Traceability Link Recovery for Relations in Natural Language Software Architecture Documentation and Software Architecture Models

Bachelor's Thesis of

Jianan Ye

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr.-Ing. Anne Koziolek
Second examiner:	Prof. Dr. Ralf H. Reussner
First advisor:	M.Sc. Dominik Fuchß
Second advisor:	M.Sc. Sophie Corallo

19. Juni 2023 – 19. October 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

PLACE, DATE

. . . . . .

(Jianan Ye)

Abstract

In software development, software architecture plays a vital role in developing and maintaining software systems. It is communicated through artifacts such as software architecture documentation (SAD) and software architecture models (SAM). However, maintaining consistency and traceability between these artifacts can be challenging. If there are inconsistencies or missing links, it can lead to errors, misunderstandings, and increased maintenance costs. This thesis proposes an approach for recovering traceability links of software architecture relations between natural language SAD and SAM. The approach involves the use of Pre-trained Language Models (PLMs) such as BERT and ChatGPT and supports different extraction modes and prompt engineering techniques for ChatGPT, as well as different model variants and training strategies for BERT. The proposed approach is integrated with ArDoCo, a tool that detects inconsistencies and recovers trace links between software artifacts. ArDoCo is used for pre-processing the SAD text and parsing the SAM, thus facilitating the traceability link recovery process. In order to assess the performance of the framework, a gold standard of SAD and SAM created from open-source projects is utilized. The evaluation shows that the ChatGPT approach has promising results in relation extraction with a recall of 0.81 and in traceability link recovery with an F1-score of 0.83, while BERT-based models struggle due to the lack of domain-specific training data.

Zusammenfassung

In der Softwareentwicklung spielt die Softwarearchitektur eine wichtige Rolle bei der Entwicklung und Wartung von Softwaresystemen. Sie wird durch Artefakte wie Softwarearchitekturdokumentation (SAD) und Softwarearchitekturmodelle (SAM) kommuniziert. Die Wahrung der Konsistenz und Nachvollziehbarkeit zwischen diesen Artefakten kann jedoch eine Herausforderung darstellen. Wenn es Inkonsistenzen oder fehlende Verbindungen gibt, kann dies zu Fehlern, Missverständnissen und erhöhten Wartungskosten führen. In dieser Arbeit wird ein Ansatz zur Wiederherstellung der Rückverfolgbarkeit von Softwarearchitekturbeziehungen zwischen natürlichsprachigem SAD und SAM vorgeschlagen. Der Ansatz beinhaltet die Verwendung von vortrainierten Sprachmodellen (PLMs) wie BERT und ChatGPT und unterstützt verschiedene Extraktionsmodi und Prompt-Engineering-Techniken für ChatGPT, sowie verschiedene Modellvarianten und Trainingsstrategien für BERT. Der vorgeschlagene Ansatz ist mit ArDoCo integriert, einem Werkzeug, das Inkonsistenzen erkennt und Trace-Links zwischen Software-Artefakten wiederherstellt. ArDoCo wird für die Vorverarbeitung des SAD-Textes und das Parsen des SAM verwendet, was den Prozess der Wiederherstellung der Traceability-Links erleichtert. Um die Leistungsfähigkeit des Frameworks zu bewerten, wird ein Goldstandard von SAD und SAM aus Open-Source-Projekten verwendet. Die Auswertung zeigt, dass der ChatGPT-Ansatz vielversprechende Ergebnisse bei der Beziehungsextraktion mit einem Recall von 0,81 und bei der Wiederherstellung von Rückverfolgbarkeitslinks mit einem F1-Score von 0,83 erzielt, während BERT-basierte Modelle aufgrund des Mangels an domänenspezifischen Trainingsdaten Schwierigkeiten haben.

Contents

List of Figures

List of Tables

1 Introduction

Software architecture is a crucial aspect of software engineering that improves the design and understanding of software systems [5]. There are different ways to describe software architecture, such as architecture documentation (SAD) and software architecture models (SAM). While SAD provides an informal, natural language description of the architecture, SAM gives a formal and graphical representation of the architectural elements and their relationships. Artifacts documenting software design decisions help to prevent software from deteriorating [19] but need to be maintained, as failure to properly maintain these artifacts can lead to inconsistencies and sometimes remain undetected, leading to increased costs [10]. However, maintaining consistency and traceability between artifacts can be a time-consuming task when manually updating each change that has been made, as finding all locations is challenging and made worse when occurrences are differently worded, which is a common occurrence causing inconsistencies [30] and therefore making it hard to use simple string searches. A solution to this is to use links that connect occurrences of entities across different artifacts, which are also called trace links. If the software has not been properly maintained, these trace links have to be recovered in a process called Traceability Link Recovery (TLR), which is sought to be automatic.

Figure 1.1 illustrates this process. On the left side, two different artifacts describing the architecture are shown. The artifact documentation is textual and written in natural language. In the first sentence it describes two entities, 'logic component' and 'sql datastore' highlighted in orange, and its relation 'is connected to' is highlighted in blue. The artifact model displays two components, Composite Component Logic and Basic Component SQL Datastore, which are connected by a blue line, describing the architecture from a graphical view. As both artifacts describe the same relation, it is wanted to create a link between these two occurrences. The tables on the right side are representations for each occurrence that can identify the location of each occurrence, containing the sentence, entities, relation, and type for the documentation and entities and relations for the model. The process is complete by connecting these two representations, shown in the figure as the trace link arrow.

This thesis presents an approach to address this problem of TLR for software architecture relations. It involves using pre-trained language models (PLMs) for relation extraction and integrates with a framework for recovering software architecture entity trace links. The foundation for this is laid out in Chapter 2, introducing natural language processing techniques, pre-trained language models, a modeling language called palladio component model, and a TLR framework architecture documentation consistency. This is followed by Chapter 3, which reviews the related literature in relation extraction and traceability link

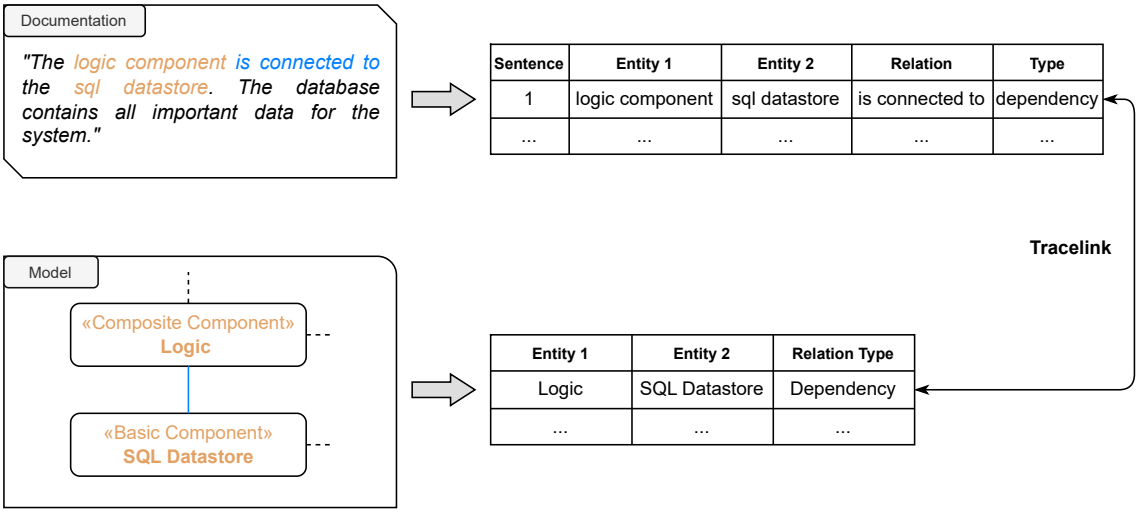


Figure 1.1: Traceability Link Recovery for Relations in SAD and SAM

recovery. Chapter 4 presents the general approach to relation extraction using ChatGPT, a conversational general-purpose PLM based on GPT [20], and BERT [3], a state-of-the-art language model for multiple natural language tasks. Chapter 5 presents the method using ChatGPT along with its implementation and the process of creating prompts. Chapter 6 presents the method using BERT and details the training process for the used BERT models. Chapter 7 then describes the representation for relations in model and documentation and the process of finding and connecting them. In Chapter 8, the different extraction modes and model variants used are evaluated, and the results are discussed. Finally, Chapter 9 concludes this thesis with challenges and limitations of the approach, as well as potential directions for future research.

2 Fundamentals

This chapter lays out the foundation for comprehending the key concepts discussed in this thesis. It encompasses two primary subjects: Traceability Link Recovery in Software Architecture Documentation and the utilization of Large Language Models for relation extraction. Section 2.1 offers an introduction to Natural Language Processing techniques, outlining the necessary steps for relation extraction. Section 2.2 introduces LLMs and provides an insight into two well-known models, GPT and BERT. Section 2.3 provides insights into the software architecture modeling language known as the Palladio Component Model. Section 2.6 describes the framework ArDoCo and its processing pipeline.

2.1 Natural Language Processing

Natural Language Processing (NLP) is a research field that focuses on how computers interact with humans using written or spoken natural language. NLP techniques are used in various TLR processes, including the automated analysis of textual artifacts such as documentation and source code. The NLP techniques relevant to this thesis include Part of Speech (POS) tagging (Subsection 2.1.1), Dependency Parsing (Subsection 2.1.2, and Relation extraction (Subsection 2.1.3).

2.1.1 Part of Speech Tagging

POS Tagging involves assigning grammatical categories, such as nouns, verbs, adjectives, and adverbs, to words in a sentence, aiding in understanding the syntactic structure of written pieces. Techniques like the Penn Treebank [18] POS tagging are frequently used in TLR to enhance the analysis of natural language text. An example is shown in Figure 2.1.

John	has	a	book	on	his	desk
NNP	VBP	DT	NN	IN	PRP$	NN

Figure 2.1: POS Tagging by Stanford CoreNLP

This figure uses POS tagging through Stanford CoreNLP to assign grammatical tags to the words in the sentence. The tags consist of Proper Nouns (NNP) for specific names, Verbs

(VBZ) for actions and states, Determiners (DT) for articles, Nouns (NN) for common objects, Prepositions (IN) for linking phrases, Possessive Pronouns (PRP$) indicating ownership, and Adjectives (JJ) for describing nouns.

2.1.2 Dependency Parsing

Dependency Parsing is a technique used for syntactic analysis of natural language sentences to determine their grammatical structure. It is particularly useful in TLR while analyzing textual requirements and source code. Dependency parsing helps identify relationships between words and phrases in sentences. Figure 2.2 shows an example of dependency parsing.

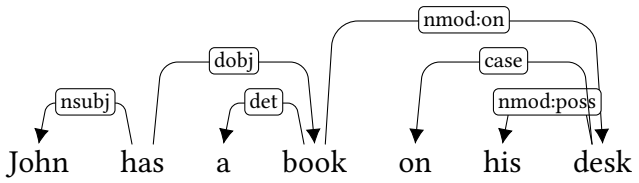


Figure 2.2: Dependency Parsing by Stanford CoreNLP

This figure shows how dependency parsing is used to identify the syntactic relationships between words in a sentence. The arrows connect the words to indicate their relationships. The "nsubj" tag identifies the nominal subject of a verb. In the example, "John" is the subject of the verb "has". The "dobj" tag identifies the direct object of a verb. In the given sentence, "book" is the direct object of the verb "has". The "det" tag indicates a determiner like an article or any word that specifies a noun. In the given sentence, "a" is the determiner for "book". The "nmod:on" tag represents a prepositional modifier that establishes a relationship between the words. In this context, it connects "book" and "desk," indicating that the "book" is positioned "on" the "desk". The "nmod:poss" tag indicates a possessive modifier, which shows ownership or association. In this sentence, it links "desk" and "his," revealing that the "desk" belongs to "him". Finally, the "case" tag reflects the case marking of a preposition, linking "desk" and "on" and conveying that "desk" is governed by the preposition "on".

2.1.3 Relation Extraction

Relation Extraction (RE) is the process of identifying and classifying semantic relationships between entities that are mentioned in the text. This thesis uses relation extraction techniques to detect textual connections between software architecture entities. The goal is to establish links between textual relations and the ones present in the model. An example is created using Stanford CoreNLP with OpenIE shown in 2.3.

Figure 2.3 displays two extracted relation triplets for a sentence. The first triplet consists of "John" (highlighted in blue) as entity 1, the relation "has" (highlighted in darker green),

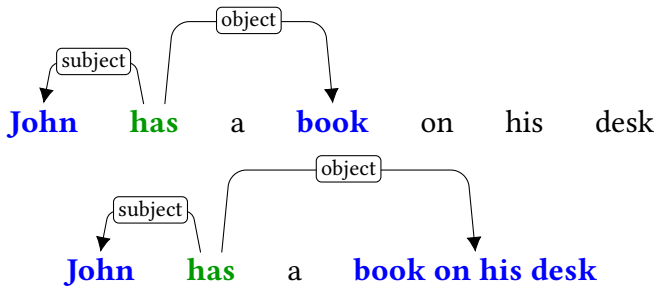


Figure 2.3: Relation Extraction by Stanford CoreNLP using OpenIE

and "book" (highlighted in blue) as entity 2. The second triplet involves the same entity 1 and relation but with a more complex entity 2, "book on his desk" (highlighted in blue). In this case, the relationship extends to the detailed context.

2.2 Large Language Model

Large Language Models (LLMs) represent a significant breakthrough in NLP and have revolutionized various NLP tasks and TLR. LLMs are built upon the transformer model and leverage self-attention mechanisms. This feature enables more efficient parallelization during training on text data compared to earlier neural network models [25], making unsupervised training on large volumes of text data more practical. LLMs used in fine-tuning are typically pre-trained on natural language before fine-tuning and are commonly referred to as pre-trained language models (PLMs).

2.2.1 Generative Pretrained Transformer

Generative Pretrained Transformer (GPT) is a well-known language model that is trained to predict the next word of a sentence, also known as completion. Models like GPT-3.5 and GPT-4 are general-purpose and can be used for a wide range of language-related tasks. ChatGPT, a conversational model based on GPT, is specially trained for conversations with humans. Users can interact with ChatGPT using natural language prompts. In the context of TLR, ChatGPT can be used for different NLP tasks, including information retrieval. It excels in Open Information Extraction (OpenIE) settings [14] and also demonstrates impressive performance using zero-shot approaches, even compared to fully supervised models [28].

2.2.2 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) is an LLM that has significantly improved the performance of several NLP tasks. BERT has the ability to

predict missing words in a sentence by considering the context from both left and right. This unique training method, also known as matching the blank, is particularly useful for relation classification tasks, and by analyzing the context and semantic meaning of a word in a sentence, BERT is able to understand its meaning better than previous models that only looked at text in one direction [22].

The thesis BERT approach relies on a framework[27] based on Soares et al. work titled "Matching the blanks: Distributional similarity for relation learning" [22]. The paper presents a novel approach to learning relation representations directly from text without using any predefined ontology or labeled training data. The process involves filling entity mentions within relation statements with a special symbol and training a relation encoder to predict whether or not two statements express the same relation. The relation encoder is built on top of the BERT model and uses entity markers and entity start states to represent relations. The paper evaluates the method for various relation extraction and classification tasks and shows that it achieves state-of-the-art results. It outperforms previous methods that use human-labeled data or knowledge graphs. The paper also shows that the method is effective in low-resource settings and can reduce the annotation effort required for creating relation extractors.

2.3 Palladio Component Model

The Palladio Component Model (PCM) is a modeling language used for component-based software architectures and helps predict the architecture's performance and reliability [1]. The model is based on the concept of CBSE roles, which involves four developer roles (component developer, software architect, system deployer, and domain expert) and their corresponding modeling languages. In this work, PCM is used as a modeling tool for the software model artifact. PCM can model a variety of component relations, but only the provided/required interfaces shown in Figure 2.4 and the composite component structures shown in Figure 2.5 are taken into account for this approach.

Figure 2.4 has two components and an interface. The Logic component provides the ILogic interface, which the UI component requires. The provided interfaces represent the services that a component offers to its environment, while the required interfaces represent the services that a component expects from its environment.

Figure 2.5 displays the composite component Logic containing an assembly context of the component UI and including their provided and required interfaces. The composition structure defines how components are instantiated and interconnected in a system architecture.

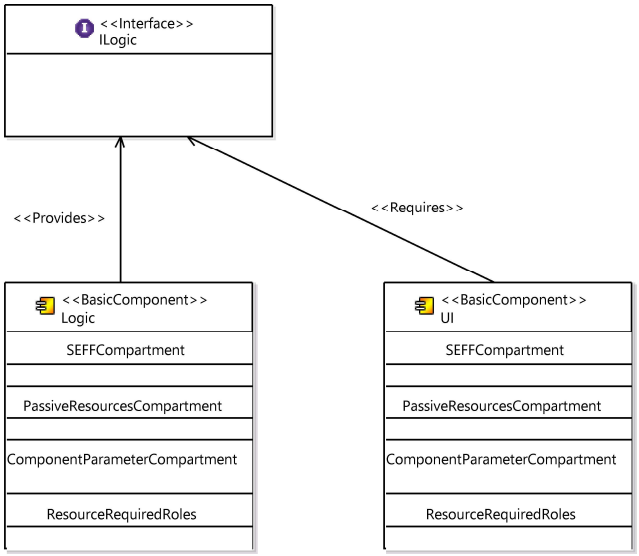


Figure 2.4: An Example of PCM Provided and Required Interface Relation

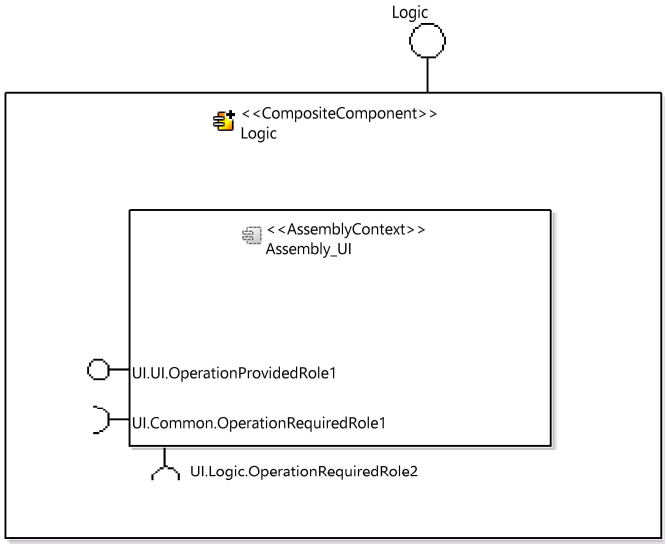


Figure 2.5: An Example of a PCM Composite Component

2.4 Architecture Documentation Consistency

Architecture Documentation Consistency (ArDoCo) [10] is a framework for inconsistency detection between SAD and SAM and uses an adaptation of SWATTR [11], a framework for TLR between natural language SAD and SAMs. ArDoCo can detect two kinds of inconsistencies, one being existing components in SAM not mentioned in the SAD, termed Unmentioned Model Elements (UMEs), and the other being mentions in SAD that are not part of the model, Missing Model Elements (MMEs). For detecting UMEs, it achieved an excellent accuracy of 0.93, while MME detection has a good accuracy of 0.75. An overview of ArDoCo's approach for TLR and inconsistency detection is shown in Figure 2.6.

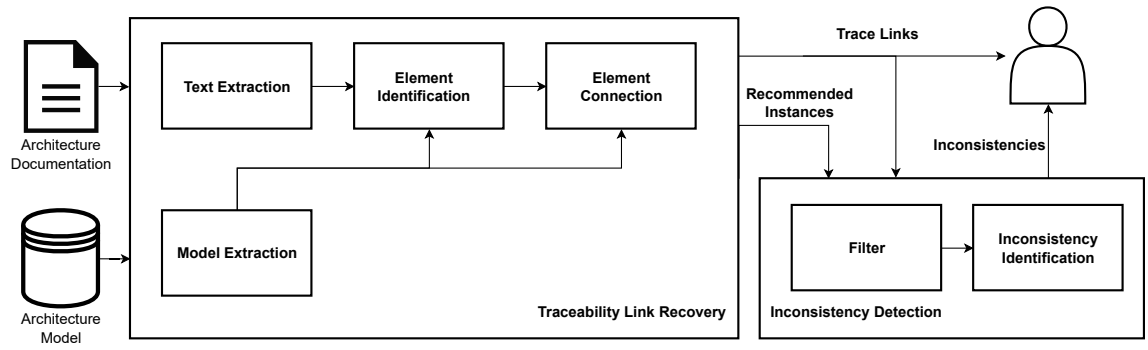


Figure 2.6: Overview of ArDoCo [10] for Traceability Link Recovery and Inconsistency Detection

Figure 2.6 depicts the two processes used in ArDoCo. On the left side is the trace link recovery pipeline adapted from SWATTR, and on the right is the process for inconsistency detection. The TLR part consists of four processing steps: text extraction, model extraction, element identification, and element connection. The pipeline first pre-processes the SAD using NLP techniques like POS tagging and dependency parsing. Then, the model extraction step parses the model data into an internal representation. The text extraction step then uses heuristics and agents to search for names and types of possible model elements, which are then grouped by similarity and given a confidence rating. In the next step, element identification, the confidence gets adjusted based on the element types of the component model and forms a recommended instance. Finally, the element connection step creates trace links by connecting RI to model instances based on string similarity. The inconsistency detection consists of a filtering and inconsistency identification step and builds upon the previously created trace links and recommended instances. The filtering step uses threshold, occurrence, and unwanted word filters to reduce the amount of false positives. After filtering out recommended instances the remaining ones are marked as inconsistencies.

This thesis builds upon ArDoCo and utilizes the pre-processed text, the parsed model, and the recovered trace links to apply TLR to relations.

3 Related Work

This chapter explores the literature related to the main topic of the thesis, which is establishing trace links between SAD and SAM entity relations. To achieve this, relationships from the software architecture domain need to be extracted. Two key research areas are relevant to this thesis. The first area is relation extraction, which is crucial for the success of this work. It is important to be familiar with state-of-the-art techniques, alternatives, and their advantages. The second area is automatic traceability link recovery, which is essential for establishing the connections between SAD and SAM entity relations. For both of these areas, LLMs are considered state-of-the-art and are used in this thesis. Therefore, prompt engineering techniques are also discussed.

3.1 Relation Extraction

This section reviews the literature on relation extraction and the benchmarks used for evaluating this task. Relation extraction is the process of identifying and categorizing the semantic relationships between entities in text data. Previous studies on relation extraction relied on pattern-based methods that used handcrafted linguistic rules and patterns [7, 9]. Recent advances in this field have incorporated deep-learning techniques and large-scale datasets. Neural networks such as CNN-based [32, 26] and RNN-based models [34, 13] are used to effectively capture complex contextual information. Pre-trained language models like BERT [2] and GPT [20] have been used to enhance the accuracy of relation extraction. BERT-based models [22, 24] have achieved the State-Of-The-Art (SOTA) in sentence-level relation extraction benchmarks like Semval 2010 task 8 [8]. This benchmark involves a sentence and two tagged nominals, where the relation and the direction of the relation are predicted. A dataset was created with nine semantic relations and an additional OTHER relation. For document-level relation extraction, the BERT-based models [33, 17] have achieved the SOTA in DocRED [31]. This dataset includes document-level relation extraction tasks and benchmarks. It is constructed using Wikipedia and Wikidata and has human-annotated coreference, intra-sentence, and inter-sentence relations. Tang et al. [23] have explored relation extraction in the software engineering domain using BERT to pre-train on a dataset created from user-generated Stack Overflow content. This work is similar to the relation extraction part of this thesis but is not specific to the software architecture domain.

3.2 Traceability Link Recovery

This section provides an overview of the current research on automated traceability link recovery in software engineering. Over the years, various approaches have been suggested, such as similarity-based methods that use information retrieval (IR) to evaluate the similarity between artifacts. Nevertheless, these methods are still far from achieving automated recovery of links [16]. Deep learning and neural networks have introduced powerful techniques for traceability link recovery. RNN-based methods [6] have demonstrated the ability to learn complex patterns and representations from software artifacts, showing significant promise in this field. BERT-based models have improved the accuracy of natural language and code artifact traceability compared to classical IR and RNN techniques. Lin et. al [15] presented a BERT-based model that generates trace links between natural language and programming language artifacts using pre-trained BERT. All these approaches applied TLR to natural language documentation and programming language, which is a different problem than TLR between SAD and SAM. This problem has been worked on by Keim et. al. [11] but is not applied to relations.

3.3 Prompt Engineering

In utilizing conversational language models, such as ChatGPT, prompt engineering plays a vital role in enabling consistent and efficient utilization. In this thesis, ChatGPT is used for the task of relation extraction, which can be achieved through zero-shot prompting. Wei et. al.'s work [28] demonstrated the effectiveness of frameworks based on ChatGPT in information extraction tasks, even surpassing some supervised models. The paper introduced ChatIE, an information extraction framework based on ChatGPT, which transforms the information extraction task into a multi-turn question-answering problem with a two-stage framework. The work by White et al. [29] provides a catalog of prompt patterns that can solve common problems occurring in conversational language models. Each pattern is explained in detail and structured into name and classification, intent and context, motivation, structure and key ideas, an example, and the consequences of it. Some of these patterns are applicable to this thesis. For example, the "Template Pattern" allows for defining a consistent structure for the output, while the "Meta Language Pattern" and the "Persona Pattern" enable the setting of the context for a software architecture model documentation domain. Additionally, when creating prompts, it is necessary to consider further constraints for valid relation types. The presented prompts are general examples and are not applied to the domain of SAD, therefore this approach has to create prompts for the ChatGPT method of relation extraction.

4 Relation Extraction for Software Architecture Documentation

This chapter presents the ideas for using LLMs for relation extraction in SAD. The general approach is presented in Section 4.1. Section 4.2 shows how entities are processed from an ArDoCo result. Section 4.3 describes the result that the LLM relation extraction process delivers.

4.1 General Approach

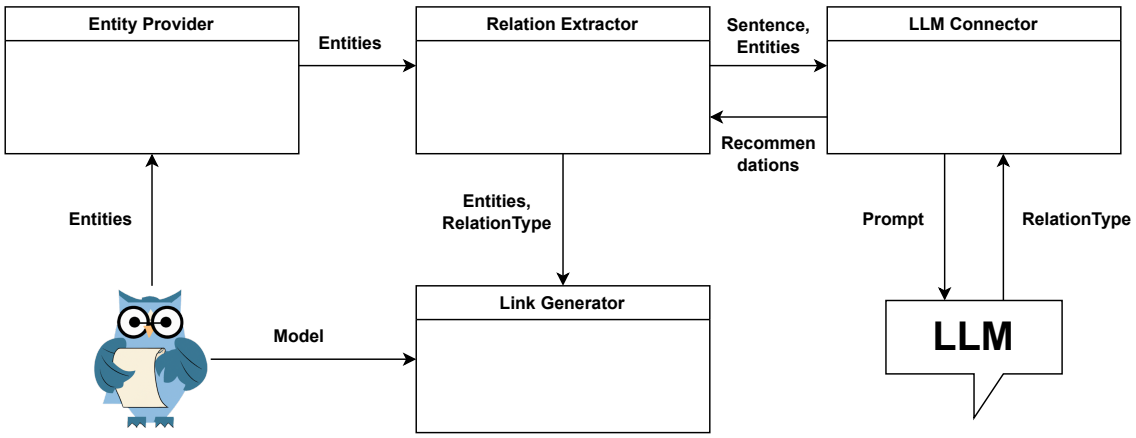


Figure 4.1: General Approach for Implementation

Figure 4.1 shows the general approach for relation extraction from SAD. The approach uses ArDoCo for creating entity trace links and LLMs for relation extraction from SAD. The implementation connects the elements and performs the following tasks: First, it processes text into a format that can be passed to an LLM. The framework must provide these if the LLM needs configuration data or other input forms like prompts. Then, the output from the LLM is filtered for invalid results and processed into an object containing all the information needed to create trace links. Finally, the extracted relations need to be connected to their model counterpart.

The implementation comprises four main components: Relation Extractor, Entity Provider, LLM Connector, and Link Generator. There are two external components: ArDoCo, depicted as the blue owl, and LLMs, shown in the speech bubble. The Relation Extractor

is the starting point and central component for relation extraction. It connects the other parts and is responsible for pre-processing the input data and post-processing the output data of the LLM. The Entity Provider connects ArDoCo to the relation Extractor and processes the data needed from ArDoCo results. This component receives project data from ArDoCo, which includes software architecture documentation and the found entity trace links. Entities extracted with ArDoCo are instances of software architecture elements and are components, packages, and objects [21] for which a trace link exists. The LLM Connector is responsible for communication between the implementation and the LLM. It provides the necessary tools for accessing the LLM. If the LLM is accessed through a web service, these tools would include functions for sending HTTP requests and the configuration data needed to set up the prompt. The output received from the LLM is then built into a Recommendation. Afterward, the Recommendation is passed to the Relation Extractor for further processing. A Recommendation is an object containing information about the extracted relation. This includes the entity pair, the classified relation type, a textual representation of the relation, and possibly a double value for the probability of this result. Relation types for this framework are Dependency, Containment, or None, as shown in Table 4.1. The Invalid type is set in post-processing for faulty relations.

Relation Type	**Description**
Dependency	Entity is reliant on another entity
Containment	Entity is part of another entity
None	Entities are not in a relation, or the type is neither dependency nor containment.

Table 4.1: Relation Types for this framework

4.2 Entity Provider

This section presents the Entity Provider, which has the responsibility to connect the Relation Extractor to ArDoCo. It is mainly used for running ArDoCo and processing the resulting entity trace link into an Entity object. In addition, it has some utility functions for Entities.

As shown in Figure 4.2, the class contains the attribute ArDoCoResult. This attribute is used to save the result calculated by ArDoCo for a given project to avoid multiple calculations of the same project. The functions shown are the most important ones of this class. The first two functions use ArDoCo to get Entities mapped to their sentence number of the given project. The first function receives its entities from TraceLinks, while the second function obtains them from Recommendations. The difference between the two functions is that one has a counterpart in the model, while the other iindicates a possible missing model entity. The third function is used to find a model ID for an Entity based on string similarity. The fourth function builds unique pairs out of a list of WordEntity.

EntityProvider
result: ArDoCoResult
getTracelinkEntities(Project, double, int) : Map<Integer, List<Entity» getRecommendedEntities(Project, double) : Map<Integer, List<Entity» getSimilarEntityId(List<WordEntity>, Entity) : String getUniqueEntityPairs(List<WordEntity>) : List<Pair<WordEntity, WordEntity» ...

Figure 4.2: UML Class for Entity Provider

4.3 Representing Relation Extraction Result

This section displays how the extracted results are modeled. A result extracted from either GPT or BERT approach is modeled as a Recommendation, shown in Figure 4.3.

The class Recommendation comprises of two entities of type Entity, a relation of type Relation, and a probability for the likeliness of this prediction with the type double. An Entity has a name for the occurrence in text and an ID to identify the component it represents in the model. A Relation extends Entity and has, in addition, the attributes relationType of type RelationType and direction of type RelationDirection. RelationType is an enum with the entries DEPENDENCY, CONTAINMENT, and NONE. RelationDirection is an enum with E1_E2 and E2_E1 to indicate the order of direction of the relation.

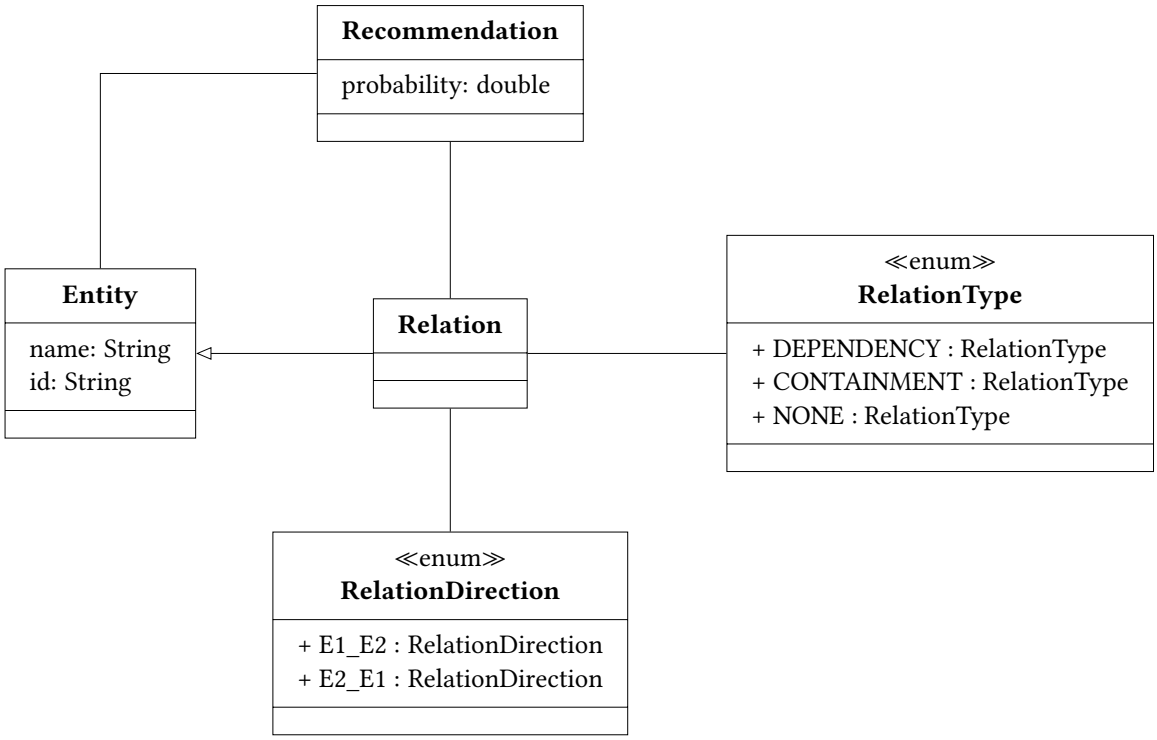


Figure 4.3: UML Model for Extracted Result

5 Relation Extraction with ChatGPT

In this chapter, we will discuss the concept of extracting relations SAD using GPT. An overview of this approach is presented in Section 5.1, while Section 5.2 covers the implementation behind the approach. Additionally, Section 5.3 details the journey of how the prompts used in this approach are built.

5.1 Overview

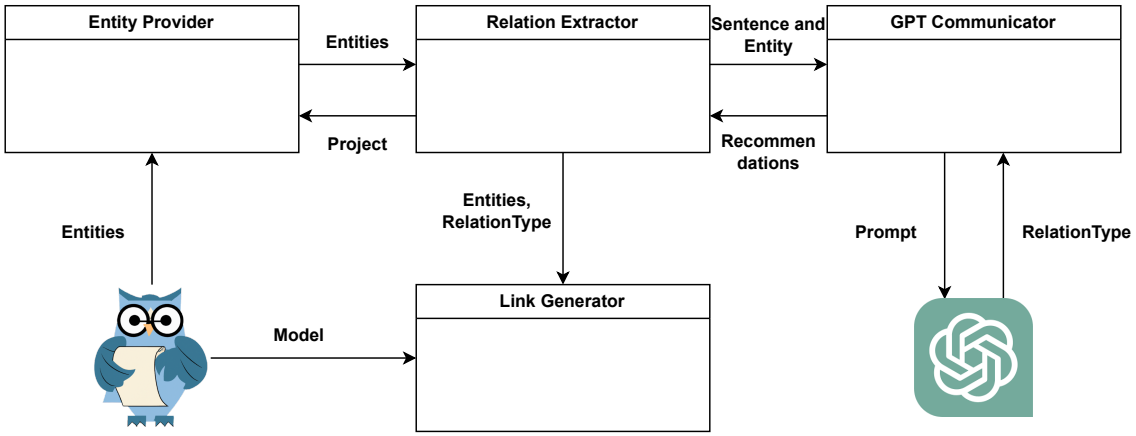


Figure 5.1: Data Flow Diagram GPT Approach

Figure 5.1 presents an overview of the concept, which involves four components implemented in this work and two external ones. The implemented components are the Relation Extractor, Entity Provider, GPT Communicator, and Link Generator. They are depicted with a rectangle and flow in the enumerated direction. The external components are represented by the blue owl icon (ArDoCo) and the green icon (GPT API).

The approach uses the Relation Extractor as the starting point and manages the other components. First, the Entity Provider is called with a project parameter to get the entities of each sentence of that project from ArDoCo. Then, the entities and the sentence are given to the GPT Communicator to send a request to the GPT API. Finally, the response containing the entities, relation, and relation type is returned and given to the Link Generator to create trace links.

Relation Type	Description
Dependency	Entity is reliant on another entity
Containment	Entity is part of another entity
None	Entities are not in a relation, or the type is neither dependency nor containment.

Table 5.1: Relation Types for this framework

5.2 Implementation

Figure 5.2 shows a more detailed variation of Figure 5.1. The classes part of the previously introduced components are Relation Extractor, Entity Provider, GPT Communicator, Serializer, Filter, Link Generator, and the enum Prompt Templates. The implementation of the Relation Extractor, GPT API Client, and Filter will be explained in the following section. The other classes are explained in Chapter 4 for Entity Provider and Chapter 7 for Link Generator.

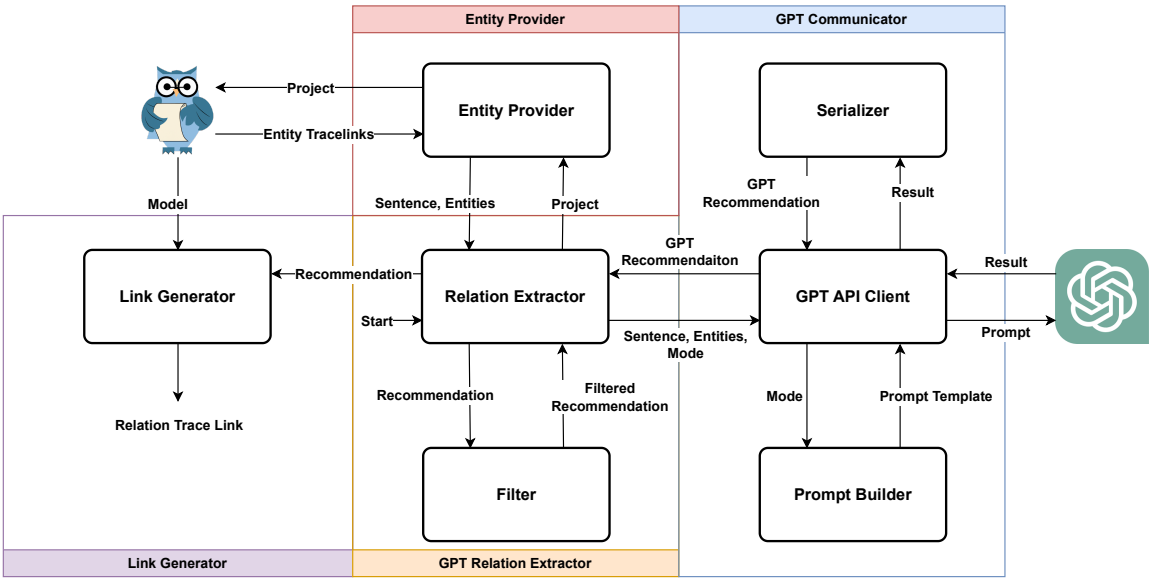


Figure 5.2: Data Flow Diagram for GPT Classes

The relation extraction process begins with initializing the Relation Extractor using an ArDoCo project. The project is an enum for ArDoCo benchmark projects. The extractor is then called with an extraction mode. The available extraction modes are entity recognition mode, entity set mode, and entity pair mode, which are explained in the following section. Next, the Entity Provider is called with the project. The Entity Provider uses ArDoCo to calculate the entity trace link of that project. It maps a list of entities and sentences to their sentence number and returns them to the Relation Extractor. The entities extracted using ArDoCo are instances of software architecture elements, such as components, packages, and objects [21].

In the next step, the GPT Communicator is called with the entities, sentence, and extraction mode. It then builds the entities and associated sentences into a prompt template using the Prompt Builder. The prompt template used depends on the mode chosen. Afterward, a request is sent with GPT parameters, shown in Table 5.5, to the GPT API. The received response is in a format set by the prompt and contains two entities, their relation, and the relation type.

Next, the data from the response is parsed into a GPT Recommendation and is returned to the Relation Extractor. It uses the Filter to filter out invalid results. This is done by comparing the entity list with the received results. Results that have entities that are not in the entity list will be marked as invalid. The valid Recommendation receives the model ID from the entities of the entity list.

5.2.1 Relation Extraction

To extract a relation, the class GPTRelationExtractor can be used as a starting point. The class is illustrated in Figure 5.2.1 and can be instantiated with four parameters, explained in Table 5.2. Once the GPTRelationExtractor is instantiated, the public function extractRelation can be called with an ExtractionMode enum, described in Table 5.3. The function initially calls the EntityProvider (described in Section 4.2) to obtain a list of entities that are mapped to their respective sentence numbers. This entity list is then passed to the function getEntitiesStringByMode, which returns a List of entity Strings mapped to their sentence number. This function builds unique entity pairs if the mode ENTITY_PAIR is selected. Next, the function extractRelationWithGPT is called with the sentence, entities, and ExtractionMode. This function goes through each map entry, calling the GPTAPIClient through the function useGPTClientByMode. It then serializes the returned String with GPTSerializer into a Recommendation. The resulting List of Recommendations, mapped to their respective sentence numbers, is then returned. In the last step, Recommendations are given to GPTFilter, which removes any Recommendations missing entities, that cannot be assigned a relation type, or have no entity counterpart in the model.

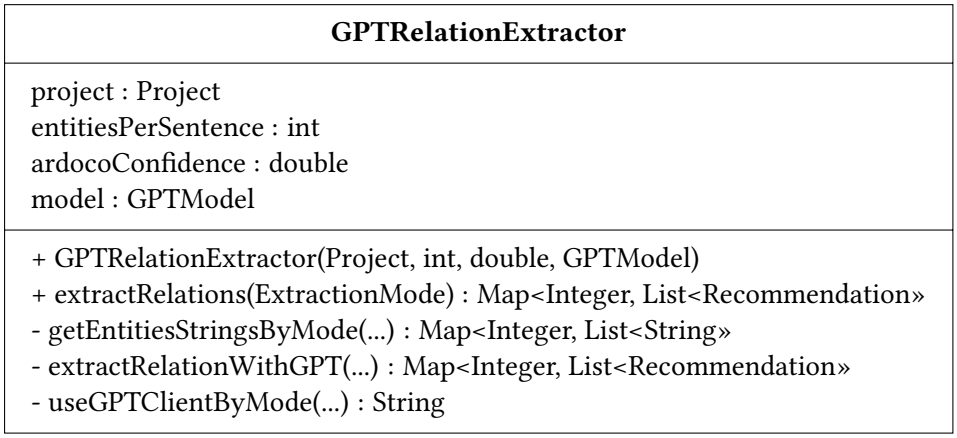


Figure 5.3: UML Class for GPT Relation Extractor

The instantiation of GPTRelationExtractor takes four parameters that are shown in 5.2. It takes the parameters: ArDoCo Benchmark project, an integer entitiesPerSentence, a double for the confidence of the entities, and an enum GPTModel.

A Project contains the text file for the documentation and model files as UML or PCM repositories. Possible projects are JABREF, MEDIASTORE, BIGBLUEBUTTON, TEASTORE, and TEAMMATES. The integer entitiesPerSentence determines how many valid entities are needed to put the sentence and its entities into the map to be extracted. Valid entities are entities for which a trace link has been recovered using ArDoCo. The double ardocoConfidence is the threshold of the confidence of the recovered trace link by ArDoCo. The enum GPTModel determines which GPT model is used for relation extraction. There are two possible models GPT-3.5-turbo and GPT-4.

Type	Name	Possible Value
Project	project	{JABREF, MEDIASTORE,..}
int	entitiesPerSentence	[0,..]
double	ardocoConfidence	[0,1]
GPTModel	model	{GPT_3_5_TURBO, GPT_4}

Table 5.2: GPTRelationExtractor Constructor Parameters

In Table 5.3, the different extraction modes are displayed. Each mode has unique prompts that request information from the GPT API and fill out the prompt with a different amount of entities. A description of the prompts used for each mode can be found in Section 5.3.

The ENTITY_RECOGNITION mode only sends the sentence where relations are sought to the GPT API. This allows the GPT to determine which entities to extract relations from. In the ENTITY_PAIR mode, GPT is restricted to only two entities for which the relation has to be determined. The ENTITY_SET mode is a middle ground between the two previous modes. It provides GPT with a set of entities that restricts the search, allowing GPT to determine for which entities of the set relations are extracted.

ExtractionMode	Description
ENTITY_RECOGNITION	GPT decide which entities are used to extract relations.
ENTITY_SET	GPT extract the relation from the given entity set.
ENTITY_PAIR	GPT extract the relation from the given entity pair.

Table 5.3: EntityMode Enums and Description

5.2.2 GPT Communicator

The GPT Communicator component consists of three classes: GPTAPIClient, GPTSerializer, and the enum GPTPromptTemplates. GPTAPIClient, shown in Figure 5.4, is initialized with the GPTmodel parameter, an integer value for max tokens, and a double value for temperature, which are explained in Table 5.4.

The GPTAPIClient is responsible for sending requests to the GPT API. It has three functions for setting up prompts for the entity extraction modes: gptEntityRecognition, gptEntitySet, and gptEntityPair. Each of these functions takes a sentence as a String and, except for the entity recognition mode, entities as a String. They use GPTPromptTemplate to get the prompt template for the assigned mode and fill in the sentence and entities. The filled prompt, along with the initialized attributes and API token, are then built into a JSON and sent to the API endpoint using the sendRequest function. The parameters used are described in Table 5.4. After receiving a response, it is serialized, and the completion content for the prompt is returned.

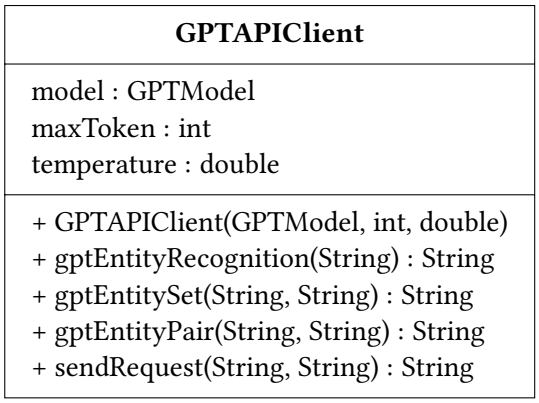


Figure 5.4: UML Class for GPT API Client

To set up GPT, each request sent to the GPT API requires four parameters, which are shown in Table 5.4. The model parameter determines the GPT model used for completion. The max_token parameter sets the maximum number of total tokens allowed, which includes prompt tokens along with completion tokens. If the completion goes beyond this value, the response will be truncated. Typically, a token in English consists of about four characters or 0.75 words[1]. The temperature parameter controls the randomness of the response. A value of 0.0 means little randomness, resulting in an almost deterministic response to the same prompt, while a value of 1.0 has the highest randomness. The message parameter includes the prompt that GPT needs to complete.

Parameter	**Description**
model	The GPT model
max_tokens	Max amount of prompt and completion tokens
temperature	Randomness of the completion
message	The prompt message to complete

Table 5.4: GPT API Parameter

[1]https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them

5.3 Prompting

Crafting effective prompts is vital to achieving a good performance with the GPT concept. This section will discuss the process of creating prompts from the initial iteration to the final prompts. Creating prompts that align with the requirements of this framework involves an iterative process of experimentation and adaptation. The development of prompts is explored to provide insights into successful prompts and those that did not yield the intended results. For the best results, a prompt has to be precise. This can prevent misinterpretation and variations in the responses. In addition, a prompt should also be short, as longer prompts require more tokens, which has a higher cost.

The ideas behind GPT for relation extraction are similar, and it is done by having a program send requests to the ChatGPT API containing a prompt and then using the received information to create trace links. There are various methods for configuring the model and writing a prompt. In this section, the focus is on prompting. All answers given by GPT use the parameters shown in Table 5.5.

Parameter	Value	Description
max_token	300	The allowed amount of completion tokens
temperature	0.0	The Randomness of the completion
model	gpt-3.5-turbo	The model used for completion

Table 5.5: Request parameters for GPT API

The max_token parameter is mainly used to keep each request's cost and response time low. For this program, the value of 300 is chosen. The temperature parameter is the randomness of the received completion. The value 0.0 is chosen as no randomness is wanted. The value 0.0 is near deterministic. This means that the response should be the same for the same prompt sent to GPT. The model parameter defines the GPT variant used for the completion. The option gpt-3.5-turbo is chosen as a cost-efficient chat model, and it allows a maximum amount of 4096 completion tokens, which is more than needed.

5.3.1 First Iterations

For the first iterations, a simple prompt asks for the relation of a given sentence. In Listing 5.1, an example is given for the sentence "The logic component includes the SQL datastore".

```
Give me the relation of the following sentence "The logic
   component includes the SQL datastore".
```

Listing 5.1: Example for a Simple Prompt

In this case, GPT responds with a sentence in natural language, seen in Listing 5.2. The response GPT gives suggests that it has understood that SQL datastore and logic components are in a composite relation. But as the response is in natural language, it would require natural language processing to make it usable.

```
The sentence suggests that the SQL datastore is a part or
   component of the logic component.
```

Listing 5.2: GPT Response to a Simple Prompt

Using templates is a method for solving the problem of an unstructured response. A template can specify which information is needed and how it should be formatted. The data required from GPT are the entities, the relation, and the classified relation type. Listing 5.3 shows these data filled into a template. This template uses the JSON format and has placeholder variables in caps. In addition, a short sentence asking GPT to respond in JSON instead of natural language is added.

```
Give me the relation of the following sentence
"The logic component includes the SQL datastore".
Please answer using only the provided template.
{
 "entity1": ENTITY1,
 "entity2": ENTITY2,
 "relation": RELATION,
 "relationType": RELATIONTYPE
}
```

Listing 5.3: Prompt with JSON Template

The response from GPT is now in the defined format as seen in Listing 5.4. The entities and relations have been recognized, even without specifying the corresponding placeholder. But as there is no description for relation type, GPT completes it with "is included in".

```
{
    "entity1": "logic component",
    "entity2": "SQL datastore",
    "relation": "includes",
    "relationType": "is included in"
}
```

Listing 5.4: GPT Response with JSON Templates

A sentence describing it is added to let GPT understand what is wanted for relationType. The relation types that GPT should classify into are dependency, containment, and none. The improved prompt can be seen in Listing 5.5.

```
Give me the relation of the following sentence
"The logic component includes the SQL datastore".
```

```
The relationType is restricted to only "dependency",
"containment", and "none".
Please answer using only the provided template.
{
    "entity1":      ENTITY1,
    "entity2":      ENTITY2,
    "relation":     RELATION,
    "relationType": RELATIONTYPE
}
```

Listing 5.5: Prompt with Defined Relation Types

The result GPT gives (see Listing 5.6) is now usable for relation recognition and classification. But as this sentence is trivial, the prompt has to be modified for more advanced sentences.

```
{
    "entity1": "logic component",
    "entity2": "SQL datastore",
    "relation": "includes",
    "relationType": "containment"
}
```

Listing 5.6: GPT Response with Defined Relation Types

5.3.2 Advanced Cases

This subsection discusses more advanced sentences and improvements for the prompt. The cases in this section include:

1. Sentence without a SA relation.
2. Sentence with multiple SA relations.
3. Sentence with SA and non-SA relations.

For the Case 1, the sentence **"The software architecture document describes the overall structure of the system"** is used as a negative case with neither a dependency nor containment. With this sentence put into the prompt in Listing 5.5, GPT returns the response shown in Listing 5.7. GPT classified the relation "describes" as a dependency, even though the sentence appears to provide a general description rather than specifying a direct relationship.

```
{"entity1": "software architecture document",
    "entity2": "overall structure of the system",
    "relation": "describes",
    "relationType": "dependency"}
```

Listing 5.7: GPT Response for Negative Test

Many additions and modifications to the prompt have been tried such as "**If the relation is neither a dependency nor a containment, classify it as "none**", but all have been unsuccessful in having the negative case classified as relationType 'none'. The problem has its roots in the naming of the relations. The relation names are not mutually exclusive, so GPT tends to go for positive cases rather than negatives. Combined with the type 'none' being able to be interpreted as having no relation between the entities, it led to the classification of the type 'dependency' and, in some cases, 'containment'. This led to many failed attempts for an actual negative case. The solution to this problem is to rename the negative case 'none' to 'other', which GPT could better interpret as neither dependency nor containment. Changing the previously created prompt shown in Listing 5.5 from 'none' to 'other', yielded the true negative result shown in Listing 5.8.

```
{
    "entity1": "software architecture document",
    "entity2": "overall structure of the system",
    "relation": "describes",
    "relationType": "other"
}
```

Listing 5.8: GPT Response for Negative Test with 'Other'

For Case 2, sentences can contain multiple SA relations. This needs modification to the prompt to support results that can contain more relations. A sentence was added that asks for all results to be returned. Furthermore, the JSON template was modified to an array to allow the result to be parsed. The modified template is shown in Listing 5.10. Listing 5.9 shows an example sentence containing two components, one package and one class instance entity, which are connected by a dependency and a containment relation.

```
In the software architecture, the PaymentGateway component
relies on the CreditCardProcessor for payments, while the
OrderManagement package contains order management classes".
```

Listing 5.9: Example Sentence with Multiple Relations

```
Give me the relation of the following sentence
"[The Sentence]"
The relationType is restricted to only "dependency",
"containment", and "other".
Please answer using only the provided template, and
if multiple results are available, include all of them.
[{
    "entity1":      ENTITY1,
    "entity2":      ENTITY2,
    "relation":     RELATION,
```

```
    "relationType": RELATIONTYPE
}]
```

Listing 5.10: Relation Extraction Entity Recognition Mode

The output from GPT, shown in Listing 5.11, contains an array with two JSON objects. It contains both relations in the sentence from Listing 5.9 and is correctly classified.

```
[{
    "entity1": "PaymentGateway",
    "entity2": "CreditCardProcessor",
    "relation": "relies on",
    "relationType": "dependency"
},
{
    "entity1": "OrderManagement",
    "entity2": "order management classes",
    "relation": "contains",
    "relationType": "containment"
}]
```

Listing 5.11: GPT Response for Multiple Relations Example

In Case 3, a sentence is given that fulfills both requirements of the previous cases. Which contains multiple relations including the negative type Other and can be seen in Listing 5.12.

```
The CustomerManagement package contains customer data
classes, the OrderProcessing component relies on it for
customer information, and the BillingSystem interfaces
with external payment gateways.
```

Listing 5.12: Example Sentence with Multiple Positive and Negative Relation Cases

The sentence shown in 5.12 is put into the prompt in Listing 5.10 and gives the following result shown in Listing 5.13. The first extracted relation is a trivial case that has been correctly extracted. The last relation is a true negative. While the second relation is not trivial as CustomerManagement is referred to using an "it". This insight motivates further prompts for cross-sentence relation extraction in Section 5.3.4. With this case complete, a prompt was found that can fulfill the relation extraction task for more advanced cases.

```
[{
    "entity1": "CustomerManagement package",
    "entity2": "customer data classes",
    "relation": "contains",
    "relationType": "containment"
},
{
```

```
    "entity1": "OrderProcessing component",
    "entity2": "CustomerManagement package",
    "relation": "relies on",
    "relationType": "dependency"
},
{
    "entity1": "BillingSystem",
    "entity2": "external payment gateways",
    "relation": "interfaces with",
    "relationType": "other"
}]
```

Listing 5.13: GPT Response for Multiple Positive and Negative Relation Cases

5.3.3 Prompt Variations

This subsection discusses prompt variations for different GPT relation extraction approaches. These variations are used to create different modes of extraction. The modes are listed in 5.3.3. Each mode has its purpose.

Mode 1 lets GPT do the recognition of the entities in text, and only the sentence is provided. This mode is useful for finding relations of entities that haven't been recognized by ArDoCo. The previously crafted prompt shown in Listing 5.10 is using this mode for relation extraction.

1. Relation extraction using entity recognition.
2. Relation extraction with a set of entities given.
3. Relation extraction with entity pairs given.

For the second Mode 2, a set of entities is given. The relations to look for should only be of those entities. This mode is particularly useful when using entities provided by ArDoCo, as these entities have a counterpart in the model, which is needed for creating trace links. From here on, the JSON template is using a placeholder for better readability.

```
Give me the relation of the following sentence
"[The Sentence]"
for only the following entities
"[The Entities]"
The relationType is restricted to only "dependency",
"containment", and "other".
Please answer using only the provided template, and
if multiple results are available, include all of them.
[JSON Template]
```

Listing 5.14: Relation Extraction Entity Set Mode

The sentence used in Listing 5.12 is inserted into the prompt in Listing 5.14, along with the entities "CustomerManagement, OrderProcessing, BillingSystem". GPT correctly responds, as shown in Listing 5.15 with the first entity pair (CustomerManagement, OrderProcessing) which is classified correctly as a dependency. The next pair (CustomerManagement, BillingSystem) is correctly classified as a dependency, and their relation, "interfaces with", is also correctly identified. However, the entities are not connected using this relation. The relation interfaces are between "BillingSystem" and "external payment gateways". Whereas for the other possible entity pairs like (OrderProcessing, BillingSystem), GPT predicted it to be not in a relation. As seen in this example, it is sometimes hard to interpret a non-trivial sentence. The placement of commas and the formulation of longer sentences can cause ambiguity in the author's meaning, which is hard for humans, just as for LLMs to understand. With the tendency of GPT to go for positive classification, the increased occurrence of false positives should be kept in mind.

```
[{
    "entity1": "CustomerManagement",
    "entity2": "OrderProcessing",
    "relation": "relies on",
    "relationType": "dependency"
},
{
    "entity1": "CustomerManagement",
    "entity2": "BillingSystem",
    "relation": "interfaces with",
    "relationType": "dependency"
}]
```

Listing 5.15: GPT Response for Entity Set Mode

Mode 3 takes the previous mode into an even more controlled direction by providing entity pairs to extraction relations. This mode reduces the entity set to just a pair, which can help avoid misinterpretations like those that occurred in Mode 2. However, the performance of this mode may worsen if the entity pairs are not well-chosen. To use Mode 3, the previous prompt can be used without modification, and the given entities can be limited to two. The responsibility of passing over only entities in pairs lies within the program that uses these prompts. As only a single entity pair is given, only one relation and its type can be found, and therefore, it is possible to remove the part having multiple results. However, the array is still included for compatibility reasons in parsing. The prompt can be seen in Listing 5.16 which is a slightly modified version of Mode 2.

```
Give me the relation of the following sentence
"[The Sentence]"
\textbf{for only the given entity pair}
"[The Entities]"
The relationType is restricted to only "dependency",
"containment", and "other".
```

```
Please answer using only the provided template.
[JSON Template]
```

Listing 5.16: Relation Extraction Entity Pair Mode

5.3.4 Prompt for Cross Sentence Relation

Cross-sentence prompts are necessary for identifying relations that span multiple sentences, such as coreference and inter-sentence relations. In the previous example (Listing 5.12), an instance of such a case in partial sentences was observed, where a previously defined component is referred to as "it". In addition, utilizing cross-sentence prompts has the added advantage of reducing the overhead per prompt, making it a more cost-effective option.

To further illustrate the significance of cross-sentence prompts, a short text consisting of five sentences (see Listing 5.17) is created. This text contains multiple relationships, including a cross-sentence relationship between the third and fourth sentences. By utilizing cross-sentence prompts, complex relationships spanning multiple sentences can be identified, which would not have been possible otherwise.

```
1. The system's core functionality is managed by the
   CoreModule, which handles data processing, user
   authentication, and access control.
2. The CoreModule relies on the DatabaseManager component to
    store and retrieve data efficiently.
3. The AuthenticationService component, part of the
   CoreModule, verifies user credentials before granting
   access.
4. It communicates with the UserDatabase component to
   authenticate users securely.
5. The PaymentProcessing module, which is separate from the
   CoreModule, interacts with external payment gateways for
   processing transactions.
```

Listing 5.17: Example Document with Multiple Sentences

The prompt seen in Listing 5.10, which is used for Mode 1, can be reused for this purpose by using all the sentences of the document at once. The results this prompt delivers are seen in 5.18 and are shortened using a tuple representation (Entity1, Entity2, Relation, RelationType) for oversight. The given answer is recognized and classified correctly for the four result entries. One of those entries is the cross-sentence relation.

```
{
(CoreModule, DatabaseManager, relies on, dependency),
(CoreModule, AuthenticationService, part of, containment),
```

```
(AuthenticationService, UserDatabase, communicates with,
   dependency),
(PaymentProcessing module, external payment gateways,
   interacts with, dependency)
}
```

Listing 5.18: GPT Response Using Entity Recognition Mode for Multiple Sentences

For comparison purposes, the same text is analyzed using Mode 1 with one sentence at a time, and the results are presented in 5.19. The method yields more ambiguous relations that are harder to interpret. In sentence five, the relation between the pair (PaymentProcessing module, CoreModule) is classified as 'other', even though it was previously classified as 'dependency'. Such inconsistency is a common occurrence in LLMs, and although attempts are made to modify the prompt to make the results more consistent, they remain unsuccessful. The relation in sentence four is the cross-sentence relation identified in the previous method. This method can only recognize the entity pair (It, UserDatabase component), with "It" being unsolvable as the entity it refers to is missing.

```
1. {(system's core functionality, CoreModule, is managed by,
    dependency)}

2. {(CoreModule, DatabaseManager, relies on, dependency)}

3. {(AuthenticationService, CoreModule, part of, containment
   ),
(AuthenticationService, user credentials, verifies,
   dependency)}

4. {(It, UserDatabase component, communicates with,
   dependency)}

5. {(PaymentProcessing module, CoreModule, separate from,
   dependency),
(PaymentProcessing module, external payment gateways,
   interacts with, other)}
```

Listing 5.19: GPT Response Using Entity Recognition Mode with Single Sentences

One issue with the prompting method is that it's ambiguous which sentence the relation is being extracted from. To resolve this, the prompt template would need modification to include an additional entry sentence number. Additionally, the implementation in GPTSerializer would need modification to parse the additional entry.

5.3.5 Other Prompting Approaches

The previously crafted prompts are often plagued by a high number of false positives in their results. Several attempts have been made to reduce this problem, but so far, they have not been very successful. Even modifications to the prompt, such as using "The Persona Pattern" [29] and setting the context by using the sentence "**Suppose you are a relation classification model.**" have not had a significant impact on the results. Similarly, changing the relations to "**Dependent-Depender**" and "**Component-Whole**" instead of using "Dependency" and "Containment" has not yielded meaningful improvements either. An alternative approach involves a two-stage process similar to the one used in [28]. In the first stage, only the relations that have occurred are requested. These relations are then used to extract the relation triplet in the second stage. However, this method did not lead to the desired improvements. In an attempt to reduce the number of tokens used, the System-User feature of GPT is tested. The goal was to send a system prompt once, containing instructions for what to do, and then send user prompts containing only the sentences and entities that would be applied to the system prompt. This approach is similar to how it is possible in ChatGPT to send one instruction prompt followed by data that gets applied to the instruction. However, tokens can not be saved as the system prompt had to be included, leading to even more used tokens.

6 Relation Extraction with BERT

This chapter presents a BERT-based method for extracting SAD relations, which builds upon the idea discussed in Chapter 4. Section 6.1 provides an overview of the relevant components and data flow, while Section 6.2 goes into detail on the implementation. Additionally, Section 6.3 covers the training process, including pre-training and fine-tuning.

6.1 Overview

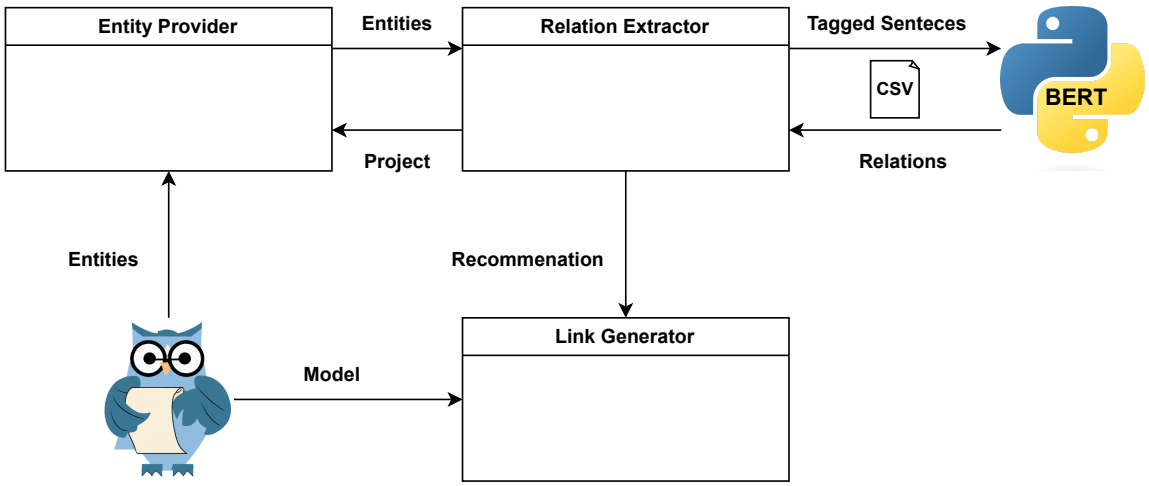


Figure 6.1: Data Flow Diagram BERT Approach

In the relation extraction approach using BERT, there are three main components: the Entity Provider, Relation Extractor, and Link Generator. Figure 6.1 provides an overview of this process. To find entity trace links, the approach uses ArDoCo. For relation extraction, it employs a PyTorch implementation of BERT. However, since the project is implemented in Java and the BERT implementation is in Python, data communication must be done manually by passing data as a CSV file.

6.2 Implementation

In this section, we'll discuss how the BERT relation extraction component is implemented. The BERTRelationExtractor, as shown in Figure 6.2, serves as the starting point for this approach. To begin with, we call the writeTaggedSentencesToCSV function with the Project as a parameter, along with the ardocoThreshold and entitiesPerSentence values. This function, in turn, uses these parameters to invoke the BERTEntityTagger, which tags the sentences that contain at least entitiesPerSentence entities with a value of ardocoThreshold or higher. The EntityProvider is utilized by the BERTEntityTagger to obtain unique entity pairs that are tagged in the sentence. The tags that are used have the following format: [E1]ENTITY[/E1] and [E2]ENTITY[/E2]. The tagged sentences are then written to a CSV file containing sentence ID, entity1 text, entity1 ID, entity2 text, entity2 ID, and the tagged sentence.

After that, a Python script reads the file, and the trained BERT model is fed with this information. The classified relation types are then saved in another CSV file, which is again read into the Java project using the getBERTResults function and converted into a Recommendation. The BERT model is trained on the SemEval2010 task 8 dataset and also determines the direction of the relation. Although the direction is not used in the evaluation, it is saved in the recommendation.

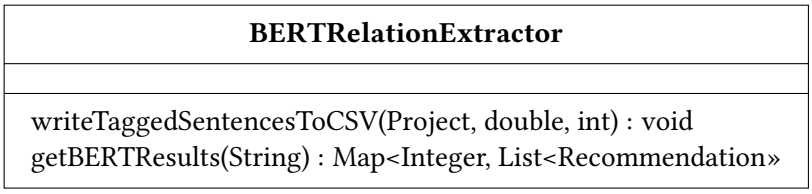


Figure 6.2: UML Class for BERT Relation Extractor

6.3 Trainings Process

The BERT approach utilizes a PyTorch implementation [1] based on BERT by GitHub user plkmo, which is trained using Matching The Blank (MTB) [22]. The framework provides Python scripts for both pre-training and fine-tuning. After fine-tuning, a console interface is available for relation extraction, with two modes: one for tagged sentences and one without tags. The mode without tags uses the built-in entity tagger to infer entity pairs, but it is not reliable in detecting software architecture entities. Therefore, the Python-based framework had to be extended to process multiple tagged sentences and output the results to a CSV file, as it could not directly connect to the Java implementation of this thesis. The AI machine used for training the model is a Tesla V100S with 32 GB VRAM. The models BERT-base-uncased, BERT-large-uncased, and ALBERT-base-v2 were obtained from the

[1] https://github.com/plkmo/BERT-Relation-Extraction

HuggingFace website [2]. The training process involved pre-training and fine-tuning for relation extraction using MTB.

6.3.1 Pre-Training

Pre-training MTB is done on a large corpus of text. This approach allows pre-training using non-annotated data, followed by fine-tuning with annotated data. The CNN dataset is used as the source of entity-linked text for pre-training. Relation statements are extracted that contain at least two grounded entities within a fixed window of 50 tokens. One or both entity mentions are replaced with a special [BLANK] symbol to create positive and negative pairs of relation statements based on whether they share the same pair of entities or not. The BERT model is then trained on them by minimizing estimation loss that encourages the similarity between positive pairs and discourages the similarity between negative pairs. During attempts to pre-train on the AI machine, it was found that pre-training on BERT-base and BERT-large was unsuccessful. The training process would terminate before reaching a checkpoint without any error message, making it difficult to identify the cause. However, the author of the framework stated that fine-tuning without pre-training still produces reasonable results, so this step is skipped for BERT-base and BERT-large. It is believed that the resource requirements of the BERT models are the cause of this issue. Therefore, a less resource-intensive variant of BERT, ALBERT [12], was tested and found to be successfully pre-trained on the dataset. As a result, it will be included in the approach.

6.3.2 Fine-Tuning

The annotated training dataset provided by SemEval2010 task 8 is used for fine-tuning. This particular training data is suitable for the software architecture domain, some of which are applicable to software architecture. The SemEval relations and the transferred type for software architecture relation and a short description are depicted in Table 6.1.

Fine-tuning with the SemEval dataset could be successfully done without pre-training for BERT-base, BERT-large, ALBERT, and pre-trained ALBERT. The results in Listing 6.1 show classification for the tagged sentence mode and infer mode for two different sentences. The results correctly identified the relation in the first sentence for the entity pair (visit, frenzy) as Cause-Effect and the direction (e1,e2). The second sentence used the inferred mode, for which entities in the sentence are recognized using Spacy NLP and then tagged. The results correctly identified the relation Cause-Effect for the entity pair (After eating the chicken, sore throat) and its direction.

```
Sentence: The surprise [E1]visit[/E1] caused a [E2]frenzy[/
   E2] on the already chaotic trading floor.
Predicted: Cause-Effect(e1,e2)
```

[2] HuggingFace.co

SemEval Relation	Mapped Relation	Description
Cause-Effect	Dependency	An event yields an effect
Instrument-Agency	Dependency	An agent uses an instrument
Product-Producer	Dependency	A producer creates a product
Content-Container	Containment	An object is stored in a container
Entity-Origin	Dependency	Entity coming from an origin
Entity-Destination	Dependency	Entity moving to a destination
Component-Whole	Containment	An object as part of a whole
Member-Collection	Containment	A member forms a collection
Communication-Topic	Dependency	Communication about a topic.
Other	None	If no relation can be assigned

Table 6.1: SemEval Relations and Relation Type Mapping

```
Sentence: [E2]After eating the chicken[/E2] , [E1]he[/E1]
   developed a sore throat the next morning .
Predicted: Other

Sentence: After eating the chicken , [E1]he[/E1] developed
   [E2]a sore throat[/E2] the next morning .
Predicted: Other

Sentence: [E1]After eating the chicken[/E1] , [E2]he[/E2]
   developed a sore throat the next morning .
Predicted: Other

Sentence: [E1]After eating the chicken[/E1] , he developed
   [E2]a sore throat[/E2] the next morning .
Predicted: Cause-Effect(e1,e2)

Sentence: After eating the chicken , [E2]he[/E2] developed
   [E1]a sore throat[/E1] the next morning .
Predicted: Other

Sentence: [E2]After eating the chicken[/E2] , he developed
   [E1]a sore throat[/E1] the next morning .
Predicted: Cause-Effect(e2,e1)
```

Listing 6.1: Result of BERT Large MTB with only Fine-Tuning

After fine-tuning the language model on the training dataset, the evaluation resulted in an F1-score of 0.80 for BERT-large-uncased on the testing dataset. Figure 6.3 illustrates the performance of the F1 score on the testing dataset after each of the ten epochs. An epoch is completed when the language model has been trained on all the data of the dataset

and adjusted its weight parameters. The model BERT-base-uncased achieved an F1-score of 0.75, which is 0.02 points lower than the reference score provided by the author of 0.77, despite following the instructions diligently. ALBERT and pre-trained ALBERT both achieved an F1 score of approximately 0.80 after fine-tuning.

In order to evaluate the effectiveness of models in detecting software architecture relationships, simple examples were used. The results for the BERT-Large model, as shown in Listing 6.2, indicate that the trained model is proficient in recognizing 'contains' relationships and identifying the direction of the relationship (indicated with (e1,e2) or (e2,e1)). However, for the most part, it classifies 'dependency' relationships as 'Other'. The other model performed similarly to BERT-Large. This suggests that there is a lack of labeled data for dependency-type relationships in the software domain for the models to train on. As a result, an alternate dataset is being sought after.

```
Sentence:  [E2]Logic[/E2] depends on [E1]UI[/E1]
Predicted:  Other

Sentence:  [E2]Logic[/E2] calls the [E1]UI[/E1]
Predicted:  Other

Sentence:  [E2]Logic[/E2] uses [E1]UI[/E1]
Predicted:  Instrument-Agency(e1,e2)

Sentence:  [E2]Logic[/E2] contains [E1]UI[/E1]
Predicted:  Member-Collection(e2,e1)

Sentence:  [E2]Logic[/E2] composes of [E1]UI[/E1]
Predicted:  Member-Collection(e2,e1)

Sentence:  [E2]Logic[/E2] includes [E1]UI[/E1]
Predicted:  Member-Collection(e2,e1)
```

Listing 6.2: BERT-Large-Uncased Easy Example Results

Tang et. al [23] created an annotated dataset using StackOverflow user content which contains 17061 relation instances. This dataset is used for training relation extraction using tagged entities, similar to the one created in SemEval. The dataset has four relation types and the relation 'semantic' which represents the other relation types. These relation types are shown in Table 6.2. A request to access the dataset for use in the thesis has been submitted since it is not publicly available, but no response has been received by the end of this thesis.

Relations by Tang et. al	Mapped Relations	Description
Use	Dependency	An object uses another
Inclusion	Containment	An object includes another
Brother	None	An object is similar to another
Consensus	None	An object is a synonym of another
Semantic	None	Other semantic relations

Table 6.2: Relations by Work of Software Knowledge Relation ExtractionFramework

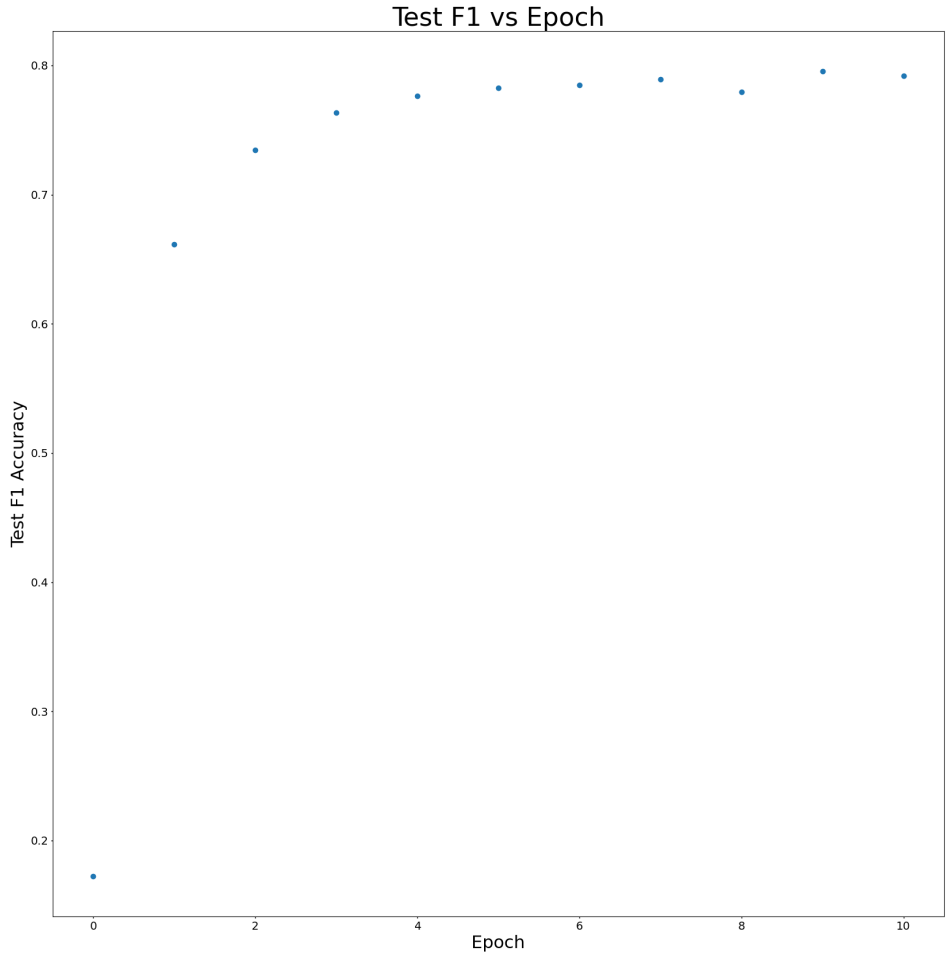


Figure 6.3: Evaluation of BERT Large Uncased MTB using SemEval Testing Dataset

7 Link Generation

This chapter discusses the approach of connecting relations in SAD with SAM. In Section 7.1, the representation of the trace link is presented. Section 7.2 explains the implementation for creating trace links.

7.1 Link Representation

Relation links are represented using the sentence number for the documentation and a list of model interface IDs for the model. The sentence number points to where in the documentation the relation occurs. The model IDs are represented using different approaches. The model representation for the type DEPENDENCY is explained in Subsection 7.1.1 and for the type CONTAINMENT in Subsection 7.1.2. The relation type NONE is represented using an empty list "[]". It is important to note that the ArDoCo Benchmark models do not contain any composite components that classify as CONTAINS relations. Therefore, a modified PCM repository is used to test the algorithm.

7.1.1 Dependency Representation in Model

A DEPENDENCY relation is only valid if the model components for the entity pair exist. In addition, one component must provide an interface, and the other must require one. This is shown in Figure 7.1 for direct connections and in Figure 7.2 for transitive connections. Model data is provided by an ArDoCo interface which parses PCM repositories. As the interface does not offer the IDs for the provided and required arrows, the interface connecting the components is used to determine the relation instead.

Figure 7.1: Dependency Relation in Model with Direct Connection

Figure 7.1 shows a direct connection between Component A and Component B. The interface between them represents the relation in the model, and the interface ID is saved as the relation ID.

Figure 7.2 depicts a transitive connection between Component 1 and N. It is a path with any amount of interfaces connected with components between the starting and end

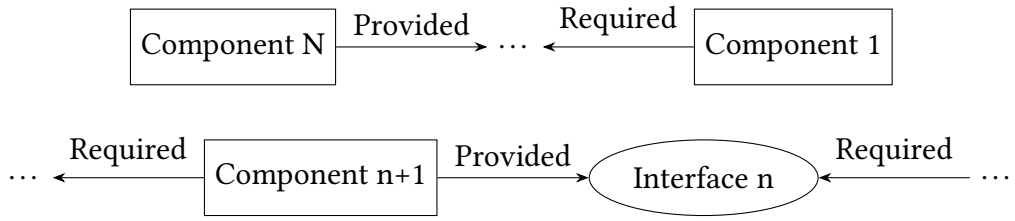


Figure 7.2: Dependency Relation in Model with Transitive Connection

components. Each component must have a provided and a required arrow to interfaces on the path and has to alternate between required and provided. The relation is represented using the ID of each interface between these components saved in a list. It starts from the component with a required arrow and ends with the component with a provided arrow. Therefore, the direction of the relation is: Component 1 depends on Component N.

7.1.2 Containment Representation in Model

A CONTAINMENT relation in the model is valid if a component is a composite of another. In the model, this is represented using the IDs of the components starting from the composite component. Figure 7.3 shows Composite component A containing Component B. The direction of this relation is: Component A contains Component B.

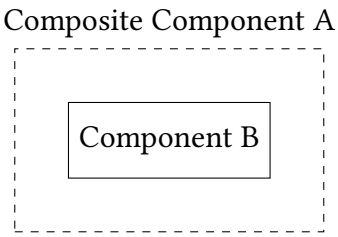


Figure 7.3: Containment Relation in Model

7.2 Link Generator Implementation

This section presents the implementation for creating trace links. It is done by explaining the functions of the class LinkGenerator. The starting point for creating a trace link is Algorithm 1. It determines, based on the relation type, whether to call Algorithm 2 to find relation representation for DEPENDENCY or to call Algorithm 3 to look for model reference of CONTAINMENT. As relation direction is not implemented, it checks for both directions.

Algorithm 1: Generating Trace Link

```
function generateTraceLink(recommendation, int sentenceId)
    e1Id ← recommendation.getEntity1().getId()
    e2Id ← recommendation.getEntity2().getId()
    if recommendation.getRelationType() == RelationType.DEPENDENCY then
        path ← findDependencyPath(new List, e1Id, e2Id)
        if path.isEmpty() then
            path ← findDependencyPath(new List, e2Id, e1Id)
    else if recommendation.getRelationType() == RelationType.CONTAINMENT then
        path ← findContainmentPath(e1Id, e2Id)
        if path.isEmpty() then
            path ← findContainmentPath(e2Id, e1Id)
    recommendation.setRelationId(path)
    return RelationTraceLink(recommendation, sentenceId)
```

7.2.1 Connecting Dependency Links

Algorithm 2 is used to get a List of interface IDs that connect two components. This list of interface IDs is a path in which a component goes by alternating between provided and required interfaces. If there is no path between these two components, an empty list is returned. The algorithm searches the path starting with the provided interfaces. To look for the direction E1 depends on E2. the algorithm has to use E2 as the component to start and E1 as the component to find. The other way around would imply that E2 depends on E1. But as the goal is to find the trace link representation regardless of direction, this function is called for both cases.

The function is a recursive path-finding algorithm that takes three parameters: compIdToStart is the starting component ID to search from, compIdToFind is the component ID to look for, and visitedComps is a list of component IDs to track which components have been visited.

The algorithm starts iterating over every component that has a required interface that the starting component provides. If the ID matches the component to find, then it returns a list containing the provided interface ID. If no match is found, then it recursively calls this function with the current component as the starting component. If the called function returns a nonempty result, then it has found the ending component and returns the path with the current interface ID added to it.

7.2.2 Connecting Containment Links

Algorithm 3 checks for the CONTAINMENT relation by getting the inner components of the starting component. And looking for a match of an inner component with the

Algorithm 2: Algorithm for Finding Dependency Path

```
function findDependencyPath(List of visitedComps, compIdToFind, compIdToStart)
    providedInterfaces ← getProvInterfacesFromComp(compIdToStart)
    for provInterface in providedInterfaces do
        compWithReqInterface ← getCompListWithReqInterface(provInterface)
        for comp in compWithReqInterface do
            if visitedComps.contains(comp) then
                continue
            visitedComponents.add(comp)
            if comp.equals(compIdToFind) then
                path.add(provInterface)
                return path
            path ← findDependencyPath (visitedComps, compIdToFind, comp)
            if not path.isEmpty() then
                if not path.contains(provInterface) then
                    path.add(provInterface))
                return path
    return empty path
```

component to find. If there is a match, then a list with the IDs of both components will be returned, else it returns an empty list.

Algorithm 3: Algorithm for Finding Containment Path

```
function findContainmentPath(compIdToFind, compIdToStart)
    List<String> path ← new ArrayList();
    component ← getComponentById(compIdToStart);
    for innerComp in component.getInnerComponents() do
        if inner.getId() == compIdToFind then
            path.add(component.getId());
            path.add(innerComp.getId());
            return path;
    return path;
```

8 Evaluation

This chapter evaluates the performance of different approaches to test the thesis goal of traceability link recovery, intermediate goal relation extraction, and classification. Both concepts have variations of models and parameter configurations to evaluate. The GPT concept is evaluated for the model GPT-3.5-turbo and GPT-4. In addition, different prompts are evaluated. The BERT concept has the model option for BERT-base, BERT-large, ALBERT, and pre-trained ALBERT.

Model / Mode	Entity Recognition	Entity Set	Entity Pair
GPT-3.5-turbo	X	X	X
GPT-4	X	X	X
BERT-base			X
BERT-large			X
ALBERT			X
ALBERT-pre			X

Table 8.1: Evaluated Relation Extraction Modes for Different Models

The evaluated methods for relation extraction and the used models are depicted in Table 8.1. The table shows that the GPT models are used for entity recognition, inferring entity pairs from a set of entities, and relation extraction using given entity pairs. In contrast, the BERT-based models are trained to match the blank, which requires a tagged entity pair.

The following sections are structured as follows: Section 8.1 explains the process of creating a gold standard for evaluating relation extraction. In Section 8.1.2, the additions made to the gold standard to facilitate the evaluation of trace links are presented. Section 8.2 introduces the metrics used to measure the performance of the approaches. Section 2.1.3 details the implementation behind the evaluator for the relation extraction task, including the results. Section 8.4 presents the trace link recovery step's implementation and results. Finally, Section 8.6 discusses possible issues that may affect the evaluation.

8.1 Gold Standard

In this section, the gold standard used in the evaluation is presented. First, a gold standard is created for evaluating the relation extraction capabilities shown in Subsection 8.1.1. Then Subsection 8.1.2 presents the gold standard for trace link recovery, which is created by adding references of the model to the previously created gold standard. For creating this

gold standard, the ArDoCO Benchmark dataset [4] is used and is composed of extracted data from open-source SAD and SAM.

8.1.1 Gold standard for Recognition and Classification of Relations

Two essential sub-goals are required before traceability links can be created. First, occurrences of potential software architecture relations in the documentation have to be recognized. Then, the found relation needs to be classified. These two sub-goals can be grouped as the relation extraction task. For this task, a relation extraction gold standard is created.

Sentence	Entity1	ID	Entity2	ID	Relation	Relation Type
9	Logic	_1	Model	_2	depend on	DEPENDENCY
122	Datastore	_3	Logic	_1	hides	CONTAINMENT

Table 8.2: Relation Extraction Gold Standard Example.

The relation extraction gold standard entries contain all data required for identifying a relation in text with the classified relation type depicted in Table 8.2. The table shows two example entries for the gold standard. One is from the relation type DEPENDENCY, and the other is of type CONTAINMENT. The required data consists of the sentence number, the entity pairs, the relation, and the relation type. The entities in the gold standard are identified using their model ID. For a better overview, the model name is also included. The gold standard is created in Comma Separated Values (CSV) with ',' being the delimiter.

The gold standard is created using software architecture documentation from the ArDoCo benchmark. This is done by writing down all the software architecture relations with the required data for every sentence. For this process, only relations of type DEPENDENCY and CONTAINMENT are considered. Every relation not included in the gold standard is implicitly classified as the relation type NONE. This is because the NONE type represents the negative case where the type is neither DEPENDENCY nor CONTAINMENT, which includes entity pairs that are not in relation. Only relations between two valid software architecture entities are considered. The ArDoCo Benchmark gold standard for entity trace links determines valid entities. Every valid entity has a counterpart in the model. From this standpoint, valid relations can be derived from valid entities in a relation.

8.1.2 Gold Standard for Traceability Links

The existing gold standard for relation extraction is expanded to create a gold standard for trace link recovery. The sentence number and model IDs can represent a trace link, as explained in Chapter 7.1. Table 8.3 provides an example of a gold standard entry. The sentence number represents where the relation occurs in SAD, while entity IDs determine the correct relation. The relation in the model is represented as a list of the model interface IDs that connect the entities of that relation. Additionally, a relation type is used for

classification. Since most of these values are already present in the relation extraction gold standard, the Model Interface IDs are simply added to it.

Sentence	Entity1 ID	Entity2 ID	Model Interface IDs	Relation Type
3	_1	_2	[_1, _2]	CONTAINMENT

Table 8.3: Traceability Link Gold Standard Example

This gold standard is created using the ArDoCo Benchmark PCM repositories. The process is as follows: For each entry in the relation extraction gold standard, if it is a dependency relation, check whether the entities are directly or transitively connected. Add the ID of each model interface to the list of Model Interface IDs in the order they are connected. If it is a containment relation, add the model IDs of both entities to the list.

8.2 Metrics

In this section, the metrics used for the evaluation are described. For the evaluation, a confusion matrix is created. With this matrix, values like True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) can be easily obtained. This is particularly useful in calculating weighted metrics, which is essential as the used dataset is heavily weighted in favor of DEPENDENCY against CONTAINMENT. The performance metrics most useful for this approach are Recall, Precision, and the F1-Score. Another useful metric is Accuracy, which takes TN into consideration, but because TN is not important for the performance of this approach, it won't be included.

Precision, also referred to as the positive predicted value, is a measure that indicates the ratio of correctly identified positive instances out of all instances that were predicted as positive (as demonstrated in Equation 8.1). This metric is useful in determining the reliability of a positive prediction. In essence, it represents the proportion of true positives to the total number of predicted positives for a particular class.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{8.1}$$

Recall, also known as the true positive rate, is the proportion of correct positive results out of all actual positive results (see Equation 8.2). This metric indicates the rate of actual positive instances that have been correctly identified.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{8.2}$$

F1-Score is the harmonic mean of precision and recall. This metric represents a single metric that considers both false positives (FP) and false negatives (FN), and is calculated using the formula in Equation 8.3.

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8.3}$$

Weighted Metrics can be a useful method for calculating averages in evaluations where there is an imbalance in the number of instances across multiple classes. To compute a weighted metric, the previous metric is multiplied by the weight of the positive class, which is the ratio of actual class instances to the total number of instances, and then divided by the total number of instances N. This approach provides a more accurate evaluation by taking into account the data imbalance. Equation 8.4 demonstrates this calculation for weighted precision.

$$\text{Weighted Metric}_i = \frac{\sum_{i=1}^{C} w_i \cdot \text{Metric}_i}{N} \tag{8.4}$$

8.3 Relation extraction

The relation extraction gold standard mentioned in section 8.1.1 is used to evaluate the relation recognition and classification task. The evaluation process involves creating a confusion matrix using predicted results and the gold standard, which is done using Algorithm 4. Next, the TP, FP, TN, and FN values are calculated by adding the corresponding cells as indicated in Table 8.4. These values are then used to calculate the precision, recall, and F1-Score.

The function Algorithm 4 requires two parameters, both of which are sentence numbers to list of recommendations maps. The first map "resultsMap" contains the predictions made by the framework, while the second map "gsMap" contains the data from the gold standard. A two-dimensional integer array is created to represent the confusion matrix. The size of this array depends on the number of relation types. The first dimension of the array represents the actual relations, while the second represents the predicted relations. To fill the confusion matrix, the function iterates over every sentence number key and the corresponding recommendations of each sentence. If a predicted recommendation matches a gold standard recommendation, the function increases the corresponding cell in the confusion matrix by one and removes the matched gold standard recommendation from the map. The ordinal of the relation type determines the cell. If there is no matching gold standard recommendation or the gold standard doesn't have an entry for this sentence number, the actual relation is assumed to be of type NONE. After iterating over every predicted result in the sentence, the function fills the remaining gold standard recommendations that have not been matched as predicted NONE.

Table 8.4 displays the entries that must be accumulated to compute a weighted average for the metrics. The confusion matrix illustrates the classification of two positive categories, namely DEPENDENCY and CONTAINMENT, and a negative category called NONE. Correct classifications are marked in green, while incorrect ones are highlighted in red. The darker color shading indicates the predicted negatives. The confusion matrix is utilized to

Algorithm 4: Fill Evaluation Matrix

```
function fillEvaluationMatrix(resultsMap, gsMap)
    relationTypes ← [DEPENDENCY, CONTAINMENT, NONE]
    relationTypeCount ← length(RelationType.values())
    confusionMatrix ← 2D array of size [relationTypeCount][relationTypeCount]
    foreach resultEntries in resultsMap do
        sentenceID ← resultEntries.getKey()
        foreach result in resultEntries.getValue() do
            if gsMap contains sentenceID then
                match ← findMatch(gsMap[sentenceID], result)
                if not match is null then
                    confusionMatrix[match.relationIndex()][result.relationIndex()]++
                    Remove match from gsMap[sentenceID]
                else
                    confusionMatrix[RelationType.NONE][result.relationIndex()]++
            else
                confusionMatrix[RelationType.NONE][result.relationIndex()]++
        foreach gs in gsMap[sentenceID] do
            confusionMatrix[gs.relationIndex()][RelationType.NONE]++
    foreach entry in gsMap do
        if not resultsMap contains entry.getKey()) then
            foreach gs in entry.getValue() do
                confusionMatrix[gs.relationIndex()][RelationType.NONE]++
return confusionMatrix
```

		Predicted		
		Dependency	Containment	None
Actual	Dependency	True Positive	FP & FN	False Negative
	Containment	FP & FN	True Positive	False Negative
	None	False Positive	False Positive	True Negative

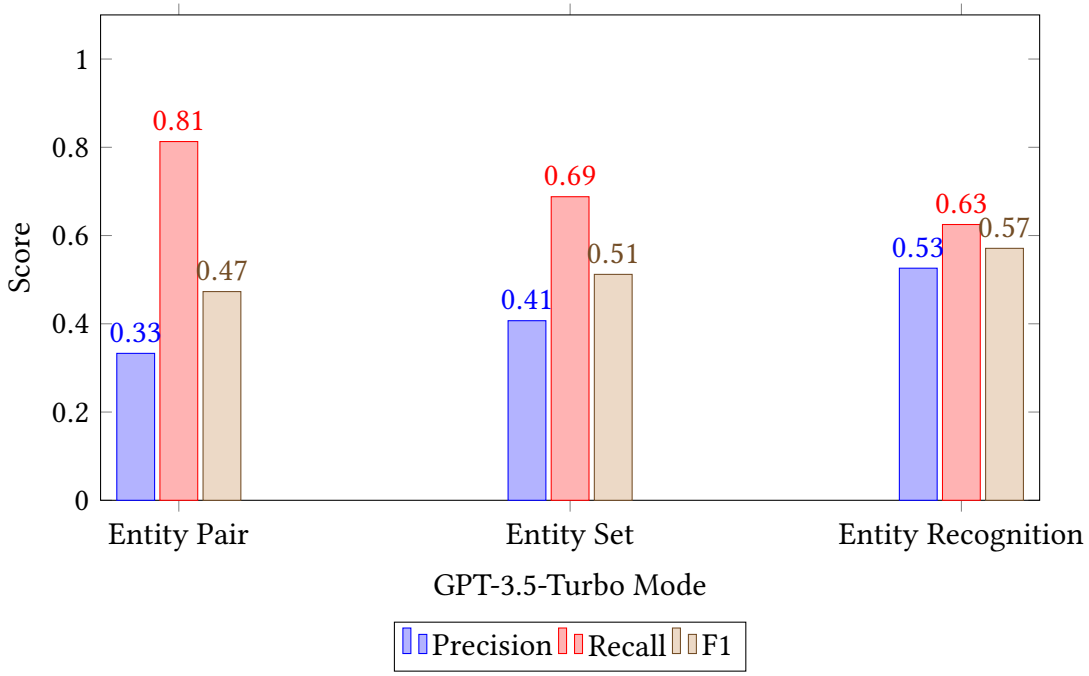
Table 8.4: Multi-Class Confusion Matrix for Relation Types Dependency, Containment, and None

determine TP, FP, TN, and FN classifications. The entries (Dependency, Containment) and (Containment, Dependency) are used as the FP and FN values. In multi-way classification, these entries are used to calculate the total predictions for precision and the total actual relations for recall.

8.3.1 GPT Relation Extraction Results

The relation extraction results for GPT-3.5-Turbo is shown in Figure 8.1 and for GPT-4 in Figure 8.2.

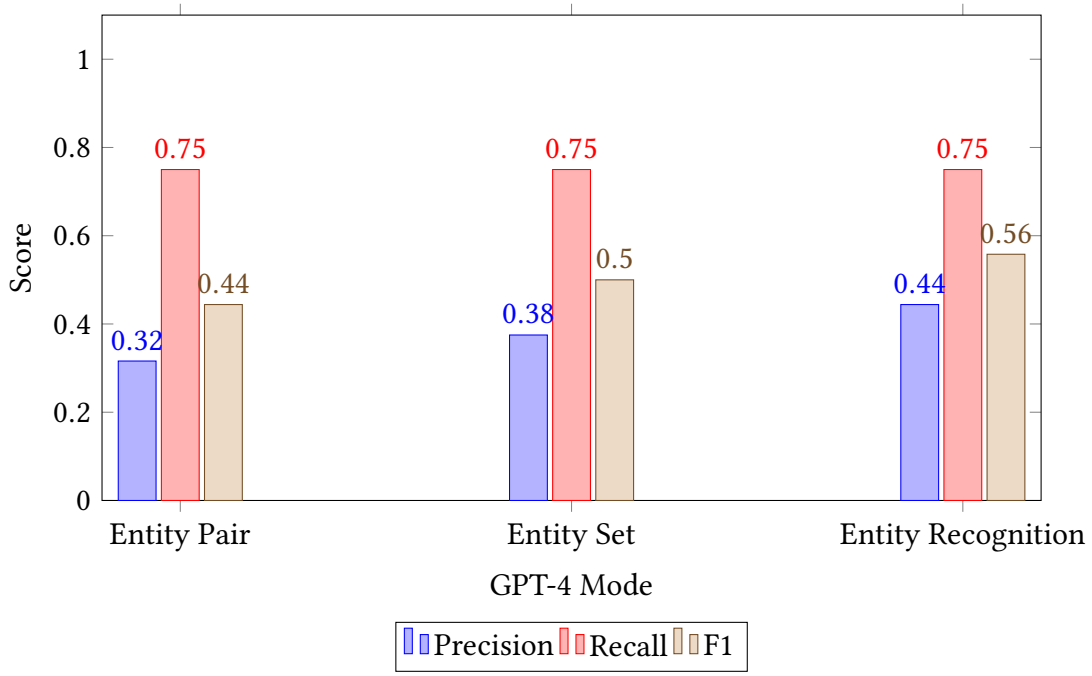
Figure 8.1: Relation Extraction Performance Metrics by GPT-3.5-Turbo Mode



The results for the GPT-3.5-Turbo modes are presented in Figure 8.1. Each extraction mode has three bars representing precision in blue, recall in red, and F1-Score in brown. There is an observable pattern in the performance of Entity Pair (EP), Entity Set (ES), and Entity Recognition (ER) modes. The precision of these modes is increasing from 0.33 EP, 0.41 ES, to 0.53 ER, while their recall is decreasing from 0.81 EP, 0.69 ES, to 0.63 ER. This can be explained by the fact that EP mode has the highest number of total predictions, followed by ES and ER. Using more attempts to make predictions increases the likelihood of getting a correct prediction but reduces its precision. The low precision performance is possibly caused by GPT's responses favoring positives if there is even the slightest hint of a relation. The harmonic mean for the modes is 0.47 EP, 0.51 ES, and 0.57 ER, and slightly favors modes with fewer predictions, indicating that making more predictions is slightly less effective in achieving the correct prediction.

The performance of GPT-4 modes is depicted in Figure 8.2. In contrast to GPT-3.5-Turbo, GPT-4 consistently recalls 0.75 by correctly identifying three out of four actual relations for

Figure 8.2: Relation Extraction Performance Metrics by GPT-4 Mode

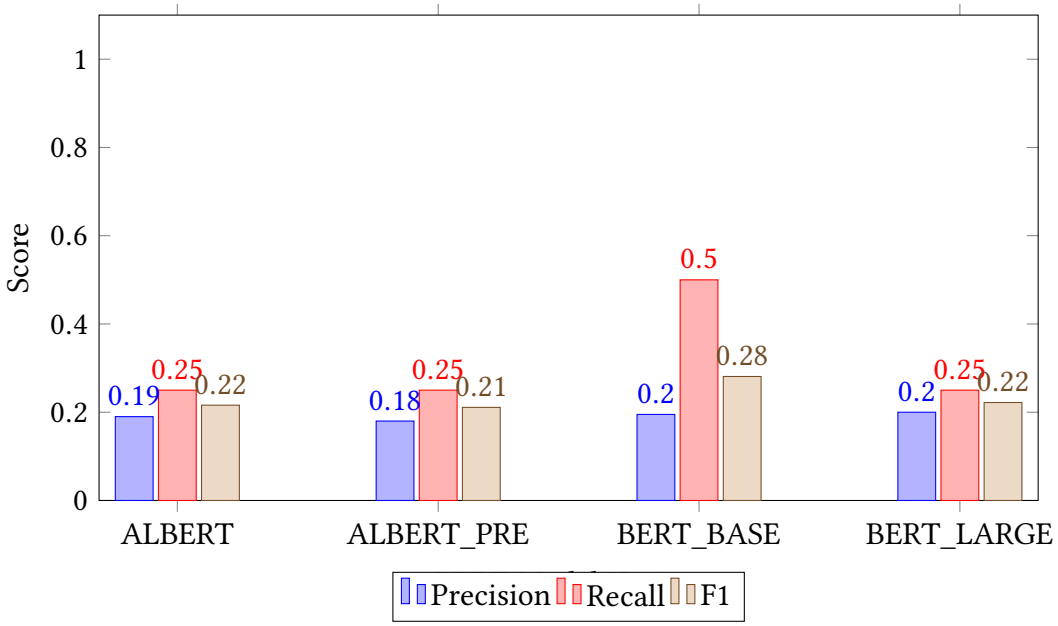


all three modes. This observation indicates GPT-4 likely being more consistent in responses. The precision of GPT-4 is similar to that of GPT-3.5-Turbo, with more predictions for EP, followed by ES and ER. However, the precision is slightly lower than that of GPT-3.5-Turbo, indicating more predictions made by GPT-4.

8.3.2 BERT Relation Extraction Results

The performance of various BERT variants in relation extraction is shown in Figure 8.3. The graph displays performance metrics for the fine-tuned BERT models ALBERT, pre-trained ALBERT, BERT-base, and BERT-large. The overall performance is quite low compared to ChatGPT's, with an average F1-score of 0.52 for GPT-3.5 and 0.52 for GPT-4. Another comparison is the BERT using MTB performance on SemEval test data using the SemEval training data for fine-tuning, which achieved an F1-score of 89.5. One of the reasons for this difference is the lack of domain-specific training data for fine-tuning, which leads to a high number of 'Other' classifications. The variants exhibit a similar performance. The precision ranges between 0.18 to 0.20 points, while the recall is consistent at 0.25 with the exception of BERT_BASE, which shows double the recall rate of 0.5. This means that BERT_BASE is more likely to classify a positive relation than the other variants. Regarding ALBERT and pre-trained ALBERT, their classifications are quite similar. In contrast, BERT_LARGE has different classifications from both ALBERT and BERT_BASE. This suggests that the size of the model has a greater influence on determining classification than pre-training.

Figure 8.3: Relation Extraction Performance Metrics by BERT Model



8.4 Traceability Link Recovery

This section evaluates the approach's ability to establish connections between text and model using the extended gold standard defined in Subsection 8.1.2. The evaluation process is a binary classification based on whether a trace link can be established or not, similar to the one described in Section 8.3, and the evaluation algorithm is analog to Algorithm 4. A trace link is counted as established if a prediction exists for a given sentence with matching entity IDs, relation IDs, and relation classification.

		Predicted	
		Positive	Negative
Act.	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Table 8.5: Binary Confusion Matrix for Trace Link

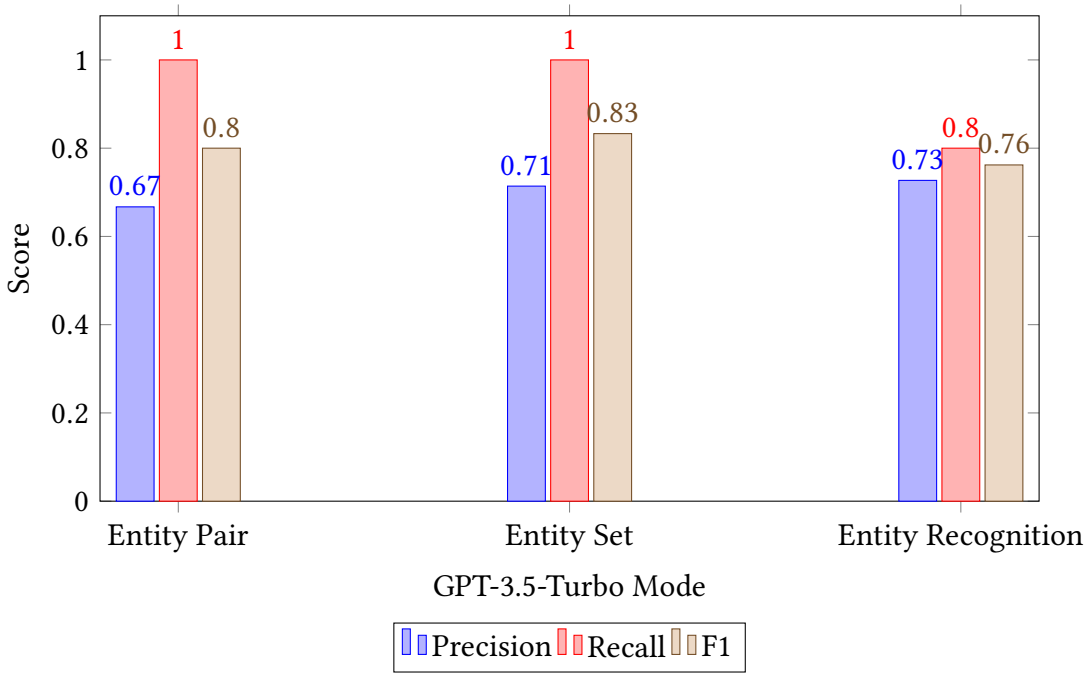
Table 8.5 shows the cases where TP, FP, FN, and TN occur. Each value appears only once in the matrix so that it can be taken directly from it. Correct classifications are marked in green, while incorrect ones are highlighted in red. The darker color shading indicates the predicted negatives and the lighter shading indicates the predicted positives.

8.4.1 GPT Traceability Link Recovery Results

The TLR results depend on the relation extraction result calculated earlier and are shown in Figure 8.4 for GPT-3.5-Turbo and in Figure 8.5 for GPT-4. Only valid recommendations

and existing model relations are taken into account for the performance evaluation. GPT-3.5-Turbo has a precision score of 0.67 EP, 0.71 ES, and 0.73 ER, following the trend of its relation extraction performance. Its recall achieved 1.0 for both EP and ES, with ER performing worse at 0.8 and is high overall, with EP and ES modes finding all existing relation trace links. However, it is crucial to note that the evaluation only considered links with a valid model entity and a modeled relation between them. For the right predictions, the algorithm only needs a correctly predicted entity pair and relation type and avoids existing relations that are not mentioned in the documentation, provided it is correctly implemented. GPT-4 performed similarly to GPT-3.5-Turbo in recall with a score of 1.0 for both EP and ES, but its ER mode is slightly better at 0.9. Its precision is going against the previous trend with a score of 0.71 EP, 0.71 ES, and 0.64 ER. This is also caused by the evaluation environment only including predictions with matching entity pairs.

Figure 8.4: Traceability Link Recovery Performance Metrics by GPT-3.5-Turbo Mode



8.4.2 BERT Traceability Link Recovery Results

The TLR performance of the BERT-based variants is shown in Figure 8.6. The precision of the models is 0.23 ALBERT, 0.38 ALBERT-pre, 0.8 BERT-base, and 0.6 BERT-large. The TLR performance of the models has a greater variety and better performance than the similar RE performance. This indicates that for some models, the FP of the RE step could be filtered out. The recall performance has a similar distribution to the RE results, with 0.3 for ALBERT, ALBERT-pre, and BERT-large. BERT-base performed better with a score of 0.8. After filtering out invalid relations, the performance of BERT_BASE is overall better in the TLR evaluation than in the relation extraction step, with both precision and recall at 80%. BERT_LARGE has significantly increased its precision, followed by ALBERT.

Figure 8.5: Traceability Link Recovery Performance Metrics by GPT-4 Mode

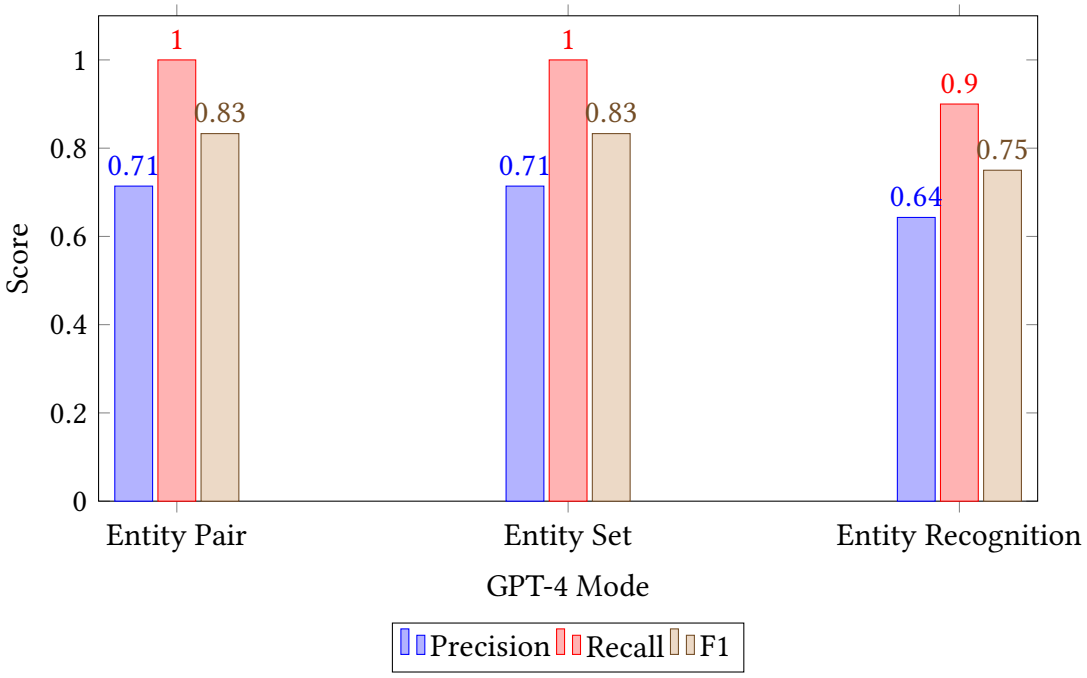
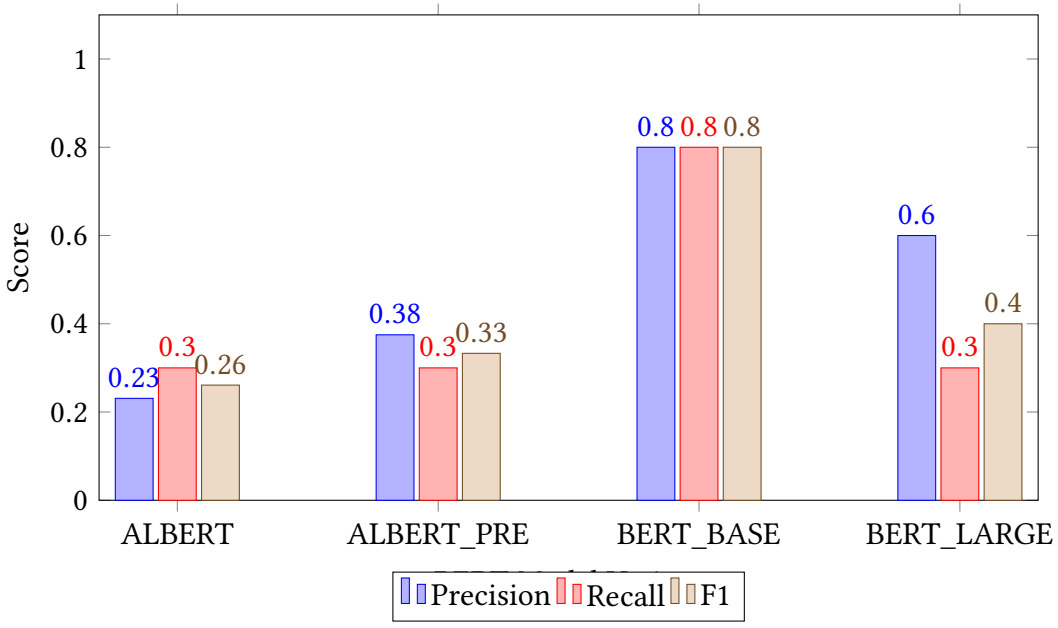


Figure 8.6: Traceability Link Recovery Performance Metrics by BERT Model



8.5 Performance of GPT and BERT

After evaluating each GPT extraction mode and BERT model, this section discusses how they performed against each other. For relation extraction, ChatGPT based on GPT-3.5-turbo and GPT-4 performed well across all extraction modes with an average recall of 0.71 and F1-score of 0.52 for GPT-3.5-turbo and an average recall of 0.75 and F1-score of 0.56 for GPT-4. BERT, on the other hand, scored across all models an average of 0.31 for recall and 0.23 for F1-score. Even though fine-tuned BERT models using the SemEval dataset showed an average F1-score of 0.77 on the SemEval test data, indicating the problem is at the adaptation of SemEval dataset for software architecture domain purposes. This is further underlined by the fine-tuned models having difficulties classifying DEPENDENCY relations, as shown in Figure 6.2 on trivial dependency cases. The occurrence of DEPENDENCY relations vastly outnumbers the CONTAINMENT relation in SAD, causing the result of BERT models to underperform as it would mostly classify DEPENDENCY relations as NONE type. For TLR, ChatGPT performed on average 0.8 for GPT-3.5-turbo and GPT-4 and is better than the relation extraction results, as the model is taken into account to filter out most of the false positives. The BERT models also increased in performance and could score an average of 0.45 F1-score. It is important to note that these results only depend on the extracted relations and did not use the PLMs in recovering these links.

8.6 Threats to Validity

This section discusses the validity threats faced by the evaluation. One of the main issues is that the gold standard is created by only a single person and is prone to bias and subjectivity, which affects the validity of the evaluation. An issue is also the small number of different relation types with only DEPENDENCY, CONTAINMENT, and NONE. Also, DEPENDENCY and CONTAINMENT relations are not mutually exclusive and, in some cases, ambiguous, making it hard to decide which relation to assign. Another threat is the small sample size of SAD and SAM, as the gold standard is based on five open-source projects but has only 14 software architecture relations in the text. Additionally, out of these five projects, only four had a complete SAM, which further reduced the possible model relations to ten. This is worsened by the lack of occurrences of the CONTAINMENT type in SAD and no occurrence of it in SAM. Therefore, steps are taken to confirm that the approach can correctly identify CONTAINMENT relations by extending the existing benchmark project TEAMMATES with composite components using PCM bench IDE. Further, the existing XML parser of ArDoCo has also been extended to include composite components, which was done by ArDoCo's maintainer. The approach previously included a fourth relation type INVALID, for faulty predictions and was also included in the evaluation. But as this would distort the evaluation, it has been taken out.

9 Conclusion

This thesis aims to present a methodology for extracting relationships and restoring traceability links between software architecture documentation and models. The framework utilizes large language models, such as ChatGPT and BERT, to perform zero-shot and fine-tuned relation extraction from textual documentation and generate trace links to software architecture models. The methodology has been implemented as a project that integrates with ArDoCo, a framework for trace link recovery and inconsistency detection. Three different extraction modes have been implemented with the GPT approach, and more have been manually tested. The BERT approach was tested using different BERT-based models and varying degrees of training. The GPT approach yielded promising results in relation extraction, with a recall of 0.81 for EP mode and an F1-score of 0.57 for ER mode. However, the BERT-based approach did not perform well in the relation extraction step due to the lack of domain-specific training data. Through filtering and utilizing trace link entities from ArDoCo, the TLR step could achieve an F1-score of 0.83 for ES mode using GPT-3.5-turbo and achieved detection of all possible relation trace links in EP and ES mode.

Insights are made by comparing these LLMs with the prompting modes and different models. For GPT, the extraction modes entity pair, entity set, and entity recognition received different amounts of predictions by ChatGPT, resulting in the option to balance recall against precision. Another observation is that GPT-4 evaluated more consistent performance, with a recall of 0.75 across all extraction modes, than GPT-3.5-turbo, which ranges in recall between 0.63 and 0.81. For BERT approaches, an observation is that the resource-intensive pre-training of model-variant ALBERT indicated no increase in relation extraction performance without domain-specific fine-tuning. This also underlines the importance of labeled domain-specific training data for PLMs like BERT as it performed worse across all model variants with an average F1-score of 0.23 against ChatGPT based on GPT-3.5 with an average F1-score of 0.52 and GPT-4 with an average of 0.5.

Potential avenues for future research could be optimizing the prompt engineering process to include document-level extraction to find cross-sentence relations to increase recall and adding additional filters for removing unlikely predictions to increase precision. To improve the BERT approach, domain-specific annotated training data is needed, which would enable a method that does not depend on third-party services. Expanding the benchmark documentation and model dataset to include more samples, especially for containment relations, would improve the evaluation's validity. Overall, this approach serves as a step towards more advanced and efficient methodologies in software architecture traceability link recovery. It provides insights into the performance of different LLMs and their applicability in relation extraction from SAD.

Bibliography

[1] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

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

[4] Dominik Fuchß et al. “Establishing a Benchmark Dataset for Traceability Link Recovery Between Software Architecture Documentation and Models”. In: *Software Architecture. ECSA 2022 Tracks and Workshops*. Ed. by Thais Batista et al. Cham: Springer International Publishing, 2023, pp. 455–464. ISBN: 978-3-031-36889-9.

[5] David Garlan and Mary Shaw. “An introduction to software architecture”. In: *Advances in software engineering and knowledge engineering*. World Scientific, 1993, pp. 1–39.

[6] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. “Semantically enhanced software traceability using deep learning techniques”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE. 2017, pp. 3–14.

[7] Marti A Hearst. “Automatic acquisition of hyponyms from large text corpora”. In: *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*. 1992.

[8] Iris Hendrickx et al. *SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations Between Pairs of Nominals*. 2019. arXiv: 1911.10422 [cs.CL].

[9] Scott B Huffman. “Learning information extraction patterns from examples”. In: *International Joint Conference on Artificial Intelligence*. Springer. 1995, pp. 246–260.

[10] Jan Keim et al. “Detecting Inconsistencies in Software Architecture Documentation Using Traceability Link Recovery”. In: *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. IEEE. 2023, pp. 141–152.

[11] Jan Keim et al. “Trace link recovery for software architecture documentation”. In: *European Conference on Software Architecture*. Springer. 2021, pp. 101–116.

[12] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).

[13] Joohong Lee, Sangwoo Seo, and Yong Suk Choi. *Semantic Relation Classification via Bidirectional LSTM Networks with Entity-aware Attention using Latent Entity Typing*. 2019. arXiv: 1901.08163 [cs.CL].

[14] Bo Li et al. "Evaluating ChatGPT's Information Extraction Capabilities: An Assessment of Performance, Explainability, Calibration, and Faithfulness". In: *arXiv preprint arXiv:2304.11633* (2023).

[15] Jinfeng Lin et al. "Traceability transformed: Generating more accurate links with pre-trained bert models". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 324–335.

[16] Andrea De Lucia et al. "Recovering traceability links in software artifact management systems using information retrieval methods". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 16.4 (2007), 13–es.

[17] Youmi Ma, An Wang, and Naoaki Okazaki. "DREEAM: Guiding Attention with Evidence for Improving Document-Level Relation Extraction". In: *arXiv preprint arXiv:2302.08675* (2023).

[18] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. "Building a large annotated corpus of English: The Penn Treebank". In: (1993).

[19] David Lorge Parnas. "Software aging". In: *Proceedings of 16th International Conference on Software Engineering*. IEEE. 1994, pp. 279–287.

[20] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[21] Sophie Schulz. "Linking Software Architecture Documentation and Models". MA thesis. Karlsruher Institut für Technologie (KIT), 2020. DOI: 10.5445/IR/1000126194.

[22] Livio Baldini Soares et al. *Matching the Blanks: Distributional Similarity for Relation Learning*. 2019. arXiv: 1906.03158 [cs.CL].

[23] Mingjing Tang et al. "Software Knowledge Entity Relation Extraction with Entity-Aware and Syntactic Dependency Structure Information". In: *Scientific Programming* 2021 (2021), pp. 1–13.

[24] Yuanhe Tian et al. "Dependency-driven Relation Extraction with Attentive Graph Convolutional Networks". 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)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4458–4471. DOI: 10.18653/v1/2021.acl-long.344. URL: https://aclanthology.org/2021.acl-long.344.

[25] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[26] Linlin Wang et al. "Relation Classification via Multi-Level Attention CNNs". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1298–1307. DOI: 10.18653/v1/P16-1123. URL: https://aclanthology.org/P16-1123.

[27] Soh Wee Tee. *BERT-Relation-Extraction*. `https://github.com/plkmo/BERT-Relation-Extraction`. 2020.

[28] Xiang Wei et al. "Zero-shot information extraction via chatting with chatgpt". In: *arXiv preprint arXiv:2302.10205* (2023).

[29] Jules White et al. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. 2023. arXiv: `2302.11382 [cs.SE]`.

[30] Rebekka Wohlrab et al. "Improving the consistency and usefulness of architecture descriptions: Guidelines for architects". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2019, pp. 151–160.

[31] Yuan Yao et al. "DocRED: A large-scale document-level relation extraction dataset". In: *arXiv preprint arXiv:1906.06127* (2019).

[32] Daojian Zeng et al. "Distant supervision for relation extraction via piecewise convolutional neural networks". In: *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2015, pp. 1753–1762.

[33] Ningyu Zhang et al. "Document-level relation extraction as semantic segmentation". In: *arXiv preprint arXiv:2106.03618* (2021).

[34] Shu Zhang et al. "Bidirectional long short-term memory networks for relation classification". In: *Proceedings of the 29th Pacific Asia conference on language, information and computation*. 2015, pp. 73–78.