ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF CLASSICAL PHILOLOGY AND ITALIAN STUDIES

SECOND CYCLE DEGREE IN

DIGITAL HUMANITIES AND DIGITAL KNOWLEDGE

EXPLORING DEEP-LEARNING-BASED FRAME IDENTIFICATION SYSTEM:

A COMPARATIVE ANALYSIS OF PERFORMANCE ACROSS VARIOUS LANGUAGE MODELS AND HYPERPARAMETERS

Dissertation in

Natural Language Processing

Supervisor

Prof. Fabio Tamburini

Co-Supervisor

Prof. Aldo Gangemi

Defended by

Manu Srivastava

Graduation Session III

Academic Year 2022/2023

Dedication

To my Mother,
for showing me the magic in words,

To my Father,
for giving the strength to undertake challenges.

Acknowledgement

I would like to extend my heartfelt gratitude to Professor Tamburini for his unwavering support, invaluable guidance, and profound insights throughout the duration of this thesis. His mentorship has been instrumental in shaping my research journey and fostering intellectual growth.

I am deeply indebted to my friends and family for their constant encouragement, unwavering belief in my abilities, and endless motivation during this challenging endeavor. Their encouragement and support have been the cornerstone of my perseverance and determination. I am truly fortunate to have such a supportive network of individuals who have stood by me every step of the way.

I would also like to express my sincere appreciation to the University of Bologna for providing a conducive environment for personal and academic growth. The university's commitment to excellence, coupled with its provision of resources and dedicated faculty members, has played a pivotal role in fostering a nurturing academic atmosphere. I am grateful for the opportunities afforded to me by the university, which have allowed me to flourish intellectually and professionally. It is through the university's unwavering support and commitment to educational excellence that I have been able to embark on this rewarding academic journey.

Abstract

This thesis delves into the realm of computational semantics and explores the optimization of semantic parsing systems, driven by the increasing demand for sophisticated natural language processing (NLP) technologies in applications like question-answer systems and dialogue systems, including platforms such as ChatGPT. Motivated by the need to enhance the performance of AI systems and mitigate biases and miscommunications, the study focuses on deep-learning-based semantic parsing, aiming to optimize its expressivity and performance through hyper-parameter tuning and experimentation with components such as language models. The research investigates the potential benefits of using larger architecture language models and fine-tuned graph embedding generators. Leveraging a frame identification system combining the prowess of modern language model and Adaptive Graph Encoder, the study systematically explores the impact of hyper-parameter tuning and component replacements on system performance. Through meticulous experimentation and performance evaluation, insights into the system's robustness, efficiency, and adaptability are gained, informing future optimizations in frame identification using deep learning methods. The thesis is structured to provide comprehensive insights into the domain, computational semantics, existing literature, experimental setup, results, and conclusions, offering a road-map for future explorations in semantic parsing and NLP advancements.

Keywords : Computational semantics, Semantic parsing, Natural language processing, Deep learning, Hyper-parameter tuning, Language models, Frame identification, Graph embeddings, Performance evaluation.

Contents

List of Figures

List of Tables

1. Introduction

We wouldn't be far from the truth to say that the phenomenon of language hides its complexity in its facade of intuitiveness. Native speakers of any language have at some point felt lost in their understanding of why certain phrase structures make sense but others don't, or why certain words inflect according to a pattern but at certain times deviate from it. They can tell what's right from wrong but find it hard to put a rationale behind it. The quote below by Matt Groening humorously captures this dilemma of the native speakers.

> **"I know all those words, but that sentence makes no sense to me."**
> — Matt Groening (*creator of 'The Simpsons'*)

The need for meaning-making and understanding its process is probably as old as the need to communicate and has been studied in many fields of study, quite prominently in Philosophy, Linguistics, Logic, and relatively recently in Computer science. Wittgenstien said and we quote,

> **"The limits of my language means the limits of my world."**
> — Ludwig Wittgenstein (*from Tractatus Logico-Philosophicus*)

He quite aptly highlights one of the core reasons why we engage in this process and that is to understand what can be expressed through language and how. If we were to apply his quote to the world of AI, it no longer remains a metaphorical limitation but a genuine one. The race to make better Language models in recent times is a testament to that.

This work also belongs to the vast body of work done towards understanding semantics through computational methods and technologies.

1.1 Goal

During the course of this work we deal with the challenges posed by Frame semantic parsing which is type of semantic parsing based on the semantic Framework developed by Charles J. Fillmore. This framework, commonly called as Frame Semantics, aims to capture the latent semantic in a text through an analysis of its individual words as agents who play their role and interact with other words around them to create meaning. The task of doing Frame Semantic then becomes akin to that of correctly identifying the roles which these words play and which words play what role in the sentence.

Jurafsky states that the problem of Frame-Semantic parsing is of value in numerous NLP applications [Gildea and Jurafsky, 2002], such as question-answer systems and dialogue systems to name a few. As these technologies start to be utilised in our day-to-day life through applications like ChatGPT, it becomes imperative to improve their capacity to perform better and avoid biases and miscommunications between humans and machines. We also continue to see continuous growth in the field of natural language processing as well as in machine learning in general which also posits the challenge of elaborate testing and exploration of their potential applications. Motivated by these observations we decide to look into work with a semantic parsing system and explore how can it be optimised. We also try to map a deep-learning-based semantic parser's tendencies in order to make AI more accessible and we hope to make our work guide other future explorations and help them avoid repetitions of experiments in directions that can be non-favourable.
We state our hypothesis as follows :

Hypothesis Statment: A Deep-learning-based machine learning system dealing with semantic parsing can benefit from an extensive exploration of its expressivity and performance through exploratory tuning of its hyper-parameters as well as through experimentation on its components such as the language model used.

Following our hypothesis statement, we created these research questions to direct our attention and achieve our goals.

- **Research objective 1:** To explore how can a semantic parse benefit from using a language model that is trained with a bigger architecture and provides richer word embeddings.
- **Research objective 2:** Treating text as a graph could provide improvements in capturing its implicit syntactic and semantic information and a semantic parser will benefit from a fine-tuned graph embedding generator.

1.2 Research object

To conduct our study, we chose to work with the frame identification system proposed by Tamburini [2022] where he combined the Electra language model Clark et al. [2020] with Adaptive Graph Encoder Cui et al. [2020] to achieve a state-of-the-art performance with a system which is very light in terms of computational requirements and easily lends itself to our research objectives.

1.3 Method of experiment

This study focused on exploring the chosen Frame identification system by tuning its hyper-parameters and replacing its parts to gauge their impact on performance. This process aims to check its sensitivity and do a performance evaluation, this provides an understanding of the system's behavior under various configurations and component alterations. We start by identifying the hyper-parameters that significantly influence the AI system's performance,

such as word and sentence embeddings, network architecture, and graph embedding quality. We systematically adjust these parameters following general good practices in the field while keeping other factors constant. Additionally, we also modify the classifier components with added dropout layers to assess their comparative effectiveness. We meticulously document the system's performance metrics across different configurations in order to gain insights into the system's robustness, efficiency, and adaptability, thus informing further optimizations and advancements on the task of Frame identification using deep learning methods.

1.4 Thesis structure

The thesis is structured as follows —

1. **Chapter 1 -** provides a glimpse of the domain of work and our goals and also gives a brief introduction to the systems we utilise in this work as well as a summary of the findings.

2. **Chapter 2 -** provides a background about Computational semantics and Frame-Semantics

3. **Chapter 3 -** provides a look in the state of the art and the existing literature to guide our choices in setting up and performing our experiments.

4. **Chapter 4 -** covers the details of the Frame Identification system we experiment with and goes in depth to describe the different aspects we work with when performing the experiments and lays the foundations for understanding the results and experiments.

5. **Chapter 5 -** covers the experiment with using the graph representaion along with a pre-trained LLM to train our model on multi-task and we share our benchmarking setup.

6. **Chapter 6 -** contains the results obtained from all the different experiments and highlights the key observations and patterns.

7. **Chapter 7 -** contains our conclusions derived from the results and discussion about what can be done it future.

2. Background of the domain

2.1 On Semantic Analysis

Why perform semantics analysis

Semantic Analysis, situated within Natural Language Processing (NLP), endeavors to interpret the meaning inherent in human language. While comprehending Natural Language may appear intuitive for humans, machines face a formidable challenge due to the intricate nature and subjectivity of language. The goal of Semantic Analysis in NLP is to grasp the meaning conveyed in a given text, considering factors such as context, the logical structure of sentences, and the roles played by grammar.

Categories of semantic analysis

Depending on the level of text we are dealing with, Semantic Analysis of Natural Language can be classified into three broad parts:

1. **Lexical Semantic Analysis**: Lexical Semantic Analysis involves understanding the meaning of each word of the text individually. It basically refers to fetching the dictionary meaning that a word in the text is deputed to carry.

2. **Compositional Semantics Analysis**: Although knowing the meaning of each word of the text is essential, it is not sufficient to completely understand the meaning of the text.

3. **Discourse Semantic Analysis**: It involves understanding the meaning making which happens through long term dependencies found in text and works at the level of a big textual body.

2.2 On computational semantics

Defining Computational semantics

Having understood the basic idea behind what it means to do semantic analysis and why we do it, we can start to also understand how this task can become incredibly cumbersome due to the complex and dynamic nature of natural language itself. This is where computers come in and this gave rise to the field of computational semantics which Blackburn and Bos

[2003] define as a combination of formal semantics, computational linguistics and automated reasoning. To recapitulate, the aim of computational semantics is to facilitate the generation of rich semantic representations in an automated fashion, enabling their use in inferences.

Computational semantics deals with linguistic meaning with a computational approach to natural language and it shares many of the concerns of semantic theory and may apply some of its formal treatments of meaning. Lappin [2005] says that, while semantic theory try to formalise the knowledge that speakers posses in order to interpret the sentences of their language, computational semantics works in parallel to introduce methods and techniques to create representations of these meanings in a manner which is computationally viable.

One of the core aspect of computational semantics is to model a way in which the meanings of phrases and sentences are computed systematically from the meanings of their syntactic constituents [Lappin, 2005]. It is possible to divide this inquiry along two directions. The first aims to answer how should meanings be represented while the second deals with how are the semantic representations of syntactically complex expressions emerging from the meanings of their component parts.

Lappin [2005] underscores that we generally work with the underlying assumption that generating a semantic interpretation for a sentence depends upon the a well made representation which does justice to its syntactic structure.This makes it fundamental concern for any semantic theory to enquire about the manner in which semantics and syntax affect each other.

Meaning Representation in semantic analysis

While it is intuitive for humans to know and understand the meaning of a textual information, the same cannot be said about a computer. Thus, machines tend to represent the text in specific formats in order to interpret its meaning. This formal structure that is used to understand the meaning of a text is called meaning representation. We can usually find the following units for creating a semantic system for MR as shown by Semantic Analysis-web article [2021].

1. **Entity**: An entity denotes a distinct unit or individual, such as a person or a location, like ‘GeeksforGeeks’ or ‘Delhi’.

2. **Concept**: A concept serves as a broader abstraction of entities, representing a general class of individual units, such as Learning Portals, Cities, or Students.

3. **Relations**: Relations establish connections between different entities and concepts, facilitating the understanding of their relationships, as seen in statements like ”CourseEra is a Learning Portal” or ”Paris is a City.”

4. **Predicate**:Predicates embody the verb structures within sentences, conveying actions or states of being.

Formal systems for making meaning representations

There are many semantic systems which can be used to perform meaning representation but how does one select the right one for capturing the essence of human language? This

is largely dependent upon what we want to do with this representation, the level of detail we want to work at and what kind of language phenomenon we are interested in studying. We also need to keep in mind the capacity of natural language to be context sensitive which makes a textual expression be dubious at times and without context, can be assigned different meanings.

We List down some of the most popular approaches used to perform MR.

- First-order predicate logic (FOPL)
- Semantic Nets
- Frame Semantics
- Conceptual dependency (CD)
- Rule-based architecture
- Case Grammar
- Conceptual Graphs

In our work, we are using the 'Frame' as our formal system for performing automated semantic analysis using a deep learning architecture.

Paradigm of text processing

Another factor to consider when working with the above stated assumption of computational semantics is the paradigm of representing text while preserving its syntactical and semantic information. Overtime, the paradigm of choosing textual representation has changed significantly. In general, there are three different ways in which we can represent natural language when preparing them for any kind of computational processing :

1. **BAG** : This method, the simplest and oldest approach is to represents natural language as a collection of words or tokens. It disregards the specific order of tokens within the text and only considers the frequency of unique tokens. Even if the text is shuffled randomly, its meaning remains unchanged from this perspective. Notably, topic modeling techniques, such as those proposed by Blei et al. (2003), adopt this view to represent input texts as mixtures of topics, with each topic modeled as a combination of words.

2. **SEQUENCE** : Representing natural language as a sequence of words or tokens mirrors the way humans typically speak and write. This approach captures more nuanced information about text, such as the sequence of tokens and the frequency of word pairs occurring in local contexts. Notable techniques leveraging this view include linear-chain Conditional Random Fields (CRF) by Lafferty et al. (2001), which considers sequential dependencies in predictions, and word embeddings methods like word2vec by Mikolov et al. (2013), which learn embeddings by predicting context words for a target word.

3. **GRAPH** : Representing natural language as a graph is a prevalent approach in NLP. While sequential data seems the most intuitive, the NLP community has a long history of using various graph representations for text or world knowledge. These include dependency graphs, constituency graphs, Abstract Meaning Representation (AMR) graphs, Information Extraction (IE) graphs, lexical networks, and knowledge graphs. Text graphs can encompass multiple hierarchical elements like documents, passages, sentences, and words. Compared to the other paradigms, the graph view captures richer relationships among text elements. Traditional graph-based methods, such as random walk and label propagation, have proven effective in challenging NLP tasks like word-sense disambiguation, name disambiguation, co-reference resolution, sentiment analysis, and text clustering.

The figure2.1 below illustrates the difference in these paradigms with three example sentences.

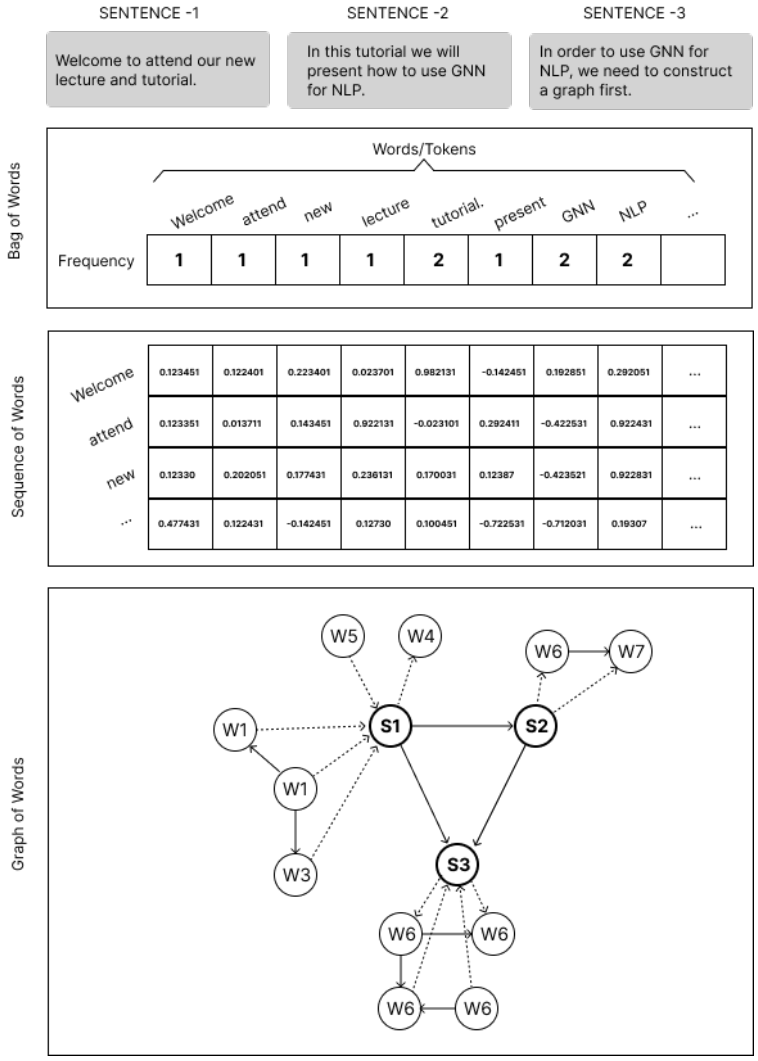

Figure 2.1: Text processing paradigms in NLP

Looking at the paradigms, it becomes clear that the graph based representation methods

allow for better integration of richness in terms of syntactical information and there exist a diverse set of graph which are suitable for different types of unit under consideration. Some commonly used types are dependency graphs, constituency graph, AMR graphs, etc.

2.3 Frame-Semantics and FrameNet

Since we will be working with a Frame Identification pipeline trained on FrameNet annotated corpus, it is required to develop a practical understanding of the selected framework for performing semantic analysis. We will look at what are cognitive frames and its implications and how frame semantics incorporates it in performing semantic analysis.

Frame-semantics is one of the paradigms for doing computational semantics and FrameNet is a resource created for this task. Fillmore et al. [2003] informs us that the central idea of Frame Semantics is that the meaning of a word/s must be made in relation to schematic representations of conceptual structures and patterns of beliefs, practices, institutions, images, etc. They call the schematic representations as Frames and they constitute the foundation for meaningful interaction in a given speech community.

2.3.1 Theory of Frame-Semantics

Cognitive Frames

There exists a broad conceptual framework known as a"frame" [Minsky, 1974; Goffman, 1974; Tannen, 1993], alongside related concepts like schema [Bartlett, 1932; Rumelhart, 1975], idealized cognitive model [Lakoff, 1987], script [Schank and Abelson, 1975], and even meme [Dawkins and Kraljevstvo, 1976], among others, which have been extensively explored in cognitive sciences since the 1970s. These frames can be defined as organized packages of knowledge, beliefs, and patterns of practice that shape and enable humans to comprehend their experiences.

According to Brachman and Schmolze [1985], these cognitive frames are typically expressed as "slot-filler representations", comprising interconnected roles with constraints on the potential or likely fillers of those roles. Humans access various frames due to factors such as living on Earth, being subject to its daily and annual cycles, possessing bodies that respond to gravity and biological needs, and having perceptual faculties. Additionally, cultural membership influences our response to institutions, symbols, artifacts, and values, while participation in speech communities further shapes our schematic knowledge.

Thus, humans possess schematic knowledge encompassing diverse topics, including gravity, heat, living and non-living distinctions, colors, emotions like pain and joy, societal constructs like marriage and government, and cultural elements like weekends and military titles.

Semanticists believe that the complete meaning of a text surpasses its linguistic form alone, suggesting that a dictionary limited to the standing definitions of individual words would inadequately aid text comprehension. Similarly, it is challenging to envision a computer program recognizing the humor in a passage solely based on its linguistic analysis. Our access to various frames, influenced by factors such as planetary existence, bodily functions,

cultural norms, and speech communities, underscores the richness and complexity of human comprehension.

Frame invocation and frame evocation

Frame theory makes a distinction between two cognitive acts: frame invocation and frame evocation. Fillmore and Baker [2009] say that frame invocation occurs when an interpreter, possibly unconsciously, engages cognitive processes to comprehend incoming information. In contrast, frame evocation is the interpreter's cognitive experience, triggered by language-specific associations linking linguistic signs to particular frames. The discovery and analysis of such associations, acquired during language acquisition, form the basis of Frame Semantics.

Fillmore and Baker [2009] emphasize the importance of distinguishing between:

The utilization of cognitive frames by individuals to interpret experiences, irrespective of whether those experiences are communicated through language and frame Semantics, which explores how linguistic forms (such as words, fixed phrases, and grammatical patterns) are associated with cognitive structures (frames) within our language knowledge, influencing the interpretation process and outcomes.

2.3.2 Scope of Frame Semantics

As described before, any formal system for meaning representations relies on defined units. These units define the scope of that formal system. The FrameNet project [FrameNet, 2011] uses these four types of units to define its scope of use. We also show example of these units from the FrameNet.

- Events :
  - **Being born**: LUs: born.v, come into the world.v
  - **Giving birth**: bear.v, beget.v, birth.n, birth.v, bring forth.v, carry to term.v, have.v .
  - **Death**: croak.v, death.n, demise.n, die.v, end.n, expire.v, kick the bucket.v . . .
- Relations
  - **Being relevant**: irrelevant.a, pertinent.a, play (into).v, relevant.a
  - **Personal Relationship**: adultery.n, affair.n, affianced.a, amigo.n, bachelor.n, beau.n
- States
  - **Being in operation**: off.prep, on.prep, operate.v, operational.a
  - **Being located**: find.v, lie.v, located.a, sit.v, situated.a, (ten)-twenty.n, whereabouts.n
- Entities
  - **Gizmo**:appliance.n, centrifuge.n, contraption.n, device.n, gear.n, machine.n,

2.3.3 About the FrameNet data-structure

The Berkeley FrameNet Project FrameNet [2011] endeavors to transcend piecemeal observations by constructing a comprehensive frame-based database. Their database now comprises hundreds of frames and many of which encompass substantial sets of words from the English vocabulary. Each frame is further accompanied by sentence annotations, serving both as evidence for analyses and as a repository of examples for further research.

They use the following steps in the FrameNet lexical analysis process are as follows:

1. **Characterising the frames**, i.e., the situation types for which the language has provided special expressive means.

2. **Describing and naming the frame elements (FEs)**, i.e., the aspects and components of individual frames that are likely to be mentioned in the phrases and sentences that are instances of those frames.

3. **Selecting lexical units (LUs) that belong to the frames**, i.e., words from all parts of speech that evoke and depend on the conceptual backgrounds associated with the individual frames.

4. **Creating annotations** of sentences sampled from a very large corpus showing the ways in which individual lexical units in the frame allow frame-relevant information to be linguistically presented.

5. **Automatically generating** lexical entries, and the valence descriptions contained in them, that summarise observations derivable from them.

This type of semantic analysis involves characterizing the situation types evoked by lexical units (LUs), determining the participant roles (frame elements) required to complete instances of any frame, and documenting how these elements are syntactically realized. Frame elements represent entities or properties that may or must be present in any frame instance, serving as topics of conversation once a frame is established. The classification in FrameNet distinguishes between "core" FEs and "peripheral" FEs, denoting varying degrees of importance or centrality within frames.

FrameNet Dataset description

Following this process, multiple versions of annotated data have been released by the berkeley project for FrameNet(FN)and we can look at its salient features as it changed over the past versions. Note that there have been further releases of FN since 1.5 but we only deal with FN 1.5.

Relations in FrameNet elements

As the number of frames has grown, it has become obvious that they are not simply a collection of separate entities, but there are networks or hierarchies of frames, and some are instances of others, some are components of others, etc. FrameNet resolves this with the creation of a Frame-to-Frame relation systems which allows assertions about semantic types

	R1.2	R1.3	R1.5	R1.6	R1.7
Frames	609	795	1,019	1,205	1,221
(non-lexical frames)	58	74	111	134	135
FEs in lexical frames	4,909	7,124	8,884	10,333	10,503
FE/lexical frame	8.91	9.88	9.78	9.65	9.7
Proportion non-lexical	9.5%	9.3%	10.9%	11.1%	11.1%
Frame relations	550	1,152	1,507	1,805	2,070
FE relations	2,770	6,311	8,252	11,791	12,393
Lexical Units	8,869	10,195	11,829	13,312	13,572
LUs/lexical frame	16.1	14.14	13.03	12.43	12.5
LUs w/ lexicog. anno	6642	6,815	7,711	8,313	8,390
Pct. LUs w/ lexicog. anno	74.9%	66.8%	65.2%	62.5%	62%
AnnoSets in lexicog anno	13,3846	139,439	149,931	157,739	174,017
Lexicog. AnnoSets/annotated LU	20.2	20.5	19.5	19.0	20.73
AnnoSets in full text anno	0	11,671	23,087	28,160	28,208
Total AnnoSets	133,846	151,110	173,018	201,695	202,225
Full Text anno/Total anno	0%	7.7%	13%	22%	14%

Table 2.1: FrameNet versions and their dataset statistics. *Data from Release notes of FrameNet 1.7*

to be made at the appropriate level of generality. Fillmore and Baker [2009] groups the seven common relations into three groups, Generalization, Event structure, and Systematic.

- Generalisation relations:

  1. **Inheritance**:All FEs of the parent frame are bound to FEs of the child frame, but the child FEs need not have the same name. The child can have more FEs, and the child's semantics is a subtype of the parent's semantics. For example, the Revenge frame inherits from the Rewards and Punishment frame, since it also involves a person inflicting a punishment on another. It differs explicitly from Rewards and Punishments in being outside of institutional or judicial control.
  2. **Perspective on**: Different lexical items (e.g., buy, sell) evoke frames with different perspectives on an abstract event (Commercial transaction), a kind of figure:ground relation [Gawron, 1985]. Specifically, buying takes the perspective of one participant in the goods-transfer, and selling takes the perspective of the other. In FN, they are linked to the abstract event via Perspective on relations.
  3. **Using**: The child frame depends upon background knowledge provided by the parent frame; at least some of the core FEs of the parent are bound to child FEs, but not all of them.

- Event Structure relations:

  1. **Subframe**: These are sub-events of a complex event, often with temporal ordering, e.g., in FN the Giving frame is linked to two sister frames, called, for lack of

better names, Pre-Giving and Post-giving, which provide information about who has what when (and inherit, indirectly from the Possession frame). These three frames together constitute the Giving scenario.

2. **Precedes**: This relation specifies temporal ordering, e.g., Giving Precedes Post-giving.

- Systemic relations:

1. **Causative of**: The parent frame represents the causative corresponding to the child frame, e.g., Cause change of position on a scale (LUs raise, lower) is the Causative of Change position on a scale (rise, fall).
2. **Inchoative of** : The parent frame represents the inchoative, and the child represents the stative. Change position on a scale (rise, fall) is the Inchoative of Position on a scale (high, low).

There is also the relation of **'See also'** but it is a special case that we don't look at in this work. Also, all of these frame-to-frame relations have accompanying frame element-to-frame element relations.

2.3.4 Subtasks in Frame Semantic parsing

The figure 2.2 shows the pipeline constructed for performing frame semantic parsing in 2010, and the work relied on a heuristical method for the target identification and subsequently performed frame identification and argument labeling using statistical methods on a full-text-annotation. This is still considered the standard pipeline for doing Frame semantic parsing where the stages and their challenges are:

1. Target identification : This step involves selecting a word/token or a sequences of words which are called the target predicate, since they are responsible for evoking a Frame within a given sentence. These target words could be verbs, nouns, adjectives, and even prepositions under specific conditions.

2. Frame Identification : This step follows the previous one but deals with determining the precise frame evoked by a target predicate in a given sentence, akin to a word-sense disambiguation task where frames can be seen as senses.

3. Argument identification : is sometimes also known as semantic role labeling. This step involves identifying words and phrases as the appropriate arguments of the target predicate, known as Frame Elements in the FrameNet schema and out task become to assign them specific roles.

2.3.5 Differences between different semantic analysis resources

As stated above, FrameNet is not the only resource created for performing computational semantic parsing but it surely offers a novel perspective on approaching the task. We will look

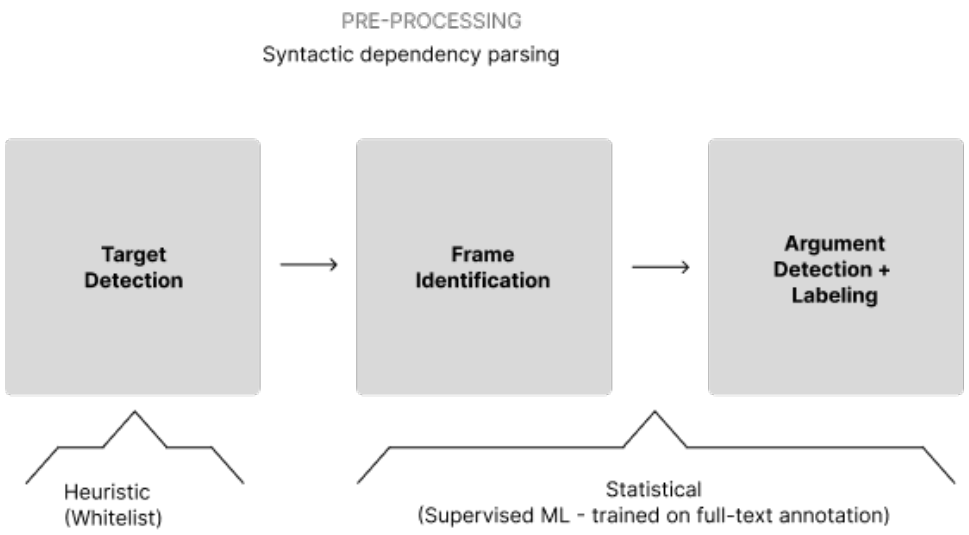

Figure 2.2: FN parsing steps

at how it differentiates itself from some other resources and what could be the advantages of doing so.

We briefly show the differences in the resource like Wordnet - Propbank - AMR and FrameNet.

FN vs WordNet

They both contain very different lexical information and are in many ways, complementary to each other. For example, while FN has little to say about common nouns, Wordnet has a good hierarchy for nouns and whereas wordnet has little syntagmatic information FN has a lot.

POS	**WordNet**	**FrameNet**
Noun	146,312	5,177
Verb	25,047	4,879
Adj	30,002	2,270
Adv	5,580	(other)387
total word sense	206,941	12,713

Table 2.2: FrameNet vs WordNet

WordNet (WN) features separate hierarchies for nouns, verbs, adjectives, and adverbs exclusively, while each FrameNet (FN) frame can encompass words of any part of speech. WN establishes hyper-/hyponymy relations between synsets, essentially between lexical units (LUs) that are synonymous, whereas FN does not articulate LU relations explicitly but delineates various types of frame relations. In terms of content, FN offers annotated examples elucidating the syntax and semantics of each LU and describes roles (Frame Elements) for every frame, providing a more detailed and comprehensive perspective compared to WN.

Furthermore, the FN frame hierarchy often yields superior generalizations compared to the WN synset hierarchy.

FN vs PropBank

Proposition Bank (Palmer et al., 2005, CL) initiated its work by annotating verbs along with their arguments and adjuncts in the Wall Street Journal (WSJ). It utilizes Penn POS tags and Penn TreeBank parse structures as foundational elements. Over time, it expanded its scope to include nouns and roles associated with verbs, resulting in a comprehensive annotation scheme. Notably, Proposition Bank has undergone significant developments in various languages including Chinese, Korean, Arabic, Hindi, and the biomedical domain. It benefits from the availability of efficient semantic role labeling systems. Unlike FrameNet, Proposition Bank does not feature frames; instead, it employs two levels of role labels: one set that is general (e.g., ARG0-ARG5, ARGM-Time, ARGM-Loc) and another set specific to each lexical unit or word sense. We can also see the comparison between how both the systems deal with annotations and role names

FN vs AMR

AMR (Abstract Meaning Representation) encompasses coreference, named entities, and value expressions, aiming to encapsulate all content words and their relationships, albeit with shallower lexical semantics compared to FrameNet (FN). Originally, the objectives of AMR annotation prioritized cost-effectiveness and utility for natural language processing (NLP), contrasting with FN's focus on lexicography. While AMR relies on PropBank rolesets for event predicates, which are lexicalized and coarsely disambiguated, FN employs rich semantic groupings in its frames. FN annotations typically consist of labeled spans within sentences, whereas in AMR, graph components may not align explicitly with sentence elements, and not all concepts correspond to words in the sentence. Moreover, FN lacks explicit relationships across frame annotations for a given sentence, whereas AMR emphasizes the composition of predicates and highlights explicit shared arguments.

2.4 Important works in Frame-Semantics domain

Since the creation of FrameNet there have been many works aimed at solving its tasks which have bought various innovations in the the domain. We look at some of the key works to get a general idea of how the approach has changed over the years.

2014

Das et al. [2014] provided a two stage approach to performing frame semantic parsing using semi-supervised methods which set a new high-score on the SemEval 2007 benchmark data set.

2017

Hartmann et al. [2017] bought to focus the issue of domain dependence of NLP systems and their application in large-scale text analysis, also how it restricts the applicability of FrameNet semantic role labeling (SRL) systems and also proposed methods to mitigate this issue.

2018

Devlin et al. [2018] introduced a bidirectional encoder representations from transformers (BERT) approach to semantic parsing. BERT is a type of neural network that has been shown to be effective for a variety of natural language processing tasks.

2019

The work by Popov & Sikos which talked about how small variations in data processing can have big consequences for ranking parser performance [Popov and Sikos, 2019] and they proposed an open-source standardized processing pipeline, which can be shared and reused for robust model comparison. Also Tan and Na [2019] produced a transformers based work for target disambiguation and frame classification. This work used position attention in combination with BERT.

2021

Jiang and Riloff [2021] present a new model for frame identification that uses a pre-trained transformer model to generate representations for frames and lexical units (senses) using their formal definitions in FrameNet. Their frame identification model assesses the suitability of a frame for a target word in a sentence based on the semantic coherence of their meanings.

Su et al. [2021] proposed a Knowledge-Guided Frame Identification framework (KGFI) that integrates three types frame knowledge, including frame definitions, frame elements and frame to frame relations, to learn better frame representation which are subsequently used for Frame Identification task.

2022

KID [Zeng et al., 2020] proposed Frame Semantic Graph (FSG) to represent frame semantic structures extracted from the text with graph structures. In this way, they transform frame semantic parsing into an incremental graph construction problem to strengthen interactions between subtasks and relations between arguments.

Around the same time Tamburini [2022] proposed a much simpler but equally effective model which leveraged the combined benefit of graph embeddings from Adaptive Graph Encoder (AGE) with word embeddings generated by the Electra-base model which achieved a new state of the art score in the Frame Identification task.

2.5 Challenges in computational Frame-Semantics

Challenge of domain

Issue: Frame semantics heavily depends on the identification of frames and their associated lexical units, which can vary across different domains. Pre-trained models may not generalize well to new domains. Challenge: Adapting frame semantic parsers to diverse domains requires domain-specific knowledge and training data, making it challenging to achieve broad coverage across various contexts.

Size of FN dataset

- Issue: Annotated datasets for frame semantic parsing are often limited in size and may not cover the wide range of linguistic variations present in natural language. Challenge: Insufficient training data may lead to challenges in capturing the diversity of frame-evoking expressions, affecting the parser's ability to handle nuanced and less common usages.

Syntactic complexity

Issue: Complex sentence structures and syntactic constructions can complicate the identification of frame-evoking elements and their associated roles. Challenge: The parser needs to navigate through intricate syntactic relationships to accurately assign frame elements, demanding sophisticated algorithms and syntactic analysis.

Frame Evolution and Shifting

Issue: The meaning of frames may evolve or shift over time, and new frames may emerge with changes in language use. Challenge: Keeping frame semantic models up-to-date and adaptable to evolving language patterns is a challenge, particularly in dynamic environments where frame semantics can undergo shifts. Also talk about language dependency of Frame and Frame elements.

Multi-word Expressions (MWEs)

Issue: Many frames are expressed through multi-word expressions, and their identification and interpretation pose additional challenges. Challenge: Recognizing and parsing MWEs requires handling varying word orders, discontinuity, and non-compositional meanings, which adds complexity to the parsing process.

3. State of the Art in NLP

Today, machine learning has become an inseparable part of NLP and we almost always need to use/train a machine learning model to perform NLP tasks. Consequently, we have seen a tremendous rise in works that deal with the process of making better models that can process and represent text in a way that is amenable to downstream learning. This can be loosely viewed as developing general-purpose knowledge that allows the model to "understand" text.

The research of language modelling(LM) has received extensive attention in the literature. Zhao et al. [2023] provides an overview in which they divided it into four major development stages:

- **Statistical Language Model (SLM)**: depend on statistical learning methods that rose in the 1990s which utilised Markov chains for predicting the next work based on its most recent context to build a word prediction model. These models usually dealt with a fixed context length n to perform their prediction and were also commonly referred to as n-gram language models, e.g., bigram and trigram language models.

- **Neural Language Model (NLM)**: can be understood as those models which utilised neural networks to calculate and assign the probability of word sequences. Some famously used architectures used to build such models are recurrent neural networks (RNNs) and Multi-Layer Perceptron (MLP). Further improvements were made on such models through the introduction of the concept of distributed representation of words by Bengio et al. [2000] which built the word prediction function based on the aggregated context features (i.e., the distributed word vectors).
  Later, word2vec by Mikolov et al. [2013b,a] showed that a simplified shallow neural network could be quite proficient in learning distributed word representations.

- **Pre-trained Language Model (PLM)**: With the introduction of more powerful neural network architectures such as bidirectional LSTM (biLSTM) and the highly parallelisable transformer architecture with self-attention mechanism proposed by Vaswani et al. [2017], a new class of models such as ELMo proposed by Peters et al. [2018] became more popular due to their capacity to better capture the context of a word by overcoming the limitation of a fixed word representation. Such models were commonly pre-trained to capture a context-aware word representation and later fine-tuned for any specific downstream task. One extremely popular model that still continues to be used for general-purpose semantic feature extraction tasks is BERT made by Devlin et al. [2018], was specially trained on large-scale unlabelled corpora.

- **Large Language Model (LLM)**: Researchers find that scaling PLM (e.g., scaling model size and data size) often leads to an improved model capacity on downstream tasks (i.e., following the scaling law formulated by Kaplan et al. [2020]). A number of studies have explored the performance limit by training an ever larger PLM (e.g., the 175B-parameter GPT-3 and the 540B parameter PaLM). Although scaling is mainly conducted in model size (with similar architectures and pre-training tasks), these large-sized PLMs display different behaviors from smaller PLMs (e.g., 330M-parameter BERT and 1.5B parameter GPT-2) and show surprising abilities (called emergent abilities by Wei et al. [2022]). In existing literature, there is no formal consensus on the minimum parameter scale for LLMs but Zhao et al. [2023] uses a slightly loose definition of LLMs, and mainly focuses on discussing language models with a model size larger than 10B.

3.1 About Pre-trained Language Models

Since we will be dealing with textual data, we look at the existing PLMs that populate the scene and what are their differences. This helps us guide our choices when selecting the right PLM to use in our own experiments. Zhou et al. [2023] says that there are mainly three branches of PLMs according to the word representations approach and the key factors that distinguish them from one another is their direction of word prediction and the way they use contextual information in a text.

3.1.1 Modelling approaches in PLMs

The differences in the PLMs come from their training approach and this leads to them performing differently and this makes some more suitable for certain downstream tasks than others. We see their key differences in this section.

Autoregressive language modelling

This class of models are usually made and trained for Natural Language Generation (NLG) tasks such as text summarisation and machine translation and they are focused on predicting the next sequence of words based on the preceding word or words in the input. They perform feature extraction on the sequence of text preceding the output to generate the representation of the generated output and usually only work with the context in the backward direction.

Contextual language modelling

Due to the previously stated issue with the directionality of context awareness in The autoregressive LM, alternative methods were proposed such as ELMO by Peters et al. [2018] which combined two LSTMs in a backward and forward configuration such that it allowed the model to gather context of a word from both the preceding as well as the succeeding words at any point in the text sequence.

Permuted language modelling

Permuted LM aims to combine the advantages of the autoregressive LM and the autoencoder LM. It greatly improves upon the flaws of the previous two types of models allowing it to be used as a foundational resource for further tasks. Permuted LM no longer models text as an ordered sequence but rather provide all possible permutations of sequences to maximize the expected logarithmic likelihood of the sequence. This approach allows them to take advantage of the context from all the positions and this strengthens the encoding of text at any point in the sequence. This is further strengthened by the use of bidirectional encoding. A common method for training a permuted language model is through masking. Some key works which use this type of training are ELECTRA, SpanBERT, DeBERTaV3.

3.1.2 Boosting in PLMs

Today, we can see that the PLMs can perform really well in both NLG and Natural Language Understanding (NLU) tasks but they have their caveats. Survey done by Zhao et al. [2023] sheds light on various directions one can take to improve the performance and what has been done in the past. We look at three directions here:

Computational performance

One of the biggest challenges in working with PLMs is the pre-requisite of having a large pre-training dataset which consequently creates a need for a compatible hardware. This makes the task of training,re-training and fine-tuning a model very challenging and computationally demanding. Over the course of years, several efforts have been made to bypass this issue which led to advent of models such as ERNIE Tiny, and Tiny BERT where the models are miniaturised by reducing the number of layers or sometimes the hidden size of layers to increase the prediction speed at the cost of accuracy. A common trend observed in such attempts is the that compression of models usually leads to a sharp decline in their performance.

Multi-task learning

Zhou et al. [2023] point how in recent year, there has been a change in the pre-training paradigm where it has become a good practice to use multiple tasks during to maximally leverage the power of self-attention in transformers. Task embedding is one such technique which uses applies characteristics to a task. Multi-task training was implemented in ERNIE 2.0 where the researchers used continual learning to make the model retain knowledge from previous tasks during the pretraining process thus enabling the model to acquire long-distance memory. This required the model to have two part, which includes a transformer encoder along with task embedding. Such works have made it a common practice to pre-train on multiple tasks simultaneously such that there is a consistent loss function. Modeling in a such a way also allows parameter sharing on multiple tasks and consequently results in a good generalisation ability in performing both NLU and NLG tasks.

Task orientation

In the context of pretraining models, the management of size and compatibility with various downstream tasks is of paramount importance. Zhou et al. [2023] provide several examples of how specialized pretraining models have emerged to address this concern citing works such [Diao et al., 2019; Tsai et al., 2019]. Another such example is the work by Cui et al. [2021] which introduced the BERT-whole word masking model (BERT-WWM), which directly applies BERT to Chinese text with random masking following the original Masked Language Model (MLM) training approach. However, these method have their own caveats.

3.1.3 Word embeddings vs Sentence embeddings

For tasks such as Semantic Textual Similarity(STS), we require to use the information at a sentence level and simple word representations are not enough. We create sentence embeddings for such tasks. Making sentence embeddings is a fundamental problem in natural language processing which requires language models to project sentences into a vector space based on their semantics.

While one can perform STS with older PLMs like BERT, it usually incurred a massive computational overhead when working with sentences due to its architectural limitations. Usually tasks such as semantic similarity and unsupervised sentence clustering required the model to load multiple long text sequences on the network and requires immense computation. One of the first solutions to this problem was proposed in the work by Reimers and Gurevych [2019] where they created two different architectures namely Siamese network and triplet network for creating semantically meaningful sentence embeddings that could be compares using cosine distance

The work of Gao et al. [2021] presented a framework (SimCSE) which showed that the contrastive learning is an effective technique to generate sentence embeddings using BERT in both supervised and unsupervised way. Various other works have further explored the contrastive learning method. For example, DiffSCE done by Chuang et al. [2022] used a replaced token detection loss during its contrastive learning.

Another important work was done by Ni et al. [2021] in creating SentenceT5 (ST5) where they show that encoder-decoder structure such as T5 Raffel et al. [2020], could be used for generating sentence embeddings and further demonstrate that it's capable of improvements with scaling. However, directly using large language models (LLMs) to generate sentence embeddings remains an area of ongoing research.

3.2 Ways of doing PLM evaluation

Its commonly seen that different PLM which are proposed for created a text embedding or other language modelling tasks, are only tested on a small set of datasets with a focus on either a single task or a few. This makes it quite difficult and tedious task to compare one model to another and make an informed choice when doing fine-tuning or doing transfer learning. For example, a model which performs very well in STS might night lend itself to other tasks such as clustering or re-ranking. This gave rise to some benchmarks which have

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Figure 3.1: Description of tasks in GLUE. *Picture from Wang et al. [2018]*

become a standard for evaluation acroos the field and some of the most prominent ones are General Language Understanding Evaluation(GLUE) by Wang et al. [2018], SuperGLUE by Sarlin et al. [2020] and Massive Text Embedding Benchmark(MTEB) made by Muennighoff et al. [2022]. We shall briefly discuss how they perform their evaluation.

3.2.1 GLUE and SuperGLUE

One of the earliest benchmarking tools and resources was proposed in the paper by Wang et al. [2018] which is commonly reffered to as GLUE today. The authors of GLUE pointed out how the evaluation of language models was severely limited by the lack of standard benchmarking tool and a diverse set of datasets for performing a multi-faceted evaluation.

GLUE comes with several tasks and was designed to favor and encourage models that share general linguistic knowledge across tasks. It was also one of the seminal works to prove that multi-task training on all tasks can make a model performs better than training a separate model per task.

GLUE use three direction to evaluate a model's competence by testing its performance when dealing with single sentence tasks, similarity and paraphrasing tasks and lastly with inference tasks.The figure 3.1 provides a quick overview of the tasks.

Wang et al. [2019a] pointed out that the performance of NLP models on the GLUE benchmark has recently surpassed the level of non-expert humans and implied that there is a limited headroom for further research when using it. To overcome this problem they created a new benchmark styled after GLUE but with a new set of more difficult language understanding tasks among other things. Some of these tasks are the same the ones in GLUE but most have been changed to better cater to the evaluation of recent and more capable models.

It must be notes that both these benchmarks were constructed for English language only.

3.2.2 MTEB

Muennighoff et al. [2022] point out how evaluating embedding methods on many tasks re-

Corpus	|Train|	|Dev|	|Test|	Task	Metrics	Text Sources
BoolQ	9427	3270	3245	QA	acc.	Google queries, Wikipedia
CB	250	57	250	NLI	acc./F1	various
COPA	400	100	500	QA	acc.	blogs, photography encyclopedia
MultiRC	5100	953	1800	QA	$F1_a$/EM	various
ReCoRD	101k	10k	10k	QA	F1/EM	news (CNN, Daily Mail)
RTE	2500	278	300	NLI	acc.	news, Wikipedia
WiC	6000	638	1400	WSD	acc.	WordNet, VerbNet, Wiktionary
WSC	554	104	146	coref.	acc.	fiction books

Figure 3.2: Division of tasks and datasets in SuperGLUE. *Picture from Wang et al. [2019a]*

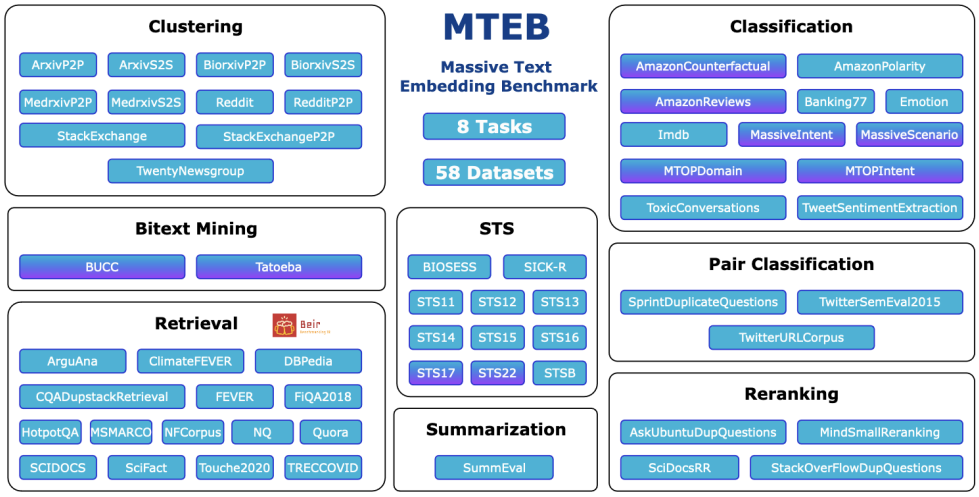

Figure 3.3: Division of tasks and datasets in MTEB. *Picture from Muennighoff et al. [2022]*

quires implementing multiple evaluation pipelines. Implementation details like pre-processing or hyper-parameters may influence the results making it unclear whether performance improvements simply come from a favorable evaluation pipeline. MTEB allows us to have a clear picture of the capabilities of a model on a variety of embedding tasks and thus serves as the gateway to finding universal text embeddings applicable to a variety of tasks. MTEB includes 58 datasets covering 112 languages from 8 embedding tasks: Bi-text mining, classification, clustering, pair classification, re-ranking, retrieval, STS and summarisation as shown in Figure 3.3. Note that MTEB also includes bi-text mining task which allows for evaluation and comparison of multilingual models, something which GLUE and SuperGLUE do not cater to.

3.3 About Graph Neural Networks

As discussed previously in the paradigms for text representation, some NLP tasks can be best addressed by processing text as a graph. There has seen a surge of interests in applying and developing different Graph Neural Network (GNNs) variants for many NLP tasks such

as classification [Henaff et al., 2015; Huang and Carley, 2019], semantic role labeling [Luo and Zhao, 2020; Gui et al., 2019], and relation extraction [Qu et al., 2020; Sahu et al., 2019], to generation tasks like machine translation [Bastings et al., 2017; Beck et al., 2018], question generation [Pan et al., 2020; Sachan et al., 2020], and summarisation [Fernandes et al., 2018; Yasunaga et al., 2017]. These works showed that graph based methods provide a highly expressive system for solving such tasks.

The sentence structural information in text sequence can be utilised to augment original sequence data by incorporating the task-specific knowledge in form of syntactic trees or dependency tree amongst many other approaches for presenting text as a graph. Similarly, the semantic information in sentences can be parsed like through methods like Abstract Meaning Representation (AMR) which further allows us to enhance original sequence data as well. Therefore, these graph structured data can help us encode complicated pairwise relationships between entity tokens for learning more informative representations in the form a a node embedding.

3.3.1 Challenges in using graph based learning algorithms

Despite the successes that the existing research has achieved, deep learning on graphs for NLP still encounters many challenges. In the survey done by Wu et al. [2023], they point the following problems,

- Transforming original text sequences into highly graph-structured data presents a significant challenge in natural language processing (NLP), as most NLP tasks primarily rely on textual input sequences. This makes the automatic construction of graphs from text sequences quite difficult but a crucial preliminary step for leveraging graph neural networks in NLP applications.

- The selection of appropriate techniques for graph representation learning is pivotal in developing tailored Graph Neural Networks (GNNs) capable of capturing the distinct characteristics of various types of graph-structured data, including undirected, directed, multi-relational, and heterogeneous graphs.

- Effectively modeling complex data posits another significant challenge, particularly in NLP tasks that require mapping of entailment between graph-based inputs and other highly structured output data forms, such as sequences, trees, and multi-typed graph data encompassing diverse node and edge types.

3.3.2 Taxonomy of GNNs in NLP

To aid the creation of a standard pipeline for performing graph based machine learning, Wu et al. [2023] provide us with a taxonomy to divide the use of different types of GNNs along four axes as shown in Figure 3.4.

This taxonomy helps us to understand the purpose of a model and what kind of benefits it offers. We go a bit deeper in the details of graph construction methods. We will focus on two major graph construction approaches as show in the taxonomy, namely static graph

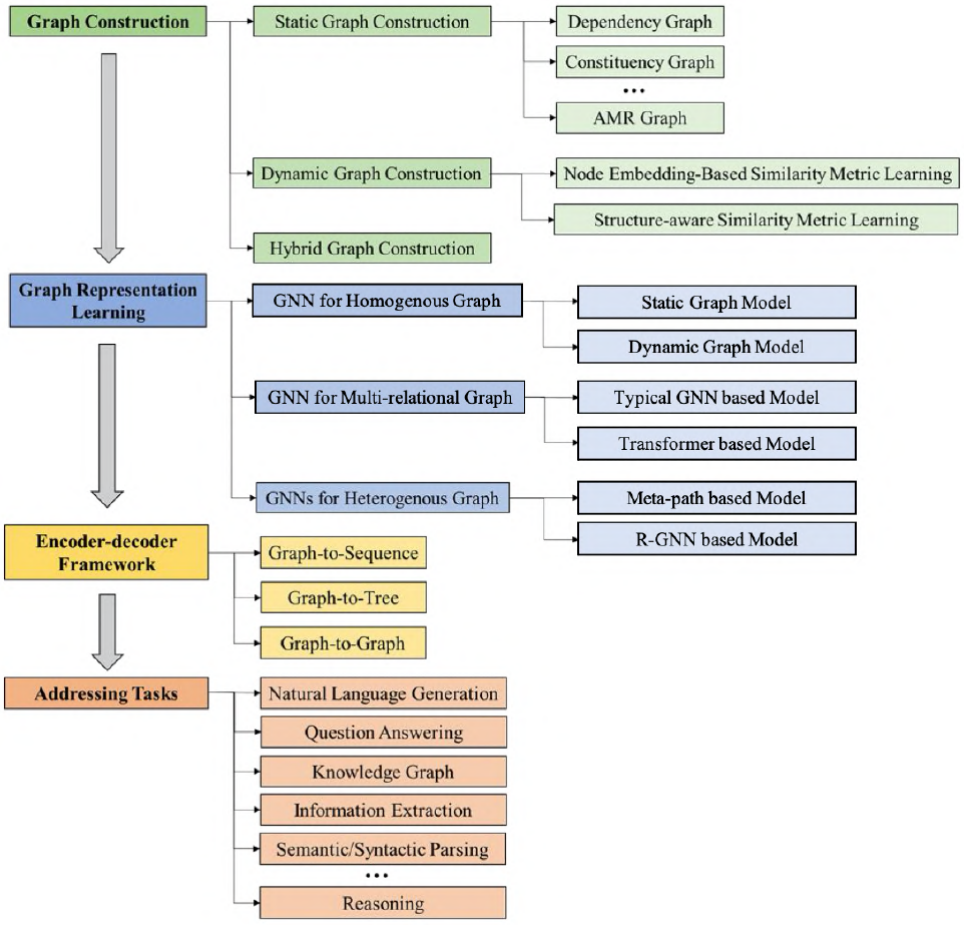

Figure 3.4: Taxonomy of GNNs along Four axes. *Picture from Wu et al. [2023]*

construction and dynamic graph construction for constructing graph structured inputs in various NLP tasks. Conceptually, a static graph incorporates different domain/external knowledge hidden in the original text sequences, which augments the raw text with rich structured information.

Static graph construction

To leverage the power of graph created from a input of a sequence of text, one requires an extensive human effort and domain expertise and the process of graph creation is critical for a successful usage with a GNN. It should also be noted that the manual construction of the graph structure might be error-prone (e.g., noisy or incomplete) and since the graph construction stage and graph representation learning stage are normally disjoint, the errors introduced in the graph construction stage cannot be corrected and can cause error accumulation during the downstream task which can result in degraded performance. The static graph constructed are also limited in their expressivity due to the fact that they only consider one specific relation between nodes. To keep the training simple, such graph are also restricted to only showing one type of relation between the nodes.

Dynamic graph construction

Dynamic graphs are constructed to overcome the limitation of the static graphs talked previously, there is an increasing interest in building a hybrid graph by combing several graphs together in order to enrich the semantic information in graph [Jia et al., 2020; Sahu et al., 2019; Xu et al., 2018; Zeng et al., 2020; Xu et al., 2020; Christopoulou et al., 2019; Yao et al., 2019]. Usually the methods for constructing a hybrid graph is application specific.

Instead of building a fixed static graph based on domain expertise, dynamic graphs jointly train a graph structure learning module together with the graph embedding learning module in order to learn an optimal graph structure considering not only the semantic meanings of the words but also the conversation history and current question. These dynamic graph construction approaches typically consist of a graph similarity metric learning component for learning an adjacency matrix by considering pair-wise node similarity in the embedding space, and a graph sparsification component for extracting a sparse graph from the learned fully-connected graph. In our work we rely upon a dynamic graph construction method.

3.3.3 Graph Similarity Metric Learning Techniques

After the formation of graph from the required data, once can train a GNN with the final objective of creating a graph representation which can subsequently be used for node-level tasks or can even be used for graph-level tasks.

The goal of graph representation learning,similar to word representation learning, is to find a way to incorporate information of graph structures and attributes into a low-dimension embeddings via a machine learning model [Hamilton et al., 2017]. To mathematically formalize this problem, we give the mathematical notations for arbitrary graphs as $G(V, E, T, R)$, where V is the node set, E is the edge set, $T = \{T_1, T_2, \ldots, T_p\}$ is the collection of node types, and $R = \{R_1, \ldots, R_q\}$ is the collection of edge types. Moving forward we can discuss similarity as metric for learning the graph representation that can then be used the constructed graphs for various NLP tasks. This method has an underlying assumption that node attributes contain useful information for learning the implicit graph structure and this had led to recent works which have explored the idea of looking at graph structure learning problem as the problem of similarity metric learning defined upon the node embedding vector space. These learned similarity metric function are later applied to an unseen set of node embeddings to infer a graph structure, thus enabling inductive graph structure learning.

Wu et al. [2023] show that various similarity metric functions exist for graph structure learning for GNNs and depending upon the types of information sources utilized, they can be grouped into two categories:

1. **Node Embedding Based Similarity Metric** are those functions which are devised to acquire a weighted adjacency matrix by evaluating the pairwise node similarity within the embedding space. Typical metric functions utilized for this purpose encompass attention-based and cosine-based metric functions.

2. **Structure-aware Similarity Metric** learning approaches are inspired by structure-aware transformers which put emphasis on the existing relations and its intrinsic information so as to be not limited by just the information from the nodes.

3.3.4 Evaluation methods for graph based methods

The choice of evaluation methods used in graph representation is directly related to the task for which they are being made and trained on. Its common to do node classification and clustering to check how well a model can capture the features from the data. Another important task commonly performed is Link prediction. To measure the performance of node clustering methods, we can employ two metrics:

Normalized Mutual Information (NMI)

It's a measure used to quantify the similarity between two sets of labels assigned to the same set of data points which makes it suitable for clustering algorithms where its required to evaluate the agreement between the clustering results and the ground truth labels.

To understand NMI we can look at its components:

Mutual Information (MI): Mutual Information looks at the common information between two variables an in the context of clustering, the variables can be the ground truth labels and the clustering results produced by the model. The Mutual Information between two sets of labels XX and YY is calculated as:

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

Here, $p(x,y)$ is the joint probability of x and y, and $p(x)$ and $p(y)$ are the marginal probabilities of x and y respectively.

Normalization: Normalized Mutual Information (NMI) Mutual information needs to be normalised to be applicable and comparable across the dataset and at different scales. This normalisation is performed as :

$$NMI(X,Y) = \frac{MI(X,Y)}{\sqrt{H(X)H(Y)}}$$

where $MI(X,Y)$ is the Mutual Information and $H(X)$ and $H(Y)$ are the entropies of X and Y respectively. Entropy measures the uncertainty or disorder of a random variable and is defined as:

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

$H(X)$ and $H(Y)$ represent the amount of information inherent in the ground truth labels and the clustering results. The NMI score ranges between 0 and 1, where 0 indicates no agreement between the sets of labels, and 1 indicates perfect agreement. Higher NMI scores assures us that the learning algorithm has not strated away from the ground truth in creating its representations. Its also advantageous because it accounts for both the clustering purity and the entropy of the clusters. It penalizes the clustering results that partition the data into many small clusters or merge distinct clusters, thus providing a more nuanced evaluation of clustering quality compared to metrics like accuracy.

Adjusted Rand Index (ARI)

This measure is useful when dealing with clustering results and we need to evaluate similarity while considering the chance agreement. This measure takes the ground truth labels to check the similarity between the different clustering outputs provided to it. It can be calculated the following way :

Here's how the Adjusted Rand Index is calculated:
True Positive (TP): The number of pairs of elements that are in the same cluster in both the predicted and true clustering.
True Negative (TN): The number of pairs of elements that are in different clusters in both the predicted and true clustering.
False Positive (FP): The number of pairs of elements that are in the same cluster in the predicted clustering but in different clusters in the true clustering.
False Negative (FN): The number of pairs of elements that are in different clusters in the predicted clustering but in the same cluster in the true clustering.

The Adjusted Rand Index is then computed using these quantities:

$$ARI = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

The ARI ranges from -1 to 1: ARI = 1: Perfect agreement between the two clustering results. ARI = 0: Expected value of the index when the clustering results are completely random. ARI = -1: Indicates complete disagreement between the clustering results.

The Adjusted Rand Index adjusts for chance agreement that can occur due to the stochasic nature of machine learning algorithms by considering how much better is the clustering results compared to what is possible if the result was just produced with random chance. It can be used with any number of clusters and size of the dataset but it can be sensitive to noise and outliers in data and should be used with caution in situations where the ground truth label is not available or unreliable.

Area Under Curve (AUC)

It is a commonly used metric in machine learning and statistics and it deals with the Receiver Operating Characteristic (ROC) curve at varying threshold values produced in the context of binary classification problems but also can be applied to multi-class classification problems.

We need to understand ROC before going to AUC :

Receiver Operating Characteristic (ROC) Curve: The ROC curve is a graphical representation of the performance to illustrate the variance produced by threshold changes and plots the true positive rate (TPR), often also mentioned as sensitivity or recall, against the false positive rate (FPR).

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

The ROC curve typically plots TPR on the y-axis and FPR on the x-axis, with each point on the curve representing a different threshold setting.

Area Under the Curve (AUC): The AUC quantifies the overall performance of a binary classification model across all possible threshold settings. It represents the probability that the classifier will be able to to assign a higher ranking of similarity to randomly chosen positive instance as compared to a randomly chosen negative instance.

The AUC ranges from 0 to 1, where: $AUC = 0.5$: Indicates that the model performs no better than random guessing.
$AUC > 0.5$: Indicates that the model performs better than random guessing, with higher values indicating better performance.
$AUC = 1$: Indicates ideal classification.

AUC is a useful metric because it is threshold-independent, meaning it is agnostic to the specific threshold chosen for doing classification. Its singular output which depicts the effectiveness of the model's performance across all possible thresholds.

Average Precision (AP)

It is a metric commonly used to evaluate the performance of information retrieval systems, binary classifiers, and ranking algorithms. It is widely used in scenarios where the focus is on achieving high precision and recall scores.

Average Precision can be calculated using the following metrics :
Precision: Precision measures the proportion of relevant instances among the retrieved instances. It is the ratio between the number of actual true positives and the sum of the number of predicted instances which includes both true positives and false positives.

$$\text{Precision} = \frac{\text{Correct Predictions}}{\text{All Predicted Instances}}$$

Recall: Recall measures the proportion of relevant instances that have been retrieved over the total amount of relevant instances in the dataset. It is calculated as the ratio between the number of true positives and the sum of relevant instances which includes both true positives and false negatives.

$$\text{Recall} = \frac{\text{Correct Predictions}}{\text{All Possible Correct Predictions}}$$

Precision-Recall Curve: It is usually desired that both precision and recall scores are high but precision pushes the model towards a higher confidence threshold for making predictions while recall pushes the model to work with a smaller confidence threshold to avoid missing true positives and this requires a trade-offs. The Precision-Recall (PR) curve is a plot of precision values against corresponding recall values for different threshold settings of a binary classifier or ranking algorithm.

Average Precision (AP): Average Precision calculates the area under the Precision-

Recall curve. It summarizes the precision-recall trade-off and provides a single scalar value that reflects the overall performance of the classifier or ranking algorithm. AP is computed by calculating the area under the precision-recall curve using numerical integration or the sum of trapezoids under the curve.

Interpretation: AP ranges from 0 to 1, where a higher value indicates better performance. An AP of 1 indicates perfect precision and recall, meaning that all retrieved instances are relevant and all relevant instances are retrieved. An AP of 0 indicates poor performance, where either precision or recall is zero.

Average Precision vs. Area Under the Curve (AUC): Average Precision focuses on the precision-recall curve and is sensitive to changes in recall at low recall values. Area Under the Curve (AUC), on the other hand, focuses on the Receiver Operating Characteristic (ROC) curve and is sensitive to changes in false positive rate. For all the metrics, a higher value indicates better performance.

4. AGE + ELECTRA : A simple Method for frame disambiguation and labeling

4.1 Introducing the model

As mentioned in the Background chapter, there have been numerous works done in the past to tackle and we choose one of the latest work which proposed a relatively small deep learning model for Frame Identification (FI), based on pre-trained text encoders trained discriminatively (ELECTRA) and graphs embedding (Adaptive Graph Encoder (AGE)), producing state of the art performance. We systematically compare a diverse set of pretrained language models to discern their impact on the pipeline's performance. The comprehensive analysis sheds light on the nuanced interplay between different embedding from a pre-trained language model and the AGE component's ability to capture intricate frame-semantic relationships. This work contributes insights into the model's behavior and aims to make it more explainable and opens avenues for optimizing performance based on the selection of pre-trained language models.

4.2 Architecture of Model

The past works dealing with Frame Identification (FI) task have described it in this way where given a sentence with n words, which also contained a highlighted predicate t which is responsible for evoking a Frame which sets the semantic tone for that sentence, given that the ground truth label of the target is available the task is seen as a classification task over the entire set of defined and annotated frames provided by the selected version of FrameNet. Popular works such as those done by [Das et al., 2014; Swayamdipta et al., 2017] can be looked at for an elaborate description of this task.

Tamburini [2022] first extract frame embeddings $D_{\text{embs}} = [d_1, \ldots, d_k] \in \mathbb{R}^{k \times d}$ from FrameNet frame definitions $D = \{d_1, \ldots, d_k\}$, with d_i the definition of f_i, by using the Sentence-BERT model [Reimers and Gurevych, 2019], for all the k frames in a given FN version and uses them as starting node features for the AGE model. d represents the dimension of Sentence-BERT embeddings. Then, he extract the FrameNet subgraph $\mathcal{FNG}$ projected by the Inheritance, Subframe and Using frame relations from FN, as done by Popov and Sikos [2019], and uses them as graph-structure input in the AGE model for obtaining the

structurally-informed frame embeddings $AGE_{\text{embs}} = [f_1, \ldots, f_k] \in \mathbb{R}^{k \times y}$, where y is the AGE output embedding dimension. These embeddings are then used by the classifier to do the final classification.

The Figure 4.1 Depicts the architecture.

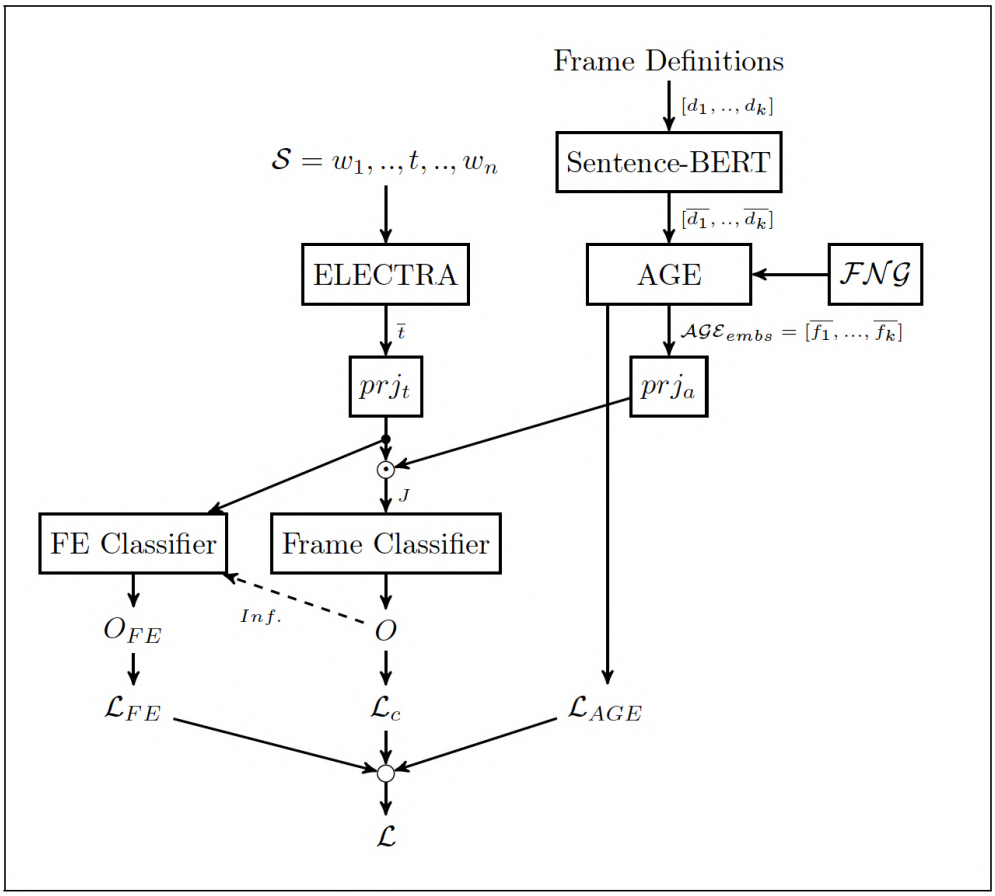

Figure 4.1: The various modules comprising the full model for Frame Identification. *Picture from Tamburini [2022]*

4.2.1 PLM Module

As depicted in Figure 4.1 the FI system makes use of two different sets of embeddings from the FN dataset. The first set of embeddings are word embeddings for the target word in the annotated corpus which is responsible for Frame evocation. The second set of embeddings are made from the definition of Frames in the selected version of FN being used for performing the tasks. Both these embeddings are made by using different PLMs. We shall discuss how they are made and later used in the model and what were the original PLMs used for making them.

Target word embeddings

Since a single lexical unit has the potential to evoke different frames when used in different ways. We illustrate this point in below inside boxed text where the word "Leaving" can evoke three different frames namely "Departing","Causation" and "Abandonment", which makes it a critical job to create a contextual aware word embedding for these target words.

We hurried down the village street and found, as we had expected, that [the inspector]$_{Theme}$ was [just]$_{Time}$ **LEAVING**$_{[\text{DEPARTING}]}$ [his lodgings]$_{Source}$.
Mysteriously, the Anasazi vanished from the valley around a.d. 1150, **LEAVING**$_{[\text{CAUSATION}]}$ [it]$_{Affected}$ [to be repopulated by the Southern Paiutes, another hunter-gatherer tribe]$_{Effect}$.
True to wild-West stereotypes, Stewart was slain by a neighboring farmer, **LEAVING**$_{[\text{ABANDONMENT}]}$ [his strong-willed wife, Helen]$_{Theme}$,[to assume the duties of the ranch]$_{Explanation}$.

The entire sentence is firstly transformed into a graph with edges reflecting the dependency of the words to each other. This graph is then passed to Mapper-one in the pipeline which makes word embeddings for each word. These embedding are stored in dictionary with their position as key. These are then used at later stage during the training and inference by the classifier. The original model utilised pretrained ELECTRA$_{small}$ for making the word embeddings for the target words. We experiment with various other pretrained LMs released more recently. The detailed description of this stage is provided in the data-processing section later in this chapter.

Sentence embeddings for the Frame Definitions.

For all the frames defined in FN, there exists an annotated definition which highlights the target word/s as well as the core FEs. In figure 4.2 we have shown the the definition of the *Abandonment* frame. It becomes clear when we look at the format of the defintion that it lends itself quite well for making semnatically rich embeddings for the frames themselve but treating the sentence embedding of the Frame's defintion as it's representation. Since these defintions also contain the core FE and some of the typical lexical units which evoke that frame, they also provide contextual information.

The results produced by the model in the work done by Tamburini [2022] performed the task of making the sentence embeddings with SentenceBERT [Reimers and Gurevych, 2019]. In our own experiments we test the performance of the model with sentence embeddings made from other PLMs. The step by step process of making sentence embeddings from definitions is explained in the later sections in this chapter.

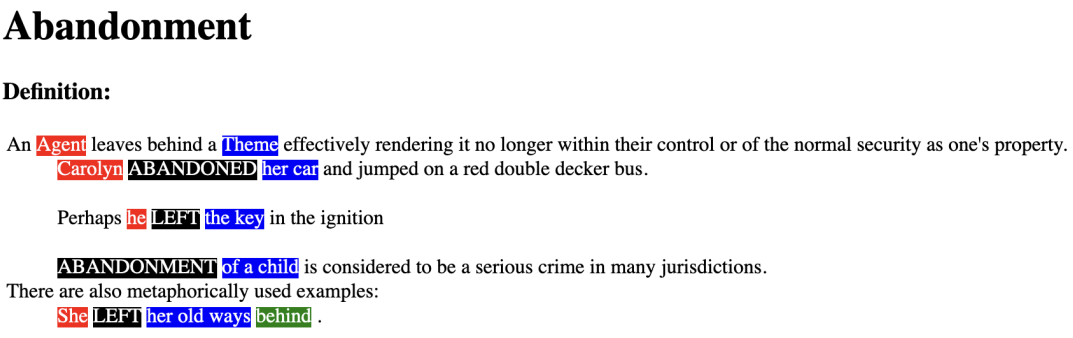

Figure 4.2: Annotated defintion of the `Abandonment` Frame

4.2.2 AGE Module

Drawbacks of GCNs

Since the Architecture of the system utilises a graph embedding for performing FI, the choice of a good graph representation maker becomes critical. Most of existing GCN-based Network Embedding methods are based on graph autoencoder (GAE) and variational graph autoencoder (VGAE) [Kipf and Welling, 2016] and are made up of a Graph Convolutional Netowork (GCN) encoder and a decoder for reconstructing the original graph. Cui et al. [2020] points our three flaws in their architecture, as shown in Fig 4.3.

First issue is caused by the convolutional filter (H in Fig. 4.3). The author cites the work of Wu et al. [2019] who showed that the entanglement of the filters and weight matrices during the training do not performance gain for semi-supervised graph representation learning, and even harms training efficiency since it deepens the paths of backpropagation due to the fact that a convolutional network usually consists of multiple layers of filters.

Second issue points that previous research done by Li et al. [2018] shows in theory the convolutional filter used in the GCN are actually Laplacian smoothing filters [Taubin, 1995] applied on the feature matrix for low-pass denoising. Cui et al. [2020] further show that existing graph convolutional filters are not optimal for acting as a low-pass filters since they cannot filter out noises in some high-frequency intervals and fail to provide the best smoothing effect.

At last they also argue that the training objectives for these algorithms such as reconstructing the adjacency matrix [Pan et al., 2018; Wang et al., 2019b] or feature matrix [Park et al., 2019; Wang et al., 2017] cannot be used as it is for real world applications since the process treats the recreated adjacency matrix as the ground truth for calculating the similarity of nodes which is not correct due to lack of feature information. Further they state that implementing methods for Recovering the original feature matrix can will force the model to remember high-frequency noises in features which can give rise to a different complication.

Motivated by such observations, Cui et al. [2020] proposed Adaptive Graph Encoder (AGE), a unified framework for attributed graph embedding in this chapter. Their solution disentangle the filters and weight matrices and can be seen as two modules:

1. A nonparametric **Laplacian smoothing filter** to efficiently extract smooth features

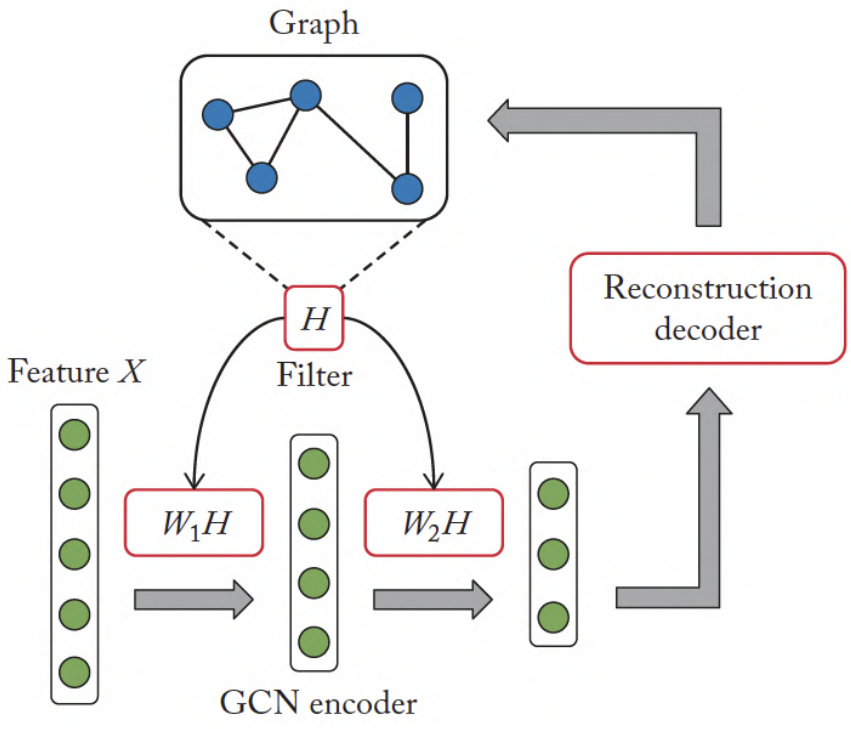

Figure 4.3: Issues with the architecture of graph auto-encoders highlighted in red box. *Picture from Cui et al. [2020]*

for the nodes.

2. An **adaptive encoder** for the purpose of learning better representative node embeddings from the graphs.

Key parts of AGE

The Figure 4.4 depicts the AGE framework and it can be broadly broken down into two key parts - The Laplacian filter and the Adaptive Encoder. Cui et al. [2020] provide us detailed explanation of how they work and in the end they provide a general algorithm provided for implementing it.

1. **Laplacian Filter** : This filter is implemented to have a smooth node features on the graph manifold in such a way that it preserves the low-frequency component which could be useful for calculation similarities and differentiation the nodes from one another. Their choice of using a Laplacian filter for this task was also motivated by the high computational efficiency provided by them. Hence, to achieve low-pass filtering, the frequency response function $1 - k\lambda$ should be a decrement and non-negative function. Stacking up t Laplacian smoothing filters, we denote the filtered feature matrix $\tilde{X}$ as

$$\tilde{X} = H^t X. \qquad (eq - 0.1)$$

Note that the filter is non-parametric at all.

Choice of K: In practice, with the renormalisation trick $\tilde{A} = I + A$, they employ the symmetric normalized graph Laplacian

$$\tilde{L}_{\text{sym}} = \tilde{D}^{-\frac{1}{2}} \tilde{L} \tilde{D}^{-\frac{1}{2}} \qquad (eq - 0.2)$$

, where $\tilde{D}$ and $\tilde{L}$ are degree matrix and Laplacian matrix corresponding to $\tilde{A}$. Then the filter becomes

$$H = I - k\tilde{L}_{\text{sym}} \qquad (eq - 0.3)$$

. Note that they set k = 1, which makes the filter become like a GCN filter. However, Cui et al. [2020] also provide us with an optimal value to use for λ_{max} based on their experiments with various graph datasets.

2. **Adaptive Encoder** : Filtered by t-layer Laplacian smoothing, they create output features which are smoother and preserve abundant attribute information. To learn better node embeddings from the smoothed features, they choose pairwise similarity learning task as their unsupervised optimisation task which was inspired by Deep Adaptive Learning proposed by Chang et al. [2017]. For the attributed graph embedding task, the relationship between two nodes is crucial, which requires the training targets to be suitable similarity measurements. GAE-based methods usually choose the adjacency matrix as true labels of node pairs. However, they argue that the adjacency matrix only records one-hop structure information, which is insufficient. Meanwhile, they state that the similarity of smoothed features or trained embeddings are more accurate since they incorporate structure and features together. Due to these reasons, they decide to

adaptively select node pairs of high similarity as positive training samples, while those of low similarity as negative samples.

Training sample selection :After calculating the similarity matrix, they rank the pairwise similarity sequence in the descending order. They represent the rank of node pair (v_i, v_j) as r_{ij}. Then they set the maximum rank of positive samples as r_{pos} and the minimum rank of negative samples as r_{neg}. Therefore, the generated label of node pair (v_i, v_j) is

$$l_{\text{ij}} = \begin{cases} 1 & \text{if } r_{ij} \leq r_{\text{pos}} \\ 0 & \text{if } r_{ij} > r_{\text{neg}} \\ \text{None} & \text{otherwise} \end{cases} \qquad \text{(eq - 1)}$$

In this way, a training set with r_{pos} positive samples and $n^2 - r_{\text{ne}}$ negative samples is constructed. Specially, for the first time they construct the training set, since the encoder is not yet trained, then they directly employ the smoothed features for initializing S:

$$\mathbf{S} = \frac{\tilde{\mathbf{X}}\tilde{\mathbf{X}}^{\top}}{\left\|\tilde{\mathbf{X}}\right\|_2^2} \qquad \text{(eq - 2)}$$

After construction of the training set, they train the encoder in a supervised manner. As graphs depicting real-world relations and entities are almost always unbalanced in the proportion of similar to dissimilar nodes, calculating similarities directly on the entire graph can be skewed. They solve this by selecting more than r_{pos} negative samples from the training set and then they balance positive/negative samples by randomly choosing r_{pos} negative samples in every epoch. They denoted this balanced training set as O. Accordingly, they provide the cross entropy loss as following -

$$L = \sum_{(v_i, v_j) \in O} \left(-l_{ij} \log(s_{ij}) - (1 - l_{ij}) \log(1 - s_{ij})\right). \qquad \text{(eq - 3)}$$

Threshold update : Inspired by the work on curriculum learning done by Bengio et al. [2009], the authors of AGE also set a threshold for controlling the size of the training set by implementing a strategy which updates r_{pos} and r_{neg}. At the beginning of training process, they use more samples for the encoder to find rough cluster patterns. Subsequently, samples with higher confidence are utilised for training, forcing the encoder to capture refined patterns. In practice, r_{pos} decreases while r^{st}_{pos} increases linearly as the training procedure goes on. They set the initial threshold as r^{st}_{pos} and r^{st}_{neg}, together with the final threshold as r^{ed}_{pos} and r^{ed}_{neg}. This gives them $r^{ed}_{\text{pos}} \leq r^{st}_{\text{pos}}$ and $r^{ed}_{\text{neg}} \geq r^{st}_{\text{neg}}$. They mathematically describe their threshold update stratergy as following considering the update happens T times,

$$r'_{\text{pos}} = r_{\text{pos}} + \frac{r^{ed}_{\text{pos}} - r^{st}_{\text{pos}}}{T} \qquad \text{(eq - 4)}$$

$$r'_{\text{neg}} = r_{\text{neg}} + \frac{r^{ed}_{\text{neg}} - r^{st}_{\text{neg}}}{T} \qquad \text{(eq - 5)}$$

They inform that as the training process goes on, every time the thresholds are updated, they reconstruct the training set and save the embeddings. For node clustering, they perform Spectral Clustering [Ng et al., 2001] on the similarity matrices of saved embeddings, and select the best epoch by Davies-Bouldin index [Davies and Bouldin, 1979] (DBI), which measures the clustering quality without label information. For link prediction, they select the best performed epoch on validation set. Algorithm 1 presents the overall procedure of implementing the training of the Adaptive encoder

Algorithm 1 Adaptive Graph Encoder as shown by Cui et al. [2020]

Require: Adjacency matrix A, node feature matrix X, filter layer count t, iteration number max_iter, and threshold update number T

Ensure: Node embedding matrix Z

Obtain graph Laplacian $\tilde{L}_{\text{sym}}$ from Eq. (0.2);
$k \leftarrow 1/\lambda_{\max}$;
Get filter matrix H from Eq. (0.3);
Get smoothed feature matrix $\tilde{X}$ from Eq. (0.1);
Initialize similarity matrix S and training set O;
for $iter = 1$ to max_iter **do**
    Compute Z;
    Train the adaptive encoder with loss in Eq. (3);
    if $iter \mod (max\_iter/T) == 0$ **then**
        Update thresholds with Eq. (4) and (5);
        Calculate the similarity matrix S;
        Select training samples from S by Eq. (1);
    end if
end for

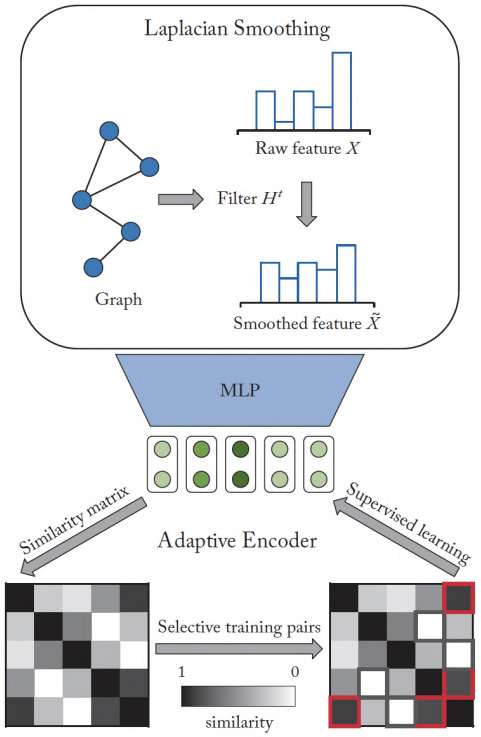

Figure 4.4: AGE framework and its modules. *Picture from Cui et al. [2020]*

4.2.3 Classifier

The model created by Tamburini [2022] taken an input sentence S and processed it using one contextual embedding technique based on transformers [Vaswani et al., 2017] which provides him with the contextualized word embeddings for each word in S which also accounts for the target word t and he accounts for multi-word expression targets by calculating the average over the corresponding embeddings. In his work, he relied on ELECTRA, which is created by Clark et al. [2020], which is a text encoder pre-trained with a more sample-efficient task called 'replaced token detection' as a discriminator rather than a generator in a set-up commonly found in generative adversarial networks. Further, he combines AGE and target embeddings by projecting them into a shared joint target-label vector space of dimension h with the help of two linear layers with bias which are referred to as prj_a, and prj_b. Later he performs a component-wise multiplication between prj_t which is the projection of the target word embedding and the previously created vector space. He provides the mathematical expression to do as follows:

$$J = [prj_a(f_1) \odot prj_t(t), \ldots, prj_a(f_k) \odot prj_t(t)] \in \mathbb{R}^{k \times h}$$

This allows the model to measure the compatibility between the target and each possible frame as done in Pappas and Henderson [2019], which provides him with the final probabilities that can be used for the classification task over all possible frames they had computed.

To further enhance the expressivity of the trained classifier, Tamburini [2022] also imple-

ments simultaneous training on a multi-labeling task for predicting the Frame Elements(FE) involved in a specific frame invocation from the predicted target. This is accomplished by further adding a linear layer with bias P.

Layers of Classifier and Weight initialisation

The whole system does not require a very big model consisting of only 1.6 million parameters. We shall briefly look at the individual layers of the classifier and the how they work together.

During the implementation of the model architecture, Tamburini [2022] used the **kaiming-uniform** method created by He et al. [2015] for setting the starting weights of the network before the training start. This method of weight initialization, often referred to as the Kaiming initialization or He initialization, is widely used in machine learning for initializing weights in deep neural networks. It initializes the weights of the neural network layers with values drawn from a uniform distribution, centered around zero, while adjusting the scale of the distribution based on the number of input and output units. The Kaiming initialization was proposed specifically designed for rectified linear unit (ReLU) activation functions and it addresses the vanishing and exploding gradient problems commonly encountered during the training of deep neural networks by initializing the weights such that the variance of the activations remains relatively constant across different layers. By ensuring that the gradients neither vanish nor explode during backpropagation, the Kaiming initialization facilitates more stable and efficient training of deep neural networks. This method has become a standard practice in machine learning due to its ability to improve the convergence speed and overall performance of deep learning models, particularly those utilizing ReLU activations, which are widely used in modern neural network architectures.

4.2.4 Optimisation, Losses, and Inferences

All deep learning models require a loss function and an optimiser for training. The loss function is responsible for measuring how accurate is the model in performing its task and gives a quantitative value that can be used for adjusting the model's parameters to achieve better results. The choice of loss funcion is also related to the kind of task the model is being trained on. But a loss function cannot adjust the weights the model's parameters, this job is done by the optimiser which takes the loss value from the loss function and uses it to calculate the new set of parameter weights for the model to use. We shall briefly describe the optimser and the loss functions being used by our selected model.

Selected Optimizer

The FI system uses **AdamW** [Loshchilov and Hutter, 2017] as its optimiser. It is an extension of the Adam optimizer, which introduces a weight decay term to the optimization process. The key idea behind AdamW is to address the issue of weight decay not being effectively applied to the weights updated by adaptive optimisers like Adam. By decoupling weight decay from the parameters' update steps, AdamW ensures that the regularization

term contributes to the gradient descent process more consistently, helping prevent overfitting and improving generalization performance. The benefit of AdamW lies in its ability to effectively handle weight decay regularization while maintaining the advantages of adaptive learning rates offered by optimisers like Adam, thereby enhancing the stability and convergence speed of deep learning models during training. We optimize our model by using the AdamW algorithm with respect to three losses:

Loss Functions and Combined Loss

(a) the standard loss for link prediction proposed by Cui et al. [2020], L_{AGE}, for the AGE model - binarycrossentropy with logits;

(b) the **cross-entropy loss**, L_c, for the classifier output O. Cross-entropy loss is a common loss function used in classification tasks, particularly in scenarios involving multiple classes. It calculates how different is the actual distribution of labesl from what the model is predicting. The key aspect of cross-entropy loss lie in its ability to effectively penalize models for making inaccurate predictions while encouraging them to output well-calibrated probabilities. By optimizing cross-entropy loss, models learn to assign higher probabilities to the correct classes and lower probabilities to the incorrect ones, leading to improved classification performance. Additionally, cross-entropy loss is differentiable, making it suitable for gradient-based optimization algorithms commonly used in training neural networks. Its robustness and effectiveness make cross-entropy loss a widely used choice for classification tasks across various domains.

(c) the **binary cross-entropy with logits loss**, L_{FE}, for the multi-label FE classifier output O_{FE}. Binary cross-entropy loss, also known as logistic loss or log loss, is a loss function used in binary classification tasks, where each instance belongs to one of two classes (positive or negative). It measures the discrepancy between the predicted probability distribution and the actual binary labels. Binary cross-entropy loss is particularly suited for multi-label classification tasks because it provides a robust way to handle instances that can belong to multiple classes simultaneously. By treating each class independently and applying the sigmoid activation function to obtain probabilities for each class, binary cross-entropy loss allows the model to learn the likelihood of each class being present in the input instance. It penalizes the model based on the difference between the predicted probabilities and the actual binary labels, encouraging the model to output high probabilities for the correct labels and low probabilities for the incorrect labels, ultimately leading to more accurate predictions in multi-label classification scenarios.

Tamburini [2022] combine these losses and optimises the model by using the joint loss

$$L = \gamma_2 \left[\gamma_1 L_c + (1 - \gamma_1) L_{AGE}\right] + (1 - \gamma_2) L_{FE}$$

with $\gamma_1, \gamma_2 \in [0, 1]$. We can look at the individual factors used in the combined loss function:

- The term γ_1 determines the relative importance of the classification loss (loss_c) compared to the embedding loss (loss_e).

- When $\gamma_1 = 1$ and $\gamma_2 = 0$, the combined loss is dominated by the classification loss, prioritizing accurate classification over embedding quality.
- Conversely, when $\gamma_1 = 0$ and $\gamma_2 = 1$, the combined loss is dominated by the embedding loss, prioritizing the quality of the learned embeddings over classification accuracy.
- The term γ_2 controls the contribution of the combined loss to the final loss function. When $\gamma_2 = 1$, only the combined loss is considered, while $\gamma_2 = 0$ ignores the combined loss entirely.

By adjusting γ_1 and γ_2, you can trade-off between classification accuracy and embedding quality based on the specific requirements of your model and the task at hand.

Things to keep in mind when changing the values of γ_1 and γ_2:

1. **Balance between Loss Components**: Ensure that the sum of γ_1 and γ_2 is equal to 1 to maintain a balanced contribution between the classification and embedding losses.
2. **Effect on Model Behavior**: Changing the values of γ_1 and γ_2 will influence the behavior of the model during training. Consider how these changes affect the model's convergence, generalization performance, and ability to learn meaningful embeddings.
3. **Task Requirements**: Adjust γ_1 and γ_2 based on the specific requirements of your task. For instance, if classification accuracy is more critical, prioritize the classification loss by setting γ_1 closer to 1. Conversely, if the embedding quality is paramount, prioritize the embedding loss by adjusting γ_2 accordingly.

Overall, careful tuning of γ_1 and γ_2 is essential to strike the right balance between classification accuracy and embedding quality, ultimately leading to improved model performance.

Doing Inference

During inference on output frames, the model narrows down the range of potential frames that could be potential outputs by checking the potentiality of the frames to evoke the target word concerned. Similarly, when making decisions about Frame Elements (FEs) classification, Tamburini [2022]'s algorithm narrows down the possible FEs to those that could belong to any frame potentially triggered by the target word. This is achieved by setting a threshold at 0.0 on the logits that determine the final decision regarding the FEs involved in a specific instance of the problem.

4.3 Data processing steps

4.3.1 Making graph from text

Since the system is made as such that it requires embeddings to train and later do the inference, we perform text processing to transform the FN data into a format suitable for the training. One of the first things to do after selecting the FN version to work with, we choose the appropriate text and make create a dependency graph which contains the semantic and

syntactic information provided in the data. This graph is then used by the PLM to create word embeddings for the target words. We show the implementation overview in Algorithm 2 below.

Algorithm 2 Text to graph generator

Require: Annotated Corpus A, Mapper-One Mpr_1, Get-graph function t, PLM path max_iter

Ensure: Correct root path is set for Resource Manager R_{mngr}

1: make R_{mngr} object with the correct root path;
2: make PLM object with the selected plm-path;
3: make Mpr_1 object with the selected PLM object;
4: **for** $data - split$ in $corpus$ **do**
5: Fetch the data using R_{mngr}
6: store the data from the data-split
7: make graph from the split using the Get-graph function and store them in a list
8: Pass the PLM-name,Split-list,root-path and split-type to the $Mpr_1.(saveBERT)$ method to create the text embeddings and save them. Details of this method provided in Algorithm 4;
9: **end for**

4.3.2 Making graph of frame relations

As discussed previously, FrameNet has an intricate relation system for expressing Frame-to-Frame as well as FrameElement-to-FrameElement relations and this allows us to leverage its structural information and Frame definition to build graph representation embeddings. Figure 4.5 shows a limited portion of the network of the **DEPARTING** frame. It's clear from the example figure that we need to store the graph data as a matrix before passing it to AGE module for making the graph embedding.

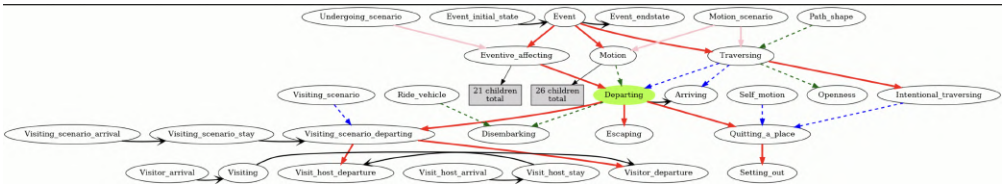

Figure 4.5: Frame relations in FN

Since the AGE module of the pipeline requires the graph with the frame embeddings for creating the graph embedding, we utilise the Frame-to-Frame information provided for all the types of relations we are interested and the Frame definition embeddings in our selected FN version and create an attributed graph using this data.

Algorithm 3 Graph relation Network for AGE

Require: file with Frame-Frame relations for all relation types A, Frame Definitions and Frame Defintion embeddings.

Ensure: Nomenclature of the files after the processing should be according to how the AGE accepts the file names.

- making graph from relations
- load the right FN-v definitions
- add definition to graph
- restructure as per AGE req
- store the pickle

for $iter = 1$ to max_iter **do**
  Compute Z with Eq. (11);
  if $iter \mod (max\_iter/T) == 0$ **then**
    Update thresholds with Eq. (16) and (17);
  end if
end for

4.3.3 Making target word embedding

After obtaining the graph out of the example sentences in the FN data, we pass it to the Mapper-one as show in Algorithm 2. This Mapper is responsible for making the target word embeddings and storing them in a suitable format for the FI system to use later during its training stage. We show the implementation overview of this step in Algorithm 4 below.

In practice, we rely upon the hidden states of the PLM, such as those produced by transformers, to derive word embeddings through a process commonly referred to as contextual embedding. These **hidden states** capture contextual information about each token in a given sequence, representing the learned representations of the input text. Typically, the **last hidden state** of the model, often corresponding to the final layer or token, is used to generate word embeddings. However, in certain scenarios, such as in tasks requiring deeper contextual understanding or in applications where longer-term dependencies are crucial, using all or part of hidden states can provide richer representations.

The last hidden state is not always the same as the other hidden states due to the architecture of the language model. Each layer of the model captures different levels of abstraction and context, with earlier layers focusing on local syntactic structures while deeper layers capture more abstract semantic information. Consequently, the last hidden state may contain higher-level semantic representations, while earlier hidden states may capture more fine-grained syntactic details.

The decision of whether to use the last hidden state or all hidden states depends on the specific requirements of the downstream task. In our tasks, which requires more nuanced understanding and where capturing long-range dependencies is critical, utilizing all hidden states yields better results. Ultimately, we decided to work with the average of the last four hidden states to create our word embeddings since it proved to work better than only using the last hidden state.

Algorithm 4 Target word embedding generator

Require: Dependency graph of example sentences *sentList* from the annotated corpus of FN using Algorithm 2.

Ensure: the example sentences graphs are in a list.

make dictionary object *dictAll* for storing the word embeddings for each sentence.
for *sent* in *len*(*sentList*) **do**
  split *sent* into words using and store the words in list *wrds*;
  for *w* in *wrds* **do**
    add special token [CLS] to the start of *wrds*
    tokenise *wrds* using the tokeniser of the selected PLM;
    truncate the tokenised output at 511 and add [SEP] token to this new list *tknList*;
    add padding to the *tknList* as required;
    create attention mask *attn* for *tknList*;
    make the inputIDs *inpIDs* for the words in *tknList*;
    convert $inpIDs, attn$ into tensors;
    pass the $inpIDs, attn$ tensors to the PLM to get its hidden states;
    calculate the average mean of the values of each token at the last four hidden layer to make the word embeddigs for each word in the sentence.
  end for
  make a key for the *sent* to store the word embeddings, the sentence in the *dictALL*
end for
Save *dictALL* as pickle file for using later.

4.3.4 Making sentence embedding for Frame-Definition

We shall describe the method used for making the sentence embeddings from the Frame definitons provided by FrameNet and provide the implementation overview in Algorithm 5.

sentence embedding method

One of the methods used by us for making the sentence embeddings during the experiments was mean pooling. It's a technique commonly used to create sentence embeddings, representing the meaning of a sentence as a fixed-size vector. The logic behind mean pooling is straightforward: it aggregates the word embeddings of all the words in the sentence by taking their average. This process effectively captures the overall semantic content of the sentence while maintaining a constant-size representation regardless of the sentence length.

Mean pooling works by first converting each word in the sentence into its respective word embedding using a pre-trained word embedding model. Then, it calculates the average of all these word embeddings to obtain the final sentence embedding. Mean pooling is computationally efficient and simple to implement, making it a popular choice for various natural language processing tasks, including sentiment analysis, text classification, and semantic similarity measurement. However, mean pooling may lose some nuanced information present in the sentence structure, as it treats all words equally without considering their positions or importance within the sentence. Despite its limitations, mean pooling remains a powerful and widely used technique for generating sentence embeddings due to its simplicity

and effectiveness in capturing the semantic content of sentences.

Algorithm 5 Frame Definition embedding generator

Require: Frame definitions in text for the selected FN version, SentencePLM for making the embeddings.

Declare a dictionary $dictDef$ for storing the FrameName as key and defintion as the value;
open the file containing both frame name and defintions;
split the text such that we have two lists: Frame Name and Defintions;
store both the values in the $dictDef$;
declare a new dictionary $dictEmb$ for storing the embeddings for the defintions;
for $Fname, FdefindictDef$ **do**
    Pass the $Fdef$ to the SentencePLM for making the embedding;
    Tokenise the sentence using the SentencePLM tokeniser;
    Pass the tokenised inputIDs to the model to get all hidden states;
    Do mean pooling to get the sentence embedding from the word embeddings;
    return sentence embedding;
end for
store the returned embedding as value and $Fname$ as key in $dictEmb$
store $dictEmb$ as a pickle file for using later in the classifier.

5. Experiments

We are seeing a boom in research in NLP and there is a great influx of pre-trained language models(PLMs) which show great results on various language understanding benchmarks. Another growing body of research is that of graph based deep neural network which have created a shift in the paradigm of how we approach text processing. Due to these reasons it is easy to miss upon fully exploring the strengths and limits of an older architecture simply due to a newer ones being continuously published. This motivated us to take upon a previous state of the art model designed for frame identification to see how it performs with different component and can it be made more efficient simply by replacing an older component with a newer one. We were also motivated to see if some common patterns emerge in the model's behavior as we experiment with its hyper-parameters. In this chapter we will describe the different experiments done with the selected architecture for FI described in the previous chapter. We shall look into the two different axes in which we conducted our exploration :

1. **Different PLMs** selected from different Benchmarking leaderboards.
2. **Different hyper-parameters** for the classifier and AGE module.

5.1 Selecting PLMs

The field of research on language modeling is generating more output than ever before and is producing newer and better models for various tasks each day.
We used the MTEB [Muennighoff et al., 2022] leaderboard on Hugging face which keeps track of some of the best-performing models both proprietary and open-source on various tasks related to language comprehension such as semantic-textual-similarity(STS), classification,etc by benchmarking them on a diverse set of datasets such as STS16, STS17 and many others. We also selected two other models, namely 'Deberta' and 'GTE'.

5.1.1 ELECTRA-base & ELECTRA-large

Since the original architecture has used embeddings from the Electra-base model which has a smaller hidden size, we also tranied the model with word embeddings from Electra-large which has the bigger hidden size, and was trained on a bigger corpus. This choice was made to see if the architecture benefits from a simple scaling of the hidden size of the language encoder component.

Architecture

Clark et al. [2020] state that the model architecture and most hyperparameters are the same as BERT's base version. The large version of Electra share the architecture with BERT's large version. During the finetuning and evaluation stage of the model, the authors used various classifier heads for different tasks but we shall not discuss that since in our own experiments we are only using the hidden layers without any head on top.

Training Data

We are informed by Clark et al. [2020] that during the pre-trainng stage, Electra models use the same data as BERT, which consists of 3.3 Billion tokens from Wikipedia and BooksCorpus [Zhu et al., 2015]. However, for the Large model the authors pre-trained on the data used for XLNet [Yang et al., 2019], which was created by combining the BERT dataset with data from ClueWeb [Callan et al., 2009], CommonCrawl, and Gigaword resulting in upto 33B tokens. Clark et al. [2020] also notify that all of the pre-training and evaluation of the models was done with English data only.

Training Details

Electra models were trained using the *replaced token detection (RTD)* technique proposed by Clark et al. [2020]. This pre-training task requires the model learn to distinguish real input tokens from synthetically generated replacements which are plausible tokens. The task of creating these plausible replacement tokens is done by a small masked language model called generator. These models are then pre-trained to act as a discriminator which can differentiate between the original token which was replaced with the plausible but corrupt tokens. The generator is discarded after the training process and the discriminator can be used for downstream tasks.Fig 5.1 illustrates this process. The authors of Electra state that a key advantage of this discriminative task is that the model does not exclude masked out tokens but rather is learning from all input tokens. Another key difference between the base and the large version of pre-trained Electra is that of the training time. Electra large has been trained for a much longer period compared to the base version.

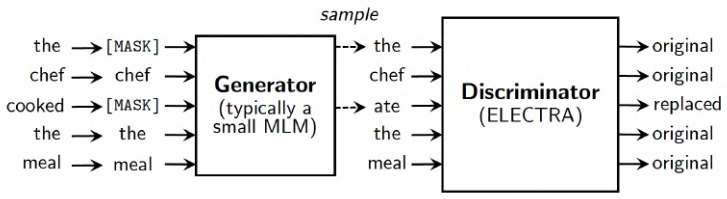

Figure 5.1: The model configuration for Replaced Token Detection training process. *Picture from Clark et al. [2020]*

5.1.2 DeBERTa V3

The DeBERTa model is another transformer based model trained by Microsoft which introduced the novel approach of storing the position information and word information in separate vectors instead of a single vector and they refer to this innovation as disentangled attention mechanism. In our experiments we have chosen to work with the V3 version of the model which were proposed in the work done by He et al. [2021] and wherever we mention DeBERTa, we are referring to DeBERTA V3. This model was trained with Electra style **replaced token detection** in combination with **disentagled attention**.

Architecture

DeBERTa uses L stacked Transformer blocks [Vaswani et al., 2017] where each block contains a multi-head self-attention layer followed by a fully connected positional feed-forward network. Two novelties introduced in the architecture are **DA (Disentangled Attention)** and an **enhanced mask decoder**.Unlike existing approaches that use a single vector for representing both the content and the position of each input word, the disentanglement mechanism instead creates two separate vectors where one is used for the content and the other for the position.

Training Data

The authors of DeBERTa state that they used data from Wikipedia for pre-training and the bookcorpus [Zhu et al., 2015] were used as training data, following the base model configuration of Devlin et al. [2018].

Training Details

Since RTD in ELECTRA has been proved to be very sample efficient, DeBERTa used it as its training objective to combine its strength with disentangled attention.

The generator used during its pretraining uses a neural net of half the size of the discriminator. They use a batch size of 2048 tokens and the model is trained for 125,000 steps with a learning rate of 5e-4 and warmup steps of 10,000. They validate the effectiveness of DeBERTaV3 testing it two different NLU tasks.

5.1.3 GTE

At the time of this work, the GTE model proposed in the paper "Towards general text embeddings with multi-stage contrastive learning" done by Li et al. [2023] was at the top of the MTEB leaderboard during the time of selection. The key highlight of the model is that it was trained with **multi-stage contrastive learning** to provide textual embeddings for both words and sentences which could act universally and be agnostic towards downstream task.

Architecture

The backbone of GTE embedding model is a deep Transformer encoder [Vaswani et al., 2017] which was initialized by its creators using pre-trained language models such as BERT [Devlin et al., 2018]. The model works by combining a vanilla dual-encoder architecture with a mean pooling layer at the end on top of the contextualized token representations produced by the previous hidden layers. Li et al. [2023] inform that the pooling operation only applies across the first dimension to get the text representation .

It utilises contrastive learning objective for creating text embeddings which the authors show to be capable of distinguishing semantically relevant text pairs from irrelevant ones. It uses the popular contrastive objective propsed in the InfoNCE loss [Oord et al., 2018], to estimates the similarity between two pieces of text by utilising the vector distance between them embeddings.

Training Data

Li et al. [2023] state that they did the unsupervised training part where they used a mix of resources with in total, $\sim 800M$ text pairs text pairs. They also provide a simple statistics and data distributions of the used data illustrated in Table 5.1

Source	Datasets	Prop. Size
Web Page	3	18.7% (147M)
Academic Paper	5	5.7% (45M)
Hyperlink	4	13.4% (106M)
Social Media	2	41.5% (327M)
Knowledge Base	2	4.8% (38M)
Community QA	7	1.5% (12M)
News	5	0.4% (3M)
Code	2	2.5% (20M)
Others	3	11.6% (91M)
Total	33	100% (788M)

Table 5.1: Statistics of data used in pre-training. *Data from Li et al. [2023]*

They subsequently have a supervised fine-tuning stage where they use relatively lower-sized datasets with human annotation of the relevance between two pieces of text and optional hard negatives mined by an extra retriever to form text triples. The authors state that this data was roughly $\sim 3M$ pairs of examples for fine-tuning.

Training Details

The work of Li et al. [2023] states that the training process of GTE model consists of two stages: unsupervised pre-training and supervised fine-tuning and both stages employ the learning objective of contrastive learning. Firstly, we will introduce the basic framework of the model.

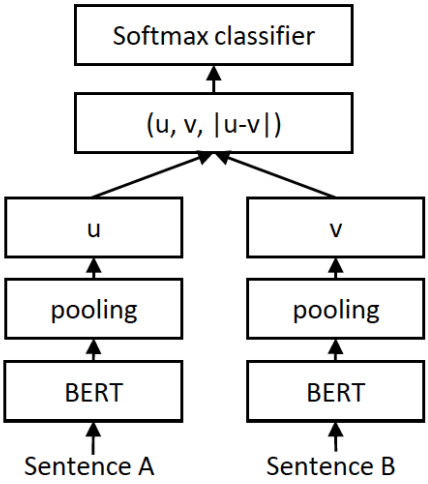

Figure 5.2: Architecture of Siames configuration of SBERT. *Picture from Reimers and Gurevych [2019]*

The authors point out that in the initial stage of unsupervised pre-training, data sources differ significantly in terms of the number of training instances and they address this imbalance by employing a multinomial distribution to sample data batches from different data sources, taking into account their respective sizes.

5.1.4 SBERT

Architecture

SBERT was created by using outputs from BERT/RoBERTa with an added pooling operations at the end for making sentence embeddings [Reimers and Gurevych, 2019]. The authors experimented with three pooling strategies which involved using the output of the CLS-token, computing the mean of all output vectors (refereed to as MEAN-strategy), and lastly they experimented with computing a max-over-time of the output vectors (refereed to as MAX-strategy). They choose the default configuration as the MEAN. They also finetuned selected BERT / RoBERTa using siamese and triplet networks similar to Schroff et al. [2015]. This was done to update the weights in such a way that the produced sentence embeddings are semantically meaningful and lend themselves to be used with cosine-similarity.

Reimers and Gurevych [2019] inform that siamese configuration uses a classification objective function where it concatenates the sentence embeddings u and v with the element-wise difference $|u - v|$ and multiply it with the trainable weight $W_t \in \mathbb{R}^{3n \times k}$:

$$O = \text{softmax}(W_t(u, v, |u - v|))$$

where n is the dimension of the sentence embeddings and k is the number of labels. It optimizes the cross-entropy loss. This structure is depicted in Figure 5.2.

Training Data

Reimers and Gurevych [2019] inform us that SBERT was trained on the combination of the SNLI by Bowman et al. [2015] and the Multi-Genre NLI by Williams et al. [2017] dataset and describe that SNLI consists of 570,000 sentence pairs which are annotated with the labels *contradiction, entailment, and neutral.*The authors choose these datasets to encorporate text from a diverse set of genres considering both spoken and written text.

Training Details

SBERT is fine-tuned using a 3-way softmax classifier objective function where it was provided with three sentences they call as an anchor sentence a, a positive sentence p, and a negative sentence n. The tune the models using the triplet loss with the objective of having a distance between a and p that is smaller than the distance between a and n which makes the model to create semantically rich sentence embeddings.

5.1.5 Statistical description of PLMs

We compared the three selected models on various attributes such as their vocabulary size and their tokenisation methods and how their hidden layers are configured to correctly setup the tokenisation pipeline and correctly use them to generate word-embeddings for our dataset. The statistics from the comparison can be found in Table 5.2 and Table 5.3. Note that the tokeniser of the models also differ in the way they encode the input text due to which we noticed that WordPiece based tokeniser resulted in fitting more text within 512 tokens as compared to Byte-pair encoding which splits texts into sub-words resulting in requiring larger token sequences. Since the max token length is set at 512, there is a possibility that certain example sentences were truncated more than what wordpiece based tokeniser would do.

	hidden layers	hidden size	attention heads	intermediate size	Parameters
Electra-base	12	768	12	3072	110M
Electra-large	24	1024	16	4096	335M
Deberta-v3-large	24	1024	16	4096	304M
GTE-large	24	1024	16	4096	330M

Table 5.2: Pretrained Language Model architecture details

	vocab size	max token length	encoding type
Electra-base	30522	512	WordPiece.
Electra-large	30522	512	WordPiece.
Deberta-v3-large	128000	unset	Byte-Pair
GTE-large	30522	unset	WordPiece.

Table 5.3: Pretrained Language Model Tokeniser details

5.2 Hyper-parameter tunning

Learning algorithms related to artificial neural networks and to a greater degree for Deep Learning involves setting up values for multiple points in the algorithm which work together to make in the desired way. These tune able values are called hyper-parametersand using different values for these hyper-parameters can yield different results. Bengio [2012] provides a concise definition of hyper-parameters as follows :

> "Hyper-parmeter for a learning algorithm A is any variable that need to be set to prior to the application of A on the selected data and this variable's value cannot be directly selected by the algorithm itself."

Many good practices and standard approaches have been established for tuning different hyper-parameters to increase efficiency. During our experiments, we configured our classifier with different values for the following hyper-parameters of the model.The motivation behind tweaking different aspects lies in optimising the model's efficiency and efficacy.

5.2.1 Tuning batch size

The mini-batch size can be roughly understood as how much input data is passed through the neural network and its value can vary from 1 to a few hundreds but over the years it has becomes a good practice to use a mini-batch size of 32. The mini-batch size (also called simply batch size) has a direct impact on the computations and a larger batch size usually leads to a faster computation(not always true) but requires more example data to reach an optimal error value since there are less epochs and thus the model sees fewer updates. One can fine tune the batch size by looking at the training curves and validation curves to reach an optimal number of batches. One also need to consider learning rate when working with different batch sizes and they can be tune together.

5.2.2 Tuning Laplacian feature smoothing in AGE

As described in the architecture of the AGE module, the Laplacian tuning depend upon the choice of K for performing the smoothing operation and there is no specific value of K which is one size fits all. In our own experiments we start with the value K set to 8 which provides the best results at FI.

We used other values such as 10 and 5 to see check how increasing and decreasing the amount of smoothing affect the functioning of our model with respect to the dataset being used.

5.2.3 Tuning Projection space dimension

Since we are training our classifiers with multiple embeddings, embedding projection serves as a crucial technique for integrating information from diverse embedding sources into a unified representation space. This practice enables the model to effectively capture and

leverage the complementary information embedded within different embeddings, thereby enhancing the model's capacity to generalize and discern patterns across varied data modalities. The embedding projection process involves mapping embeddings from their original spaces to a common projection space through linear transformations or neural networks, thereby aligning them in a unified feature space for downstream tasks.

The dimensionality of the projection space plays a pivotal role in the training process, exerting a significant impact on the model's performance and generalization capabilities. A higher-dimensional projection space can accommodates a more expressive feature representation, enabling the model to capture intricate patterns and nuances present across multiple embedding sources. However, increasing the dimensionality of the projection space may also lead to an escalation in model complexity and computational overhead, potentially resulting in overfitting and reduced efficiency during training. Conversely, a lower-dimensional projection space may result in a loss of discriminative information and reduced model capacity, limiting the model's ability to discern subtle differences and variations in the data.

The choice of the projection space dimensionality necessitates a delicate balance between model complexity and performance requirements. Guided by considerations such as dataset size and computational resources, we choose to work with 2 different sizes for evaluating the performance of the classifier. Through empirical evaluation and iterative experimentation, practitioners can ascertain an optimal projection space dimensionality that strikes an equilibrium between model expressiveness and computational efficiency, thereby facilitating the development of robust classifiers capable of effectively leveraging multiple embeddings for improved performance on our downstream tasks.

5.2.4 Tuning batch size for AGE

The AGE module works with a subset of the nodes which is made of half similar nodes and half dissimilar nodes to generate Frame embeddings for the nodes of the graph and since our Frame-relations graph ,like all real-world graphs, has far more dissimilar node pairs than positive pairs, the choice of the size of this subset for updating the node similarity during the training of the classifier becomes crucial.

The baseline configuration used a value of 500 nodes(half having positive simlairty and half having negative similairty) for the subset. we also tried to train the model with an even smaller size of subset of 256 nodes to see if a smaller subset would force the model to be provide more robust and better node embeddings for the frames.

5.2.5 Adding dropout layer to Classifier

Dropout layers constitute a fundamental technique in training classifiers, particularly within the realm of deep learning, serving as a regularization mechanism to mitigate overfitting and enhance model generalization. Srivastava et al. [2014] inform that dropout operates by randomly deactivating a fraction of neurons within the network during each training iteration, thereby preventing co-adaptation among neurons and promoting the robustness of the model to noise and variations in the data . By including dropout layers into the architecture of our classifier,we aim to make it less prone to over-reliance on specific features or

correlations present in the training data which in turn allows us to have a more generalized representation of the underlying patterns in our data.

In practice, several good practices have emerges for an effective use of dropout layers in classifier training. Firstly, the dropout rate, denoting the probability of neuron deactivation, must be carefully calibrated to strike a balance between regularization strength and model capacity. Setting the dropout rate too high may hinder the model's ability to learn useful features, while setting it too low may fail to curb overfitting effectively. Moreover, dropout layers should primarily be incorporated during training, with the dropout functionality deactivated during inference to ensure consistent and deterministic predictions. Additionally, when employing dropout in conjunction with other regularization techniques, such as weight decay or batch normalization, practitioners must judiciously tune hyperparameters to prevent unintended interactions and optimize overall model performance.

Furthermore, we also choose to apply dropout strategically within the network architecture, targeting layer with a high capacity for memorization and complex feature representations. Commonly, dropout is applied to fully connected layers and convolutional layers in neural networks, while sparing certain architectural components, such as input and output layers, from dropout regularization. We employ a constant dropout during our training.

Ultimately, as mentioned by Srivastava et al. [2014] we employ a max norm regularisation after a dropout for effective generalisation.

5.3 Baseline configuration

The model presented in the original paper came with certain hyper-parameters which produced the best result and we repeated the training with no changes and confirmed the results. This configuration become our baseline for measuring the effects of the changes made during our experiments. From this point onwards the baseline is refers to the following hyper-parameters and their values -

Parameter	**Value**
Batch Size	32
GNN Layers (affect the k in Laplacian filter)	8
Projection Dimension	256
AGE Batch Size (subset of nodes for updating node similarity)	250

Table 5.4: Baseline configuration of Hyperparameters during experiments

6. Results

In this chapter we present the results obtained from the experiments conducted to satisfy our research objectives and we present them in the parts according to the aspect of the Frame Identification system we tweaked. We have also shown the results gathered by the repetition of the FI system without any modification as shown in the GitHub repository and this acts as a comparative baseline for all our experiments as it allows us to clearly see how different configurations of the model affect its performance. We monitor the training on various metrics and visualise the the system's behaviour during this process as well as results achieved.

It is known that the training process involves some level of randomness due to the stochastic nature of the training initialisation process and different runs with the same configuration of hyper-parameters might result in slightly different training trajectories. Due to time limitations and lack of computational power, the results produced in this work are only based on single runs and should be verified further with repeated runs and calculating mean and deviation for each setup.

6.1 Dependency stack

For any one interested in repeating the experiments done in this report, we have provided the code and the data-set on our GitHub repository. It is known that the results may vary due to different GPU models, drivers, CUDA SDK versions, and random seeds, we also declare our dependency stack for the the experiments in Table 3.3. Full dependency list is available on the GitHub repository. of this work. The training was performed on 1 Nvidia GTX1060 (6GB) card.

Library	**Version**
CUDA	11.7
pytorch	2.0.1
Transformers	4.33.2

Table 6.1: Dependencies

6.2 Observations on target embeddings

Name	Configuration	Accuracy Score				FE-JS
		Total	Ambiguous	Unambiguous	Unknown	
Electra-base	baseline	92.29	83.33	99.74	95.12	37.38
	batch size - 16	92.04	82.78	99.74	95.09	36.75
	AGE layers - 10	92.01	82.81	99.67	90.24	35.65
	Projection dim - 512	92.17	83.01	99.79	97.56	36.90
Electra-large	baseline	91.54	81.67	99.75	95.12	35.17
	batch size - 16	92.03	82.81	99.71	92.68	37.57
	AGE layers - 10	91.74	82.17	99.71	92.68	35.80
	Projection dim - 512	90.90	82.41	99.79	97.56	34.27
GTE-large	baseline	91.09	80.74	99.71	92.68	32.47
	batch size - 16	90.93	80.34	99.75	95.12	29.28
	AGE layers - 10	90.64	79.70	99.75	95.12	26.41
	Projection dim - 512	91.36	81.33	99.71	92.68	37.54
DebertaV3-large	baseline	84.82	68.74	98.63	24.98	19.45
	batch size - 16	84.92	68.59	98.52	21.95	17.37
	AGE layers - 10	84.58	67.80	98.56	24.39	19.15
	Projection dim - 512	84.65	68.34	98.23	04.87	14.15

Table 6.2: Results FI and FE Classfication performance

Name	**Configuration**	**Parameters**	**Final Epoch**	**AUC**	**AP**
Electra-base	baseline	1,541,496	181	.9326	.9357
	batch size - 16	1,541,496	149	.9374	.9330
	AGE layers - 10	1,541,496	189	.9077	.8763
	Projection dim - 512	2,490,232	137	.9337	.9250
Electra-large	baseline	1,607,032	142	.9332	.9127
	batch size - 16	1,607,032	180	.9398	.9371
	AGE layers - 10	1,607,032	221	.9260	.9055
	Projection dim - 512	2,621,304	131	.9266	.9131
DebertaV3-large	baseline	1,607,032	154	.9369	.9284
	batch size - 16	1,607,032	168	.9347	.9315
	AGE layers - 10	1,607,032	243	.9300	.9254
	Projection dim - 512	2,621,304	109	.9220	.9135
GTE-large	baseline	1,607,032	179	.9324	.9243
	batch size - 16	1,607,032	189	.9392	.9355
	AGE layers - 10	1,607,032	194	.9034	.8661
	Projection dim - 512	2,621,304	204	.9341	.9278

Table 6.3: Results: AGE performance and Epoch count

We find that the original model is still better in terms of performance for all measures performed during and after the training and a simple modification of the PLM used for making the target embedding is not sufficient for making improvements in its performance.

Having said that, we still found how the model behaves with respect to individual changes made which help us understand the model's capacity in a better way. We share our findings below and talk about possible reasons for the observations.

By looking at the recorded epoch at which the model training is stopped, we see a pattern of quicker learning in models with a higher dimension of the projection layer. This could be the result of increased parameter count resulting in more complex modeling of weights for doing the classification. We also observe that the average precision also goes down when increasing the dimension size.

We notice that the training and test accuracy of the model for each configuration tends to move away from each other after the 20th epoch and by the end of the training there is a difference of 7 percent for frame instance classification. The model performs better with a smaller embedding size for target instances and has a smoother and slower convergence as shown in Fig1 and also has less difference between training and test accuracy but only by marginal value.

Another hyper-parameter that we experimented with was the number of AGE layers which correspond to how many times the filtering happens for obtaining the smooth feature matrix for the frame definitions. This hyperparameter is not parametric but is directly related to how well the AGE component of the mode can create frame-embeddings. The baseline has 8 layers and we experimented with 10 layers to see if an increase smoothing could result in a more robust model for predicting the frames. This resulted in a decrease in both the AP and AUC metrics and could be a sign of over-smoothing which is making it harder to produce sufficiently distinguished embedding for each frame. We also observe a rise in AUC and AP when working with higher dimension projection, which uses more parameters, which could result in a better calculation of node similarity of Frames and consequently make better frame embeddings.

We see that the increased vector size of the target embeddings with all the PLMs didn't result in a better performance. The model performed slightly worse even with Electra-large which is just a slightly bigger version of the base model but also pretrained with a different dataset compared to the Electra-base version.

Embedding size difference between target and frame definition. - During our experiments we did not touch upon the frame defintion features which are made from SBERT and not modified from the original paper. The results observed during the experiments could also be casued due to the difference in the way the embeddings are produced by newer language models like GTE and deberta which rely upon different training task and pooling mechanism compared to SBERT.

We observe a rise in AUC and AP when working with higher dimension project using more parameters. this could result in better updating of similarity due to less compression and loss of information when performing projection to a lower dimension.

From these observations of the performance of the model with the previously shown configurations, we decided to work with the baseline configuration of the hyperparameters in our further experiments and we explicitly mention whenever we deviate from these configurations moving forward. Also we choose to not perform further experiments with the Electra-large since it showed no promise over Electra-base.

6.3 Observation on Graph node embeddings

We made new Frame definition embeddings using two different PLMs namely GTE and miniLM-L12 from SentenceBERT and compared the performance of the system with the original architecture. The embeddings made by GTE have a hidden size of 1024 and they are normalised before being used. The embeddings from miniLM are much smaller in comparison having a hidden size of only 384 which results in a relatively smaller size of trainable parameters and leads to a quicker training time. We only use the baseline configuration to measure the effects of the sentence embeddings on the model. The results of these changes are shown in the Table 6.4 and Table 6.5

Name	Configuration	Accuracy Score				FE-JS
		Total	Ambiguous	Unambiguous	Unknown	
Electra-base	baseline + GTE	91.92	82.46	99.79	97.56	.3627
	baseline + miniLM	91.99	82.61	99.79	97.56	.3550
GTE-large	baseline + GTE	91.22	81.03	91.71	92.68	.3493
	baseline + miniLM	91.22	81.03	99.71	92.68	.3394
DebertaV3-large	baseline + GTE	84.27	67.45	98.27	07.31	.1326
	baseline + miniLM	85.03	68.88	98.47	19.51	.1357

Table 6.4: Results FI and FE Classification performance

Name	**Configuration**	**Parameters**	**Final Epoch**	**AUC**	**AP**
Electra-base	baseline + GTE	2066040	198	.7013	.6360
	baseline + miniLM	1000440	186	.8386	.7881
GTE-large	baseline + GTE	2131576	227	.7093	.6506
	baseline + miniLM	1065976	208	.8620	.7977
DebertaV3-large	baseline + GTE	2131576	157*	.6748	.6332
	baseline + miniLM	1065976	167*	.7801	.7220

Table 6.5: Results: AGE performance and Epoch count

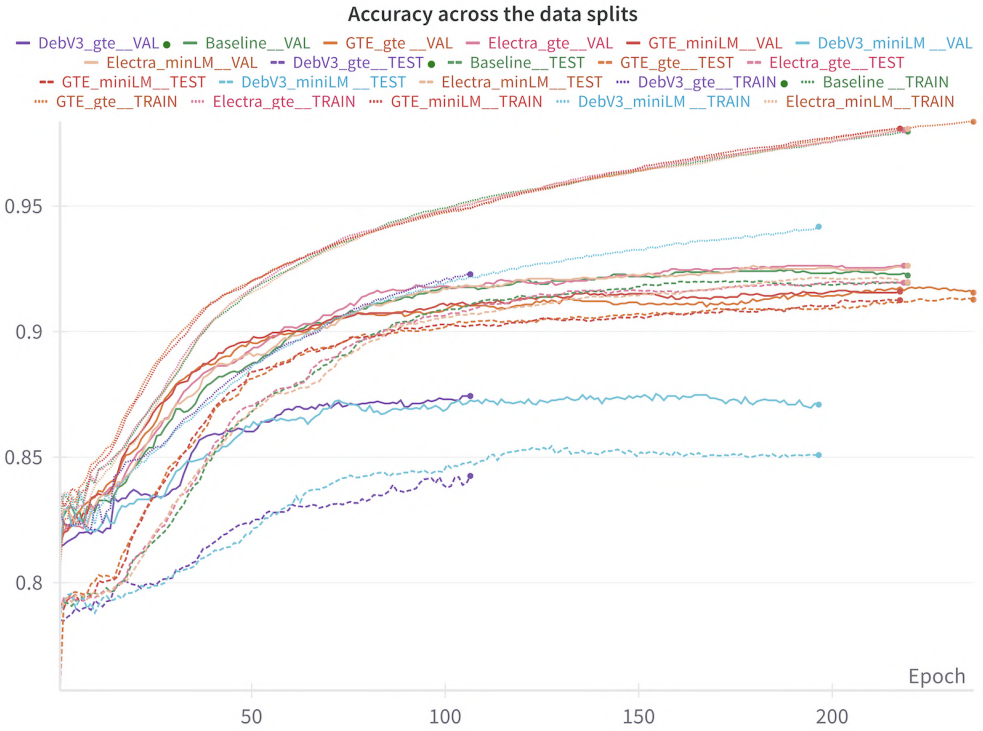

Figure 6.1: Accuracy of the selected models during training and evaluation

Figure 6.2: Combined Loss of the selected models during training

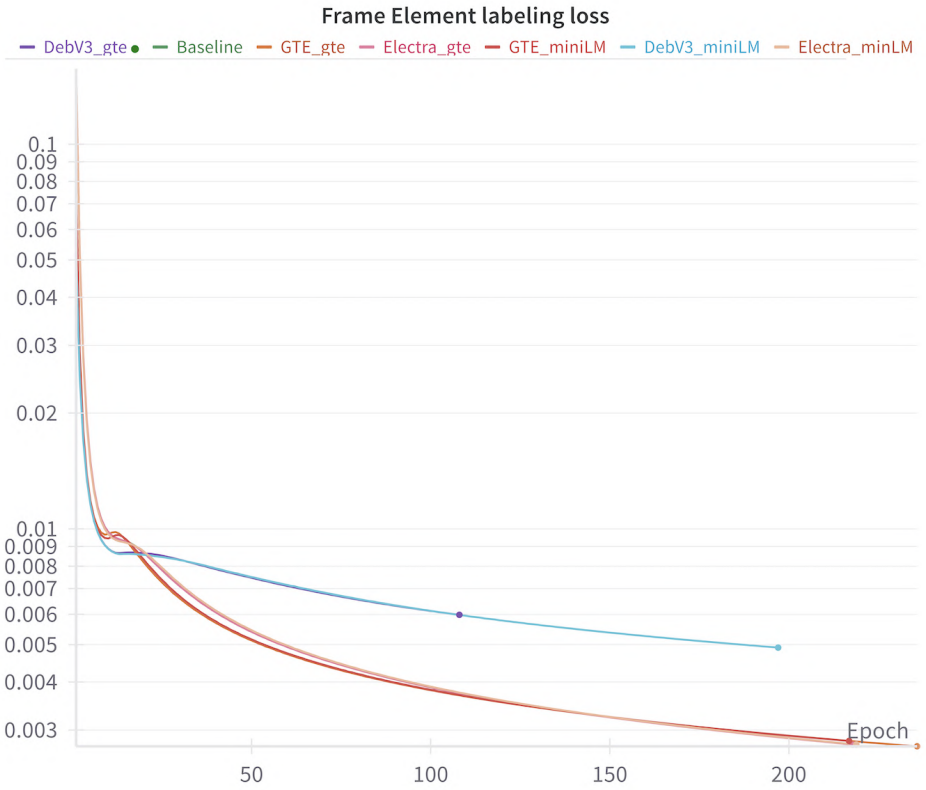

Figure 6.3: Frame Element loss during training

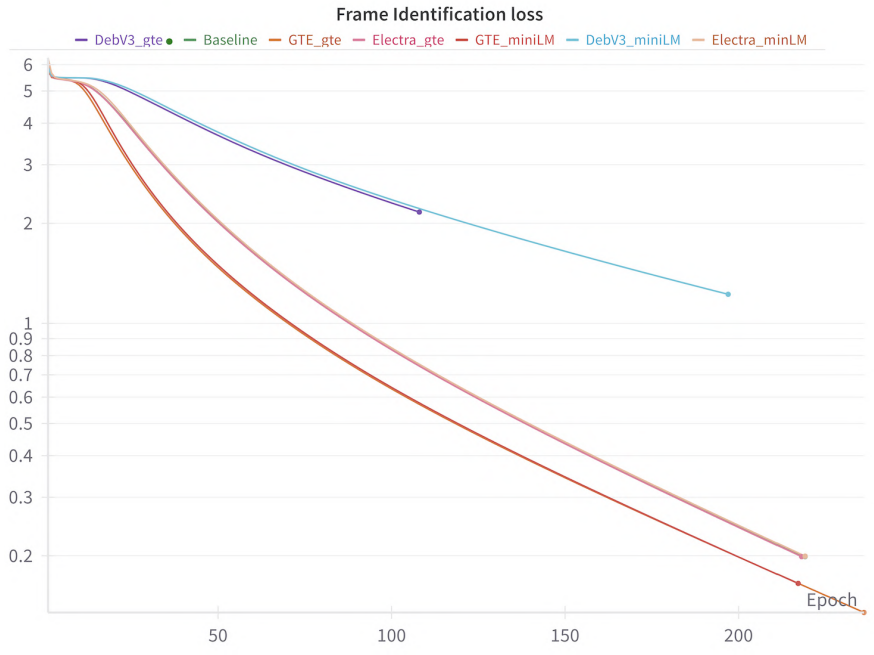

Figure 6.4: Frame Identification loss during training

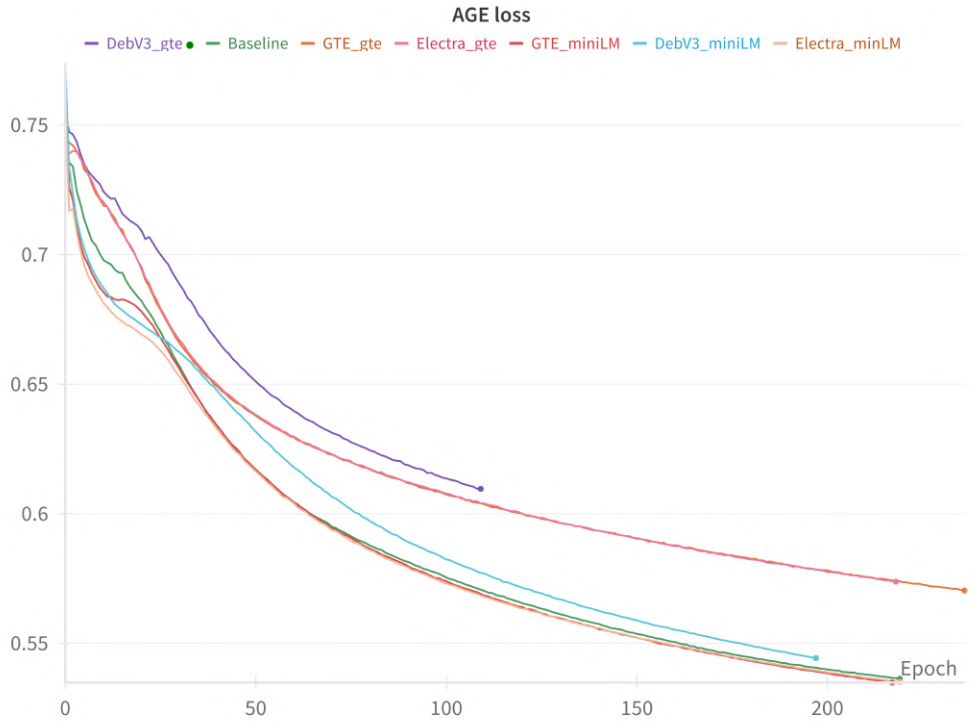

Figure 6.5: AGE Loss with different sentence embeddings

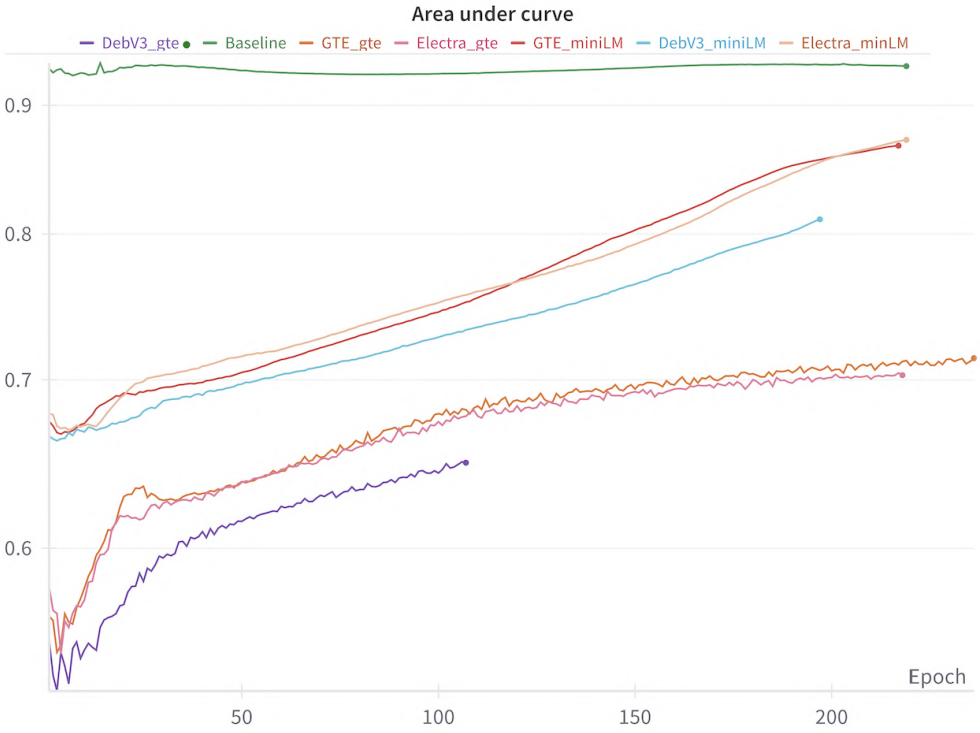

Figure 6.6: Area under curve with different sentence embedding

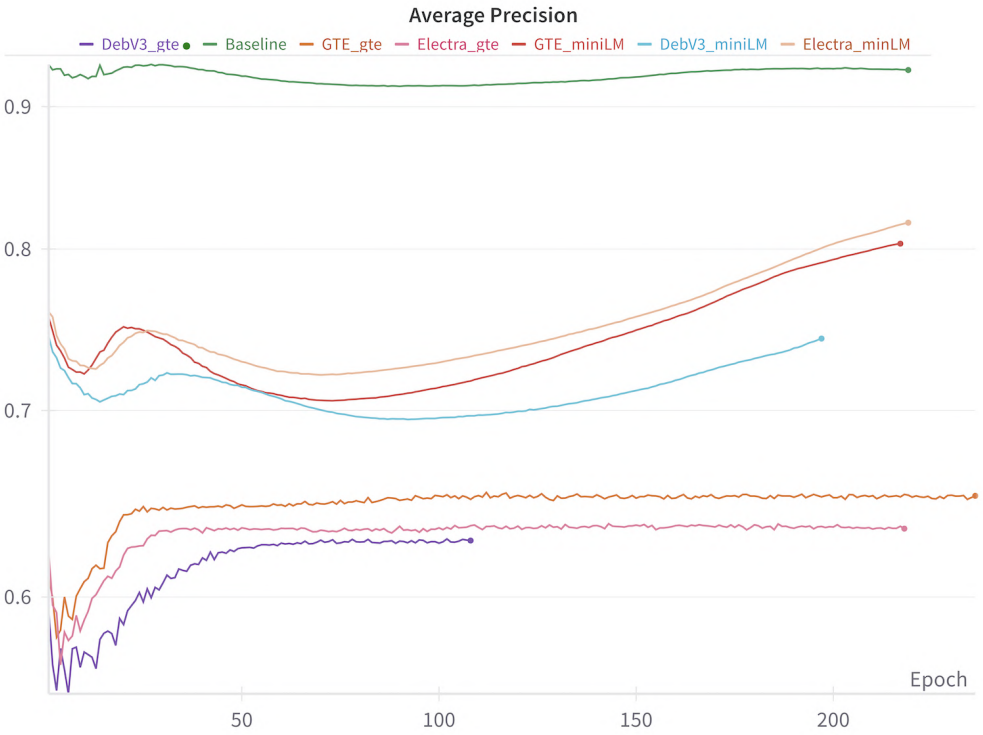

Figure 6.7: Average Precision with different sentence embedding

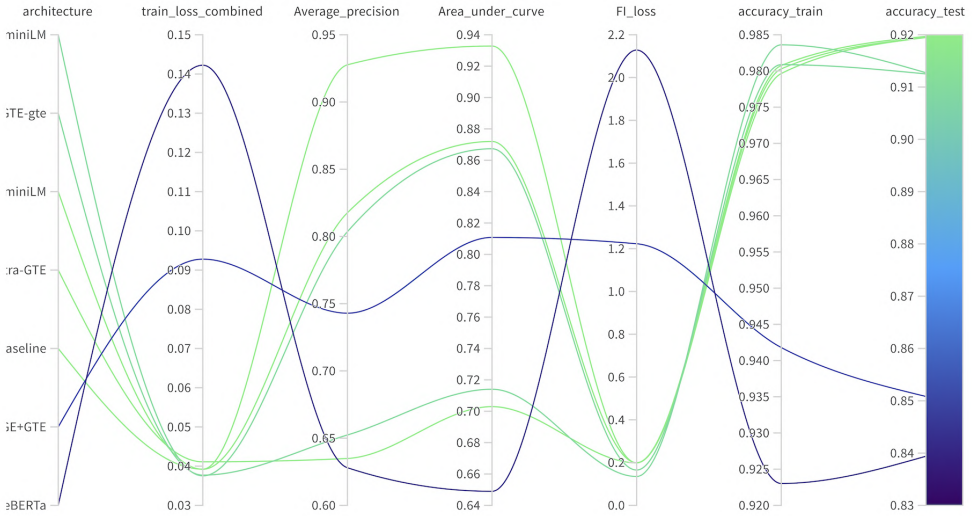

Figure 6.8: Sweep comparison of multiple measures for all different sentence embeddings

After replacing the original sentence embeddings with the GTE sentence embeddings and miniLM embeddings, we do not see any significant changes when combined with the target word embeddings from Electra-base but it does suffer a minuscule loss in its accuracy. Meanwhile, when combined with target embeddings from GTE, we see a minor improvement in its accuracy. We also see that the target embeddings from DeBERTaV3 still show the previously observed trend but perform even worse when dealing with unknown tokens and due to their much slower convergence rate as shown in Figure 6.17, we had to stop both the runs before it could reach normal completion to save computational resource and time and we show the results with the evaluation performed on the last epoch model reached. Another aspect of DeBERTa target embeddings is their tendency to reach a plateau around the Epoch 90-100 and the model sees almost no reduction of Frame Identification loss over the next 100 epochs as shown in the Figure 6.20. Lastly, we can also see from the Figure 6.17 that training with DeBERTa suffers the highest amount of overfitting as it shows the maximum difference between the training accuracy and the test accuracy which shows it's not able to generalise for unseen examples. Citing the previously stated reasons, we choose to drop trying further experiments with DeBERTa embeddings.

Another interesting observation is on the average precision of the model where we observe from Figure 6.7 that the sentence embeddings from miniLM perform significantly well as compared to the ones from GTE while being less than half the size of GTE. This keeps the model at a very small size of roughly 1 million trainable parameters. In contrast to the behavior of miniLM, when training with GTE sentence embeddings tend to make the average precision climb very quickly in the early stages but very quickly reach a plateau quite similar to DeBERTa's behavior.

6.4 Observations on classifier tuning

We also evaluated the model's performance with an added dropout of 0.2. Since dropout has been proven to improve the robustness of learning in neural networks and could help avoid over-fitting, we tested it with three configurations,

1. dropout to the AGE embedding projection
2. dropout to the target word embeddings.
3. dropout to the Frame possibility matrix output + AGE embedding projection.

Note that for the 3rd dropout configuration, we are using a learning rate of $2e-03$ while in the baseline configuration and the rest of the configurations we continue to use $2e-05$. The reason for doing this was to avoid the added computational time required due to having two dropouts as we were observing almost double the time required for training as compared to without dropouts. The results obtained are shown in Table 6.6 and Table 6.7

We can see from the results that the D2 configuration with the double dropouts sees a dramatic decrease in the number of epochs required to reach an optimal convergence and when implemented with the Electra-base target embeddings, we have a slight decrease in the accuracy but it also shows an improved score in Frame element labeling with a Jaccard

Name	Configuration	Accuracy Score				FE-JS
		Total	Ambiguous	Unambiguous	Unknown	
Electra-base	baseline + D1.1	92.03	82.71	99.79	97.56	.3901
	baseline + D1.2	91.88	82.37	99.79	97.56	.3189
	baseline + D2	91.29	81.08	99.79	97.56	**.4515**
GTE-large	baseline + D1.1	91.07	80.69	99.71	92.68	.3529
	baseline + D1.2	91.22	81.03	99.71	92.68	.3087
	baseline + D2	91.13	80.83	99.71	99.68	**.4408**

Table 6.6: Results FI and FE Classification performance

Name	**Configuration**	**Parameters**	**Final Epoch**	**AUC**	**AP**
Electra-base	baseline + D1.1	1,541,496	255	92.99	.9255
	baseline + D1.2	1,541,496	185	.9316	.9227
	baseline + D2	1,541,496	**20**	92.05	.9169
GTE-large	baseline + D1.1	1,607,032	249	93.22	.9220
	baseline + D1.2	1,607,032	215	.9229	.9123
	baseline + D2	1,607,032	**25**	92.58	.9239

Table 6.7: Results: AGE performance and Epoch count

score of .45 and a similar trend is seen when training with the GTE target embeddings where the Jaccard score is .44. These scores are much higher than the baseline configuration and the model tend to reach a very low loss when performing Frame Element labeling as can be seen from the Figure 6.11. At the same time, we also observed a poor performance in the AGE loss convergence when using D2, see figure 6.13 configuration and it could not reach its optimal convergence due to the extremely early stopping of the training and consequently, we also see a poor area under the curve when using D2. D1.1 and D1.2 seem to not affect the overall accuracy by much but do show a tendency to make the model reach a slightly longer convergence time.

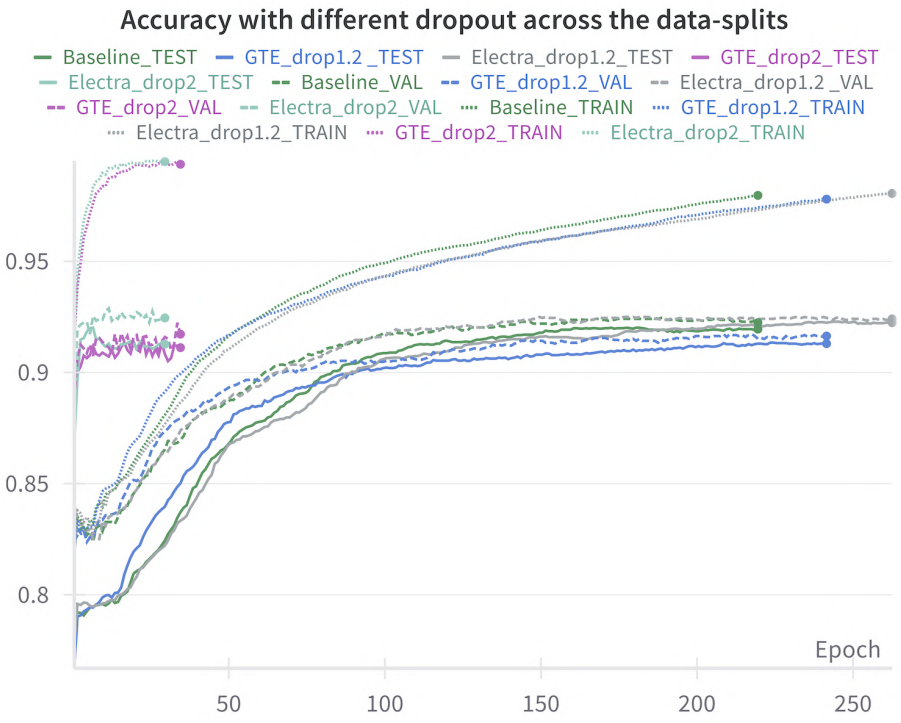

Figure 6.9: Accuracy of the Classifier with different dropouts during training and evaluation

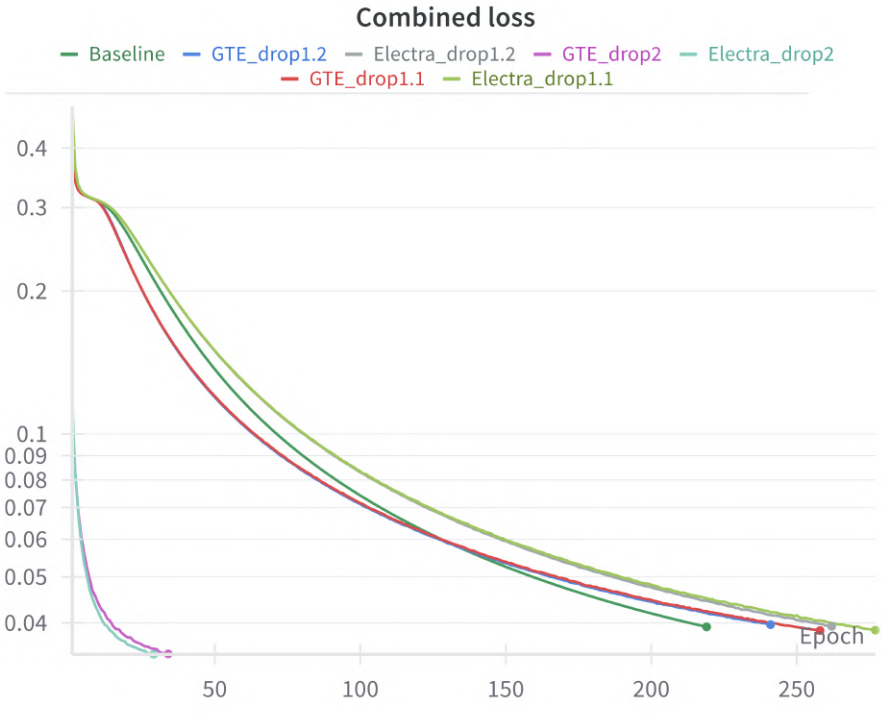

Figure 6.10: Combined Loss with different dropout configurations during training

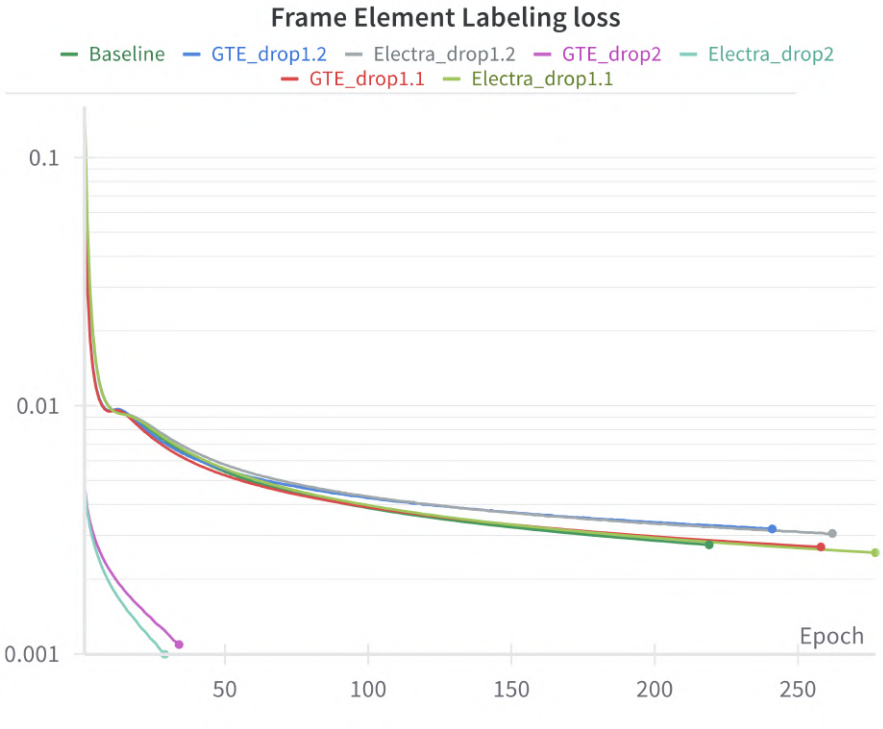

Figure 6.11: Frame Element loss with different dropouts over the selected models

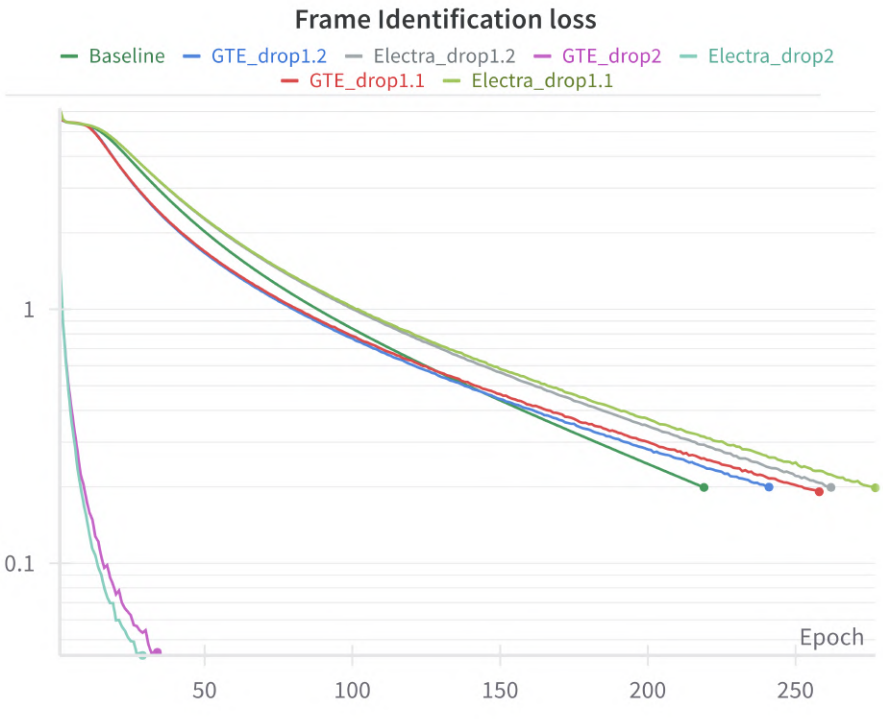

Figure 6.12: Frame Identification loss with different dropouts over the selected models

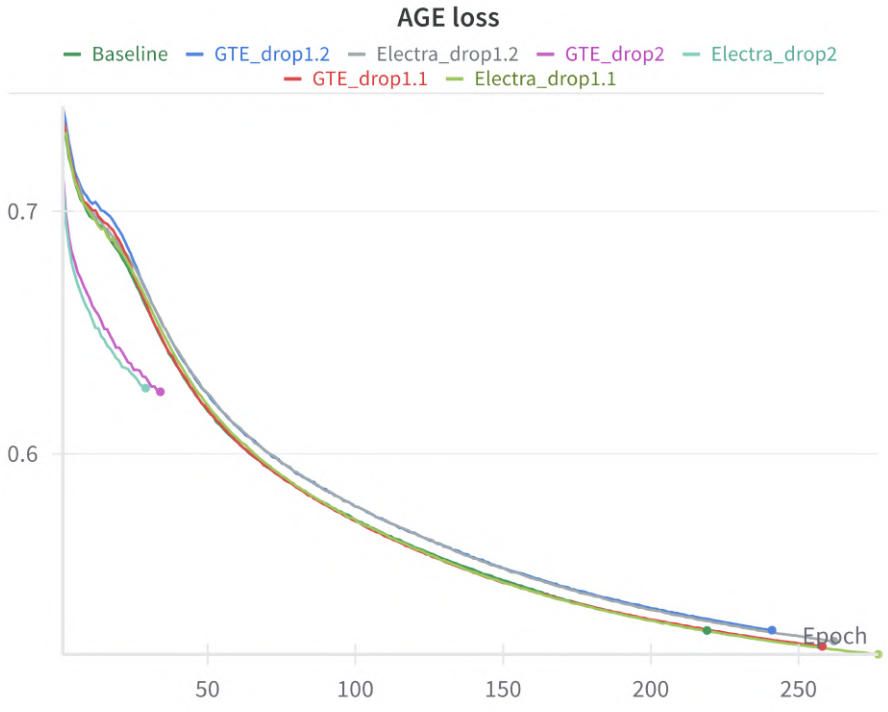

Figure 6.13: AGE Loss with different dropouts over the selected models

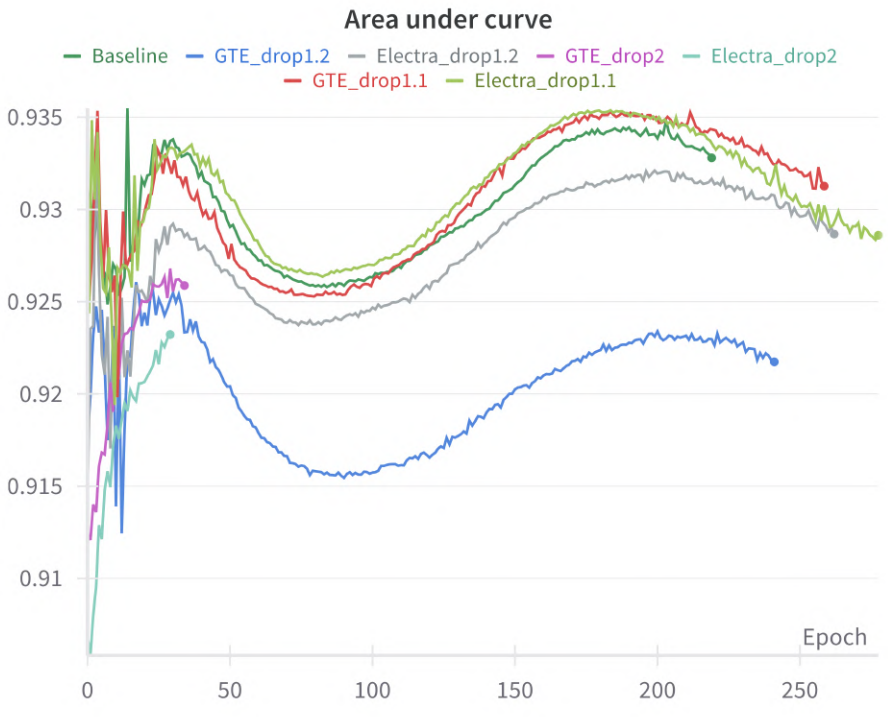

Figure 6.14: Area under curve with different dropouts over the selected models

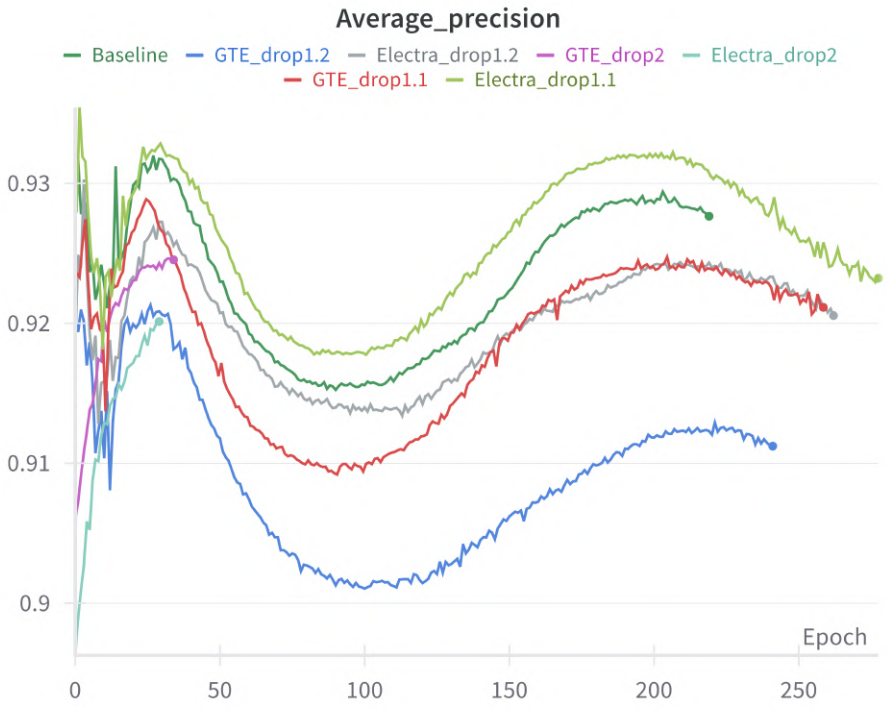

Figure 6.15: Average Precision with different dropouts over the selected models

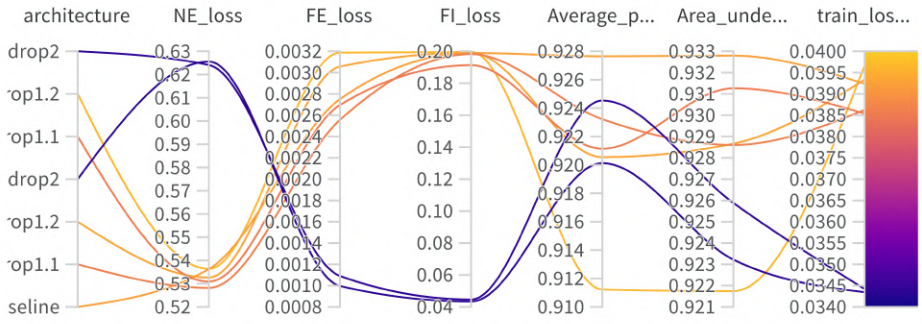

Figure 6.16: Sweep comparison of multiple measures for all different dropouts over the selected models

6.5 Observations on AGE tuning

We have taken one aspect of the AGE to tune, one being the batch size used for the similarity calculation of the node embeddings and we show the results with two values we worked with. The results are provided in Table 6.8 and Table 6.9 Note that that baseline configuration uses 512 as the batch size for AGE to create the node embeddings.

Name	Configuration	Accuracy Score				FE-JS
		Total	Ambiguous	Unambiguous	Unknown	
Electra-base	baseline	92.29	83.33	99.74	95.12	.3738
	baseline 256	92.08	82.81	99.79	97.56	.3483
GTE-large	baseline	91.09	80.74	99.71	92.68	.3247
	baseline 256	91.27	81.13	99.71	92.68	.3431

Table 6.8: Results FI and FE Classification performance

Name	**Configuration**	**Parameters**	**Final Epoch**	**AUC**	**AP**
Electra-base	baseline	1,541,496	181	.9326	.9357
	baseline 256	1,541,496	171	.9338	.9274
GTE-large	baseline	1,607,032	179	.9324	.9243
	baseline 256	1,607,032	215	.9405	.9382

Table 6.9: Results: AGE performance and Epoch count

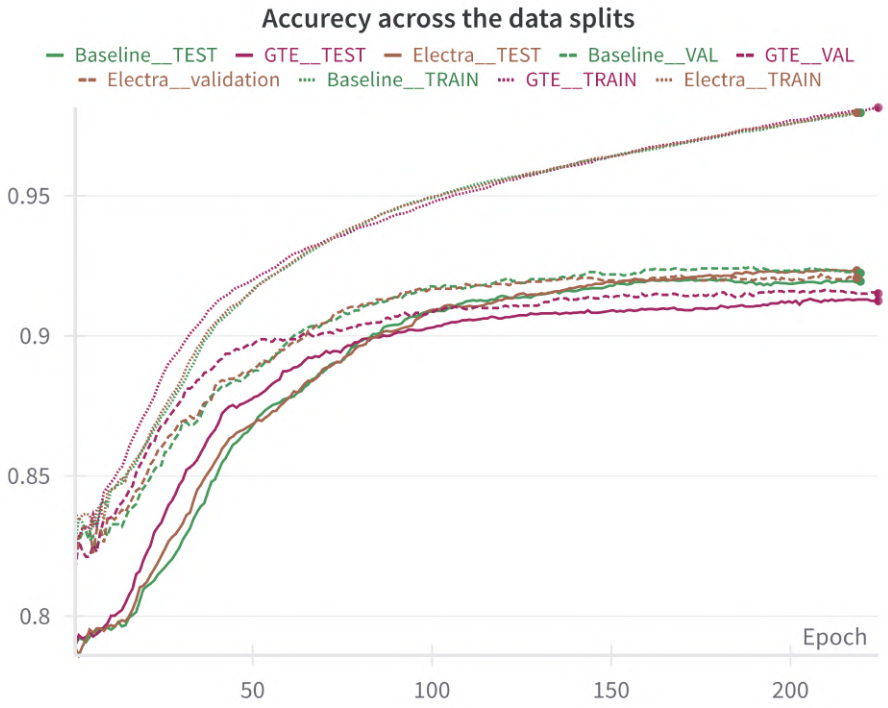

Figure 6.17: Accuracy of the selected models with different AGE batch sizes over the selected models

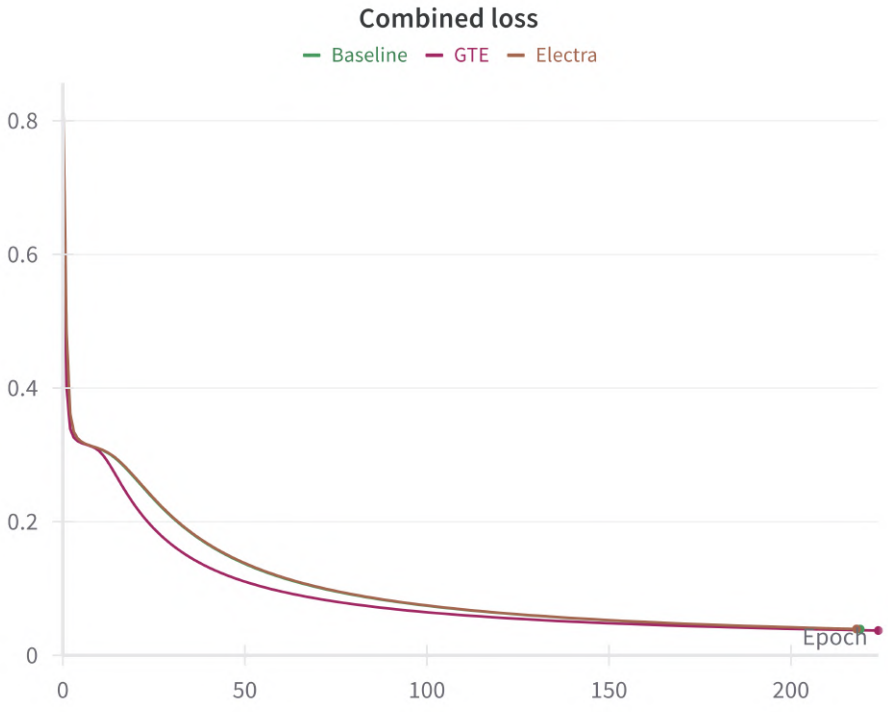

Figure 6.18: Combined Loss of the AGE batch size over the selected models

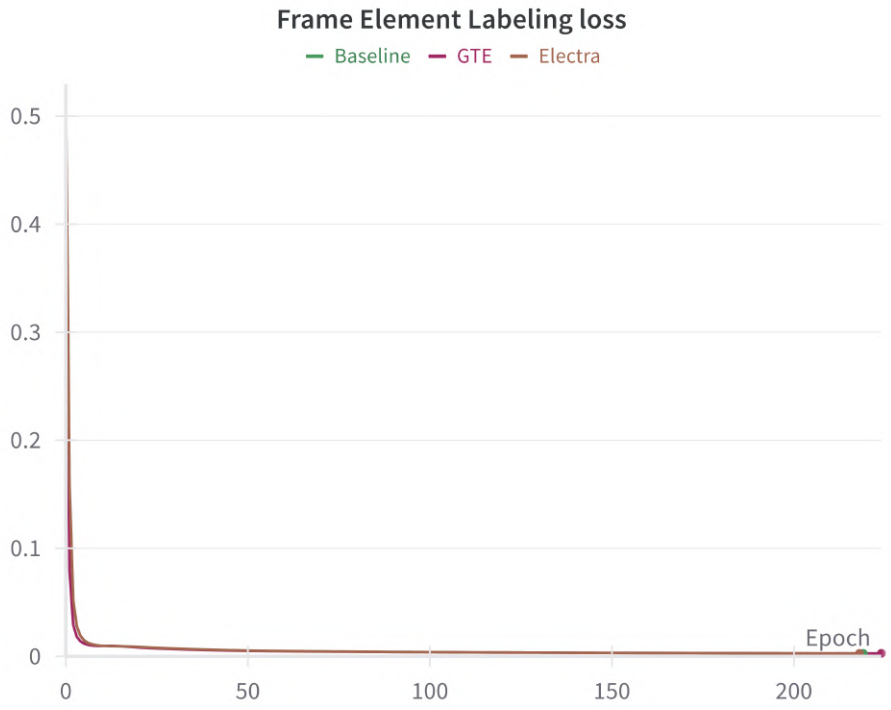

Figure 6.19: Frame Element loss for the different AGE batch size over the selected models

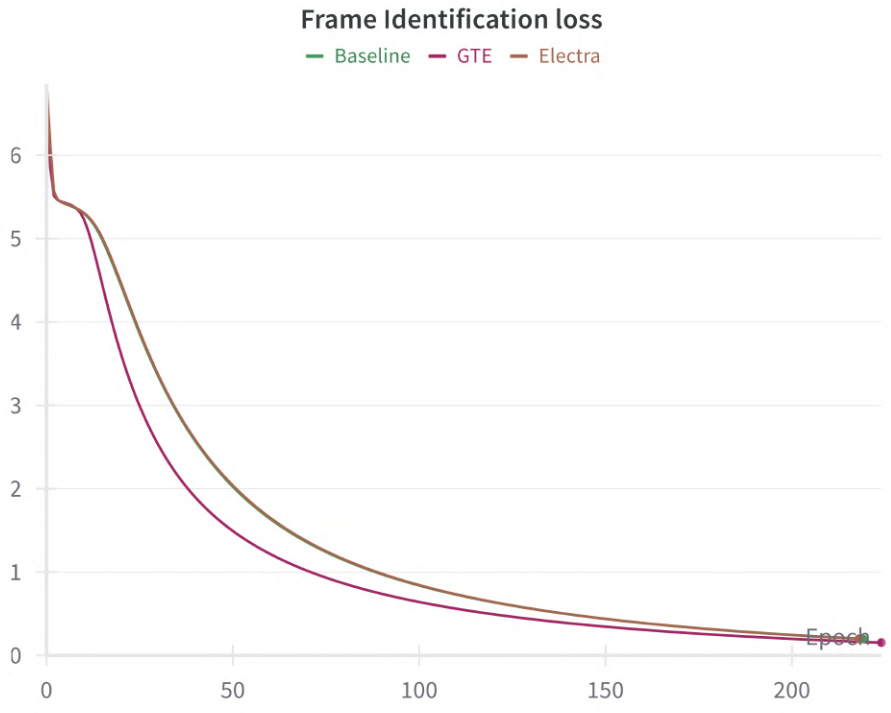

Figure 6.20: Frame Identification loss for different AGE batch size over the selected models

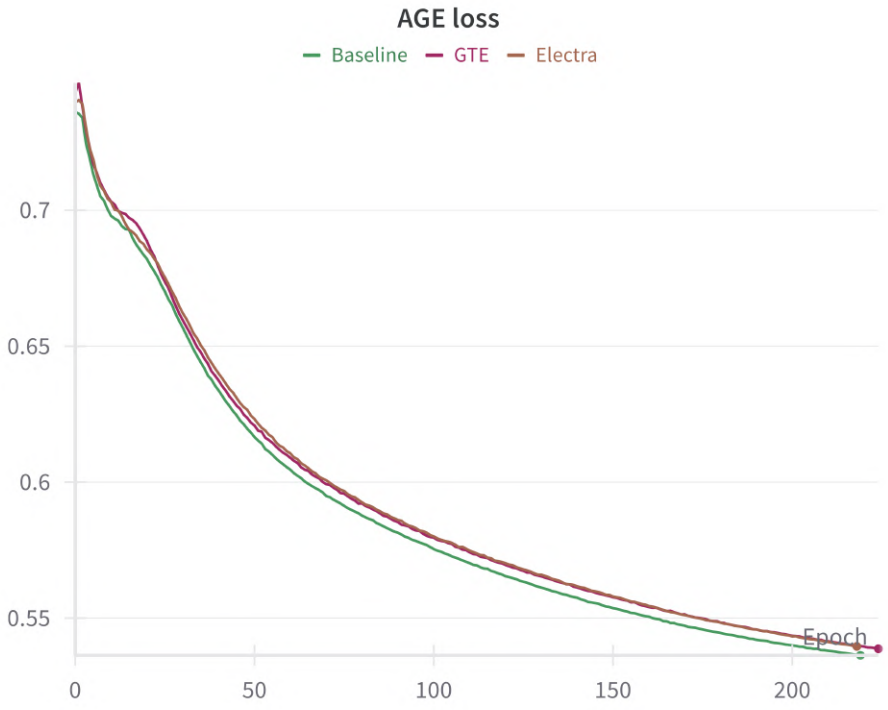

Figure 6.21: AGE Loss with different AGE batch size over the selected models

The core idea behind using a smaller batch size for AGE was to update the weights with lesser information with the intuition that this should force the model to update the weights in a way that makes it applicable to a more general input and not try to over-fit for a large number of examples. This should in theory also provide it with a better contrast between the positive and negative samples since it has to only use a smaller sample set.

We observe that the GTE target embeddings benefit more from a smaller AGE size as compared to Electra-base which shows more or less the same behavior as the baseline where it works with an AGE batch size of 512. We see in figure 6.23 and 6.22 that both average precision and area under the curve are better for GTE than Electra.

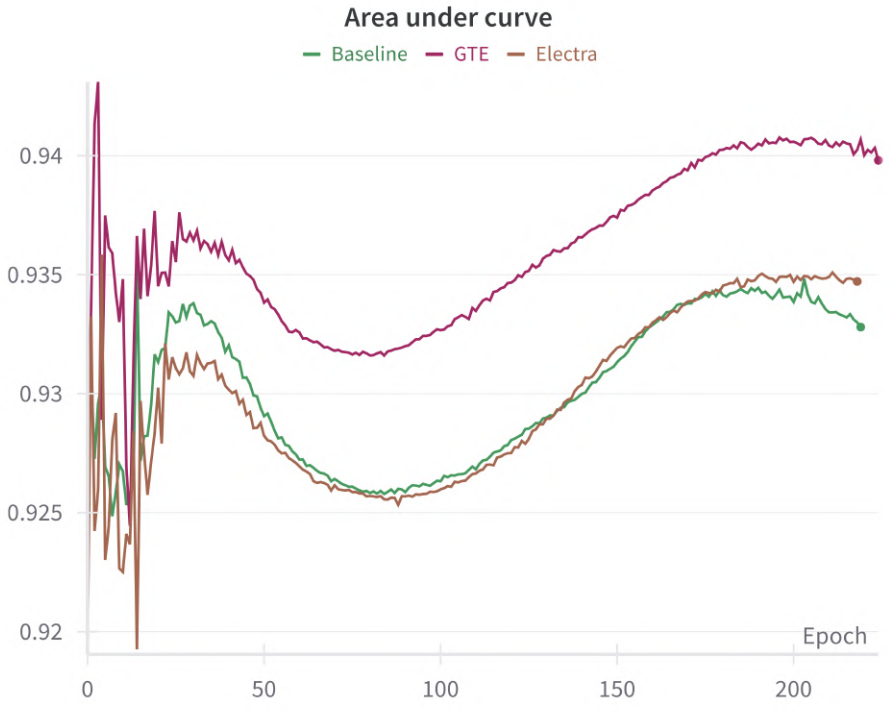

Figure 6.22: Area under curve with different AGE batch size over the selected models

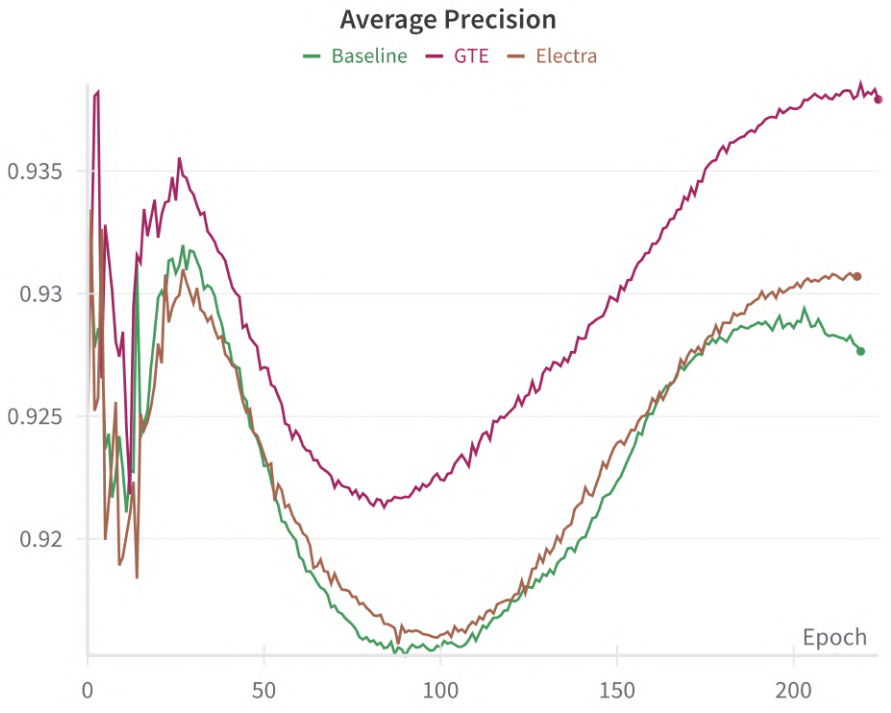

Figure 6.23: Average Precision with different AGE batch size over the selected models

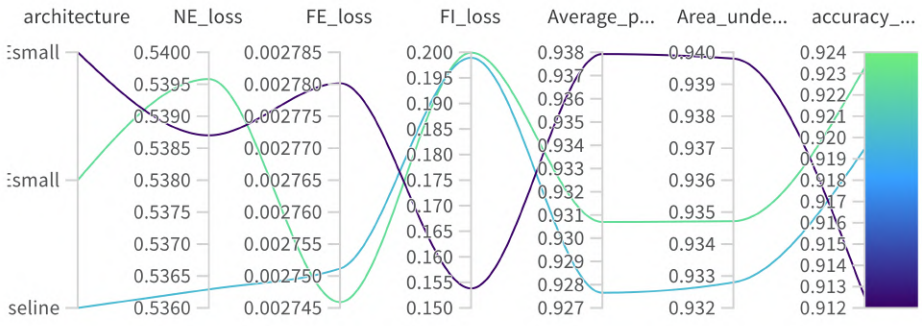

Figure 6.24: Sweep comparison of multiple measure for all different AGE batch size over the selected models

7. Conclusions

Following the set of experiments conducted during this work, we present our conclusions from the results obtained and also define the directions in which future work needs to be done.

7.1 Conclusions from Hyperparameter tuning

In the initial observations concerning hyperparameter tuning, several noteworthy trends emerge. Firstly, it appears that larger hidden sizes of word embeddings derived from pre-trained Language Models (PLMs) tend to exhibit a rapid adaptation to training data but demonstrate sub-optimal performance on the test set. This observation underscores the nuanced relationship between embedding size and model generalization. Additionally, experiments employing smaller batch sizes seem to impede the system's efficacy in performing Frame Element Labeling tasks. This phenomenon could stem from the inherent constraints imposed by reduced batch sizes on the model's learning capacity, thus warranting further investigation. Furthermore, an increase in Area Under the Curve (AUC) and Average Precision (AP) metrics is noted when employing higher-dimensional projections, albeit at the expense of increased parameter usage. This enhancement in performance suggests that projecting the target and AGE embeddings into a bigger vector space could facilitate more accurate similarity updating by mitigating information loss inherent in lower-dimensional projections.Regarding the utilization of word embeddings from various PLMs, these experiments yield outcomes that are either sub-optimal or comparable to the original configuration of the Frame Identification system. We think that this disparity in performance is due to the nature of the bigger PLMs like DeBERTa to generate hyper-contextualised embeddings. Another plausible reason for this observation could be the difference in the dataset used during the pre-training of these models.

It is important to acknowledge that the scope of this study is limited to evaluating model performance across a narrow range of hyperparameter values. To establish robust conclusions, future investigations should encompass a broader spectrum of hyperparameter configurations to facilitate generalisation of findings and enhance the comprehensiveness of insights garnered from this work.

7.2 Conclusions from changing Sentence Embedding

The performance metrics of Average Precision (AP) and Area Under the Curve (AUC) are significantly influenced by the characteristics of sentence embeddings. Notably, the Frame Identification (FI) loss demonstrates notable advantages when utilizing smaller-sized sentence embeddings, resulting in achieving a lower FI loss compared to the baseline model. In contrast, both General Text Embedding (GTE) model and DeBERTa exhibit a slower convergence of loss for the Adaptive Graph Encoder (AGE) in comparison to the baseline, while the miniLM model displays the highest convergence, albeit marginally surpassing the baseline model's convergence capacity. We attribute this divergence to the embedding size of the sentences, indicating the pivotal role of embedding dimensions in influencing the convergence behavior of the models. Furthermore, among various Pre-trained Language Models (PLMs) employed for sentence embeddings, a consistent pattern emerges in the behavior of the AP metric over time. However, training with sentence embeddings made by GTE deviate from this trend by inducing a hyperpaced learning phase in the initial 30 epochs, followed by an almost stagnation in improvements. The precise reasons for this distinctive behavior remain uncertain, and potential factors such as embedding size, training data variations used in the PLM's pretraining, or differing pretraining objective used by them can be seen as potential contributors to this observed phenomenon, warranting further investigation in future.

7.3 Conclusions from adding Dropouts

The utilization of a faster Learning Rate (LR) with D2 configuration manifests a noteworthy enhancement in system performance, particularly in Frame Element labeling tasks derived from predicted Frames for target words, resulting in a performance boosts of up to 15%. Remarkably, this enhancement does not compromise the system's Average Precision score or its high Area Under the Curve metric, thereby affirming the efficacy of this approach. However, the D2 configuration encounters challenges in converging to optimal loss values due to its rapid learning rate, often plateauing in predicting Frames. Notably, D2's dropout configurations demonstrate better performance, particularly when coupled with General Text Embedding (GTE) model's embeddings, suggesting a correlation with their larger hidden size. Conversely, D1.1 and D1.2 configurations exhibit comparable behavior to the baseline, albeit with prolonged convergence times. Interestingly, GTE embeddings with larger word embeddings tend to benefit more from the D1.1 and D1.2 dropout configurations compared to Electra embeddings, displaying lesser discrepancies in model accuracy across training and test datasets as compared to the training without the dropouts. These observations highlight the nuanced interplay between dropout configurations and the size of the embeddings amongst other things.

7.4 Conclusions from changing batch size of Adaptive Graph Encoder

Analyzing the impact of changing the batch size of the Adaptive Graph Encoder (AGE) yields several notable conclusions. Primarily, reducing the batch size exhibit minimal influence on test accuracy, with differences observed being negligible across the board. Interestingly, the embeddings from General Text Embedding (GTE) when trained with this smaller batch size of the Adaptive Graph Encoder (AGE) demonstrate a smoother and faster convergence when a smaller AGE size of 256 examples (half and half positive and negative samples) is employed, suggesting improved optimization efficiency. However, it is important to note that the Network Embedding loss for AGE experiences a slight delay in convergence and fails to attain the same level as the baseline. Moreover, both Area Under the Curve (AUC) and Average Precision (AP) metrics reap benefits from the implementation of a smaller AGE batch size, with GTE displaying more substantial improvements compared to Electra. These findings underscore the nuanced relationship between batch size configurations and model performance, particularly in terms of convergence dynamics and predictive accuracy across different embedding methodologies. Further exploration into the mechanisms underlying these observations could illuminate strategies for optimizing AGE configurations and enhancing the overall effectiveness the systems.

7.5 Scope for future works

There are multiple avenues open for future experimentation within the realm of improving the performance and to build a better understanding of the Frame identification system we work with. Firstly, altering the node embedding for the target words presents an avenue for further exploration as potentially diversifying the testing with other PLMs could strengthen the insights about the patterns observed in this work and possibly discover new ones. Additionally, fine-tuning the language model to be used for making the target word embeddings can be tailored to the nuances of the FrameNet corpora by using training objectives such as masked token prediction and Replaced Token Detection as they can render the model more adept at handling the specific textual characteristics Clark et al. [2020] inherent in FrameNet data. Multi-task training further offers a promising strategy for imbuing the model with robustness and adaptability across various linguistic tasks. Although this still suffers from the caveat of domain dependency.

Moreover, fine-tuning the Adaptive Graph Encoder (AGE) to better align with the structural complexities of the FrameNet graph holds potential for optimizing the encoder's efficiency and efficacy when used in the context of Frame identification. Furthermore, a comprehensive error analysis targeting problematic examples in Frame Identification tasks can provide valuable insights into the underlying challenges and provide more transparency about the limitations and allow for targeted improvements in system performance.

Bibliography

Bartlett, F. (1932). *Remembering: A Study in Experimental and Social Psychology*. Cambridge psychological library. Cambridge University Press.

Bastings, J., Titov, I., Aziz, W., Marcheggiani, D., and Sima'an, K. (2017). Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*.

Beck, D., Haffari, G., and Cohn, T. (2018). Graph-to-sequence learning using gated graph neural networks. *arXiv preprint arXiv:1806.09835*.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478. Springer.

Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Blackburn, P. and Bos, J. (2003). Computational semantics. *Theoria: An International Journal for Theory, History and Foundations of Science*, 18(1(46)):27–45. http://www.jstor.org/stable/23918435.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the kl-one knowledge representation system. *Cogn. Sci.*, 9:171–216.

Callan, J., Hoy, M., Yoo, C., and Zhao, L. (2009). Clueweb09 data set.

Chang, J., Wang, L., Meng, G., Xiang, S., and Pan, C. (2017). Deep adaptive image clustering. In *Proceedings of the IEEE international conference on computer vision*, pages 5879–5887.

Christopoulou, F., Miwa, M., and Ananiadou, S. (2019). Connecting the dots: Document-level neural relation extraction with edge-oriented graphs. *arXiv preprint arXiv:1909.00228*.

Chuang, Y.-S., Dangovski, R., Luo, H., Zhang, Y., Chang, S., Soljačić, M., Li, S.-W., Yih, W.-t., Kim, Y., and Glass, J. (2022). Diffcse: Difference-based contrastive learning for sentence embeddings. *arXiv preprint arXiv:2204.10298*.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Cui, G., Zhou, J., Yang, C., and Liu, Z. (2020). Adaptive graph encoder for attributed graph embedding. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 976–985.

Cui, Y., Che, W., Liu, T., Qin, B., and Yang, Z. (2021). Pre-training with whole word masking for chinese bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3504–3514.

Das, D., Chen, D., Martins, A. F., Schneider, N., and Smith, N. A. (2014). Frame-semantic parsing. *Computational linguistics*, 40(1):9–56.

Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227.

Dawkins, R. and Kraljevstvo, U. (1976). Sebični gen (the selfish gene). *Smederevo: Heliks*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Diao, S., Bai, J., Song, Y., Zhang, T., and Wang, Y. (2019). Zen: Pre-training chinese text encoder enhanced by n-gram representations. *arXiv preprint arXiv:1911.00720*.

Fernandes, P., Allamanis, M., and Brockschmidt, M. (2018). Structured neural summarization. *arXiv preprint arXiv:1811.01824*.

Fillmore, C. J. and Baker, C. (2009). A frames approach to semantic analysis. *Oxford Handbook of Linguistic Analysis*.

Fillmore, C. J., Johnson, C. R., and Petruck, M. R. (2003). Background to framenet. *International journal of lexicography*, 16(3):235–250.

FrameNet (2011). FrameNet. http://framenet.icsi.berkeley.edu. Last Accessed: 07-08-2023.

Gao, T., Yao, X., and Chen, D. (2021). Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

Gawron, J. (1985). Lexical representations and the semantics of complementation (valence),chapter-2.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288. https://aclanthology.org/J02-3001.

Goffman, E. (1974). *Frame analysis: An essay on the organization of experience.* Harvard University Press.

Gui, T., Zou, Y., Zhang, Q., Peng, M., Fu, J., Wei, Z., and Huang, X.-J. (2019). A lexicon-based graph neural network for chinese ner. 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 1040–1050.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Hartmann, S., Kuznetsov, I., Martin, T., and Gurevych, I. (2017). Out-of-domain FrameNet semantic role labeling. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 471–482, Valencia, Spain. Association for Computational Linguistics. https://aclanthology.org/E17-1045.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, P., Gao, J., and Chen, W. (2021). Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Huang, B. and Carley, K. M. (2019). Syntax-aware aspect level sentiment classification with graph attention networks. *arXiv preprint arXiv:1909.02606*.

Jia, R., Cao, Y., Tang, H., Fang, F., Cao, C., and Wang, S. (2020). Neural extractive summarization with hierarchical attentive heterogeneous graph network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3622–3631.

Jiang, T. and Riloff, E. (2021). Exploiting definitions for frame identification. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2429–2434.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kipf, T. and Welling, M. (2016). Variational graph auto-encoders. nips workshop on bayesian deep learning.

Lakoff, G. (1987). Cognitive models and prototype theory. In *Concepts and conceptual development: Ecological and intellectual factors in categorization*.

Lappin, S. (2005). 91 Semantics. In *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., and Zhang, M. (2023). Towards general text embeddings with multi-stage contrastive learning.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Luo, Y. and Zhao, H. (2020). Bipartite flat-graph network for nested named entity recognition. *arXiv preprint arXiv:2005.00436*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Minsky, M. (1974). A framework for representing knowledge.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2022). Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Ng, A., Jordan, M., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14.

Ni, J., Ábrego, G. H., Constant, N., Ma, J., Hall, K. B., Cer, D., and Yang, Y. (2021). Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Pan, L., Xie, Y., Feng, Y., Chua, T.-S., and Kan, M.-Y. (2020). Semantic graphs for generating deep questions. *arXiv preprint arXiv:2004.12704*.

Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. (2018). Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*.

Pappas, N. and Henderson, J. (2019). Gile: A generalized input-label embedding for text classification. *Transactions of the Association for Computational Linguistics*, 7:139–155.

Park, J., Lee, M., Chang, H. J., Lee, K., and Choi, J. Y. (2019). Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6519–6528.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations.

Popov, A. and Sikos, J. (2019). Graph embeddings for frame identification. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 939–948, Varna, Bulgaria. INCOMA Ltd. https://aclanthology.org/R19-1109.

Qu, M., Gao, T., Xhonneux, L.-P., and Tang, J. (2020). Few-shot relation extraction via bayesian meta-learning on relation graphs. In *International conference on machine learning*, pages 7867–7876. PMLR.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Rumelhart, D. E. (1975). Notes on a schema for stories. In *Representation and understanding*, pages 211–236. Elsevier.

Sachan, D. S., Wu, L., Sachan, M., and Hamilton, W. (2020). Stronger transformers for neural multi-hop question generation. *arXiv preprint arXiv:2010.11374*.

Sahu, S. K., Christopoulou, F., Miwa, M., and Ananiadou, S. (2019). Inter-sentence relation extraction with document-level graph convolutional neural network. *arXiv preprint arXiv:1906.04684*.

Sarlin, P.-E., DeTone, D., Malisiewicz, T., and Rabinovich, A. (2020). Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schank, R. C. and Abelson, R. P. (1975). Scripts, plans and goals. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence. IJCAI*, volume 1.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Semantic Analysis-web article (2021). "Understanding Semantic Analysis – NLP" is part of a series of articles on NLP published online by GeekForGeeks website.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Su, X., Li, R., Li, X., Pan, J. Z., Zhang, H., Chai, Q., and Han, X. (2021). A knowledge-guided framework for frame identification. 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 5230–5240.

Swayamdipta, S., Thomson, S., Dyer, C., and Smith, N. A. (2017). Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *arXiv preprint arXiv:1706.09528*.

Tamburini, F. (2022). Combining electra and adaptive graph encoding for frame identification. In *Proceedings of The 13th Language Resources and Evaluation Conference*, Marseille, France. European Language Resources Association.

Tan, S.-S. and Na, J.-C. (2019). Positional attention-based frame identification with bert: A deep learning approach to target disambiguation and semantic frame selection. *arXiv preprint arXiv:1910.14549*.

Tannen, D. (1993). *Framing in discourse*. Oxford University Press, USA.

Taubin, G. (1995). A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358.

Tsai, H., Riesa, J., Johnson, M., Arivazhagan, N., Li, X., and Archer, A. (2019). Small and practical bert models for sequence labeling. *arXiv preprint arXiv:1909.00100*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30. https://arxiv.org/abs/1706.03762.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019a). Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., and Zhang, C. (2019b). Attributed graph clustering: A deep attentional embedding approach. *arXiv preprint arXiv:1906.06532*.

Wang, C., Pan, S., Long, G., Zhu, X., and Jiang, J. (2017). Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 889–898.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Williams, A., Nangia, N., and Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR.

Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., Long, B., et al. (2023). Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16(2):119–328. https://arxiv.org/abs/2106.06090.

Xu, K., Wu, L., Wang, Z., Yu, M., Chen, L., and Sheinin, V. (2018). Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. *arXiv preprint arXiv:1808.07624*.

Xu, M., Li, L., Wong, D., Liu, Q., Chao, L. S., et al. (2020). Document graph for neural machine translation. *arXiv preprint arXiv:2012.03477*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Yao, L., Mao, C., and Luo, Y. (2019). Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377.

Yasunaga, M., Zhang, R., Meelu, K., Pareek, A., Srinivasan, K., and Radev, D. (2017). Graph-based neural multi-document summarization. *arXiv preprint arXiv:1706.06681*.

Zeng, S., Xu, R., Chang, B., and Li, L. (2020). Double graph based reasoning for document-level relation extraction. *arXiv preprint arXiv:2009.13752*.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., Zhang, K., Ji, C., Yan, Q., He, L., et al. (2023). A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.