \ldots, \tilde{h}_T) = (h_1; h_{\text{topic}}, .., h_T; h_{\text{topic}}), \tag{4.3}$$

where $\tilde{h}_i$ is a $2d$-dimensional vector. The drawback of using concatenation is the topic information contributes to each token in the sentence evenly. In order to address this issue, we proposed an attention based interaction method to obtain a topic-aware comprehensive

sentence representation. The idea of attention was first used in Neural Machine Translation (NMT) [Bahdanau et al., 2015]. Instead of paying attention to everything, the attention mechanism is designed to highlight the important information in a sequence. In order to calculate the attention we first need to define query, key, value used in our scenario. For the query, we obtain a d-dimensional vector h_{topic_i} by taking average of h_{topic} and h_i. We use $(h_1, h_2, \ldots, h_T)$ for both key and value. Given index i, a dot product operation is applied on h_{topic_i} and h_j, where j ranges from 1 to T. Each of the dot product operation generates a weight α_{ij}, calculated as:

$$\tilde{\alpha_{ij}} = h_{\text{topic}_i} \cdot h_j, \text{ and} \tag{4.4}$$

$$(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iT}) = \text{softmax}(\tilde{\alpha_{i1}}, \tilde{\alpha_{i2}}, \ldots, \tilde{\alpha_{iT}}). \tag{4.5}$$

In this way, given a query vector h_{topic_i}, we obtain a sequence of weights $(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iT})$. The weights are used to measure the importance of tokens in the sentence when talking about the topics. A higher weight indicates a higher importance. Afterwards, a topic-comprehensive representation for the query h_{topic_i} can be obtained by calculating:

$$\tilde{h_i} = \sum_{j=1}^{T} \alpha_{ij} \cdot h_j. \tag{4.6}$$

Similarly, we can obtain $\tilde{h_i}$ for each query vector h_{topic_i}, where i ranges from 1 to T. Eventually, a sequence of topic weighted topic-aware hidden vectors $(\tilde{h_1}, \tilde{h_2}, \ldots, \tilde{h_T})$ are obtained. Each $\tilde{h_i}$ is a d-dimensional vector.

4.5.5 Event Detection CRF Decoder

Table 4.2 shows an example of identifying took place as a trigger of event type process start.

We feed the topic-aware contextualized token representations $(\tilde{h}_1, \tilde{h}_2, \ldots, \tilde{h}_T)$ to CRFs [Laf-

the	Total	Nonestop	Action	Wresting	(	TNA	)	promotion	that	took	place	on	October	23
O	O	O	O	O	O	O	O	O	O	B	E	O	O	O

Table 4.2: An example of the tag sequence for event type Process Start annotated with the BIOE scheme.

ferty et al., 2001] to obtain the sequence of BIOE tags with the highest probability:

$$(y_1, y_2, \ldots, y_T) = \text{CRF}(\tilde{h}_1, \tilde{h}_2, \ldots, \tilde{h}_T). \tag{4.7}$$

The CRF decoder [Huang et al., 2015] can enforce the tagging consistency that captures dependency between the output tags. For example, if we already know the starting boundary of an attribute (B), this increases the likelihood of the next token to be an intermediate (I) or end of boundary (E), rather than being outside of the boundary (O). CRF contains a linear layer and a transition matrix, which are used to calculate the emission and transition scores for the tag predictions, respectively. The score for an input text sequence X that belongs to a specific topic to be assigned with a tag sequence Y can be calculated as:

$$\text{score}(X, \text{topic}, Y) = \sum_{i=1}^{T-1} \mathbf{T}_{y_i, y_{i+1}} + \sum_{i=1}^{T} \mathbf{E}_{i, y_i}, \tag{4.8}$$

where $\mathbf{T} \in \mathbb{R}^{m \times m}$ is the transition matrix, $\mathbf{T}_{ij}$ is the transition score of i-th tag to the j-th tag. $\mathbf{E} \in \mathbb{R}^{T \times m}$, $\mathbf{E}_{ij}$ represents the i-th token is assigned j-th tag in the tag set. m is the number of tags in the tag set which includes different B, I, E tags for each event type and a shared O tag. For example, given two event types, killing and cause to be included, there will be 7 tags including "B-Killing", "I-Killing", "E-Killing", "B-Cause to be included", "I-Cause to be included", "E-Cause to be included", and an O tag. Let a be the number of event types, then $m = 3a + 1$.

4.5.6 Event Detection Training

The event detection task is trained to maximize the log likelihood of (X, topic, Y) triplets in the training set, the score of given tokens, and topic that has predicted tags Y is given in Equation (4.8), and the log likelihood to maximize is defined as:

$$\log p(Y|X, \text{topic}) = \log \frac{\text{score}(X, \text{topic}, Y)}{\sum_{Y' \in \text{tagset}^T} \text{score}(X, \text{topic}, Y')}. \tag{4.9}$$

Assuming we have N samples in the training set, then the loss to minimize for the event detection task is defined as:

$$\text{Loss}_{\text{event\_detection}} = -\sum_{i=1}^{N} \log p((\hat{Y}_i|X_i, \text{topic}_i), \tag{4.10}$$

where $\hat{Y}_i$ is the ground truth label for sentence i.

4.5.7 Topic Classification Training

Our topic classifier classifies each sentence into its corresponding topic. In order to avoid information leakage, instead of using the topic-aware contextualized token embeddings $\tilde{h}$ from equation (4.3) to classify the topics, we directly use the `[CLS]` token representation denoted as h_{sentence} from the sentence encoder (the purple, dashed-border token in Figure 4.4) to classify the topic. The loss function for the topic classification task can be defined as

$$(p_1, \ldots, p_C) = \text{softmax}(W_t \cdot h_{\text{sentence}} + b_t), \text{ and} \tag{4.11}$$

$$\text{Loss}_{\text{topic}} = -\sum_{j=1}^{N} \sum_{i=1}^{C} y_{ij} \log(p_{ij}), \tag{4.12}$$

where $W_t \in \mathbb{R}^{C \times d}$, $b_t \in \mathbb{R}^C$ and C is the number of the topics in the training dataset.

4.5.8 Multi-Task Training

We jointly train TAED for event detection and topic classification in a multi-task learning [Caruana, 1997; Martínez Alonso and Plank, 2017; Yang et al., 2017] setting, by combining the loss of the two tasks:

$$\text{Loss} = \text{Loss}_{\text{event\_detection}} + \gamma \cdot \text{Loss}_{\text{topic}}, \tag{4.13}$$

where γ is a non-negative tunable hyper-parameter. By training the model in a multi-tasking setting, both the event detection and topic classification tasks will contribute to the contextualized vector representation learning for the sentence and topic tokens.

4.6 Experiments and Evaluation

In order to test our hypothesis that the event topic information can help event detection, we used the MAVEN dataset [Wang et al., 2020], which has a wide range of event types (168) and also comes with the topic labels that humans annotated. Furthermore, we test our model with the RAMS dataset [Ebner et al., 2020], which also comes with many event types (139). Though the RAMS dataset does not come with topic labels for each document, we use LDA [Blei et al., 2003] to generate pseudo topic labels. We first tested our work on the full MAVEN dataset. The MAVEN dataset only releases the gold labels for the training and validation datasets and not for the test dataset. In order to speed up our experiments with the data that has gold labels, we combined the training and validation dataset, then separated the merged dataset further into a 70%/15%/15% distribution. Additionally, the topic occurrence in the original dataset is extremely skewed, with the highest topic occurrence as 984 and the lowest topic occurrence as 1. We further sampled several topic-balanced datasets to test the effectiveness of our proposed method in a topic-balanced scenario.

Furthermore, to validate that an event topic can be used as a "bridge" to transfer knowl-

edge from high resource event types to low resource event types, we grouped event types based on their occurrences in the training dataset defined in Table 4.4. Examples of low resource event types can be from the *Rare* group, with the occurrence of the event type less than 20 times. While high resource event types can be from the *High* group, with the occurrence of the event type more than 500 times. We generated the Macro average group precision, recall and $F1$ score accordingly.

4.6.1 Performance

Altogether, we tested our topic-aware event detection framework on five settings of MAVEN which include the following: the full MAVEN dataset, two new splits of the train and validation dataset which has the gold labels, and two new splits of the train and validation datasets sampled in a topic-balanced way. In each of the settings, we included three random seeds as model parameters and report the means and standard deviations of each metric along with the P-value from a paired t-test in Table 4.3.

Model Type	**Dataset**	**P**(%)	**R**(%)	**F1**(%)	**P-Value**
BERT-CRF	Full MAVEN Data	66.15 ± 0.24	69.64 ± 0.43	67.85 ± 0.07	0.001*
BERT-CRF-TOPIC	Full MAVEN Data	66.28 ± 0.38	70.39 ± 0.40	$\mathbf{68.27 \pm 0.06}$	
BERT-CRF	Generated Data 1	65.18 ± 1.14	70.32 ± 2.96	67.63 ± 0.05	0.041*
BERT-CRF-TOPIC	Generated Data 1	66.21 ± 0.16	70.23 ± 0.16	$\mathbf{68.16 \pm 0.03}$	
BERT-CRF	Generated Data 2	65.65 ± 0.30	69.74 ± 0.38	67.63 ± 0.08	0.017*
BERT-CRF-TOPIC	Generated Data 2	66.35 ± 0.12	70.14 ± 0.34	$\mathbf{68.19 \pm 0.10}$	
BERT-CRF	Generated Data topic-balanced 1	64.09 ± 1.67	62.68 ± 1.72	63.33 ± 0.05	0.001*
BERT-CRF-TOPIC	Generated Data topic-balanced 1	63.9 ± 0.3	65.17 ± 0.18	$\mathbf{64.52 \pm 0.09}$	
BERT-CRF	Generated Data topic-balanced 2	63.93 ± 1.51	63.21 ± 1.28	63.53 ± 0.11	0.003*
BERT-CRF-TOPIC	Generated Data topic-balanced 2	64.41 ± 0.28	66.50 ± 0.24	$\mathbf{65.44 \pm 0.02}$	
BERT-CRF	RAMS	34.05 ± 0.14	33.83 ± 0.03	33.94 ± 0.05	0.004*
BERT-CRF-TOPIC	RAMS	36.67 ± 0.12	32.69 ± 0.04	$\mathbf{34.56 \pm 0.04}$	

Table 4.3: Test dataset performance on different settings. Mean and standard deviation of precision (**P** column), recall (**R** column) and $F1$ score (**F1** column) based on three executions (with different random seeds) of each setting are presented. To show the improvement of topic-aware model is significant among different settings (with the null hypothesis that the BERT-CRF-TOPIC model has the same performance as BERT-CRF model), P-Values from two-tailed paired t-test on the **F1** are included. * means the improvement is statistically significant. Note that since the test dataset of each setting that we tested on has overlapping, the t-test result might have high Type I error.

Groups	Occurrence	Event Type Count	Event Type Examples
Rare	$(0, 20]$	38	besieging, ratification
Low	$(20, 50]$	35	warning, rescuing
Medium	$(50, 100]$	35	assistance, escaping
Sub-high	$(100, 500]$	53	damaging, destroying
High	$(500, \infty)$	7	catastrophe, causation

Table 4.4: Event type groups based on its occurrence frequency in training data.

Groups	Macro P(%)		Macro R(%)		Macro F1(%)	
	topic	non-topic	topic	non-topic	topic	non-topic
Rare	26.12 ± 0.49	11.29 ± 1.09	19.63 ± 1.03	6.76 ± 0.42	$\mathbf{21.49 \pm 0.54}$	8.15 ± 0.36
Low	56.39 ± 1.17	50.25 ± 1.50	49.99 ± 1.71	35.35 ± 4.25	$\mathbf{50.66 \pm 1.46}$	38.25 ± 3.82
Medium	59.12 ± 0.42	61.16 ± 1.54	56.78 ± 0.89	51.72 ± 1.44	$\mathbf{56.51 \pm 0.47}$	54.07 ± 0.57
Sub-high	64.06 ± 0.67	62.60 ± 0.23	70.00 ± 0.38	69.26 ± 0.51	$\mathbf{66.03 \pm 0.31}$	64.61 ± 0.38
High	63.13 ± 0.75	61.93 ± 0.38	70.08 ± 0.60	71.78 ± 0.22	66.34 ± 0.38	$\mathbf{66.39 \pm 0.16}$

Table 4.5: Macro precision, recall, and $F1$ based on event type occurrence groups.

For the generation of the topic-balanced dataset, we first removed the tail topics whose topic occurrence is less than a threshold (15), and then down-sampling the head topics to the median occurrence of the topics. BERT-CRF is using BERT as the encoder and CRF as the decoder. BERT-CRF-TOPIC is our TAED architecture as shown in Figure 4.4. The topic classification weight is the weight set on the topic classification task when setting the event detection task weight as 1. The performance of the topic-aware and non-topic-aware models are shown in Table 4.3. We see that on the full MAVEN and the two generated dataset settings from splits of training and validation dataset, the topic-aware model improves the baseline by around 0.5% on the entity level micro $F1$ score, a commonly adopted metric for event detection tasks [Wang et al., 2020]. For the two topic-balanced datasets, the improvement on the $F1$ score is around 1.8%. We also saw improvement on the $F1$ score around 0.6% on the RAMS dataset. Using the paired t-test [Walpole et al., 2012, Chapter 10] along with three executions (with different random seeds as model parameters) of BERT-CRF and BERT-CRF-TOPIC models on each dataset setting in Table 4.3, we concluded that the improvement on each dataset setting is statistically significant. We observed that the performance of event detection model on RAMS is relatively low. By further investigation

Groups	P-Value
Rare	0.001*
Low	0.028*
Medium	0.039*
Sub-high	0.026*
High	0.872

Table 4.6: Two-tailed paired t-test results on $F1$ score from Table 4.5, based on topic and non-topic models. Since we have five groups, we will have five null hypothesis. As an example, the first null hypothesis is topic aware and non-topic aware model have the same performance on Rare event type group. And the last hypothesis is topic aware and non-topic aware model have the same performance on High event type group. * means the improvement is statistically significant. Note that samples of the tests are generated by running the model with three random seeds as the model parameter. And the trained model is tested on the same test dataset, thus the t-test result might have high Type I error.

of the training data, 25% of event types have less than 27 labelled instances, which explains this.

The performance based on high and low resources event types classification is shown in Figure 4.5. We observed that topic-aware model is doing much better on the low resource event types like *Rare* and *Low* group, with up to 13.34% improvement on $F1$ score compared to non-topic-aware model. The statistical significance test results are presented in Table 4.6. We observed that topic-aware model has statistically significant improvement on the *Rare*, *Low*, *Medium* and *Sub-high* event type groups.

In order to have more reliable t-test results for the performance comparison, we also conducted two-tailed paired t-test with the null hypothesis that the BERT-CRF-TOPIC model has the same performance as the BERT-CRF model on non-overlapping test datasets. The results are show in Table 4.7.

4.6.2 Ablation Study

In the ablation study, we first evaluate different ways to generate the topic embeddings. Further, we conduct experiments to show the effectiveness of different ways the topic information interacts with the main contextual sentence. Lastly, we conduct experiments to show

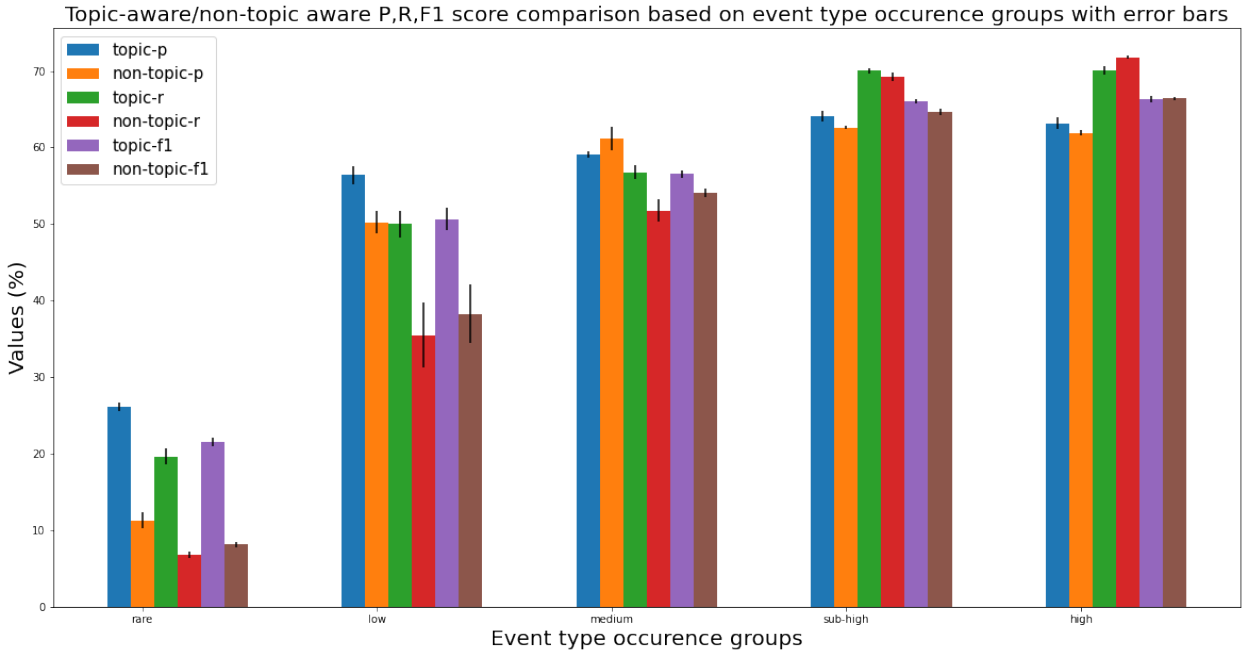

Figure 4.5: Topic-aware/non-topic-aware model macro precision, recall, and $F1$ performance based on Table 4.5 with error bars on different event type occurrence groups defined in Table 4.4.

Dataset	**P-Value**
Full MAVEN, Generated Data 1, RAMS	0.012*
Full MAVEN, Generated Data 2, RAMS	0.012*
Rare, Low groups	0.011*
Rare, Low, Medium groups	0.115

Table 4.7: Two-tailed paired t-test results on $F1$ score for non-overlapping test datasets with the null hypothesis that the BERT-CRF-TOPIC model has the same performance as the BERT-CRF. * means the improvement is statistically significant.

that auxiliary topic-classification task is effective.

Topic Name Encoding Ablation Study

As shown in Table 4.8, we observed that using the topic information to generate the context embedding without using multi-task learning (set weight = 0) is effective. We obtained improvement on the $F1$ by 1.07% from 63.66% to 64.73%.

We further tried different ways to generate the topic name embeddings. Instead of using the `[CLS]` token hidden representation, we tried to use the averaged topic name tokens

embeddings. Furthermore, aside from concatenating the topic name embedding on top of the context token embedding, we experimented with using topic name attending to the context sentence tokens. We see similar $F1$ performance for the above variations as shown in Table 4.9.

Model Type	**Topic-classification weight**	**General event word removed**	**P(%)**	**R(%)**	F_1(%)
BERT-CRF	NA	NA	66.91	60.71	63.66
BERT-CRF-TOPIC	1	True	64.44	66.52	**65.46**
BERT-CRF-TOPIC	0	True	63.52	66.04	64.73
BERT-CRF-TOPIC	0.1	True	62.37	66.66	64.44
BERT-CRF-TOPIC	0.5	True	64.17	64.92	64.53
BERT-CRF-TOPIC	2	True	63.76	65.33	64.51
BERT-CRF-TOPIC	10	True	64.26	58.73	61.38
BERT-CRF-TOPIC	25	True	63.8	47.34	54.33
BERT-CRF-TOPIC	50	True	60.36	33.06	42.65
BERT-CRF-TOPIC	75	True	55.98	25.69	34.99
BERT-CRF-TOPIC	100	True	49.58	19.38	27.79
BERT-CRF-TOPIC	1	False	65.59	64.29	64.93
BERT-CRF-TOPIC (with vocab)	1	True	64.97	65.08	65.02

Table 4.8: TAED performance with different topic-classification weights, performance of general event words kept/removed and performance of extra topic keywords added on for a specific topic.

Model Type	**Topic Embedding Type**	**P(%)**	**R(%)**	F_1(%)
BERT-CRF-TOPIC	`[CLS]`	67	63.78	65.35
BERT-CRF-TOPIC	Average Token Embedding	64.53	66.44	65.47
BERT-CRF-TOPIC	`[CLS]` freeze	64.57	66.57	**65.56**
BERT-CRF-TOPIC	Average Token Embedding freeze	65	65.64	65.32
BERT-CRF-TOPIC	`[CLS]` (topic as attention)	63.93	67.02	65.44

Table 4.9: Performance of using different ways to generate and utilize topic name embedding.

Topic Name Variations Ablation Study

The column "general event word removed" shown in Table 4.8 indicates whether or not we remove the very general word event from the topic names. Topic names with the general word

event, such as recurring event, historical event, and wrestling event, become recurring, historical, and wrestling, after removing the word event. This will help make our topic embedding more discriminatory to each other, as the word event, carrying less information to differentiate the topic phrases from one another, is removed. By adding this pre-processing step for the topic name, we can see that the performance gets enhanced, as shown in Table 4.8. We further explored adding the most important keywords of the topic along with the topic name to enrich the topic contextual embedding. We aggregated the documents that belong to one topic, and ranked the words in each topic by their tf-idf features. We used the top five keywords as the representatives and appended them to the topic name. Examples of added keywords are shown in Table 4.10.

Topic	Topic Vocabulary
earthquake	magnitude, occurred, quake, intensity, damage
winter storm	snow, blizzard, snowfall, new, winds
tennis event	open, doubles, slam, singles, djokovic
rugby match	chiefs, brumbies, sharks, final, crusaders
university boat race	oxford, cambridge, lengths, crews, goldie
war	paulo, vargas, 1930, presets, garais
military operation	bomb, manchester, ira, bombing, embassy
swimming event	golds, medals, bronze, silver, freestyle pool
cricket series	ashes, england, australia, test, wickets
civilian attack	massacre, attack, kill, police, people

Table 4.10: Sample of ten topic vocabulary terms and their top five representative keywords.

However, after adding the keywords to the topic names, the performance got a little bit worse, which could be caused by the noise brought in by the keywords. For example, the keyword new was added for the winter storm topic, and 1930 was added for the war topic.

Multi-Task Learning Ablation Study

We have conducted experiments using different weights on topic classification tasks, where γ in Equation (4.13) ranges from 0 to 100, where 0 means we ignore the topic classification loss during backpropagation. We saw that the sweet spot to achieve the best performance is to set the classification weight as 1 shown in Figure 4.6.

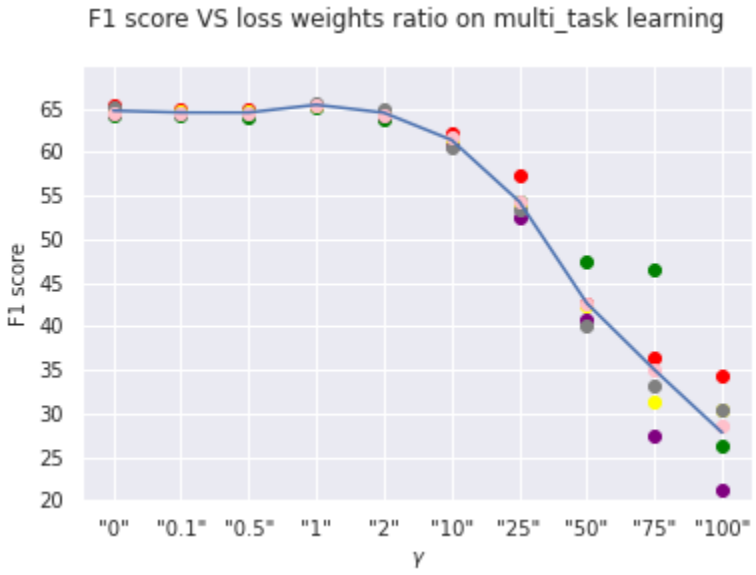

Figure 4.6: $F1$ performance vs γ. Each γ on X-axis has been run 5 times with different random seeds represented by points with different colors. The curve is the average performance of the 5 runs for each γ.

By setting equal loss weight on event detection and topic classification tasks, we further improved the $F1$ score by another 0.73% on top of the topic name embedding contribution.

4.7 Result Analysis

The performance shown in Table 4.8 combines both the performance of trigger identification and trigger classification. We further obtain the performance only for trigger identification shown in Table 4.11.

Model Type	**P**(%)	**R**(%)	**F1**(%)
BERT-CRF	77.3	77.9	77.6
BERT-CRF-TOPIC	77.93	78.59	**78.26**

Table 4.11: Performance of BERT-CRF and BERT-CRF-TOPIC only on trigger identification.

We can see that the topic-aware event detection model achieves better performance on both trigger identification and trigger classification. The error cases for event identification could come from two sources. First, the triggers are not correctly identified. Second, the

triggers are correctly identified, but the classification of the identified triggers is wrong. We conducted case studies for both of the error sources.

For one example:

> *Flight 821 is the deadliest accident involving a Boeing 737-500, surpassing the 1993 crash of Asiana Airlines Flight 733, and was the second-deadliest aviation incident in 2008, behind Spanair Flight 5022.*

The topic of the sentence is aircraft accident and the top event types for this topic are catastrophe, causation, and motions. The gold labels for the triggers, accident and incident, are both "B-catastrophe." The non-topic-aware model failed to identify the triggers in the first place, while the topic-aware model identifies the triggers and classifies them correctly into a catastrophe event.

For another example:

> *This was the first southern stadium rock show since ZZ TOP played to 80,000 people at UT Austin on September 1, 1974 and tore up the field.*

Both methods identified played as the event trigger. However, the topic-aware model predicted played as "B-competition" while the non-topic-aware model predicted it as "B-participation." The gold label for played is "B-competition." The topic of the sentence is music festival, and the top event types for this topic are social event, process start, arranging, and competition.

We can see that in both examples of the error cases, topic information plays an important role for the event detection task.

In addition, we performed case studies to understand the "bridge" behavior of document topics on transferring knowledge from high resource event types to low resource event types. For example, besieging is a rare event type. The most frequent topic that the event type belongs to is military conflict. The military conflict topic further has frequent event types including hostile encounter and attack, which are semantically related to the low resource event type besieging. Using the topic name as prior knowledge along with the introduction

of the topic classification task, we reinforce the topic information in the hidden layer sent to the decoder. The hidden layer thus carries the information of high resource event types that belong to the given topic. This further leads to information transformation to low resource event types that are semantically related to high resource event types.

4.8 Reproducibility

We implemented the Topic Aware Event Detection framework using the functionality provided in the PyTorch and Transformers packages. We adopted `bert-base-cased` version[1] of the BERT model and used the default AdamW optimizer with the Learning rate as 5×10^{-5} and Adam Epsilon as 10^{-8}. A dropout layer was introduced after the BERT encoder layer with a dropout rate 0.3. And the training batch size for the model was 16.

For the RAMS dataset topic generation, we applied the LDA algorithms on RAMS dataset for pseudo topic label generation. We chose the number of topics from the range of [10, 15, 25, 30, 35] and manually judged the quality of the topics returned and ended up using of 25 as the best fit of number of topics for the RAMS dataset. The pseudo topic name is from the outputs of the LDA, which is the combination of the top 5 important words for the specific topic. The pseudo topic names are listed below in Table 4.12.

4.9 Summary

This study proposed a topic-aware event detection method by using the topic name embedding to enrich the contextual representations of the sentences. It also adopts the multi-task set-up to combine the event detection and topic classification tasks. We showed the effectiveness of this method by testing on different datasets and conducting ablation studies. We explored different ways to generate topic embedding and different interaction methods between topic and sentence embeddings. Further analysis showed the topic-aware model

[1] `https://huggingface.co/bert-base-cased`

Topic Index	Pseudo Topic Labels
0	photo, caption, image, hide, force
1	oil, sanction, export, year, price
2	inform, secure, intelligence, government, email
3	police, attack, shoot, fan, man
4	email, investigate, fbi, hack, department
5	attack, bomb, build, kill, force
6	world, war, think, agreement
7	president, state, country, nato, unit
8	nuclear, bank, missile, system, weapons
9	attack, report, investigate, news, told
10	force, rebel, group, military, air
11	women, attack, share, right, day
12	isra, aid, human, intern, right
13	support, crimea, muslim, use
14	campaign, republican, president, donald, democrat
15	president, support, campaign, former, bush
16	million, tax, percent, wall, pay
17	foundation, million, report, government, department
18	vote, voter, elect, party, poll
19	women, sexual, rape, extradite, year
20	senate, gop, bill, seek, ryan
21	white, house, democrat, party, polite
22	question, ask, debate, cnn, death
23	polite, think, president, attempt, thing
24	state, unit, war, foreign, world

Table 4.12: Pseudo topic labels generated by LDA for RAMS dataset.

architecture beats the non-topic-aware model by a large margin in a few-shot event type scenario. Furthermore, we analyzed the event type distribution based on topics which fundamentally explains why the sentence topic information can help the event detection task.

There are limitations of the experimental design in Table 4.3 and Table 4.6. We are training the models whose parameters are initialized by three random seeds on the same train/validation dataset. And the trained models are tested on the same test datasets which leads to the consequences that the test performances are not independent. This violates the assumptions made by the t-test. Thus, our statistical test results have high type I error. The results in Table 4.3 and Table 4.6 can only validate the topic-aware model improves the

performance of event detection on MAVEN and RAMS datasets.

Chapter 5

Adaptive Scalable Pipelines for Event Data Extraction

Political event data has been increasingly important for researchers to study and predict global events. Until recently, the majority of political events were hand-coded from text, limiting the timeliness and coverage of event data sets. Recent systems have successfully employed big data systems for extracting events from text. These automated event systems have been limited by either slow performance or high infrastructure demands. In this work, we present a new approach to big data systems that allow for faster extractions when compared to existing systems. We describe a modular system, Biryani, that adaptively extracts events from batches of documents. We use distributed containers to process streams of incoming documents. The number of containers processing documents can be increased or reduced depending on available resources and load. The optimal configuration for event extraction is learned, and the system adapts to maximize the throughput of coded documents. We show the adaptability through experiments running on laptops and multiple commodity machines. We use this system to extract a new political event data set from several terabytes of text data.

5.1 Introduction

The availability of a large corpora of online news documents has made it possible for computer and social scientists to study human political behavior at previously impossible scales. One of the primary bottlenecks in deriving meaning from text documents is the resource demands of the natural language processing and information extraction that need to be performed. Current document processing pipelines suffer from a range of limitations when processing hundreds of millions of news articles for social science applications, either in terms of poor performance or ease of use. Document processing pipelines written by social scientists are easily installed and customized but tend to be single-threaded and slow for corpora larger than a few hundred thousand documents. Frameworks for distributed pipelines that are fully flexible (Apache Nifi, Kubernetes, Spark) require sophisticated infrastructure and significant technical expertise both to set up and to customize for document processing tasks, which creates a high hurdle for applied use.

In this paper, we present a simple, adaptive pipeline named Biryani for extracting political events from many news documents. The performance benef are derived from a low-overhead distributed architecture that adaptively tweaks processing parameters to optimize throughput on the processing machines using a Kalman filter. The modular, containerized architecture allows the pipeline to be executed on laptops and large clusters with minimal dependencies. The pipeline can be installed in a more straight forwad way compared to other distributed pipelines, which is a high priority for social scientists.

We begin by discussing social scientists' use of event data extracted from text. We then provide a system overview and describe the components of the system. We next present the experiment design and the evaluation of the experiment. We conclude by discussing the impact of the experiment results and giving future directions.

5.2 Background

News articles published on the web give social scientists in academia and government the ability to measure and understand political events worldwide [O'Brien, 2010; Wang et al., 2016]. In order to extract meaning from this text, social scientists have developed standardized event ontologies that define political actors and political events in a consistent form [Schrodt, 2012]. These ontologies represent political events in a framework of "who did what to whom," with standardized codes for different actors (e.g., government, military, or rebel) and the type of action (e.g., make statement, protest for leadership change, or fight with small arms). According to the ontologies, social scientists use specialized dictionary-based coders to extract events from text to transform raw text into event data.

Early systems used simple dictionary matching techniques to extract events [Schrodt, 2009]. Modern approaches use syntactic information provided by statistical parsers such as Stanford's CoreNLP [Manning et al., 2014] to better match noun and verb phrases with dictionaries [Norris et al., 2017]. Processing stories through CoreNLP and the event data extraction pipeline[1] can be very slow, both in "batch" mode and when run on stories streaming in from scrapers downloading news stories from web RSS feeds.[2] In the existing pipeline, as stories are scraped from the web, they are stored in a MongoDB and processed through a pipeline to perform CoreNLP annotations, and then run through a pipeline for event extraction, geoparsing, and other annotation tasks [Schrodt et al., 2014].

The document processing pipelines in the open-source event data research community are simple single-threaded pipelines that struggle to rapidly process large corpora in the hundreds of millions of documents. One approach to improving the processing pipeline is described in Solaimani et al. [2016]. This process uses a Spark architecture for distributed CoreNLP processing and event extraction.[3] We describe a system that can be executed

[1] `https://github.com/openeventdata/phoenix_pipeline`

[2] E.g., `https://github.com/johnb30/atlas`

[3] This approach and ours both take existing, specialized tools and reformulate them as distributed, scalable tools. An alternative approach would be to use an existing general distributed architecture like Kubernetes or Apache Nifi and modify it to perform the document processing task or to use general tools like [Rak et al.,

either as a single machine or a multi-machine distributed system. The pipeline needs lower set-up costs and without the need to set up a Spark cluster. This allows researchers to use the same tool for widely varying task sizes.

5.3 Methodology

5.3.1 System Overview

The architecture is built on Docker containers, which allow dependencies and software to be self-contained [Merkel, 2014]. The processing occurs within distributed containerized CoreNLP instances [Manning et al., 2014]. Each container pulls documents from a central RabbitMQ queue, processes them, and stores them in an SQLite database. This architecture allows the pipeline to run locally on one machine or it may be distributed across a heterogeneous cluster. The completed system is launched using a container orchestration system such as Docker Compose or Kubernetes.[4]

Containerizing and distributing the processing step has several advantages in addition to the ability to distribute across machines: the per-container threads, batchsize, and memory can be adapted to increase processing speed, containers can be automatically restarted on failure, and the architecture becomes generalizable to processes that are not natively multi-threaded (e.g., Python).

Figure 5.1 displays the components of the system. Raw data is uploaded from web pages into MongoDB. This is a common interface for storing data but it is not essential to the Biryani system. Data is transferred from MongoDB or an API to RabbitMQ. This queue is a persistent storage and the entry point to Biryani. Containers are spun up as consumers of the data queue. The containers execute the Stanford CoreNLP process with multiple threads. The data in each container is written to a local SQLite database. Finally, an

2012; Perovšek et al., 2016; Kano et al., 2011].

[4] `https://github.com/kubernetes/kubernetes`

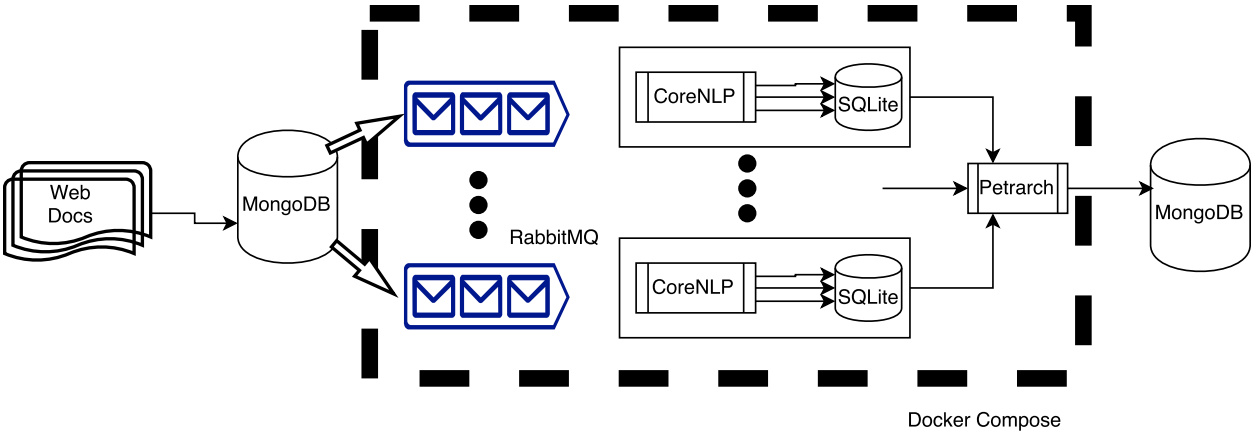

Figure 5.1: The Biryani system architecture.

existing specialized event extraction system [Norris et al., 2017] can be executed over the CoreNLP processed text.

5.3.2 Optimize System Parameters with Kalman Filter

To optimize parameters like batch and thread size automatically we employ a Kalman filter [Kalman, 1960]. The time taken to process documents can be viewed as a random variable; even with the batch size and thread number fixed, the processing time will differ each time the process runs. Randomized noise includes factors such as how many other processes are running on the system while it runs the pipeline, and exceptions that some documents could cause while being parsed. Therefore, we select the discrete Kalman filter algorithm, a Bayesian time series inference model. The Kalman filter uses a series of measurements with an estimated distribution of statistical noise and other inaccuracies, and infers a number of execution parameters of the pipeline. In our case, the parameter learned from the Kalman filter is batch size.

The Kalman filter estimates the current state variables (batch processing time) and their uncertainties. Once the outcome of the next measurement is observed, though with noise and measurement error, these measurement estimates are updated using a weighted average, with more weight being given to estimates with higher certainty.

The external parameters to the Kalman Filter are P, the a *posteriori* error estimate, R, the estimate of measurement variance, and a noise parameter Q. Because we executed the data pipeline on a machine that was not running many other processes, our noise parameter will be relatively small. We therefore set our Q, representing the stability of the system, to be small. The value Z is the observation running time of a batch by using the batch size proposed by Kalman filter. The current a *posteriori* estimation of the state is represented as X, and K is a relative weight given to the measurements. Based on the system configuration, the user may choose different hyper parameters for the Kalman filter to tune the performance better.

Kalman 0.75 filter is defined in the following way. A fixed value for Q as $0.1^{0.75}$ is used to model the stability of the system. The pseudo code for Kalman 0.75 filter is presented in Algorithm 1. We first initialize parameters P, Q, K, R, X and *batch*. For each batch in the data stream, we do the following updates. Using the estimated batch size, we obtain the observation value defined as the actual processing time of the batch. We update the weight value K, a bigger value of K indicates the estimated time taken X is less reliable. (If the estimate error P get bigger, then K will be bigger.) Next, we update the next step time taken for a batch as a combination of the actual time taken and current step time estimation weighted by K. When P is bigger, which indicates the estimation error is large, we distribute a lower weight $1 - K$ to the estimation and a larger weight K to the actual time taken. At last, based on the estimated time taken for the next step, the updated batch size can be calculated.

5.4 Experiments

In this section, we describe the experiments we conducted to demonstrate the performance of Biryani. We develop a set of experiments to examine the system's effectiveness and the efficacy of the system optimizations. The data in this system was stored on an Intel®

Algorithm 1 Pseudo code for Kalman 0.75 filter based optimization.

$P = 1$
$Q = 10^{-1}$
$K = 0.0$
$R = 0.1^{0.75}$
$X = 0$
batch = batch$_0$
while eachBatch **in** stream **do**
  $P = P + Q$
  $Z =$ **observation**(batch)
  $K = \frac{P+Q}{P+Q+R}$
  $X = K * Z + (1 - K) * X$
  $P = (1 - K) * P$
  batch $= X * \frac{\text{batch}}{Z}$
end while

Xeon® CPU X5687 @ 3.60GHz with 16 total cores and 96 GBs of RAM. These experiments' processing was performed on an Intel® Core™ i7-6950X CPU @ 3.00GHz CPU with 20 total cores and 126 GBs of RAM. The processing machine contains two 6 TB disks and has stated transfer speeds of up to 6 GB/s. For all the experiments we used only 8 cores of the machine and we limited the memory where specified.

We performed all experiments using the English Gigaword corpus [Parker et al., 2011]. This corpus contains a collection of news wire text data in English that has been acquired over several years by the Linguistic Data Consortium. The corpus contains 4 million documents (12 GB uncompressed) from The New York Times Newswire Service, Agence France Presse English Service, Associated Press' Worldstream English Service, and The Xinhua News Agency English Service. This data set mirrors web collection tasks performed by social scientists but is large enough to allow us to test scalability. The data set is preprocessed and added to a MongoDB database offline.

Our architecture already has several advantages over the current state-of-the-art. Our system runs without needing a "Big Data" cluster, which most social scientists do not have access to. Instead, Biryani can be quickly and easily deployed on any modern hardware. The distributed processing approach also allows us to add or remove machines from the pool of

workers dynamically. To further increase our processing speed, we can dynamically change each container's size and number of threads.

Given a particular machine configuration, the first experiment aims at discovering the optimal batch and thread size for the pipeline. We performed experiments varying the batch sizes and threads, using a randomized subset of documents of sizes $1K$and $25K$, respectively.

This experiment also logs the total contribution taken by each component in the pipeline, including the Stanford CoreNLP annotators. We note that skew is frequently a cause of performance problems for batch processing systems [Kwon et al., 2011]. A skew in document sizes can leave some batches waiting on a few large documents to complete processing. Additionally, the order of incoming documents can significantly impact the performance of the pipeline. To show the best practical results, we randomized the order of documents before each run of the experiments.

In the next set of experiments, we test the optimizations defined in Section 5.3.2. We compare the various parameters of the optimized pipeline with a static set pipeline configuration.

5.5 Evaluation

This section describes the results of the experiments described in Section 5.4. We first give timing numbers for different batch sizes. Next, we look at the breakdown of the proportion of time used for each component. We then examine the Kalman filter performance optimizations.

We first experiment to discover the optimal batch size. We ran each configuration three times and plotted the average time taken for each combination of the batch size and thread. Figures 5.2 and 5.3 show darker squares where the values are greater. From the graphs, we can approximate that 200 is the optimal batch size for randomized documents of different sizes from Gigaword. This confirms the need to optimize batch size decisions.

Figure 5.3 shows no significant improvement across all thread pool sizes. This suggests that Biryani will not see significant gains from optimizing thread pool sizes.

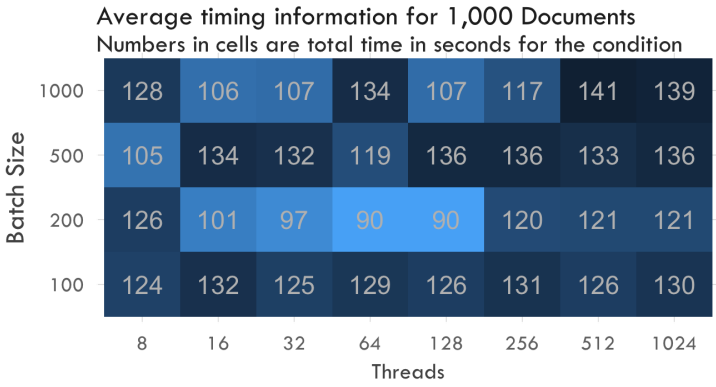

Figure 5.2: Average timing information for 1,000 Documents.

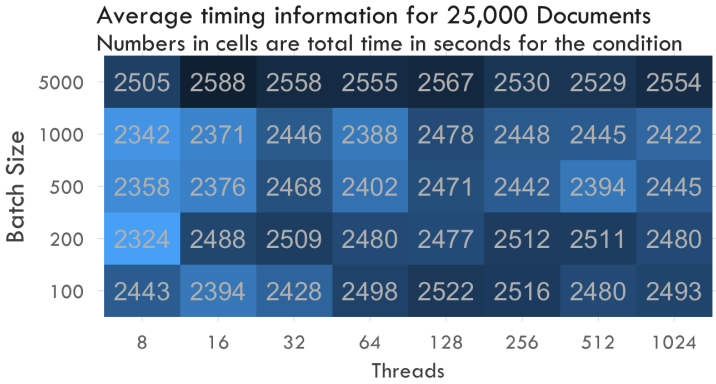

Figure 5.3: Average timing information for 25,000 Documents.

5.5.1 Adaptive Batch Size By Using Kalman Filter

From the experiments performed above we found that batch size was the most important feature to optimize. In order to effectively choose how much data should be processed per batch, we use a Kalman filter described below.

Table 5.1 shows performance gain of Kalman filter approach over static batch pipeline when performed on 150,000 documents. "Kalman 0.75" represents a pipeline with the

Pipeline	Docs	Run 1 (s)	Run 2 (s)	Avg (s)	% Gain	P-Value
Static	150K	13,555	13,619	13,587	-	0.044*
Kalman 0.75	150K	13,051	13,180	13,115	3.47%	

Table 5.1: Performance Gain of Kalman Filter approach over 150,000 documents. The null hypothesis is Kalman filter adapted system has the same performance as non Kalman filter adapted system. * means the improvement is statically significant using the two-tailed paired t-test. Note that since the test dataset of each setting that we tested on has overlapping, the t-test result might have high Type I error.

Kalman filter parameter $R = 0.1^{0.75}$ while other parameters P, K and Q remain unchanged. Note that the test data subset was reshuffled before each run of the pipeline.

5.5.2 Performance of The Pipeline on Laptop Configuration

We used Google Compute Engine (GKE) to spin up a VM that is similar to a current day laptop configuration. The configuration of the VM used was an Intel® Xeon® CPU @ 2.30GHz, 4 cores, and 16GB of RAM. We performed experiments on different pipeline approaches three times each on 12,484 random documents for each run and calculated the average.

Figure 5.4 shows the performance of the Kalman filter approach and the standard error bars of each experiment. We use the static algorithm as a baseline in the laptop configuration experiments. Kalman filter with a parameter of 0.75 sees the best speed up with a mean increase of 20.33% followed by the 1.0 Pipeline with an 16.41% mean increase. Kalman filter configurations of 0.5 and 0.25 achieved lower speedups of 14.86% and 14.04%, respectively. The lowest performing configurations still had speedups of 9.41% (Kalman 2.0) and 12.52% (Kalman 1.5). We presented the paired t-test results in Table 5.2. The performance improvements of Kalman filter 1.5, Kalman filter 1.0, Kalman filter 0.75 and Kalman filter 0.5 are significant. We concluded that Kalman filter optimization of the event data creation pipeline can also provide speedups to low resource environments.

In order to have more reliable t-test results for the performance comparison, we also

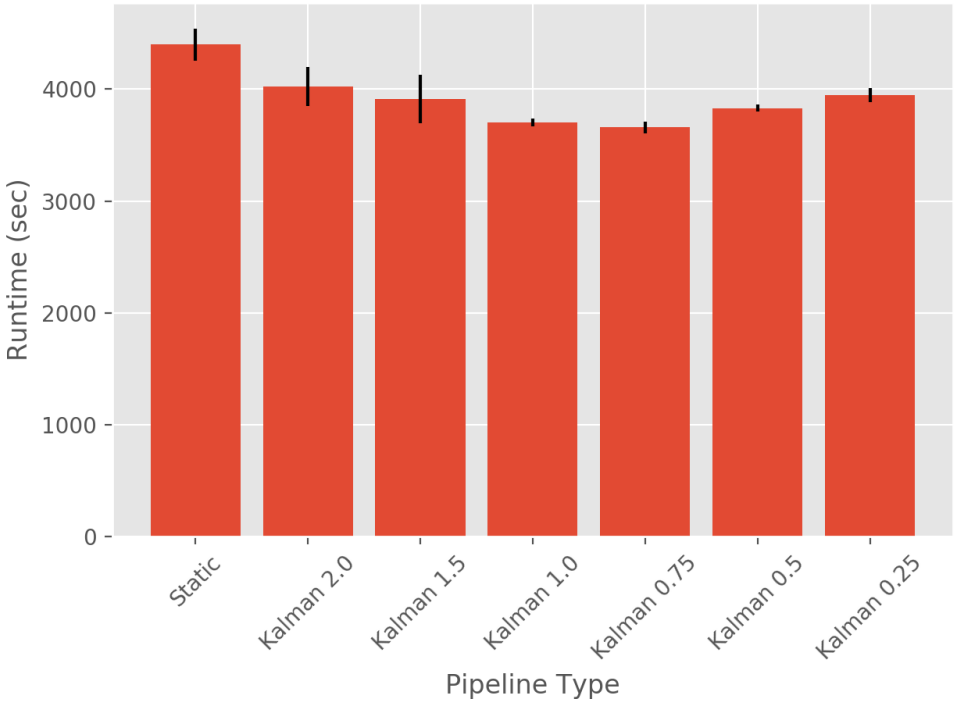

Figure 5.4: Kalman filter approaches over 12,484 sets of random documents with mean and standard deviation of three executions on each setting. The statistical significance tests for each settings are presented in Table 5.2.

conducted two-tailed paired t-test with the null hypothesis that Kalman filter based batchsize has the same performance as static batchsize on non-overlapping test datasets. The results are show in Table 5.3.

5.6 Summary

In this paper we describe a pipeline, Biryani, for extracting event data from web documents. Biryani has a container based architecture that adapts to the available systems and available load to process text. This architecture allows researchers to use whatever resources that they have to process a large data set. We were able to easily deploy into the Azure cloud in addition to the Google Compute Engine for extra processing bandwidth. We were able to publish largest machine-coded political event dataset covering 1979 to 2016 (2TB and 300 million

Kalman filter settings	P-Value
Kalman 2.0	0.169
Kalman 1.5	0.031*
Kalman 1.0	0.046*
Kalman 0.75	0.05*
Kalman 0.5	0.036*
Kalman 0.25	0.125

Table 5.2: Statistical significance test results for different Kalman filter based batchsize compared to static batchsize performance with two-tailed paired t-test. The null hypothesis is Kalman filter adapted system has the same performance as non Kalman filter adapted system. * means the improvement is statistically significant. The mean and standard deviations of three executions on each setting is presented in Figure 5.4. The test dataset was reshuffled three times to support three executions on each setting. Note that since the test dataset of each setting that we tested on has overlapping, the t-test result might have high Type I error.

Dataset	P-Value
150,000 documents, 12,484 documents	0.04*

Table 5.3: Two-tailed paired t-test results for non-overlapping test datasets with the null hypothesis that Kalman filter based batchsize has the same performance as static batchsize. * means the improvement is statistically significant.

documents have been processed) under the name TERRIER[5] (Temporally Extended Regularly Reproducible International Event Records). It is publicly shared with all researchers to study events. For future work we plan to optimize based on more system state variables. We will also integrate this processing system with a larger analytic pipeline that is used to query and study the extracted events. The code for the conducting the experiments mentioned in this chapter is open-sourced and available at `https://github.com/oudalab/biryani`.

[5] `https://terrierdata.org/`

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Event extraction is quite an essential and challenging task in the natural language processing domain. As shown in Figure 1.2, there are mainly three components in an event extraction workflow. Our work aims at scaling up the process of event extraction for all of the components: data, model, model deployment and inference. We have made the following efforts and contributions on the journey to make event extraction tasks more scalable and flexible.

On the data dimension, with the *New Techniques for Coding Political Events across Language* [Liang et al., 2018], we designed and implemented four different approaches to generate labelled events more accurately and efficiently. It can be generalized to different languages and different domains when the domain event taxonomy is given. Using the proposed approaches along with the website we developed, we were able to complete Arabic actor and verb dictionaries with coverage equivalent to the English language dictionaries in less than two years of work, compared to the two decades that the English dictionaries took to produce.

On the model dimension, with the *TAED: Topic-Aware Event Extraction*, we proposed a domain-aware event extraction method by utilizing the topic name embeddings to enrich the

sentences' contextual representations and the multi-task set-up of event extraction and topic classification tasks. With the proposed method, we conducted experiments with two event detection datasets MAVEN and RAMS, various event types. We achieved up to +1.8% on the $F1$ score compared to the baseline. Furthermore, we show that the topic-aware model we proposed can improve the few-shot event types scenario with a large margin of +13.34% on the $F1$ score and provide heuristic explanations in our case study.

On the model deployment and inference dimension, with the *Adaptive Scalable Pipelines for Political Event Data Generation* [Liang et al., 2017], we designed and implemented a pipeline, Biryani, for efficiently extracting event data from large amount of documents. Baryani has a container based architecture that adapts to the available systems and load to process text. This architecture allows researchers to use whatever resources they have to process a large data set.

6.2 Future Work

The best-performed Deep Learning method requires a large amount of training data, but using human effort to label more accurate data is a high cost. Nowadays, there is a trend to use weak-supervised data to train deep learning models. Weak-supervised data means the training data is large but the data might be noisy or obtained from an imprecise source. Since the data is noisy, the model needs to be aware of the noise and also needs a special design to deal with the noise [Song et al., 2022]. The noise-aware event extraction models are a way to explore on par performance with the model trained with accurate human labelled data.

For the closed-domain event extraction, when a new domain evolves, in order to do all four subtasks: trigger identification, trigger classification, argument identification, and argument role classification, we need a specific event-argument schema pre-defined. It will be worth the effort to explore ways on automatic schema/taxonomy generation for event

extraction on different domains. This will speed up moving the event extraction techniques into different domains.

Furthermore, the newly generated event extraction dataset like MAVEN comes with hierarchical event types as shown in Figure 2.3. The hierarchical structure of the event types is not being explored in the literature to bring in extra knowledge for event detection. As an example, because of the semantic similarity between the sibling event types, the model might be confused more to mis-classify the event into its sibling event types than more far away event types. It will be worth the effort to explore methods along this dimension to improve the performance of the event extraction.

Bibliography

Aguilar, J., Beller, C., McNamee, P., Van Durme, B., Strassel, S., Song, Z., and Ellis, J. (2014). A comparison of the events and relations across ACE, ERE, TAC-KBP, and FrameNet annotation standards. In *Proceedings of the Second Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 45–53, Baltimore, Maryland, USA. Association for Computational Linguistics.

Allan, J. (2012). *Topic detection and tracking: event-based information organization*, volume 12. Springer Science & Business Media.

Allan, J., Carbonell, J. G., Doddington, G., Yamron, J., and Yang, Y. (1998). Topic detection and tracking pilot study final report. In *Proceedings of the DARPA Broadcast News Transcriptionand Understanding Workshop*.

Armato III, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., and Hoffman, E. A. (2011). The lung image database consortium (LIDC) and image database resource initiative (IDRI): a completed reference database of lung nodules on ct scans. *Medical physics*, 38(2):915–931.

Atkinson, M., Piskorski, J., Tanev, H., van der Goot, E., Yangarber, R., and Zavarella, V. (2009). Automated event extraction in the domain of border security. In *International Conference on User Centric Media*, pages 321–326. Springer.

Atkinson, M., Piskorski, J., Tanev, H., and Zavarella, V. (2017). On the creation of a security-related event corpus. In *Events and Stories in the News Workshop Vancouver, Canada*, pages 59–65.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bach, S. H., He, B., Ratner, A., and Ré, C. (2017). Learning the structure of generative models without labeled data. In *International Conference on Machine Learning*, pages 273–282. PMLR.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet Project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98/COLING '98, page 86–90, USA. Association for Computational Linguistics.

Ben-Hur, A., Horn, D., Siegelmann, H. T., and Vapnik, V. (2001). Support vector clustering. *Journal of machine learning research*, 2:125–137.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Bosselut, A., Le Bras, R., and Choi, Y. (2021). Dynamic neuro-symbolic knowledge graph construction for zero-shot commonsense question answering. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 4923–4931.

Bossy, R., Bessières, P., and Nédellec, C. (2013). BioNLP shared task 2013–an overview of the genic regulation network task. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 153–160.

Cao, Y., Peng, H., Wu, J., Dou, Y., Li, J., and Yu, P. S. (2021). Knowledge-preserving incremental social event detection via heterogeneous gnns. In *Proceedings of the Web Conference 2021*, pages 3383–3395.

Capet, P., Delavallade, T., Nakamura, T., Sandor, A., Tarsitano, C., and Voyatzi, S. (2008). A risk assessment system with automatic extraction of event types. In *International Conference on Intelligent Information Processing*, pages 220–229. Springer.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Chen, L., Ruan, W., Liu, X., and Lu, J. (2020a). Seqvat: Virtual adversarial training for semi-supervised sequence labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8801–8811.

Chen, P., Bogoychev, N., Heafield, K., and Kirefu, F. (2020b). Parallel sentence mining by constrained decoding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1672–1678.

Chen, S., Wang, Y., Liu, J., and Wang, Y. (2021). Bidirectional machine reading comprehension for aspect sentiment triplet extraction. *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 12666–12674.

Chen, Y., Xu, L., Liu, K., Zeng, D., and Zhao, J. (2015). Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 167–176, Beijing, China. Association for Computational Linguistics.

Chen, Y., Yang, H., Liu, K., Zhao, J., and Jia, Y. (2018). Collective event detection via a hierarchical and bias tagging networks with gated multi-level attention mechanisms. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1267–1276, Brussels, Belgium. Association for Computational Linguistics.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537.

Conlon, S. J., Abrahams, A. S., and Simmons, L. L. (2015). Terrorism information extraction from online reports. *Journal of Computer Information Systems*, 55(3):20–28.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. *Advances in neural information processing systems*, 28.

De Cao, N., Izacard, G., Riedel, S., and Petroni, F. (2021). Autoregressive entity retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

DeHart, J., Xu, C., Grant, C., and Egede, L. (2021). Proposing an interactive audit pipeline for visual privacy research. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1249–1255. IEEE.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), Volume 1 (Long and Short Papers)*, pages 4171–4186.

Ding, X., Song, F., Qin, B., and Liu, T. (2011). Research on typical event extraction method in the field of music. *Journal of Chinese Information Processing*, 25(2):15–21.

Du, X. and Cardie, C. (2020). Event extraction by answering (almost) natural questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 671–683, Online. Association for Computational Linguistics.

Ebner, S., Xia, P., Culkin, R., Rawlins, K., and Van Durme, B. (2020). Multi-sentence argument linking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8057–8077, Online. Association for Computational Linguistics.

Ellis, J., Getman, J., Fore, D., Kuster, N., Song, Z., Bies, A., and Strassel, S. M. (2015). Overview of linguistic resources for the TAC KBP 2015 evaluations: Methodologies and results. In *Proceedings of the 2015 Text Analysis Conference, TAC 2015, Gaithersburg, Maryland, USA, November 16-17, 2015*. NIST.

Ellis, J., Getman, J., and Strassel, S. M. (2014). Overview of linguistic resources for the TAC KBP 2014 evaluations: Planning, execution, and results. In *Proceedings of TAC KBP 2014 Workshop, National Institute of Standards and Technology*, pages 17–18.

Ge, T., Cui, L., Chang, B., Sui, Z., Wei, F., and Zhou, M. (2018). EventWiki: a knowledge base of major events. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Ghaeini, R., Fern, X., Huang, L., and Tadepalli, P. (2016). Event nugget detection with forward-backward recurrent neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 369–373.

Gui, T., Ye, J., Zhang, Q., Li, Z., Fei, Z., Gong, Y., and Huang, X. (2020). Uncertainty-aware label refinement for sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 2316–2326.

Guo, S., Li, R., Tan, H., Li, X., Guan, Y., Zhao, H., and Zhang, Y. (2020). A frame-based sentence representation for machine reading comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 891–896.

He, K., Girshick, R., and Dollár, P. (2019). Rethinking ImageNet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hirschman, L. (1998). The evolution of evaluation: Lessons from the message understanding conferences. *Computer Speech & Language*, 12(4):281–305.

Honnibal, M. and Montani, I. (2017). Prodigy: A new tool for radically efficient machine teaching. Online Article. URL: `https://explosion.ai/blog/prodigy-annotation-tool-active-learning`. Last accessed: April 14, 2022.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *Computing Research Repository*, abs/1508.01991.

Ji, H. and Grishman, R. (2008). Refining event extraction through cross-document inference. In *Proceedings of ACL-08: HLT*, pages 254–262, Columbus, Ohio. Association for Computational Linguistics.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45.

Kano, Y., Miwa, M., Cohen, K. B., Hunter, L. E., Ananiadou, S., and Tsujii, J. (2011). U-compare: A modular NLP workflow construction and evaluation system. *IBM Journal of Research and Development*, 55(3):11:1–11:10.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Kussul, E. and Baidyk, T. (2004). Improved method of handwritten digit recognition tested on MNIST database. *Image and Vision Computing*, 22(12):971–981.

Kwon, Y., Balazinska, M., Howe, B., and Rolia, J. (2011). A study of skew in MapReduce applications. In *The 5th Open Cirrus Summit*. Moskow.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, F., Peng, W., Chen, Y., Wang, Q., Pan, L., Lyu, Y., and Zhu, Y. (2020a). Event extraction as multi-turn question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 829–838.

Li, P., Zhou, G., Zhu, Q., and Hou, L. (2012). Employing compositional semantics and discourse consistency in chinese event extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1006–1016.

Li, Q., Ji, H., and Huang, L. (2013). Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 73–82.

Li, W., Cheng, D., He, L., Wang, Y., and Jin, X. (2019). Joint event extraction based on hierarchical event schemas from framenet. *IEEE Access*, 7:25001–25015.

Li, Z., Chang, X., Yao, L., Pan, S., Zongyuan, G., and Zhang, H. (2020b). Grounding visual concepts for zero-shot event detection and event captioning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 297–305.

Li, Z., Ding, X., and Liu, T. (2018). Constructing narrative event evolutionary graph for script event prediction. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 4201–4207.

Liang, Y., Halterman, A., Irvine, J., Landis, M., Jalla, P., Grant, C., and Solaimani, M. (2017). Adaptive scalable pipelines for political event data generation. In *Big Data (Big Data), 2017 IEEE International Conference*, pages 2879–2883. IEEE.

Liang, Y., Jabr, K., Grant, C., Irvine, J., and Halterman, A. (2018). New techniques for coding political events across languages. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 88–93. IEEE.

Liao, J., Zhao, X., Li, X., Zhang, L., and Tang, J. (2021). *Learning Discriminative Neural Representations for Event Detection*, page 644–653. Association for Computing Machinery, New York, NY, USA.

Lin, H., Lu, Y., Han, X., and Sun, L. (2019). Cost-sensitive regularization for label confusion-aware event detection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5278–5283, Florence, Italy. Association for Computational Linguistics.

Liu, M., Liu, Y., Xiang, L., Chen, X., and Yang, Q. (2008). Extracting key entities and significant events from online daily news. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 201–209. Springer.

Liu, X., Huang, H., and Zhang, Y. (2019). Open domain event extraction using neural latent variable models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2860–2871, Florence, Italy. Association for Computational Linguistics.

Liu, X., Luo, Z., and Huang, H. (2018). Jointly multiple events extraction via attention-based graph information aggregation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1247–1256, Brussels, Belgium. Association for Computational Linguistics.

Lu, Y., Lin, H., Xu, J., Han, X., Tang, J., Li, A., Sun, L., Liao, M., and Chen, S. (2021). Text2Event: Controllable sequence-to-structure generation for end-to-end event extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2795–2806, Online. Association for Computational Linguistics.

Luan, Y., Wadden, D., He, L., Shah, A., Ostendorf, M., and Hajishirzi, H. (2019). A general framework for information extraction using dynamic span graphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3036–3046, Minneapolis, Minnesota. Association for Computational Linguistics.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Martínez Alonso, H. and Plank, B. (2017). When is multitask learning effective? Semantic sequence prediction under varying data conditions. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 44–53, Valencia, Spain. Association for Computational Linguistics.

Meng, H. (2015). Research on event extraction technology in the field of unexpected events. Master's Thesis, School of Computer Science, Shanghai University. Shanghai, China.

Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

Munson, M. A. (2012). A study on the importance of and time spent on different modeling steps. *ACM SIGKDD Explorations Newsletter*, 13(2):65–71.

Nédellec, C., Bossy, R., Kim, J.-D., Kim, J.-J., Ohta, T., Pyysalo, S., and Zweigenbaum, P. (2013). Overview of bionlp shared task 2013. In *Proceedings of the BioNLP shared task 2013 workshop*, pages 1–7.

Nguyen, T. H., Cho, K., and Grishman, R. (2016). Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 300–309, San Diego, California. Association for Computational Linguistics.

Norris, C., Schrodt, P., and Beieler, J. (2017). PETRARCH2: Another event coding program. *The Journal of Open Source Software*, 2(9):133.

Nuij, W., Milea, V., Hogenboom, F., Frasincar, F., and Kaymak, U. (2013). An automated framework for incorporating news into stock trading strategies. *IEEE transactions on knowledge and data engineering*, 26(4):823–835.

Osorio, J. and Reyes, A. (2017). Supervised event coding from text written in Spanish: Introducing eventus id. *Social Science Computer Review*, 35(3):406–416.

O'Brien, S. P. (2010). Crisis early warning and decision support: Contemporary approaches and thoughts on future research. *International Studies Review*, 12(1):87–104.

Parker, R., Graff, D., Kong, J., Chen, K., and Maeda, K. (2011). English Gigaword Fifth Edition LDC2011T07. Web Download. Linguistic Data Consortium, Philadelphia. `https://catalog.ldc.upenn.edu/LDC2011T07`.

Pedamonti, D. (2018). Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *Computing Research Repository*, abs/1804.02763.

Perovšek, M., Kranjc, J., Erjavec, T., Cestnik, B., and Lavrač, N. (2016). TextFlows: A visual programming platform for text mining and natural language processing. *Science of Computer Programming*, 121:128–152.

Piskorski, J., Tanev, H., Atkinson, M., Goot, E. v. d., and Zavarella, V. (2011). Online news event extraction for global crisis surveillance. In *Transactions on computational collective intelligence V*, pages 182–212. Springer.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. Technical report, OpenAI. Preprint. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Rak, R., Rowley, A., Black, W., and Ananiadou, S. (2012). Argo: an integrative, interactive, text mining-based workbench supporting curation. *Database: the journal of biological databases and curation*, volume: 2012.

Ramponi, A., van der Goot, R., Lombardo, R., and Plank, B. (2020). Biomedical event extraction as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5357–5367.

Ribeiro, S., Ferret, O., and Tannier, X. (2017). Unsupervised event clustering and aggregation from newswire and web articles. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 62–67.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI'93, page 811–816. AAAI Press.

Riloff, E. and Shoen, J. (1995). Automatically acquiring conceptual patterns without an annotated corpus. In *Third Workshop on Very Large Corpora*, pages 148–161.

Rospocher, M., van Erp, M., Vossen, P., Fokkens, A., Aldabe, I., Rigau, G., Soroa, A., Ploeger, T., and Bogaard, T. (2016). Building event-centric knowledge graphs from news. *Journal of Web Semantics*, 37:132–151.

Saha, S., Majumder, A., Hasanuzzaman, M., and Ekbal, A. (2011). Bio-molecular event extraction using support vector machine. In *2011 Third International Conference on Advanced Computing*, pages 298–303. IEEE.

Schott, L., Rauber, J., Bethge, M., and Brendel, W. (2019). Towards the first adversarially robust neural network model on MNIST. In *Proceedings of the 7th International Conference on Learning Representations (ICLR '19)*.

Schrodt, P. A. (2009). Tabari: Textual analysis by augmented replacement instructions. *Dept. of Political Science, University of Kansas, Version 0.7. 3B3*, pages 1–137.

Schrodt, P. A. (2012). *CAMEO: Conflict and mediation event observations event and actor codebook*. Department of Political Science, Pennsylvania State University, University Park, Pennsylvania. http://data.gdeltproject.org/documentation/CAMEO.Manual.1.1b3.pdf.

Schrodt, P. A., Beieler, J., and Idris, M. (2014). Three's a charm?: Open event data coding with EL:DIABLO, PETRARCH, and the open event data alliance. Paper presented at the International Studies Association meetings, Toronto, March 2014. https://parusanalytics.com/eventdata/papers.dir/Schrodt-Beieler-Idris-ISA14.pdf.

Solaimani, M., Gopalan, R., Khan, L., Brandt, P. T., and Thuraisingham, B. (2016). Spark-based political event coding. In *Big Data Computing Service and Applications (BigDataService), 2016 IEEE Second International Conference on*, pages 14–23. IEEE.

Song, H., Kim, M., Park, D., Shin, Y., and Lee, J.-G. (2022). Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*. Accepted. Preprint: `https://doi.org/10.48550/arXiv.2007.08199`.

Sundheim, B. M. and Chinchor, N. A. (1993). Survey of the message understanding conferences. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, page 56–60, Princeton, New Jersey, USA. Association for Computational Linguistics.

Tabik, S., Peralta, D., Herrera-Poyatos, A., and Herrera Triguero, F. (2017). A snapshot of image pre-processing for convolutional neural networks: case study of MNIST. *International Journal of Computational Intelligence Systems*, 10(1):555–568.

Tanev, H., Piskorski, J., and Atkinson, M. (2008). Real-time news event extraction for global crisis monitoring. In *International Conference on Application of Natural Language to Information Systems*, pages 207–218. Springer.

Tomas, M., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representaions of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Vanegas, J. A., Matos, S., González, F., and Oliveira, J. L. (2015). An overview of biomolecular event extraction from scientific documents. *Computational and mathematical methods in medicine*, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 6000–6010.

Walker, C., Strassel, S., Medero, J., and Maeda, K. (2006). ACE 2005 multilingual training corpus. Web Download. Linguistic Data Consortium, Philadelphia. `https://catalog.ldc.upenn.edu/LDC2006T06`.

Walpole, R. E., Myers, R. H., Myers, S. L., and Ye, K. (2012). *Probability & Statistics for Engineers & Scientists*. Prentice Hall, 9th edition.

Wang, W., Kennedy, R., Lazer, D., and Ramakrishnan, N. (2016). Growing pains for global monitoring of societal events. *Science*, 353(6307):1502–1503.

Wang, X., Wang, Z., Han, X., Jiang, W., Han, R., Liu, Z., Li, J., Li, P., Lin, Y., and Zhou, J. (2020). MAVEN: A Massive General Domain Event Detection Dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1652–1671, Online. Association for Computational Linguistics.

Wu, J., Cai, Z., and Zhu, X. (2013). Self-adaptive probability estimation for naïve bayes classification. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Wu, X., Wu, J., Fu, X., Li, J., Zhou, P., and Jiang, X. (2019). Automatic knowledge graph construction: a report on the 2019 ICDM/ICBK contest. In *Proceedings of 19th IEEE International Conference on Data Mining, ICDM 2019*, pages 1540–1545. Institute of Electrical and Electronics Engineers (IEEE).

Yan, H., Jin, X., Meng, X., Guo, J., and Cheng, X. (2019). Event detection with multi-order graph convolution and aggregated attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5766–5770, Hong Kong, China. Association for Computational Linguistics.

Yang, S., Feng, D., Qiao, L., Kan, Z., and Li, D. (2019). Exploring pre-trained language models for event extraction and generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5284–5294.

Yang, Z., Salakhutdinov, R., and Cohen, W. W. (2017). Transfer learning for sequence tagging with hierarchical recurrent networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Yao, Y., Ye, D., Li, P., Han, X., Lin, Y., Liu, Z., Liu, Z., Huang, L., Zhou, J., and Sun, M. (2019). DocRED: A large-scale document-level relation extraction dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 764–777, Florence, Italy. Association for Computational Linguistics.

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *Computing Research Repository*, abs/1702.01923.

You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., and Keutzer, K. (2018). ImageNet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10.

Yu, S. and Wu, B. (2018). Exploiting structured news information to improve event detection via dual-level clustering. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 873–880. IEEE.

Yuan, W., He, T., and Dai, X. (2021). Improving neural question generation using deep linguistic representation. In *Proceedings of the Web Conference 2021*, pages 3489–3500.

Zeng, Y., Feng, Y., Ma, R., Wang, Z., Yan, R., Shi, C., and Zhao, D. (2018). Scale up event extraction learning via automatic training data generation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 6045–6052.

Zhao, Y., Jin, X., Wang, Y., and Cheng, X. (2018). Document embedding enhanced event detection with hierarchical and supervised attention. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 414–419, Melbourne, Australia. Association for Computational Linguistics.