The Role of Context in Automated Requirements-to-Requirements Traceability Link Recovery

Master's Thesis of

David Mathias Bauch

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 Reussner
First advisor:	Dominik Fuchß, M.Sc.
Second advisor:	Dr.-Ing. Tobias Hey

14 July 2025 – 14 January 2026

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

The Role of Context in Automated Requirements-to-Requirements Traceability Link Recovery (Master's Thesis)

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.

Karlsruhe, 14 January 2026

. . . . . .

(David Mathias Bauch)

Abstract

When building large software systems, engineers have to keep track of a large quantity and variety of software requirements. To simplify the complexity of managing these requirements, traceability links can be used to make relationships between requirements explicit. Trace links are an important tool for many software engineering tasks, such as change impact analysis or ensuring consistency, as well as regulatory compliance. In many software projects traceability information is not available, as creating trace links manually is labor-intensive, error-prone, and expensive. Thus, researchers are investigating automated tooling that can recover trace links from existing software projects. Recent developments in the field make use of large language models (LLMs) for the trace link recovery (TLR) task. While LLMs have a broad world knowledge that is encoded during their training, they generally lack the project-specific knowledge required to fully understand a software project's requirements. This thesis proposes the use of context for LLM-based inter-requirement TLR to fill this gap. Three types of context are investigated to improve automated requirements-to-requirements TLR performance. The first kind is the automatic paraphrasing of requirements to increase the chance of semantic inter-requirement similarity. Secondly, the usage of requirements as context is investigated to give the model a broader insight into project-specific information besides the requirements of a trace link candidate. Finally, project-external trace link examples are used as context to make use of similar requirement constellations between different projects. The last context type is the first implementation of dynamic few-shot prompting for LLM-based TLR. The use of context is evaluated compared to a baseline system that is not using context information. It is found that the use of context has a small but generally positive effect on TLR performance when requirement paraphrasing or external trace links are used. Using additional requirements as context has a slight negative effect on overall TLR performance. The effect of context on TLR performance is found to vary depending on the project the task is being performed on as well as on different parameters unique to each type of context.

Zusammenfassung

Beim Aufbau großer Softwaresysteme müssen Entwickler und Requirements-Ingenieure eine Vielzahl unterschiedlicher Softwareanforderungen im Blick behalten. Um die Komplexität der Verwaltung dieser Anforderungen zu vereinfachen, können Traceability-Links verwendet werden, um Beziehungen zwischen Anforderungen explizit zu machen. Trace-Links sind ein wichtiges Werkzeug für viele Aufgaben im Bereich Software-Engineering, wie zum Beispiel die Analyse von Änderungsauswirkungen oder die Sicherstellung der Konsistenz sowie die Einhaltung gesetzlicher Vorschriften. In vielen Softwareprojekten sind Traceability-Informationen nicht verfügbar, da die manuelle Erstellung von Trace-Links arbeitsintensiv, fehleranfällig und kostspielig ist. Daher werden automatisierte Tools untersucht, mit denen Trace-Links aus bestehenden Softwareprojekten zurückgewonnen werden können. Neuere Entwicklungen in diesem Bereich nutzen große Sprachmodelle (LLMs) für die Aufgabe der Rückgewinnung von Trace-Links (TLR). LLMs verfügen zwar über ein breites Weltwissen, das während ihres Trainings kodiert wird, ihnen fehlt aber in der Regel das projektspezifische Wissen, das erforderlich ist, um die Anforderungen eines Softwareprojekts vollständig zu verstehen. Diese Arbeit schlägt die Verwendung von Kontext für LLM-basierte TLR zwischen Anforderungen vor, um diese Lücke zu schließen. Es werden drei Arten von Kontext untersucht, um die Leistung der automatisierten TLR zwischen Anforderungen zu verbessern. Die erste Art ist die automatisierte Umschreibung von Anforderungen, um die Wahrscheinlichkeit einer semantischen Ähnlichkeit zwischen verwandten Anforderungen zu erhöhen. Zweitens wird die Verwendung von Anforderungen als Kontext untersucht, um dem Modell neben den Anforderungen eines Trace-Link-Kandidaten einen breiteren Einblick in projektspezifische Informationen zu geben. Schließlich werden projekt-externe Trace-Link-Beispiele als Kontext verwendet, um ähnliche Anforderungskonstellationen zwischen verschiedenen Projekten zu nutzen. Diese letzte Kontextart ist die erste Implementierung von dynamischem Few-Shot-Prompting für LLM-basiertes TLR. Die Verwendung von Kontext wird im Vergleich zu einem Basissystem bewertet, das keine Kontextinformationen verwendet. Es zeigt sich, dass die Verwendung von Kontext einen geringen, aber insgesamt positiven Effekt auf die TLR-Leistung hat, wenn Anforderungsumschreibungen oder externe Trace-Links verwendet werden. Die Verwendung zusätzlicher Anforderungen als Kontext hat einen leicht negativen Effekt auf die Gesamtleistung von TLR. Es zeigt sich, dass der Effekt des Kontexts auf die TLR-Leistung je nach dem Projekt, in dem die Aufgabe ausgeführt wird, sowie je nach verschiedenen Parametern, die für jeden Kontexttyp einzigartig sind, variiert.

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

When building large software systems, software- and requirements-engineers have to keep track of many different requirements. These requirements describe the desired functionality and properties of a software system, allowing developers to build software that solves a specific need or problem. In order to expand or modify the requirements of a system, it is important that engineers understand how different requirements are related to each other [6]. Without understanding the connections between different requirements, it may be difficult to keep requirements consistent with each other and estimate the effects of changes to certain requirements on the entire software system.

To deal with the relations between different artifacts, such as requirements, researchers and practitioners investigate the creation and recovery of trace links. Trace links make the relation between two artifacts explicit and therefore reduce the complexity of many development related tasks [11]. Identifying all possible trace links is a difficult task that is time consuming and costly. Therefore, not all software projects embrace explicit trace link creation [38]. In many software engineering projects, trace link information is nonexistent or not readily available at any given time in the development cycle. Additionally, traceability is sometimes mandated by regulatory bodies for safety-critical systems [36]. To make the creation of trace links accessible to more projects, automated systems are proposed that can recover trace links from different software engineering artifacts such as requirements documents. This task is known as automated traceability link recovery (TLR).

Many approaches for automated requirements-to-requirements TLR do not achieve high enough performance so they can be relied upon exclusively [17, 40]. Other more performant approaches require that some trace links already exist in the project, for example, to train a machine learning based system [26]. Novel approaches using large language models (LLMs) for TLR are showing promising results [12, 19]. They do, however, face the problem that LLMs do not generally have access to any project specific information, which may be vital in identifying and constructing trace links. Especially when project or domain specific terminology is used that is not generally part of an LLM's training data, the model might not understand the meaning of a term or sentence. Additionally, some trace links can be context dependent, as a single requirement might not contain all the information that is needed to make a decision.

This can be illustrated with an example: A document management system is to be developed that manages legal documents as well as technical specifications. While legal documents should never be deleted, this restriction does not apply to technical documents. For the purpose of this example, four requirements are considered:

1. The system needs to allow for the creation of legal documents.

2. The system needs to allow the creation of technical documents.
3. The system should allow the deletion of documents.
4. Legal documents may not be deleted.

In existing LLM-based systems for automated TLR, the requirements are presented to the model in pairs of two and the model is tasked with classifying whether a trace link exists [11, 26]. If the LLM is presented with the first and third requirement from this example without any additional context, it may decide that they should be linked. The model does not have access to any of the other requirements at this stage and therefore cannot know that legal documents should not be deleted. If the model is additionally presented with the fourth requirement from the example at the same time it is asked to classify if a trace link exists between the first and third requirement, it may instead decide that there should not be a connection between those requirements. Similarly, other context may prove helpful in providing further information to the model so that it can make better decisions and its performance in automated TLR tasks can be increased.

This master thesis therefore investigates the role of context for automated TLR using LLMs. Project or task specific context is provided to the LLM with the goal of improving the performance of requirements-to-requirements TLR by eliciting better reasoning from the model. Three approaches are explored to determine if the addition of internal and external context to the model's input can yield better results than the state-of-the-art:

1. Paraphrasing requirement documents
2. Using project-internal requirements as context
3. Using trace links from other projects as context

The Linking Software System Artifacts (LiSSA) framework developed by Fuchß et al. [11] is used as a basis for experiments testing these three approaches. It is extended so that the three kinds of context can be included in the TLR process. The extended version of LiSSA is evaluated on publicly available requirements data that is commonly used to benchmark automated TLR systems. In this context, the following research questions are asked:

RQ1: How does the use of context effect LLM-based requirements-to-requirements TLR performance?

RQ2: Which one of the investigated contexts results in the highest benefit to automated TLR tasks?

RQ3: How much context information should be used to increase LLM-based TLR performance?

By answering these questions, this thesis aims to shed light on the role of context in LLM-based requirements-to-requirements TLR.

In the following chapter, the foundations required for this work are introduced and core concepts are explained. Following, related work in the field of TLR is explored. The fourth chapter explains the design of the conducted experiments, introduces the used models and

datasets, and details how the extended system is evaluated. The following three chapters present one type of context each, as listed above. The specific implementation of each type of context is described and evaluated in comparison to the use of a baseline system. Chapter 8 discusses the results of the experiments from the previous chapters. Finally, the thesis concludes with a summary of the findings and an outlook for future research avenues involving context in the field of LLM-based automated TLR.

2. Foundations

This chapter explains the foundational ideas on which this thesis builds. First, the concept of requirements traceability is explored. Then, prompting techniques which can be used to realize TLR with LLMs are described. Finally, the technique of retrieval augmented generation (RAG) and its use in the context of requirements traceability is explained.

2.1. Requirements Traceability

Requirements traceability is the ability to follow a software requirement throughout the software engineering process [14]. This means that a requirement can be traced from its inception to its deployment, including any intermediate artifacts such as code or architecture models [6]. Requirements traceability can be useful to reduce the complexity of software development tasks like bug localization [34], maintenance [30] or change management [41]. When following the traces of a requirement, it should also be possible to identify any modifications or refinements which occur to the requirement itself or its related artifacts during the development process.

The relationship between a requirement and its related artifacts is called a trace link. A trace link is a connection between any pair of artifacts which are part of the development process [6]. Therefore, trace links can exist between a requirement and the code which implements it, or between a requirement and another related requirement.

While trace links can be created manually by practitioners, this approach is time-consuming and expensive [38]. Additionally, practitioners might miss links or only spend limited effort on creating trace links in the first place [15]. To fill this gap and to make trace links explicit in projects which do not spend any effort on trace link creation, automated TLR can be used. TLR is concerned with extracting existing implicit connections between two artifacts [6] and making them explicit. Several techniques exist for automated TLR, depending on the types of artifacts being investigated.

Multiple approaches for automated TLR exist. Concepts from the fields of information retrieval (IR), machine learning and deep learning are employed [15]. IR systems aim to find trace links through the use of text similarity search. Systems like vector space modeling (VSM), latent semantic indexing (LSI) or latent dirichlet allocation (LDA) are frequently employed to determine artifact similarity [15].

Another approach for automated TLR is the use of machine learning. For this approach, TLR is treated as a classification task, where the options are the existence and absence of a

link between two artifacts. This approach requires an existing set of trace links which are used to train a classifier [15]. In the automated TLR task, this classifier creates trace links between artifact pairs.

Additionally, more advanced systems create intermediate models or use word clustering [21]. More recently, approaches using LLMs are emerging due to the increased text comprehension ability and robustness to variation of LLMs [18, 19, 32].

2.2. Information Retrieval Methods

Several information retrieval (IR) methods can be used for automated TLR tasks. The rationale behind the application of IR-based methods is that trace links most likely exist between artifacts which are textually similar [17, 18]. Historically, methods such as VSM, LSI and LDA are used, with more recent approaches using language model based embeddings.

Vector space models are models in which textual artifacts are represented by vectors. The similarity between two artifacts can then be determined through vector similarity measures like cosine similarity [17]. Vectors are usually constructed by counting how often individual terms appear in the artifact. A simple implementation is bag-of-words, in which the vector has as many dimensions as there are terms in the vocabulary. The entry for each term is set to 1 only if the term appears in the artifact [15]. VSM has the disadvantage that the vectors are constructed from individual terms. A consequence of this is that vector space models cannot consider contextual information which alter the meaning of a word and will not identify synonyms as similar to one another [15].

LSI overcomes these problems by reducing the dimensionality of the vectors by grouping multiple terms together [8, 31]. When using LSI vector dimensions refer to concepts, which consist of multiple terms. This allows LSI to capture the context in which a term appears and use that information for determining term similarity [31].

Another method used for IR tasks is latent dirichlet allocation (LDA). LDA takes a vector space model as input and derives topics occurring across the set of all artifacts. It then produces a matrix relating documents to topics, where each value in the matrix describes how likely it is for an artifact to belong to a topic [15]. This matrix contains a vector for each document, describing which topics it belongs to. That vector can then be used to calculate the similarity between different documents using similarity measures like cosine similarity or by simply comparing the topics of two artifacts [15].

Instead of using VSM, LSI or LDA to create a vector representation of a document, machine learning (ML) methods can be used to create an embedding of the document [15]. An embedding, just like the output of the previously described methods, is a vector representation of a word or document. Embeddings are created by embedding models. These models are trained on large corpora of texts using unsupervised ML methods [15, 23, 43]. In contrast to methods that count the occurrence of terms in documents, like VSM, embedding models are able to preserve the word order in their embeddings [23]. Embedding models

based on language models consider the context in which a word appears. Using statistical co-occurrence phenomena in linguistics, they can encode the semantics of the words used in a document and create vector representations that reflect its meaning [15, 42, 43]. The embedding vectors can finally be used to determine document similarity. If two documents have similar contents, their embedding vectors will point to places in the vector space that are close together. If the content of two documents is not semantically similar, their embedding vectors are further apart. This property can be used for IR to retrieve documents that are semantically similar to one another using similarity measures such as the cosine similarity.

2.3. Prompting Techniques

When interacting with an LLM, a user or system can provide information to the model in text form. The model then processes the input and produces an output, which is also in text form. The input of such an interaction is called the prompt. To increase the accuracy of LLM responses and to improve their effectiveness at solving diverse tasks, several advanced prompting techniques have been developed [28, 35].

One form of prompting is to instruct an LLM to perform a task using natural language only. This paradigm is known as zero-shot prompting, as the LLM is given zero (0) examples on how to complete the task at hand [4]. In addition to zero-shot prompting, it is possible to provide the LLM with examples on how the task at hand can be solved. This technique is known as few-shot or k-shot prompting, where k is the number of examples given to the LLM [4, 9]. Such examples are given in the prompt text, with a final task for the LLM to solve on its own. A 5-shot prompt, for example, is a prompt containing 5 examples of a task and finally the input required for the model to complete the task. In addition to the task examples, few-shot prompts may also contain natural language instructions for the LLM [13].

Instead of altering the amount of task examples being available to the model, other techniques focus on altering the natural language instructions to the model. A simple task description, such as the following example, can be used to ask an LLM to convert data from one format into another.

Prompt 1: Simple prompt
Transform the following XML data into a valid JSON object.
{xml-input}

In this example prompt, the variable XML-input would be substituted by the document which is to be transformed by the model. While this format works to get a response from the model, it has been shown that the model's output can be improved by applying the chain-of-thought (CoT) prompting style [19, 37]. The format works by decomposing the task into individual reasoning steps and asking the model to explain how it solved the task. Previous work has shown that CoT prompting can provide better results at task

solving compared to a simple prompt [19, 37]. An example for a CoT prompt can be seen in Prompt 2.

Prompt 2: CoT prompt
Transform the XML data provided after (1) into a valid JSON object.
Start by identifying constituent objects which make up the main JSON and list them, enclosed in <sub-objects> </sub-objects>. Explain why you chose those objects.
Then construct the main JSON object from those parts and reply with it enclosed in <result> </result>.

(1) {xml-input}

In this prompt, the task is broken down into the two steps of sub-object identification and construction of the full object. The model is asked to provide its reasoning before finally producing the desired output.

2.4. Retrieval Augmented Generation

While LLMs can be good at solving generic tasks using world knowledge that was encoded into the models during their training, they can struggle to solve tasks that require specific knowledge that is not part of their training data. In requirements-to-requirements TLR, this means that an LLM may not have information specific to the software project in which TLR is conducted. If the LLM does not know which requirements exist in the project, it cannot identify one of them as a trace link target. Since the input size of LLMs is limited, it is not feasible to provide the content of the entire project to the model along with the task description.

Fortunately, this gap can be filled by the use of retrieval augmented generation (RAG) [25]. RAG is a method by which information related to the task is stored outside of the LLM. This information can be stored in any form that makes it accessible for future querying and retrieval. At the time of task execution, information from the storage is retrieved and injected into the prompt for the LLM [25]. This way the prompt is augmented with information that is not encoded in the LLM's world knowledge and that is not part of the task description and data. An example of a RAG system could be a system that dynamically selects the examples for a few-shot prompt from a database of task examples. To continue the example from Section 2.3, a database could exist that contains correct XML to JSON object mappings. When a new XML structure needs to be converted into JSON format, a RAG based system can first query the database for XML data that is similar to the XML that needs to be converted using an IR method. The results of this query can then be used to construct a prompt that contains the task description, the XML to be converted, and the examples drawn from external storage. The output of this system is the JSON object created by the LLM.

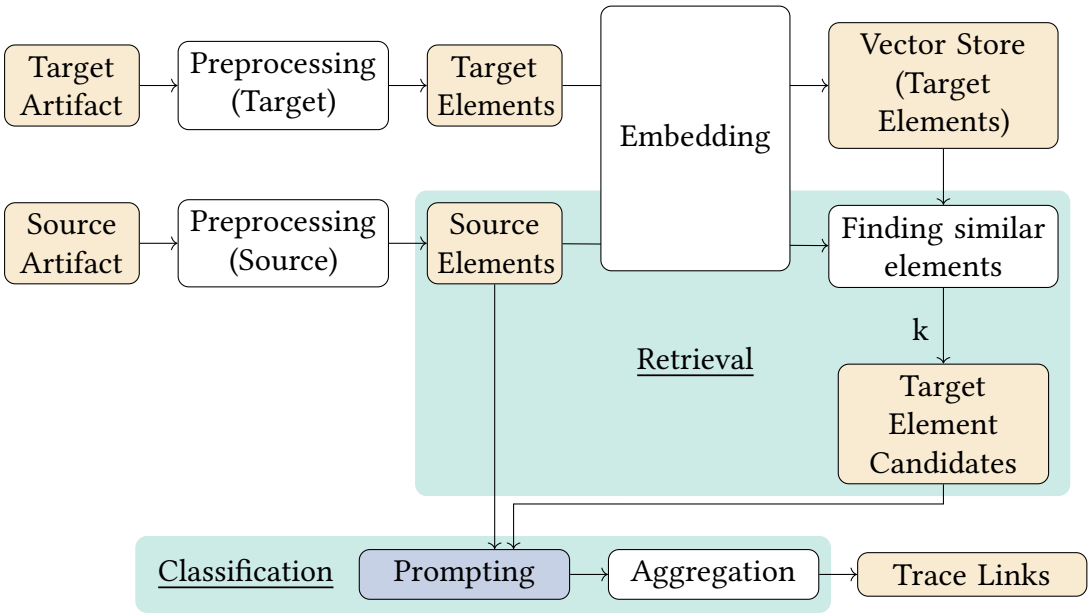

Figure 2.1.: Overview of the LiSSA pipeline by Fuchß et al. [11]. Data in orange, prompting in blue, and other processing in white.

2.5. The LiSSA Project

Linking Software System Artifacts (LiSSA) is an IR-based system for automated TLR with an LLM-based classifier proposed by Fuchß et al. [11]. This thesis heavily builds and expands on their work. In their approach, an LLM is used to classify if artifact pairs retrieved through an embedding based similarity search constitute a trace link or not. The input for the system is a set of software engineering artifacts split into source and target artifacts. These artifacts are processed through a series of processing steps, which finally lead to the creation of a list of trace links that are recovered from the input data.

2.5.1. Pipeline

The TLR task is broken down into smaller steps by the LiSSA framework. The steps are executed in sequence, forming a data processing pipeline that takes software engineering artifacts as an input and produces a list of trace links. Each of the steps can be handled by a dedicated kind of component in the framework. Figure 2.1 shows an overview of the LiSSA pipeline.

First, software engineering artifacts are loaded into the program as either a source or a target artifact. In a first step, these artifacts are preprocessed to extract individual elements. An element is one piece of information that can be extracted from an artifact [11]. It serves as the “traceable unit” [11, Sec. 3.A] on which all further operations are carried out. As Figure 2.1 shows, source and target artifacts can be preprocessed independently of one another, allowing for the use of different preprocessing steps.

After the preprocessing step, each element is a textual representation of information that can be used for retrieval [11]. In the following step, a vector representation is created from each element. An embedding model is used to create embeddings for both the source and target elements. In this step, the source and target elements are processed by the same embedding model to ensure their vector representations can be used for comparison in later steps. The vectors, along with their elements, are stored in a vector store which is the source for the IR step [11].

Following the embedding step, the retrieval step retrieves a configurable number of k target element candidates for each source element [11]. Different retrieval strategies can be used to find the top-k target elements in the vector store based on their similarity to a given source element. To determine the similarity of two elements, a similarity score is calculated based on their vector representation. After this step, k trace link candidates exist for each source element.

In the classification step, the trace link candidates consisting of a source element and one of its target element candidates are then individually presented to an LLM. The LLM is tasked with classifying whether a trace link exists between the two elements it is being presented [11]. Different approaches for classification can be used, allowing for the use of different LLMs, prompt templates, or response formats. Finally, the results of the prompting step are aggregated, and the program outputs a list of the recovered trace links.

The LiSSA framework is designed with variation points at each processing step. This allows users of the framework to develop and integrate their own components, such as element preprocessors, or modifications to the prompting and classification. The behavior of LiSSA can then be changed by configuring the framework to use different components for each of the pipeline steps.

2.5.2. Configuration

LiSSA is configured through a JavaScript Object Notation (JSON) configuration file. The configuration allows to modify which specific components are used for each pipeline step, where the system looks for its input data, and how the individual components behave. An excerpt of a LiSSA configuration can be seen in Listing 2.1.

The structure of the configuration closely follows the modules of the LiSSA pipeline, with each module being controlled by one configuration option. For most modules, the content of the configuration follows a standardized module configuration schema. An exemption are the cache directory and the gold standard configuration. These are used to configure the storage path for LLM caches and the gold standard used for the evaluation, respectively. In the sample in Listing 2.1, the module configuration for a source artifact provider can be seen. This configuration option follows the standard module configuration schema. The name attribute of the module configuration determines which module implementation is loaded for the respective pipeline step. The available values are dependent on the type of module being configured. As an example, options for the preprocessor pipeline step include *artifact* and *summarize*. If the *artifact* preprocessor is configured, LiSSA loads a simple

```
{
  ...
  "source_artifact_provider": {
    "name": "text",
    "args": {
      "artifact_type": "requirement",
       "path": "./datasets/GANNT/high"
    }
  },
  "target_artifact_provider": {...},
  "source_preprocessor": {...},
  ...
}
```

Listing 2.1.: Excerpt of a LiSSA configuration. Full samples are available in A.1.

preprocessor that transforms artifacts into elements by creating one element per artifact while retaining the exact artifact content. The *summarize* preprocessor, on the other hand, transforms artifacts into elements by summarizing their content using an LLM. To support configuration of the individual modules, each module implementation can specify additional arguments that control its behavior. By caching the responses produced by the LLMs in use in the LiSSA pipeline, the results of individual runs can be reproduced by selecting the same cache and configuration.

2.6. Metrics

In order to scientifically evaluate TLR systems, metrics are required which describe their performance and allow for the comparison of different approaches. This thesis uses three metrics to compare proposed improvements to the LiSSA system against the state of the art. All of these metrics are calculated based on the trace links that are discovered by an automated TLR system in a given test dataset. To decide which links are correct, they are compared against a gold standard, which is a collection of all trace links in a given test dataset. The comparison of the links produced by the TLR system under test and the gold standard yields four categories of link correctness, as can be seen in Figure 2.2

True positive links are all links that have been returned by the TLR system, which also appear as trace links in the gold standard. False positive links are links that have been returned by the TLR system but that do not appear in the gold standard. Trace links that have not been discovered by the TLR system but that do appear in the gold standard are categorized as false negatives.

These three categories of trace links are used to calculate the following concrete metrics. These metrics can then be used in the evaluation of different TLR systems.

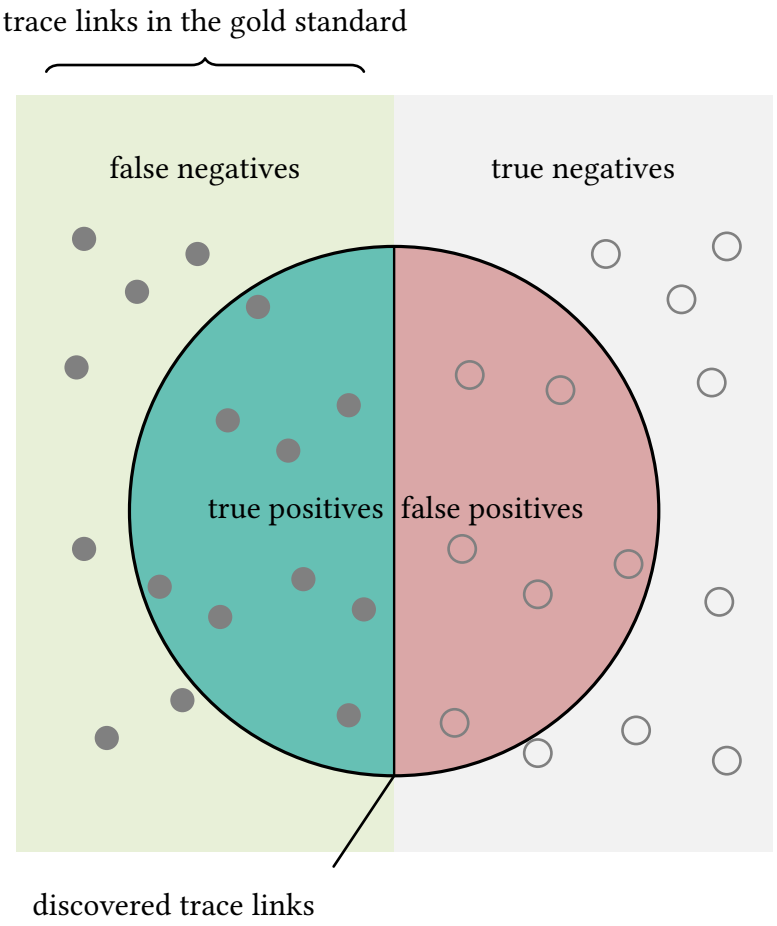

Figure 2.2.: Types of trace links for the evaluation of a TLR system

Recall The recall metric describes how many of the existing trace links have been successfully discovered by the approach [17].

$$\text{recall} = \frac{\#\text{true positive links}}{\#\text{true positive links} + \#\text{false negative links}}$$

Precision The precision metric measures how many of the trace links discovered by the approach are actually correct [17].

$$\text{precision} = \frac{\#\text{true positive links}}{\#\text{true positive links} + \#\text{false positive links}}$$

F_β score Since both the recall and precision metric are important for TLR tasks, the Fβ score exists as a combination metric. It combines recall and precision and can be weighted to favor one over the other. A β of 1 represents equal weights for recall and precision, with a $\beta > 1$ weighing recall heavier than precision [17].

$$F_\beta = \frac{1 + \beta^2}{\frac{\beta^2}{\text{recall}} + \frac{1}{\text{precision}}}$$

These metrics are commonly used for the evaluation of TLR tasks and therefore allow for comparison with other TLR systems [40]. Especially the F_1 and F_2 score can be used to qualify the overall performance of an automated TLR system. In this case, the F_1 score is suitable to indicate the overall performance of a fully automated system where precision and recall are equally important. The F_2 score gives a good indication of TLR performance for recommender systems. Since a human uses the generated trace links as suggestions, it is more important to achieve high recall and prevent missed links than to achieve higher precision [11, 19].

3. Related Work

This section explores the state of the art in automated TLR, starting with early works in the field. A special focus is set on the use of LLMs and RAG-based techniques for automated TLR tasks.

3.1. Information Retrieval for Traceability Link Recovery

Early publications on automating TLR make use of different techniques from the field of IR such as VSM, LSI and LDA. Hayes et al. [16] propose a method using vector space modeling (VSM) to retrieve target requirements for trace links to any given source artifact. They create a vector space model containing all potential target requirements and build a vector representation of the source requirement to use as a query. In addition to standard VSM, Hayes et al. [16] explore extensions to the vector model. In a first extension, they introduce key phrases that, when matched in a document, are used instead of the individual words. In a second extension a thesaurus supported retrieval algorithm is introduced that is able to match different terms that are semantically similar. Hayes et al. [16] find that while the simple VSM approach does not outperform human analysts in terms of precision or recall, it can produce results much faster. The improved methods using key phrases or a thesaurus improved the performance, with the thesaurus based approach resulting in drastic improvements in recall and improvements in precision compared to the other two options. They find that their method can reduce the workload on analysts by generating candidate links but is not suitable for unsupervised tracing.

Another approach for IR-based TLR is proposed by Zhao et al. [43]. They propose a system using embeddings to retrieve trace link candidates for a ranking approach. Zhao et al. train their own embedding model on software related wiki articles. This is done so that the embeddings are learned on a corpus belonging to the same domain as the task the embeddings are meant to solve [43]. The embedding model trained on texts from the software engineering domain is then used to create document embeddings for preprocessed software artifacts. Their approach produces a ranked list of trace link candidates. Trace link candidates are created based on the similarity of the embedding vectors of two software engineering artifacts. The higher the similarity between two artifacts, the higher it appears in the ranked list of trace link candidates. Zhao et al. [43] test their approach on the requirements-to-requirements and requirements-to-code TLR tasks. To determine which candidates from the ranked list count as trace links, they define a similarity threshold. Candidates with a similarity higher than the threshold are counted as recovered trace links,

candidates with a lower similarity are discarded. They compare their approach against a TLR method using LSI and find that for both tasks, the approach using embeddings performs better than the approach with LSI [43]. In further processing steps, Zhao et al. [43] refine their ranked list using a machine learning approach called Learning to Rank.

3.2. Large Language Models for Traceability Link Recovery

As large language models (LLMs) are gaining popularity, researchers started experimenting with their application to TLR tasks. Lin et al. [26] propose the use of the LLM BERT (Bidirectional Encoder Representations from Transformers) for TLR. In their publication, they present a pre-trained and fine-tuned version of BERT, called Trace-BERT (T-BERT), that they use to recover trace links between natural language artifacts, such as requirements or bug reports, and code artifacts. They first pre-train the BERT model on source code and documentation from open source projects. In a further training step, the model is tasked with identifying if a piece of documentation relates to a segment of code by classifying if two provided inputs are related or not. Lin et al. [26] use Python functions and docstrings, that have a clear one-to-one mapping for their training. Lastly, T-BERT is fine-tuned for the TLR task. This is accomplished by substituting the docstrings from the previous training step with natural language artifacts and performing the same classification task again. Lin et al. [26] call this transfer learning, as the learned task of matching a function with its docstring is transferred to the TLR task during fine-tuning. They report, that T-BERT cannot only be used to solve TLR tasks effectively, but also outperforms earlier approaches. Due to the transfer learning approach, the amount of labeled TLR data that is needed to train T-BERT is limited. Otherwise, enough training data is unlikely to be available to train T-BERT for new projects [26, 38]. T-BERT does, however, still require some pre-existing trace links to allow for project-specific fine-tuning. Lin et al. [26] use at least 500 pre-existing links per project for their testing, which is not an insignificant amount.

Besides training their own language models or fine-tuning existing models, researchers are recently starting to explore the use of generative LLMs such as the Claude, Gemini, or GPT models. Rodriguez et al. [37] are among the first to apply generative LLMs to the task of TLR. They show that it is generally possible to recover trace links by using chat-based LLMs without the need for extensive pre-training. This sets them apart from previous approaches such as by Lin et al. [26]. They find that variations in the prompt can lead to differences in model output, which can also lead to different trace link being discovered depending on the initial prompt. The report that CoT prompting provides comparatively consistent results across different datasets. Additionally, they find that classification prompts generally perform better compared to ranking prompts. Ranking prompts are prompts, in which the LLM is asked to rank multiple artifacts based on how closely they are related to a given source artifact. Classification prompts present two artifacts to the model and only ask for a yes or no answer. While Rodriguez et al. [37] show that the usage of generative LLMs for trace link recovery has potential, their approach is not fully automated and only

exploratively probes the space of generative LLM based TLR. They present the core concepts for TLR using generative LLMs but do not propose a system ready for practical use.

LLM-based methods are also applied to requirements-to-requirements by Ge et al.[12]. They apply different fine-tuning techniques to different sizes of the LLaMA LLM to improve its TLR performance. In contrast to Lin et al. [26] they focus on requirements-to-requirements traceability. To achieve an improvement in requirements-to-requirements TLR, publicly available datasets containing requirements are used to fine-tune the model before executing the actual TLR task. In addition to fine-tuning the model for use within a specific project, Ge et al. [12] also create cross-project datasets within the same domain and a cross-domain dataset for training and validation. They use techniques such as synonym-replacement and random-word-swapping to augment the training data and tune the model using various fine-tuning techniques such as Prompt-Tuning [24] and P-Tuning [29]. Additionally, Ge et al. [12] test summarization strategies to augment their training data. They report that summarized requirements can improve TLR performance and that the fine-tuned model generally outperforms baseline techniques even in cross-project settings. They also compare their approach to the generative LLMs GPT 3.5, GPT 4o and DeepSeek-r1 using classification prompts. While they report that their approach achieves better TLR performance than the generative LLMs, they still require large amounts of training data to be available for the fine-tuning step. Additionally, their approach cannot be applied to new projects or domains immediately, as the fine-tuning data used in their experiments always originates from the same datasets as the test data. As they show that data augmentation using synonyms or summarization can be effective in aiding the model's understanding, it may be possible to apply similar data augmentation techniques to the requirements datasets even when not employing fine-tuning techniques.

3.3. Retrieval Augmented Generation for Traceability Link Recovery

In addition to LLMs, researchers are exploring the use of RAG to improve performance in TLR tasks. Fuchß et al. [11] have shown that RAG techniques can successfully be applied to TLR tasks. They first create and store embeddings of the target artifacts of the TLR task. Then, they retrieve target artifacts based on their semantic similarity to a source artifact. Finally, the source artifact and retrieved artifacts are presented to an LLM which decides whether a trace link should be created. A more detailed description of their approach can be found in Section 2.5. Fuchß et al. [11] find that their approach outperforms state-of-the-art methods for some but not all TLR tasks. They discover that artifact preprocessing, such as the splitting of larger artifacts, is generally not beneficial when using LLMs for TLR. Additionally, they state that the use of LLMs for the classification of potential link candidates improves the TLR performance compared to pure IR-based methods. This suggests that the LLM classification step offers potential for further improvements to TLR performance.

Hey et al. [19] apply a RAG based TLR approach to requirements-to-requirements traceability. They report that the RAG-based approach outperforms other TLR techniques for requirements-to-requirements traceability. Additionally, they find that CoT prompting can be beneficial for requirements-to-requirements TLR compared to a simpler classification prompt. They also investigate and compare the performance of using different LLMs for classification. They find that while their performance is generally comparable, they achieved the best results using GPT-4o.

For requirement-to-code traceability, Ali et al. [1] explore the viability of LLMs with RAG using several different IR techniques. They propose a system that first indexes the code base. It then takes requirements as input and retrieves related parts of the code. Finally, it generates trace links between the input requirement and corresponding classes in the code. An elaborate system is proposed for the indexing step, which creates three separate indexes. One index is created containing embeddings of the project's source code. Another index contains embeddings of the project's source code augmented with LLM generated explanations of each class' functionality. The third index is a function dependency graph, which aims to capture structural semantics in the code. In order to retrieve potential target candidates for the trace link, the information from the three indexes is combined. A similarity search with an embedding of the source requirement as a query element is first conducted on the indexes containing code embeddings. In a second step, neighboring elements from the dependency graph are added to the set of potentially relevant documents. Finally, a filter is applied that removes elements that do not exceed a certain semantic similarity threshold from the retrieved set. The retrieved elements are presented to an LLM alongside the source requirement, and the LLM is asked to predict which target elements are related to the source requirement. In addition to the source requirement, Ali et al. [1] generate paraphrases of the source requirement, which are added to the prompt as context. While Ali et al. [1] do not describe the prompt in detail, they apparently use the initial requirement as the source element and the retrieved elements as potential targets for the trace link. The paraphrased requirements are used as context for the model, making this one of the first publications to add additional context to the prompt. The paraphrased requirements are not used, however, to modify the search space in the index that is used for the initial document retrieval.

In a very recently released preprint, Niu et al. [32] use LLMs with RAG for a requirements-to-requirements TLR task in an automotive environment. They use existing trace links from the same project to augment the generation step with examples and context. This makes them one of the first to augment the LLM-based TLR approach with additional context. First, they create embeddings of the project's source artifacts and artifacts with preexisting trace links. When investigating a source artifact, they then search for similar source artifacts with existing trace links in a vector store and retrieve the top-k similar trace links. These are then provided to an LLM as context for the classification step. They find that zero-shot prompting is underperforming compared to their approach augmented with project internal examples. Additionally, they suggest that a k of 3 for the top-3 most similar source requirements yields the best performance for their TLR task. It is noteworthy, though, that their publication has not yet been peer-reviewed at the time of writing.

Instead of retrieving additional context, Fuchß et al. [10] investigate a modification of the initial IR step for the link candidate selection. They use ensembles of small language models (SLMs) to reduce the amount of link candidates being presented to the LLM for classification. The IR-step in this approach works by first calculating all possible pairs of source and target requirements. Then, Fuchß et al. [10] explore two different strategies. In a first option, multiple SLMs are chained together. They are sorted in order of increasing parameter size. Each model filters the set of candidate link pairs and only forwards promising pairs to the next model. A second approach explored by Fuchß et al. employs a majority voting strategy. In this approach, the models all receive the same set of requirement pairs. A pair is kept if most of the SLMs identify it as a potential link; otherwise, it is discarded. Following both of these IR strategies, the retrieved requirement pairs are presented to an LLM for classification. They report that using SLMs for the retrieval stage can filter out many potential link pairs without compromising recall. The chained strategy produced better results compared to the majority voting approach. It outperforms baseline techniques such as LSI and VSM. The approach does not, however, outperform embedding-based retrieval, such as described in Section 2.5 and [11].

3.4. Few-shot prompting

Recent work has also been done on few-shot prompting and example selection. Brown et al. [4] present OpenAI's GPT-3 model and apply it to various natural language processing (NLP) benchmarks. They report that the model achieves higher performance on these benchmarks when few-shot prompting is used compared to one-shot or zero-shot prompting. They also find that few-shot prompting allows the model to achieve benchmark results that are comparable to state-of-the-art models that are fine-tuned to solve the specific tasks.

Liu et al. [27] conduct experiments using GPT-3 to determine which strategies are suitable for example selection. They note that the few-shot performance of GPT-3 is dependent on the examples provided in the prompt. While examples can be drawn randomly from a set of examples considered relevant to the task, Liu et al. [27] investigate more systematic approaches for example selection. They propose to provide examples to the LLM that are close to the actual task query in the embedding space of the model. This means that examples that are semantically similar to the original query are selected [27]. In order to select examples for any given task, they create embeddings for a set of examples to use as the search space. Then an embedding is created using the same embedding model for the task query of a specific task. Based on the embedding of the task query, the k nearest neighbors are selected as examples based on their distance in the embedding space. The examples are ordered by their distance from the task query and concatenated to form the context. This context is then passed to the LLM in the same prompt as the actual query. They find that their approach of selecting semantically similar examples as context heavily outperforms random example selection [27]. The higher the semantic similarity between the context and the content of the current task, the better the performance of the LLM. They also find that fine-tuning the embedding model and thereby improving how well the

model can capture semantic similarity between two sentences can lead to better example selection, further improving the performance of the LLM [27]. While Liu et al. [27] test their method for example selection on multiple NLP tasks, they do not apply their system to TLR tasks. The tasks tested by Liu et al. all require the generation of significant amounts of text. Their approach is not tested for tasks with shorter outputs, such as classification tasks. They also don't consider other forms of context besides examples for few-shot prompting in their analysis of LLM performance.

Geng et al. [13] apply few-shot prompting to the task of generating code comments. They propose a template for few-shot prompting and investigate if few-shot prompting can produce better results than zero-shot or one-shot prompting. They additionally investigate if IR-based dynamic example selection has a positive effect on few-shot performance and how the performance is influenced by example order. To this end, they test a simple code-token-based retrieval strategy as well as a strategy making use of semantic similarity. Geng et al. [13] find that few-shot prompting outperforms state-of-the-art fine-tuned models by up to 80% in the code comment generation task. Compared to random example selection, they find that both of their proposed example selection strategies achieve higher performance. They report that neither of their proposed approaches can outperform the other and that exact performance depends on the detailed parameters used for retrieval. Finally, Geng et al. [13] note that the performance of the model using the few-shot prompt increases when the provided examples are ordered based on their similarity to the problem the LLM is asked to solve.

4. Experiment Design

The goal of this thesis is to investigate the influence of context information on LLM-based TLR. An empirical evaluation is conducted on three proposed approaches for the inclusion of context in the automated requirements-to-requirements TLR task. The proposed types of context are:

1. Adding context to the vector store by **paraphrasing requirements** and thereby increasing the search space for the IR step.
2. Supplying additional project-internal **requirements as context** to the classification prompt.
3. Adding external **trace links as context** to construct a few-shot prompt.

An automated TLR system using these three types of context is compared against the state-of-the-art system. As a base for the comparison, the LiSSA system as developed by Fuchß et al. [11] is used. The proposed approaches are implemented as new modules that can be loaded into the LiSSA pipeline.

To evaluate these new additions, experiments are carried out using a set of five test datasets. An overview can be seen in Table 4.1. The test datasets are chosen because they are commonly used to evaluate tools for automated TLR [5, 43] and because they are used in previous work on LiSSA [10, 19]. Specifically, Hey et al. [19] evaluate unmodified LiSSA for the requirements-to-requirements TLR task using the same datasets. This makes them a good fit to evaluate any changes to the LiSSA pipeline that have the goal of improving requirements-to-requirements TLR results. Hey et al. [19] additionally make use of the CCHIT dataset. This dataset is not considered in this thesis as it does not directly fit the

Dataset	Domain	Number of Artifacts				
		high		low		links
GANNT [20]	Project Management	17	(100%)	69	(99%)	68
CM1-NASA [17]	Aerospace	22	(86%)	53	(57%)	45
Dronology [7]	Aerospace	99	(96%)	211	(99%)	220
MODIS [16]	Aerospace	19	(63%)	49	(63%)	41
WARC [22]	Archiving	63	(95%)	89	(89%)	136

Table 4.1.: Overview of the used datasets with their domain and the number of high level requirements (high), low level requirements (low) and trace links (links). Brackets signify the percentage of requirements that appear in trace links.

```
Create a new task which typically has a start date and an end date. Tasks are
↪  activities that one or more persons resources are expected to complete in
↪  the specified time frame.
```

Listing 4.1.: Example high-level requirement from the GANNT dataset. (Requirement ID: r1)

requirements-to-requirements TLR task. The links in this dataset connect requirements to regulatory codes. Using the same datasets as previous work allows comparing the results of state-of-the-art projects to the proposed approaches. All of the datasets consist of high level and low level requirements. Each requirement is a plaintext file with the filename serving as the requirement identifier (ID). Each file contains exactly one requirement and is sorted into either a "high" or "low" directory depending on if the requirement is a high-level or low-level requirement. During the experiments, high-level requirements are always used as the source requirements, with low-level requirements serving as targets.

The content of the individual requirement files varies from dataset to dataset. In the GANNT dataset, the high-level requirements files contain natural language descriptions of desired features or functionality. An example requirement from the GANNT dataset can be seen in Listing 4.1. The low-level requirements in the GANNT dataset generally contain a more implementation specific description of the desired functionality by mentioning individual class names and the relationships between specific classes. The high-level requirements in the WARC dataset are written in a similar way to the GANNT requirements. They, too, contain a natural language description of the desired functionality. In addition to this description, each of the requirements files in the WARC dataset also starts with its requirement ID. The high-level requirements of the WARC dataset are additionally divided into functional and non-functional requirements. This distinction is not explicitly used for this thesis; however, it is part of the requirement IDs in the dataset. Functional requirements use the prefix "FR", with non-functional requirements using the prefix "NFR". Because the requirement ID is part of the requirement content in the WARC dataset, this means that the LLM has access to these abbreviations in the classification stage. The low-level requirements of the WARC dataset also contain a more technical description; they are, however, more generic than the GANNT low-level requirements and generally do not mention specific functions. Just as the previously explored datasets, the MODIS high-level requirements also contain natural language descriptions of desired features. Its low-level requirements contain many references to specific components of the software, usually wrapped in double underscores. The components mentioned in the low-level requirements are usually not mentioned in the linked high-level requirements.

```
The DPU-CCM shall collect a TASK_HBEAT from  DPU-SCUI, DPU-CCM, DPU-DCX,
↪  DPU-TMALI, and DPU-DPA .  Non-responsive tasks will be reported in
↪  DPU_HK.
```

Listing 4.2.: Example high-level requirement from the CM1-NASA dataset. (Requirement ID: SRS5.12.3.3)

```
Default message frequency

The default message frequency for UAV state messages shall be
↪ {{UAV_MESSAGE_STATE_FREQUENCY}}
```

Listing 4.3.: Example low-level requirement from the dronology dataset. (Requirement ID: DD-743)

The requirements in the CM1-NASA dataset are distinct from the previous three in that they make heavy use of acronyms that are not explained within the dataset. An example from the CM1-NASA dataset can be seen in Listing 4.2. The high-level requirements are therefore already very technical and mention different components explicitly. Low-level requirements in CM1-NASA mention specific methods and the order in which they should be called, making them very implementation specific. Finally, the dronology dataset also makes use of acronyms in its requirements. Each of the high-level requirements in the dronology dataset starts with a very short general description that can be interpreted as a title or heading. These titles are generally not complete sentences. The titles are followed by a larger piece of text that is separated from the title using newlines. The larger texts describe the desired capabilities of individual components of the system and name them explicitly. The low-level requirements in dronology use an identical style to the high-level requirements, also consisting of two pieces of text. They only differ from high-level requirements in a smaller scope and smaller length of included text. Sometimes, high-level and low-level requirements mention the same components in their title or description. Additionally, some low-level requirements contain what appear to be placeholders instead of real values. An example can be seen in Listing 4.3.

In addition to the requirements, all datasets contain a gold standard set of trace links between the requirements. The gold standard is a comma separated values (CSV) file that can be found in the root directory of the dataset. Each line of the gold standard file contains the identifiers of two of the requirements. One of the identifiers refers to a high-level requirement; the other identifier refers to a low-level requirement. One line in the gold standard file is equivalent to one trace link. A requirement identifier can appear multiple times in the same gold standard file, allowing one requirement to be part of multiple trace links. It is noteworthy that the gold standard in the CM1-NASA and MODIS datasets only covers a low percentage of the requirements compared to the other projects. This can have two reasons:

1. The gold standard is incomplete, and it is possible to find correct trace links in the dataset that are not part of the gold standard.
2. The set of requirements is incomplete, and some of the included high-level requirements are missing related low-level requirements and vice versa.

In contrast, the GANNT dataset has particularly high coverage of its requirements.

Different LLMs from OpenAI are used for the evaluation of the proposed context approaches. Specifically, GPT-4o, GPT-4o-mini, GPT-4.1, GPT-5, GPT-5-mini, and GPT-5.1 are used.

	Goal	
	Improving the performance of LLM-based requirements-to-requirements TLR by using internal and external context.	
	Question	**Metric**
1	Does the LLM identify more of the trace links when being given context compared to without context?	Recall
2	Are fewer false positives generated when using context compared to not using context?	Precision
3	Is the system overall more reliable for requirements-to-requirements TLR tasks when using context compared to the system without using additional context.	F_1 score, F_2 score

Table 4.2.: GQM plan base

OpenAI's models are chosen as they are readily available through the API and do not require extensive installation or specialized hardware to access. GPT-4o, GPT-4o-mini and GPT-4.1 are chosen due to their usage in previous work [11, 19]. At the time of the initial testing work for this thesis, GPT-4.1 was OpenAI's most recent and advertised as most powerful model. During the creation of this thesis, OpenAI released the GPT-5, GPT-5-mini and GPT-5.1 models. They are additionally considered as OpenAI advertises them as their most capable models for complex tasks. OpenAI also released the GPT-5.2 model; however, this release was too close to the final completion date for this thesis, so the model could not be considered in testing. The non-mini models are generally chosen because they are advertised as the most capable model by the vendor at the time of testing. The mini models are used additionally, as they promise faster and more cost efficient results than their non-mini counterparts. In addition, OpenAI's text-embedding-3-large model is used to create element embeddings. This model is also chosen due to its utilization in previous work by Fuchß et al. [11] and Hey et al. [19], which allows to reuse existing caches. Additionally, the model is the most capable embedding model currently offered by OpenAI.

To ensure proper goal-based evaluation of the proposed approaches in this thesis, goal question metric (GQM)-plans according to Basili et al. [2] are created. As each of the proposed approaches for adding context follows a slightly different rationale and has different variable parameters, each approach is evaluated using an approach-specific GQM-plan. While the goal of improving the performance of LLM-based requirements-to-requirements TLR by using context remains the same across all approaches, the specific GQM-plans allow to ask questions that provide insights that are specific to each individual approach. To ensure consistent evaluation across all approaches, the individual GQM-plans share a common base that can be seen in Table 4.2. The GQM-plan includes the goal, the questions that can be asked to evaluate if it has been achieved, and which metrics can be used to answer the questions. As the overarching goal is to improve automated TLR performance, all proposed approaches are evaluated using Questions 1-3 from Table 4.2.

Each of the proposed approaches is benchmarked on the test datasets. Individual experiments are carried out using different configurations of the implemented modules on all of the

datasets. For each experiment, the metrics introduced in Section 2.6 are calculated, with recall and precision being measured experimentally. To reduce the variability in the LLM's responses, a fixed seed of 133742243 is used, and the model's temperature is set to zero when using OpenAI's GPT-4o. OpenAI's newer models, such as the GPT-5 family and GPT-4o-mini, do not support setting the temperature and therefore only allow for the default value of one. Since LLMs don't behave fully deterministically even when the same seed is used and the temperature is set to zero, all LLM responses, no matter the model used, are cached. This makes the experiments reproducible and ensures subsequent runs of the same LiSSA configuration always produce the same result.

To allow for the verification of the results presented in this thesis, a replication package [3] containing all of the relevant information is made available. It contains the implementation of required modules, LiSSA configurations used for the experiments, as well as the caches created during experiment execution.

5. Requirements Paraphrasing

During the retrieval step, LiSSA attempts to find requirements that are similar to each other by using document embeddings in a vector store [11]. It is possible, however, that requirements that are semantically similar use different synonyms and are therefore missed by the similarity-based search in the vector store. By paraphrasing requirements using different sentence structures or synonyms, the vector representation of the requirement created by the embedding model is changed, which has the potential to also change the calculated similarity between the source and target artifact. Since this might change the selection of target elements being presented to the LLM in the classification stage, it is possible for such an approach to uncover trace links that would otherwise be missed.

5.1. Automated Paraphrase Generation

When loading requirements into the TLR pipeline, LiSSA executes a preprocessing step before creating embeddings of the elements as described in Section 2.5. This preprocessing step can be modified with the goal of making paraphrases available in the vector store. Since the goal is to increase the amount of potential candidates that can be checked during the similarity search, paraphrases of the target requirements should be created. These paraphrases can then be made available to the similarity search by adding them to the vector store in addition to the original requirement text. An LLM can be used to automatically generate paraphrases for each requirement. The amount of paraphrases (n) that are created for each requirement is configurable. The LLM is simply asked to generate a paraphrase for a given artifact n-times using the prompt seen in Prompt 3. The *artifact_type* placeholder is replaced by the type of artifact that should be paraphrased. Since the only artifact type considered in this thesis is requirements, this placeholder is always replaced by the word "requirement". The *content* placeholder is replaced by the actual content of the requirement.

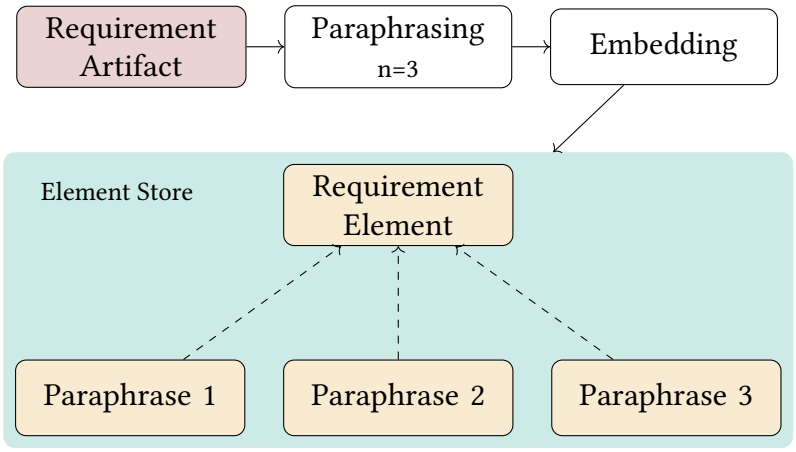

Figure 5.1.: Artifact paraphrasing process. Artifacts are shown in red, elements in yellow, and processing steps in white. The green background shows the content of the element store. The dashed arrows show that some elements hold a reference to their parent element.

Prompt 3: Paraphrase Generation
Here is the text of a software {artifact_type}:

{content}

Paraphrase the requirement, retaining its original meaning.
Answer with the paraphrased text only, without any additional explanation or formatting.

To encourage the creation of different paraphrases for the same requirement, the LLM is called with a different seed each time it generates a new paraphrase for the same artifact. The seeds are shared across artifacts, meaning that the n-th paraphrase of all artifacts is created using the same seed. To ensure reproducibility between runs using the same LiSSA configuration, the seeds used for paraphrasing are deterministically derived from a configured superseed.

In addition to the generated paraphrases, the original artifact text is retained as a further element. The preprocessing effectively transforms one artifact into $n + 1$ paraphrased elements. An illustration of this process can be seen in Figure 5.1. LiSSA allows for the hierarchical ordering of elements. Using this system, a parent element can be created that contains the text of the original artifact. The paraphrased elements can then be added as children to the first element. This structure preserves the relation between the paraphrases and the original requirement.

Following preprocessing, text embeddings are created for all elements, including the paraphrases. The text embeddings are added to LiSSA's element store. This increases the search

space for the similarity search that determines which trace link candidate pairs are presented to the LLM for classification.

In LiSSA's retrieval step, the vector store is searched for the top-k target elements that are semantically similar to a source requirement. This search can yield paraphrases as well as original requirement texts. Since the vector store contains multiple elements referring back to the same artifact, this creates multiple problems when retrieving elements to present to the LLM for classification:

1. Due to the similarity in content between an original requirement and its paraphrases, it is likely that multiple elements for the same artifact are part of the top-k.
2. Depending on the choice of n for the amount of paraphrases and the k for the amount of elements to search, the LLM may only be presented with alternatives of the same original requirement.

If the top-k similar results are passed directly to the LLM for classification, it is therefore possible that the LLM effectively considers fewer potential trace links compared to retrieving the top-k original requirements only. While the first problem only reduces the options of potential target artifacts the LLM can choose for a given source artifact, the second problem only allows the LLM to consider a trace link between a single source and target artifact pairing. Both problems are avoided by only allowing each artifact to appear once in the top-k retrieved elements. Due to the hierarchical element structure described before, the retrieved elements can be mapped to a parent element which is unique for each artifact. The resulting set of elements can then be filtered to only include each unique requirement once. Only the top-k similar results of this filtered set are then presented to the LLM for classification. The LLM can therefore still consider trace link candidates between a source requirement and k potential target requirements. Depending on which is more similar to the source requirement, the prompt used for classification can contain either the original target requirement content or the content of one of its paraphrases.

Further steps in the LiSSA pipeline, like source artifact preprocessing and classification, remain unchanged by this approach.

5.2. Evaluation

Before the requirement paraphrasing approach is used with the LiSSA pipeline, a preliminary analysis of the effect of paraphrases on the search space is conducted. Using the results from this preliminary analysis, the approach is implemented for LiSSA and evaluated compared to an unmodified baseline. Special attention is paid to the newly introduced parameters. These are the number of paraphrases created for each artifact and the models that are used for paraphrasing.

5.2.1. Preliminary Analysis

Before requirements paraphrasing is applied to LiSSA's TLR pipeline, an initial analysis is conducted. The goal of this analysis is to understand what effect the inclusion of paraphrased requirements has on the vector store. Additionally, the effect of different parameters, such as the amount of paraphrases to generate and the effect of different models, is investigated. The analysis therefore investigates the vector space created by all the vectors in the vector store.

First, metrics are calculated for the vector store containing the original requirements only. Then, the paraphrased requirements are added to the vector store with the same calculations carried out again. The prompt shown in Prompt 3 is used to paraphrase the requirements. To qualify the difference between two embeddings, the cosine similarity is used as a similarity measure. The maximum and minimum similarity between any two vectors in the vector store is calculated, as well as the average similarity over all possible vector pairs. Additionally, the range of similarities defined as the distance between the highest and lowest similarity value is calculated, and the amount of duplicate vectors is counted.

As a foundation for the analysis, the GANNT and CM1-NASA datasets are used. The datasets are chosen due to their comparatively small size and their distinct characteristics. Since the datasets are different from one another, it is hoped that the results of the preliminary analysis can be generalized for the other test datasets. Embeddings for all the requirements in the datasets and their paraphrases are created using OpenAI's text-embedding-3-large model.

5.2.1.1. Difference between LLMs

In an initial experiment, three paraphrases are derived from each requirement, leading to a total of four entries per original requirement in the element store. To generate multiple paraphrased versions of the same requirement, OpenAI's GPT-4.1, GPT-4o and o4-mini models are used. GPT-4.1 and GPT-4o are chosen because they are advertised as the most capable language models by their vendor at the time of testing. Additionally, o4-mini is investigated as it is advertised as a cheap and fast alternative with strong reasoning capabilities.

Table 5.1 shows the characteristics of the vector space when using different LLMs to paraphrase the requirements. The first column for both datasets shows the characteristics of the vector store without the addition of paraphrased requirements. For the GANNT dataset, the maximum similarity in the original dataset is 1.0. This shows, that some of the vectors in the dataset are identical, which means that the texts used to create the embeddings, and therefore some of the requirements, are also identical to one another. Closer investigation reveals that the GANNT dataset does in fact contain two pairs of identical requirements. The requirement with ID *d6_2* is equivalent to the requirement with ID *d8_1*. Similarly, the requirements with IDs *d12_6* and *d14_2* are equivalent.

	GANNT				CM1-NASA			
	ø	GPT-4.1	GPT-4o	o4-mini	ø	GPT-4.1	GPT-4o	o4-mini
max. similarity	1.00	1.00	1.00	1.00	.889	1.00	1.00	.999
avg. similarity	.329	.336	.342	.336	.372	.371	.376	.372
min. similarity	.025	-.028	.025	.007	.105	.071	.077	.046
identical vectors	2	159	147	35	0	39	52	0

Table 5.1.: Results of the preliminary vector space analysis when paraphrasing requirements using different LLMs. Three paraphrases are generated for each requirement. ø stands for the original requirements only.

The data in Table 5.1 also shows that all of the LLMs generate duplicate paraphrases for the GANNT dataset. Even though the replies from LLMs generally include a certain amount of randomness, this does not seem to be enough to avoid duplicate answers. Of note is also that GPT 4.1 produced a negative value for the minimum similarity of vectors in the GANNT dataset. This negative value can be interpreted as two elements in the vector store contradicting each other.

The average similarity across both datasets generally increases when requirements are paraphrased, with the exception of GPT 4.1 on the CM1-NASA dataset. This is the case even though the minimum similarity generally decreases when paraphrases are added to the vector store. The increase in average similarity might be due to the large amount of duplicates generated by the LLMs. Another metric that is increasing across all models is the range of similarities. An increased range indicates that the vectors representing the requirements have diversified, which may result in different requirements being selected for classification. This increase in range is generally the strongest while minimizing duplicates when o4-mini is used. In the GANNT dataset, GPT 4.1 achieved the highest range overall. This does, however, come at the cost of introducing contradictory requirements as well as a large amount of duplicates. Given that o4-mini incurs the lowest costs per request compared to the other models and achieves a wide similarity range with few duplicates, it is used for further experiments.

5.2.1.2. Amount of paraphrases

In addition to the choice of LLM used to generate the paraphrases, the characteristics of the element store can be altered by the amount of paraphrases that are generated for each original requirement. Enough paraphrases need to be generated so there is an observable effect and to cover most possible textual representations of a specific requirement. On the other hand, there should not be to many duplicates. Duplicates do not add new information to the store, but they create costs for their generation. Too many paraphrases for a single requirement may also increase the difficulty of retrieval. If, in the retrieval step, the element store is simply queried for the k most similar elements to the source element of the current classification. At this point, the element store contains the original requirement and all

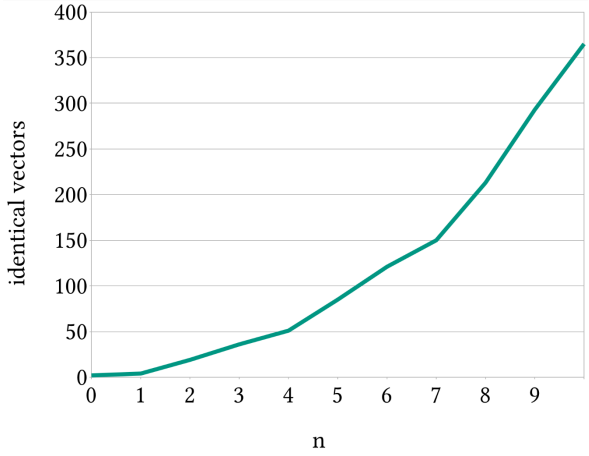

Figure 5.2.: Number of identical vectors in the vector store for different amounts of paraphrases (n) generated with o4-mini on the GANNT dataset.

of its paraphrases independently of each other. If the amount of paraphrases generated per requirement is higher or equal to the amount of elements retrieved for classification, it is possible that all retrieved elements are derived from the same requirement. Therefore, the classifier may only be presented with different variations of the same potential target requirement and can only find a trace link to that single requirement. While this can be solved by an additional filtering step as part of the retrieval, this requires extra effort that can be saved when fewer duplicates exist.

To understand approximately how many paraphrases of a requirement should be created, a second experiment is conducted. In this experiment, only OpenAI's o4-mini model and the GANNT dataset are used. For each run, the LLM is queried n-times to generate a paraphrase for a specific requirement. After each run, the same metrics as in the previous experiment are collected, and n is incremented by one. The experiment is carried out for n between 0 and 10.

Figure 5.2 shows the results of this experiment. The average similarity and number of identical vectors both increase for higher n while the minimum similarity and range stay consistent for n larger than 3. In Figure 5.3 the distribution of similarities in the element store can be seen. The overall shape of the distribution stays consistent, which is expected, as the meaning behind the requirements is not changed through paraphrasation. For higher n, the curve gets smoother, indicating that semantic gaps between requirements are filled by paraphrases. The number of duplicate elements, however, does still increase, which causes the average similarity to increase as well. The increase in the number of duplicates can be seen on the right side of the distributions shown in Figure 5.3. In the experiment with a higher number of paraphrases shown at the bottom, the tail end of the distribution appears taller than in the other two distributions. With the range of similarities and general shape of the distribution remaining consistent, there is no more information added to the vector store when generating more paraphrases. Given this observation, n should be chosen to be a low single digit number, due to higher n eventually resulting in the generation of too many duplicates without adding information.

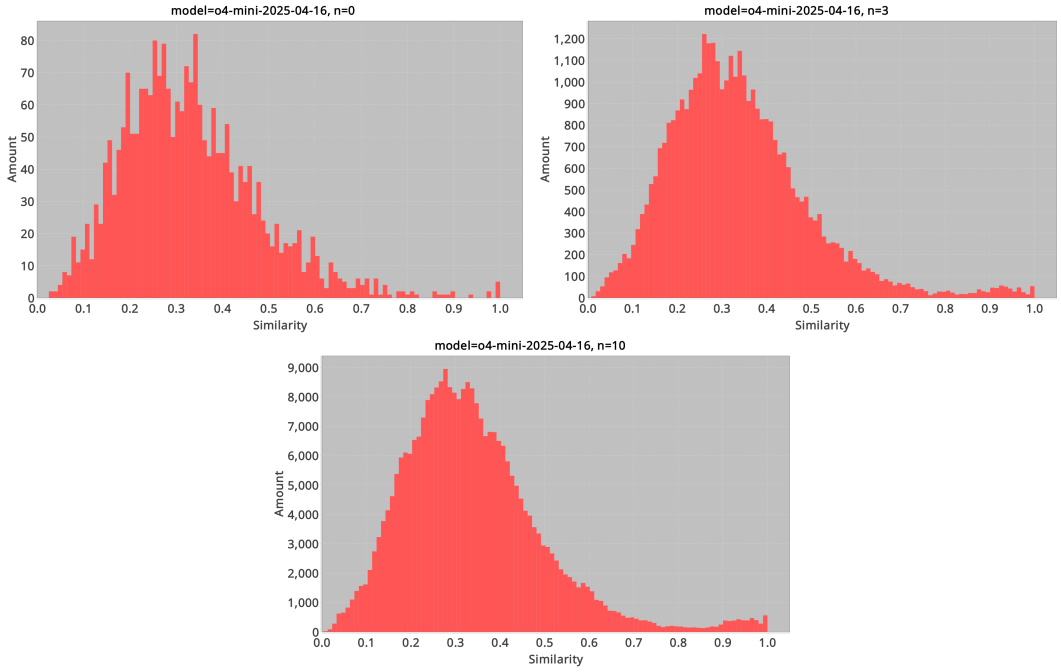

Figure 5.3.: Distribution of pairwise similarity scores for n paraphrases per original requirement.

The experiments show that the paraphrasing of requirements does affect the composition of elements in the vector store. The use of o4-mini, generating a low number of paraphrases per requirement, seems to be effective in terms of cost and the distribution of elements in the element store. Requirement paraphrasing therefore has the potential to result in different elements being chosen for classification compared to the use of non-paraphrased requirements only. The results of this preliminary study are further used to guide the implementation of requirement paraphrasing in the LiSSA pipeline for use in TLR.

5.2.2. Evaluation in the LiSSA Pipeline

The requirement paraphrasing approach described in Section 5.1 is evaluated according to the goals laid out in the GQM-plan in Table 5.2. A version of LiSSA with paraphrased requirements is compared against the state of the art LiSSA without paraphrased requirements as presented by Hey et al. [19]. Question 1 of the GQM-plan seeks to determine if more of the trace links listed in the gold standard can be found by the system with paraphrased requirements compared to the unmodified baseline. Increasing the search space is expected to have an effect on the recall metric. Question 2 asks if fewer false positives are generated when paraphrases are used for the similarity search. Finally, Question 3 aims to determine if the system is more reliable overall with requirement paraphrases compared to an unmodified baseline. The metrics described in Section 2.6 are used for the evaluation.

The embeddings used for LiSSA's IR step are created using OpenAI's text-embedding-3-large model as it is OpenAI's most capable model at the time of writing. OpenAI's GPT-4o and o4-mini are used for the paraphrasation task. o4-mini is chosen due to it's large range in

	Goal	
	Improving the performance of LLM-based requirements-to-requirements TLR through the use of requirement paraphrases	
	Question	**Metric**
1	Does the LLM identify more of the trace links when paraphrases are added to the vector store?	Recall
2	Are fewer false positives generated when paraphrases are added to the vector store?	Precision
3	Is the system overall more reliable for requirements-to-requirements TLR tasks when adding paraphrases to the vector store?	F_1 score, F_2 score

Table 5.2.: GQM plan for the addition of requirement paraphrases

similarities as shown in Section 5.2.1. GPT-4o is chosen as an additional point of comparison as it is the state-of-the-art model offered by OpenAI at the time of the evaluation. GPT-4o is also chosen for the classification task as it was shown to be effective in previous work [19]. The CoT prompt proposed by Fuchß et al. [11] is used for the classification task.

To make the experiment results reproducible, caches are used when accessing OpenAI's application programming interfaces (APIs). Embeddings, paraparaphrases, classification results are cached separately. As previous work has been conducted using the same LLMs, the caches generated there can be reused for the evaluation of this thesis. Specifically, the embedding caches from Hey et al. [19] are used, as they already contain embeddings for all low level and high level requirements of the test datasets. Additionally, the classification caches from Fuchß et al. [10] are used, as they contain classification results for all possible artifact pairings. The exact caches used for the evaluation are made available together with this thesis [3].

5.2.3. Number of Paraphrases

To determine the potential improvement in TLR performance, the models are evaluated with different numbers of paraphrases (n) inserted into the vector store. Following a baseline run without the use of paraphrases, n is varied from one through five with a constant k of 4 for the top-k retrieval mechanism. The precision and recall are measured for each run, and the F_1 and F_2 scores (see Section 2.6) are calculated. The results can be seen in Table 5.3.

When applied to the GANNT dataset, the best TLR performance across both models is achieved when using two paraphrases. The number of paraphrases is slightly higher for the CM1-NASA dataset, where $n = 3$ yields the best results across all metrics. The proposed paraphrasing approach underperforms for the dronology and MODIS datasets. The highest F_1 scores of the paraphrased approach are achieved at two paraphrases, and the highest F_2 scores at three paraphrases for these datasets. On the WARC dataset, the system performs best when three paraphrases are used.

			GPT-4o					o4-mini				
	n	0	1	2	3	4	5	1	2	3	4	5
GANNT	P	.590	**.617**	**.617**	.607	.607	.607	.593	.610	.610	.583	.593
	R	.529	**.544**	**.544**	**.544**	**.544**	**.544**	.515	.529	.529	.515	.515
	F_1	.558	**.578**	**.578**	.574	.574	.574	.551	.567	.567	.547	.551
	F_2	.541	**.557**	**.557**	.556	.556	.556	.529	.544	.544	.527	.529
CM1	P	.475	.484	.492	**.493**	.485	.485	.475	.492	.492	.492	.492
	R	.622	.667	.711	**.733**	.711	.711	.622	.689	.711	.711	.711
	F_1	.538	.561	.582	**.589**	.577	.577	.538	.574	.582	.582	.582
	F_2	.586	.620	.653	**.668**	.650	.650	.586	.638	.653	.653	.653
dronology	P	**.510**	.495	.505	.503	.498	.500	.491	.491	.490	.485	.485
	R	**.668**	.655	.659	.659	.659	.664	.650	.650	.650	.645	.645
	F_1	**.579**	.564	.572	.571	.568	.570	.560	.560	.559	.554	.554
	F_2	**.629**	.615	.621	.621	.619	.623	.611	.611	.610	.605	.605
MODIS	P	**.533**	.467	.467	.467	.467	.467	.471	.438	.444	.444	.444
	R	**.195**	.171	.171	.171	.171	.171	**.195**	.171	**.195**	**.195**	**.195**
	F_1	**.286**	.250	.250	.250	.250	.250	.276	.246	.271	.271	.271
	F_2	**.223**	.196	.196	.196	.196	.196	.221	.194	.220	.220	.220
WARC	P	**.524**	.512	.514	.520	.517	.514	.518	.518	.521	.512	.512
	R	.640	.647	**.654**	**.654**	**.654**	**.654**	.632	.647	.647	.632	.632
	F_1	.576	.571	.576	**.580**	.578	.576	.570	.575	.577	.566	.566
	F_2	.613	.615	.621	**.622**	**.622**	.621	.606	.616	.617	.604	.604
Average	P	**.527**	.515	.519	.518	.515	.515	.510	.510	.511	.503	.505
	R	.531	.537	.548	**.552**	.548	.549	.523	.537	.547	.540	.540
	F_1	.507	.505	.512	**.513**	.509	.509	.499	.504	.511	.504	.505
	F_2	.518	.520	.530	**.532**	.528	.529	.510	.521	.529	.522	.522

Table 5.3.: Precision (P), Recall (R), F_1 and F_2 score for LiSSA when n requirement paraphrases are added to the vector store at k=4. n=0 is the baseline configuration with no paraphrases.

Figure 5.4 shows the difference in F_2 score between the baseline configuration and paraphrases generated with GPT-4o at different values of n. It shows that the F_2 score only changes slightly when varying the number of paraphrases for most test datasets. The paraphrased requirements have the biggest impact on the CM1-NASA dataset with scores for other datasets remaining more stable across different values of n.

When averaging the results across all of the tested datasets, the paraphrased approach generally performs better than the baseline. When paraphrases are used, the system achieves higher recall at the cost of lower precision when compared to the baseline. This means that the LLM can identify more of the trace links when paraphrases are added to the vector store, but it also generates more false positives. The gains in recall compensate for the losses in precision, with the paraphrased approach achieving overall higher F_1 and F_2 scores than

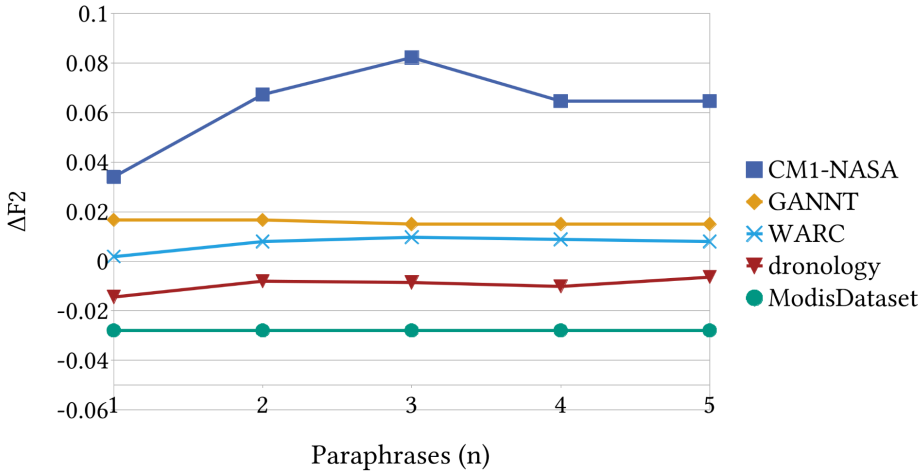

Figure 5.4.: Difference in F_2 score between the baseline and paraphrases generated with GPT-4o.

the baseline, with the best results at three paraphrases. The system is therefore on average more reliable than the state-of-the-art for automated requirements-to-requirements TLR tasks with three additional paraphrases in the vector store.

5.2.4. Difference in Models for Paraphrasing

The paraphrases generated with GPT-4o generally perform better compared to the paraphrases generated with o4-mini. For the GANNT dataset, both models outperform the baseline approach at $n < 4$. GPT-4o paraphrases also outperform the baseline at four and five paraphrases, while o4-mini performs worse than the baseline for these values. The paraphrases generated by GPT-4o perform better for all tested values for n compared to o4-mini.

Using paraphrases from either model outperforms the baseline on the CM1 dataset at all of the tested values for n. On this dataset, GPT-4o generally performs better than o4-mini with the exception of $n = 5$ at which the paraphrases generated with o4-mini achieve slightly higher precision. While paraphrases generated by both models underperform when applied to the dronology dataset, paraphrases generated by GPT-4o perform only slightly worse than the baseline. The GPT-4o paraphrases outperform the paraphrases generated with o4-mini across all metrics. As with dronology, the paraphrasing approach underperforms when applied to the MODIS dataset. MODIS is the only tested dataset in which the o4-mini paraphrases outperform the GPT-4o paraphrases in recall and in F_1 and F_2 score. Of note is that the GPT-4o paraphrases produce identical values for all metrics at all tested values of n.

o4-mini paraphrases achieve higher precision than GPT-4o paraphrases on the WARC dataset. Both models are outperformed by the baseline in this metric. The best values for recall in the WARC dataset are achieved by the GPT-4o paraphrases, with o4-mini only outperforming the baseline at two and three paraphrases. While the best F_1 score is

achieved by GPT-4o paraphrases, the baseline is only outperformed by GPT-4o at three and four paraphrases and only outperformed by o4-mini at three paraphrases. In F_2 score which favors recall over precision, the GPT-4o paraphrases consistently outperform the baseline with the o4-mini paraphrases only performing better than the baseline for n at two or three. The GPT-4o paraphrases consistently perform better in F_1 and F_2 score than the o4-mini paraphrases. On average the paraphrases generated with GPT-4o yield better results compared to both the baseline and the paraphrases generated by o4-mini.

6. Requirements as Context

LLMs possess a broad knowledge of concepts that is encoded into the model during its training. While this knowledge can be applied to different tasks such as the TLR task, the LLM can not have any specific information about the project it is being used in, simply because that information was not part of the LLM's training. This means that when the LLM is presented with trace link candidates for classification, it can only make a decision based on the knowledge in its model. This chapter proposes an approach which provides additional context information to the LLM at the time of classification. The classification prompt is enriched with additional requirements as context. Context requirements are drawn from the same project as the requirements which are currently being classified. This way, the LLM can consider alternatives to the pairing which is being presented. The additional requirements can be selected dynamically for each source and target pairing. Chosen context requirements should be related to either the source or target requirement, as adding unrelated requirements as context is unlikely to add information which is valuable to the LLM's classification task. With more requirements available to the LLM during classification, it may be able to better determine if a trace link should be created.

6.1. Dynamic Prompt Generation

Previous work has shown that providing context information through few-shot prompting can improve LLM performance [4]. Even though using requirements as context does not constitute true few-shot prompting, as the requirements by themselves are not an example of solving the TLR task, the prompt used to supply context to the LLM is constructed using a similar structure. The format is well suited for a pseudo-few-shot approach since it allows for an arbitrary number of context examples in a predictable structure. Following the results of Hey et al. [19] the few-shot prompt skeleton is combined with CoT task instructions.

A proposed prompt template for providing requirements as context can be seen in Prompt 4. This template is used to dynamically generate prompts to classify each trace link candidate pair. The placeholders marked with braces are substituted by different values depending on LiSSA's configuration and the pair of requirements being classified. The *context_type*, *source_type* and *target_type* placeholders contain the types of artifacts the system is operating on. For the task of requirements-to-requirements TLR with requirements as context, the artifact type for all of these placeholders is "requirement". The *source_content* and *target_content* placeholders are dynamically replaced by the actual content of the requirements which are being classified and should form the basis for the LLM's classification.

Prompt 4: Requirements as context prompt template
Below are artifacts from the same software system. Is there a traceability link between (a) and (b)?
Consider that the numbered {search_mode} {context_type} also exist in the same software system.
Give your reasoning, then answer with 'yes' or 'no' enclosed in <trace></trace>.

(a) {source_type}: '"{source_content}"'
(b) {target_type}: '"{target_content}"'

Context:
{context_content}

The *context_content* placeholder is filled with the content of a configurable number of additional requirements acting as context information. Each requirement is prefixed with a number and its full content is inserted into the prompt. Finally, the *search_mode* placeholder describes if the supplied context requirements are source or target requirements. This allows the model to assess if one of the context requirements might be better suited as a source for the given target requirement, or if a better target requirement could be available for the given source requirement. An example prompt with fully substituted fields is available in Section A.2.1.

6.2. Loading and Modeling Context

Before context can be used in the LiSSA pipeline, it needs to be entered into the system. To achieve this, a specific context loading step is introduced, that is executed before the rest of the LiSSA pipeline (see Section 2.5).

6.2.1. Context Loaders

In line with LiSSA's modular design, different context loaders can be implemented and then configured through LiSSA's configuration (see Section 2.5.2). To support the creation of multiple contexts, multiple context loaders can be specified as part of the same LiSSA configuration. The context loading step is responsible for the loading of context information from arbitrary artifacts and for storing it in a way that is accessible to the following pipeline steps. Figure 6.1 shows how the context interacts with the rest of the LiSSA pipeline. After context has been loaded from artifacts, it is stored in a context store. The context store contains different kinds of context which are accessible by a customizable identifier. Steps in the pipeline can query the context store with an identifier to access the context information loaded by the context loaders.

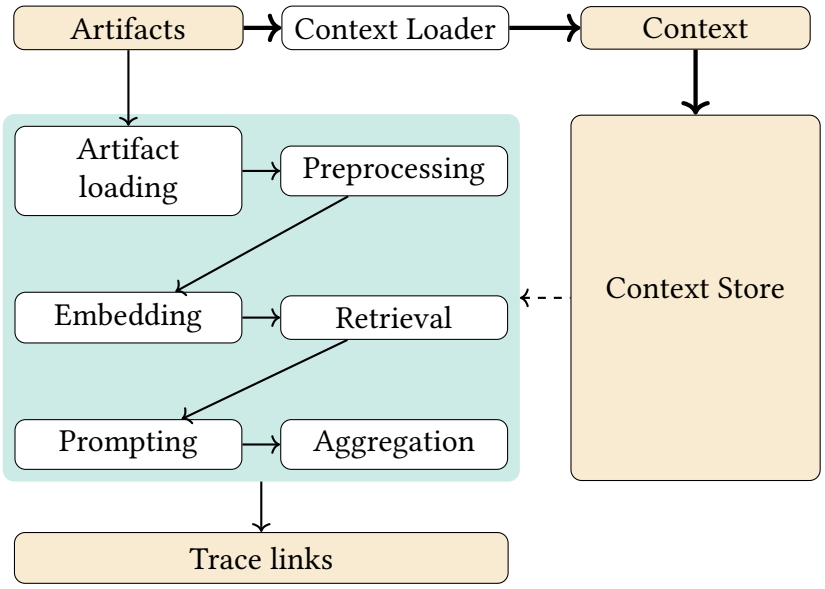

Figure 6.1.: Modified LiSSA pipeline with context loading. Orange boxes represent data and white boxes represent pipeline steps. The content of the context store is made available to all pipeline steps.

6.2.2. Context Configuration

In order to define the contexts available to LiSSA and their behavior, a new configuration option called *context_configurations* is introduced. This option configures which context loaders are used and, therefore, which contexts are available to the TLR pipeline. An example of the configuration format can be seen in Listing 6.1. The *context_configurations* configuration option takes a list of context configurations as its value. This allows to configure multiple context loaders that can each add one context to the context store. Analogous to other configuration options, the name of the context configuration determines which kind of context loader is used. The configured *context_id* is assigned to the context that is produced by the loader and can be used to retrieve the context from the context store. Each context configuration also contains a modules section. This section allows the configuration of existing LiSSA modules to be used by the context loader. That way, generic context loaders can be created that make use of other modules, such as artifact providers or preprocessors, to load different kinds of artifacts. An example of a full context configuration can be seen in Section A.1.

6.2.3. Loading Requirements as Context

Specific context loaders can be defined depending on the artifacts that make up the context and the desired format used to access the context information. For the approach of using requirements as context, a context loader called the *embeddings* context loader is implemented. It mirrors the artifact loading, preprocessing and embedding steps of the standard LiSSA pipeline. The approach can be seen in Figure 6.2. The context produced by this

```
"context_configurations":[
  {
    "name": "embeddings",
    "context_id": "requirements",
    "modules": [{...}, {...}, ...]
  },
  ...
]
```

Listing 6.1.: Excerpt from context configuration. This example configures an embedding context loader which produces a vector store and saves it to the context store with ID *requirements*.

context loader takes the form of a element store that can be queried, similar to the element store used for LiSSA's retrieval step. The element store contains both the context elements as well as their embedding vector. This design allows the context loader to reuse LiSSA modules such as the text artifact provider, artifact preprocessor and embedding component. To load the requirements from the current project into the context, the artifact provider of the context configuration is simply configured to read the same artifacts as the main LiSSA configuration.

6.2.4. Choosing Context Elements

Multiple strategies are possible for the choice of context requirements. Since the goal is to add context information that is relevant for the trace link candidate pair that is currently being classified, the selected context requirements should be related to those candidate elements. To achieve this, similarity metrics just like in the retrieval step of the LiSSA pipeline can be used. The context element store can thus simply be configured with the desired retrieval strategy and be queried by the classifier when the prompt is constructed.

Besides the retrieval strategy, the type of the context requirements may have an influence on the LLM's decision. The context requirements can either be drawn from the set of source requirements or from the set of target requirements. Additionally, the vector representation from either the source or the target requirement of the trace link candidate pair can be used as the query. The choice of the search space and the query has an implication on the semantics of the context requirements. If the source requirement is chosen as the query element and target requirements are used as context, the LLM is effectively presented with the alternatives to the candidate link pair that is currently being classified. If the target requirement is chosen as the query element and target requirements are used as context, the LLM is still presented with alternative targets. However, those may not be the same targets that were chosen for the initial classification step. The classifier can therefore be configured with a search context element store and a query context element store.

The query context is used to determine which element is used as the query to find context elements. It contains either the set of target elements or the set of source elements. When

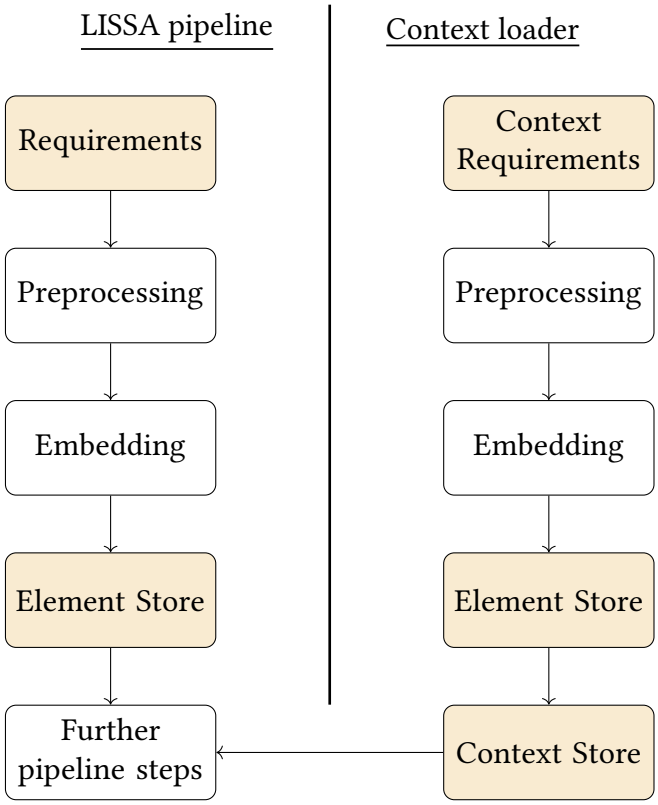

Figure 6.2.: Design of the context loader for requirements as context compared to the LiSSA artifact pipeline

constructing the prompt, the classifier attempts to find both the source and target artifact in the query context. Since the context contains only one of the two elements, the found element's vector representation is used as the query vector to find context elements. The search context also contains either the set of source elements or the set of target elements. This determines which type of element is used as context. The search context is finally searched using the query vector and the returned elements are used for prompt construction. The relationship between the source and query context is expressed as the **selection mode**. It can either be source-to-source, target-to-target, source-to-target or target-to-source.

A further aspect of context element selection is the amount of context elements to include. A larger amount of context elements may be able to give the LLM a more exhaustive overview over the project. Too many context elements may however distract the LLM from the elements that are currently being investigated.

	Goal	
	Improving the performance of LLM-based requirements-to-requirements TLR through the use of requirements as context	
	Question	**Metric**
1	Does the LLM identify more of the existing trace links when additional requirements are supplied with the prompt?	Recall
2	Are fewer false positives generated when other requirements are added as context?	Precision
3	Is the system overall more reliable for automated requirements-to-requirements TLR tasks when additional requirements are added to the prompt?	F_1 score, F_2 score
4	Which combination of query and context elements results in the largest increase in overall reliability for automated requirements-to-requirements TLR?	F_1 score, F_2 score
5	How many additional context requirements should be added to the prompt to achieve the highest overall reliability for automated requirements-to-requirements TLR?	F_1 score, F_2 score

Table 6.1.: GQM plan for using other internal requirements as context

6.3. Evaluation

The approach of using additional requirements as context is evaluated according to the goals laid out in the GQM plan in Table 6.1. Question 1 is aimed at determining if the proposed approach with additional requirements as context is able to find more of the trace links that are known in the gold standard than the baseline. Question 2 seeks to find out how many of the trace links recovered by the approach are actually correct and if that number improves compared to the baseline. Question 3 asks how the overall reliability of LiSSA changes when requirements are added as context compared to the baseline. When requirements are added as context, the context elements can either be target or source elements. Additionally, either the target or source element can be used as a query to search context elements. Question 4 therefore asks which combination of query and context element types can achieve the highest overall reliability for automated TLR tasks. Since the amount of context elements to add to a prompt can be configured arbitrarily, Question 5 tries to find an optimal number of requirements to add as context.

Experiments are carried out using the datasets introduced in Chapter 4. OpenAI's text-embedding-3-large model is used to create embeddings of all requirements for use in both the element store of the LiSSA pipeline and the element store that is used as context. For the classification step, OpenAI's GPT-4o is used. All of the element stores use the cosine similarity to calculate similarity scores. The number of target candidates for the retrieval step is fixed at k=4 because that is the average ratio of source to target requirements in the test datasets, and the same value for k has been used in previous work [19].

			target-to-target					source-to-source				
	n	0	1	2	3	4	5	1	2	3	4	5
GANNT	P	.590	.610	**.648**	.607	.632	.643	.617	.627	.597	.600	.610
	R	.529	.529	.515	.500	.529	.529	**.544**	**.544**	**.544**	.529	.529
	F_1	.558	.567	.574	.548	.576	.581	.578	**.583**	.569	.563	.567
	F_2	.541	.544	.537	.518	.547	.549	.557	**.559**	.554	.542	.544
CM1-NASA	P	.475	.491	.474	.464	.482	.400	.500	**.511**	.451	.434	.412
	R	**.622**	.578	.600	.578	.600	.578	.511	.533	.511	.511	.467
	F_1	**.538**	.531	.529	.515	.535	.473	.505	.522	.479	.469	.438
	F_2	**.586**	.558	.570	.551	.572	.531	.509	.529	.498	.494	.455
dronology	P	.510	.575	.567	.533	.533	.540	.557	.575	**.600**	.539	.584
	R	**.668**	.609	.618	.595	.586	.614	.623	.591	.600	.595	.618
	F_1	.579	.592	.591	.562	.558	.574	.588	.583	**.600**	.566	**.600**
	F_2	**.629**	.602	.607	.582	.575	.597	.608	.588	.600	.583	.611
MODIS	P	.533	.417	.364	.400	.364	.444	.500	.455	**.636**	.583	.538
	R	**.195**	.122	.098	.098	.098	.098	.122	.122	.171	.171	.171
	F_1	**.286**	.189	.154	.157	.154	.160	.196	.192	.269	.264	.259
	F_2	**.223**	.142	.114	.115	.114	.116	.144	.143	.200	.199	.198
WARC	P	.524	.601	.597	.597	.561	.585	.619	**.630**	.607	.606	.603
	R	**.640**	.610	.610	.610	.610	.610	.574	.588	.603	.610	.603
	F_1	.576	.606	.604	.604	.585	.597	.595	**.608**	.605	**.608**	.603
	F_2	**.613**	.609	.608	.608	.600	.605	.582	.596	.604	.609	.603
Average	P	.527	.539	.530	.520	.514	.522	.559	.559	**.578**	.552	.549
	R	**.531**	.490	.488	.476	.485	.486	.475	.476	.486	.483	.478
	F_1	**.507**	.497	.490	.477	.481	.477	.493	.498	.505	.494	.493
	F_2	**.518**	.491	.487	.475	.482	.479	.480	.483	.491	.485	.482

Table 6.2.: Precision (P), Recall (R), F_1 and F_2 score for LiSSA when *n* additional context requirements are supplied with the search mode in the prompt at k=4 and grouped by selection mode. n=0 is the baseline configuration with no context requirements.

Experiments are carried out for all four selection modes, using the prompt template Prompt 4. Additionally, a prompt is tested that does not include the search mode information for the source-to-source and target-to-target mode. Each of these experiments is repeated with values for the amount of context elements to include from one to five.

6.3.1. Selection Modes

To answer Question 4 of the GQM-plan, the experiment results for the different selection modes are compared. First, the source-to-source and target-to-target mode are compared

against each other. Then the source-to-target and target-to source mode are compared. The selection modes that perform the best on average in their respective comparisons can finally be compared against each other to determine which of the modes achieves the highest overall TLR performance. Table 6.2 shows the results of the experiments for the target-to-target and source-to-source context selection modes.

The GANNT dataset is a special case in this experiment, as it is the only dataset on which the proposed context approach achieves higher values for recall as well as for F_1 and F_2 score than the baseline. The target-to-target selection mode achieves higher scores than the baseline at certain numbers of context elements for both scores. The highest scores are however achieved when using the source-to-source selection mode, which also generally scores higher than the target-to-target mode.

On the CM1-NASA dataset, best results are achieved when no context requirements are used. The target-to-target selection mode achieves higher overall TLR performance than the source-to-source selection mode in this dataset. Just like on the CM1-NASA dataset, the approach using context does not outperform the baseline on the MODIS dataset. In contrast, however, the source-to-source selection mode generally achieves higher scores than the target-to-target mode. The difference in F_1 and F_2 score between the two modes on the MODIS dataset is big compared to the other datasets, with a difference of up to 10 percentage points.

Both selection modes perform similar on the dronology dataset, with only small variations in F_1 and F_2 scores. The best F_1 score in this dataset is achieved using the source-to-source selection mode. Neither of the modes is able to outperform the baseline in F_2 score for this dataset. Similar results can be seen in the experiments on the WARC dataset. Just like on dronology, the best F_1 score is achieved with the source-to-source selection mode and the best F_2 score is achieved without any context requirements. The difference in results between the two selection modes is even smaller on WARC than it is in dronology with the source-to-source mode generally achieving superior F_1 values and target-to-target mode achieving higher F_2 values.

On average, neither of the two selection modes is able to outperform the baseline configuration that is not using any additional context. Only the precision metric increases on average when adding context requirements compared to the state-of-the-art. When using source requirements as context and selecting them based on their similarity to the source requirement being classified, the average precision increases by up to 5 percentage points. For high amounts of context requirements in the target-to-target selection mode, however, the precision is lower on average compared to the state-of-the-art. The measured recall is consistently lower when using additional requirements as context across both modes. This means that the proposed approach is identifying fewer of the trace links in the datasets than the baseline. The source-to-source mode achieves higher values for average F_1 score, but the difference between the two modes is small, with up to 1.5 percentage points. For F_2 score, the target-to-target selection mode achieves higher values when only one or two context requirements are added to the prompt. For higher amounts of context elements, the source-to-source selection mode achieves higher F_2 scores than the target-to-target mode.

			source-to-target					target-to-source				
	n	0	1	2	3	4	5	1	2	3	4	5
GANNT	P	.590	.596	**.638**	.614	.632	.597	.614	.596	**.638**	.600	.593
	R	.529	.500	**.544**	.515	.529	**.544**	.515	.500	**.544**	.529	.515
	F_1	.558	.544	**.587**	.560	.576	.569	.560	.544	**.587**	.563	.551
	F_2	.541	.517	**.561**	.532	.547	.554	.532	.517	**.561**	.542	.529
CM1-NASA	P	.475	.446	.455	.458	.444	.431	.460	**.511**	.490	.458	.510
	R	**.622**	.556	.556	.600	**.622**	**.622**	.511	.511	.533	.489	.556
	F_1	**.538**	.495	.500	.519	.519	.509	.484	.511	.511	.473	.532
	F_2	**.586**	.530	.532	.565	.576	.571	.500	.511	.524	.482	.546
dronology	P	.510	.545	.552	.551	.550	.543	**.583**	.579	.569	.576	.542
	R	**.668**	.577	.582	.586	.600	.405	.577	.568	.582	.618	.591
	F_1	.579	.561	.566	.568	.574	.464	.580	.573	.575	**.596**	.565
	F_2	**.629**	.571	.576	.579	.589	.426	.578	.570	.579	.609	.580
MODIS	P	**.533**	.400	.429	.364	.333	.417	.385	.455	.429	.462	.385
	R	**.195**	.098	.073	.098	.073	.122	.122	.122	.146	.146	.122
	F_1	**.286**	.157	.125	.154	.120	.189	.185	.192	.218	.222	.185
	F_2	**.223**	.115	.088	.114	.087	.142	.141	.143	.169	.169	.141
WARC	P	.524	.563	.598	.593	.544	.568	**.642**	.618	.598	.595	.599
	R	**.640**	.588	.581	.588	.588	.581	.581	.559	.581	.574	.603
	F_1	.576	.576	.590	.590	.565	.575	**.610**	.587	.590	.584	.601
	F_2	**.613**	.583	.584	.589	.579	.578	.592	.570	.584	.578	.602
Average	P	.527	.510	.534	.516	.501	.511	.537	**.552**	.545	.538	.526
	R	**.531**	.464	.467	.477	.483	.455	.461	.452	.477	.471	.477
	F_1	**.507**	.466	.474	.478	.471	.461	.484	.482	.496	.488	.487
	F_2	**.518**	.463	.468	.476	.476	.454	.469	.462	.483	.476	.480

Table 6.3.: Precision (P), Recall (R), F_1 and F_2 score for LiSSA when n additional context requirements are supplied with the search mode in the prompt at k=4 and grouped by selection mode. n=0 is the baseline configuration with no context requirements.

Table 6.3 shows the results of the experiments when using the source-to-target and target-to-source selection modes. In source-to-target mode, target requirements are used as context. They are drawn from the context element store based on their similarity to the source requirement that is being classified. In target-to-source mode, source requirements are used as context. These are drawn from the context element store based on their similarity to the target requirement that is being classified.

For the GANNT dataset, both the source-to-target and target-to-source mode outperform the baseline that is not using requirements as context. It is noteworthy, that both modes resulted in the exact same values for all metrics when one to three context requirements are added to the prompt. Adding two context requirements using the target-to-source mode

yields the exact same results as adding one context requirement using source-to-target mode. The same is true when three context requirements are used in target-to-source mode and two context requirements in source-to-target mode. While this suggests, that target-to-source mode with one additional context requirement achieves the same results as source-to-target mode, the pattern does not continue when using four or five context requirements. Additionally, using three context requirements in source-to-target mode yields the same results as using one context requirement in target-to-source mode.

On the CM1-NASA dataset, neither of the selection modes is able to outperform the state-of-the-art in F_1 or F_2 score. The source-to-target selection mode achieves higher overall results than the target-to-source mode for this dataset, though the difference between the results of the two modes is small, with around one percentage point for most values of n. Opposite results can be seen from experiments on the dronology dataset, where the target-to-source mode outperforms the baseline in F_1 score for two out of five values of n. On this dataset, the target-to-source mode generally performs better than the source-to-target selection mode. Just as with the CM1-NASA datasets, the difference in results between the two modes is small.

Neither of the two modes is able to outperform the baseline that is not using context on the MODIS dataset. The best values on this dataset for all metrics are achieved using the baseline version of LiSSA. In general, higher requirements-to-requirements TLR performance is achieved by the target-to-source mode, with differences of up to 10 percentage points. The best result for the F_1 score when using context is however still 6 percentage points below the F_1 score of the baseline.

Both modes outperform the baseline in F_1 score on the WARC dataset. This is due to high improvements in precision compared to the baseline. The baseline achieves a recall score that is roughly 5 percentage points higher than the average recall across the two modes. This results in the baseline also outperforming both modes in F_2 score.

When scores are averaged across the test datasets, both selection modes are outperformed by the baseline configuration of LiSSA in recall, F_1 and F_2 score. The target-to-source mode generally produces better average results than the source-to-target mode. Similar to the source-to-source and target-to-target modes, the modes investigated in Table 6.3 only outperform the baseline configuration in the average precision metric. Additionally, the recall is consistently lower when context is used, meaning that these modes too are able to identify fewer trace links than the baseline approach. When comparing both modes against each other, the target-to-source mode generally achieves higher F_1 and F_2 scores than the source-to-target mode. This is interesting, since the source-to-target mode exactly replicates the IR step of the default LiSSA pipeline. In this mode, when the number of added context requirements is $k - 1$, the LLM is presented with all of the target element candidates for a given source requirement in one prompt.

To determine which of the four tested modes is most suitable for automated requirements-to-requirements TLR tasks, the average F_1 and F_2 scores can be compared across the selection modes. Figure 6.3 shows how the F_1 and F_2 scores of the different selection modes behave for different amounts of context elements. Since all of the configurations using requirements

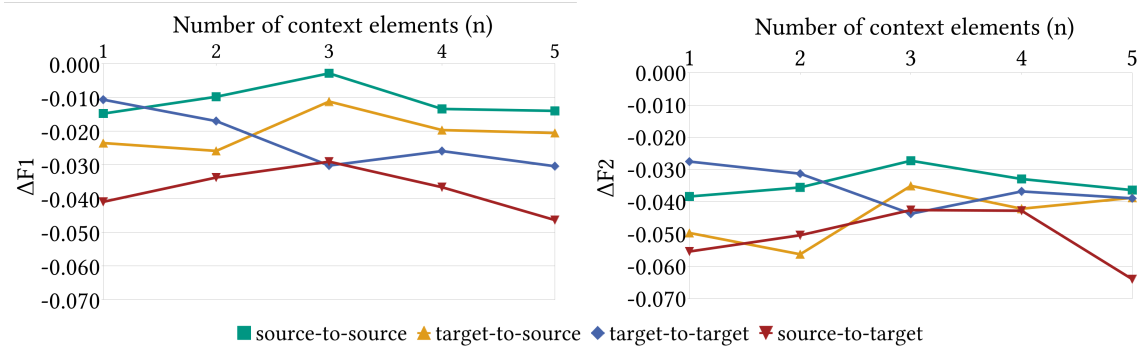

Figure 6.3.: Difference in average F_1 and F_2 score between the baseline LiSSA configuration and the different selection modes.

as context achieve worse scores than the baseline, the values are given as deltas from the baseline F_1 and F_2 scores. The figure clearly shows that the source-to-source selection mode achieves the highest F_1 scores compared to the other selection modes, except when only one requirement is added as context. Similarly, the source-to-source mode generally achieves higher F_2 scores than the other modes. When the highest scores are not achieved by the source-to-source mode, they are always achieved by the target-to-target mode. Since this is only the case for $n = 1$ for the F_1 score and $n < 3$ for the F_2 score, and in F_1 score, the target-to-target mode consistently achieves lower results than the target-to-source mode it, can be seen as performing generally worse than the source-to-source mode.

To answer Question 4 from the GQM-plan (see Table 6.1) it can be said that out of the four selection modes, the source-to-source selection mode which adds additional source requirements to the prompt achieved the best performance. The experiments also show that using additional requirements as context does not make the system overall more reliable for automated requirements-to-requirements TLR, answering Question 3 from the GQM-plan.

6.3.2. Number of Context Requirements

In addition to the context selection mode, the total number of context requirements that are added to the classification prompt can be varied. The ideal number of context requirements depends on both the selection mode and the specific dataset being used. Figure 6.4 shows the F_2 scores achieved for different amounts of context requirements. When the source-to-source selection mode is being used, the highest F_2 score on the MODIS dataset is achieved when three context elements are presented to the LLM. When the source-to-target mode is used, however, the best F_2 score for MODIS is achieved with five context elements.

Using the same source-to-target mode and five context elements, the approach produces the worst result in F_2 score for the dronology dataset out of all tested values for n. On the GANNT dataset, the best result for the source-to-target selection mode is achieved with three context elements. As has been discussed in Section 6.3.1 the GANNT, dronology and WARC datasets are the only datasets that outperform the baseline in either F_1 or F_2 score.

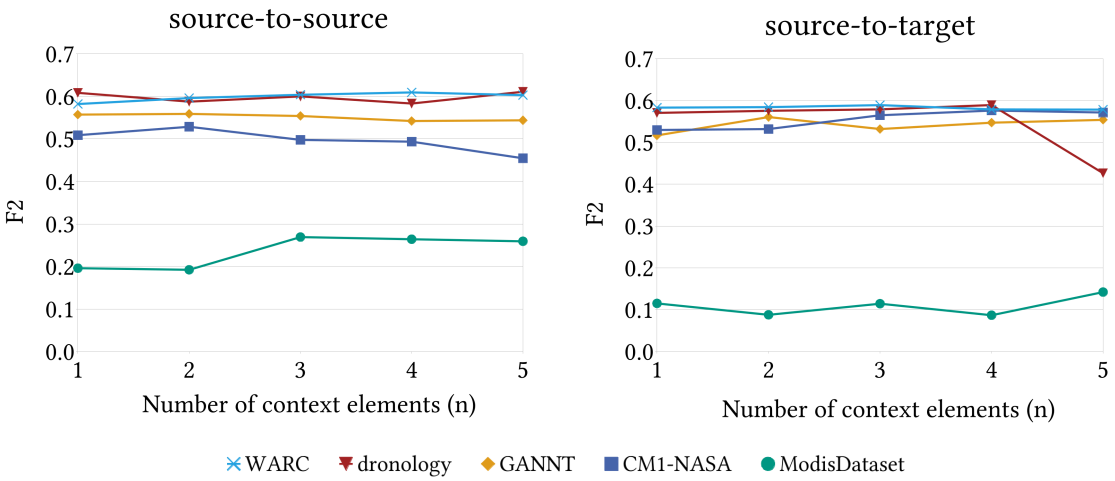

Figure 6.4.: F_2 scores achieved by LiSSA when using requirements as context. For the left graph, the source-to-source selection mode is used. On the right, the source-to-target selection mode is used.

On the GANNT dataset, the highest scores in source-to-source and source-to-target mode are achieved when two context requirements are added to the prompt. In target-to-target mode, the highest values for both scores are achieved with five context elements, and in target-to-source mode the highest scores are achieved with three context elements. The highest overall scores across all selection modes in the GANNT dataset are achieved when using two context requirements in source-to-target mode and when using three context requirements in target-to-source mode.

Dronology outperforms the baseline in F_1 score only. In target-to-target selection mode, the highest F_1 score is achieved with one additional context requirement. Source-to-source mode achieves the highest overall value for F_1 on the dronology dataset with three context requirements. Both the source-to-target and target-to-source mode achieve the highest F_1 score at four context requirements.

Like on dronology, the approach using context requirements only outperforms the baseline for WARC in F_1 score. Additionally, the highest F_1 score in target-to-target mode is also achieved using a single context requirement, just as on the dronology dataset. The absolute highest F_1 score for WARC is achieved in target-to-source mode with one context requirement. Source-to-target and source-to-source mode both achieve the highest F_1 score when two context requirements are used.

As can be seen in Figure 6.4, the fluctuation in F_2 score for different amounts of context requirements generally remains within five percentage points, with some exceptions. Notably, the MODIS dataset results jump by roughly seven percentage points going from two to three context requirements in source-to-source mode and the results for dronology drop by eleven percentage points from four to five context requirements in source-to-target mode.

When looking at the average F_1 and F_2 scores across all investigated datasets, Figure 6.3 shows that most selection modes achieve the highest scores at three context requirements. Only the target-to-target mode achieves the highest F_1 and F_2 scores when only one context

requirement is provided. In F_1 score, all other modes produce the highest results when three context requirements are used. The same is true in F_2 score, where only the source-to-target mode achieves the same result when four context requirements are used.

To answer Question 5 of the GQM-plan, the best overall requirements-to-requirements TLR performance when using context requirements can generally be achieved with three context requirements. As can be seen in Table 6.2 and Table 6.3, however, the best results are achieved when no context requirements are used at all.

6.3.3. Prompt Differences

All of the previously reported results use the prompt template Prompt 4 as described in Section 6.1. In those experiments, the *search_mode* placeholder is replaced with either the text "SOURCE" or "TARGET" depending on the type of the requirements that are added as context. To investigate whether this specific piece of information makes a difference to the classification outcome produced by the LLM, the experiment is repeated without specifying the search mode in the prompt. This means that the LLM no longer gets told if the context requirements are source or target requirements. The rest of the prompt template remains unchanged from the previous experiments. To conserve time and resources, this experiment is only carried out for the target-to-target and source-to-source selection modes. The results of this experiment can be seen in Table 6.4.

Just as in the previous experiments, the baseline is outperformed by the approach using context in all metrics only when applied to the GANNT dataset. The baseline outperforms the approach using context in F_2 score for every other dataset. In average scores, the baseline outperforms the system with requirements as context for all metrics except for precision. The highest average precision is achieved when using requirements as context in the source-to-source selection mode with one context requirement. In general, the source-to-source selection mode achieves higher average precision scores than the target-to-target mode. For all other average metrics, however, the target-to-target mode generally achieves higher values than the source-to-source mode. This is in contrast to the previous experiments, in which the source-to-source mode generally achieved higher values than the target-to-target mode. On average, neither mode is able to outperform the baseline in F_1 or F_2 score.

When comparing the prompt using context without the search_mode to the prompt with the search_mode included, the comparison is made within the same selection mode. In target-to-target mode, the average precision is higher when using a prompt without the search mode for values of $n < 5$. As soon as 5 context requirements are added to the prompt, the average precision and the precision achieved for most datasets is higher when including the search_mode in the prompt. When investigating recall values for target-to-target mode, the prompt without the search_mode consistently outperforms the prompt with search_mode. The search_mode prompt only outperforms the non-search_mode prompt in recall on the WARC dataset when four or five context requirements are used. These results carry over to the F_1 and F_2 scores, where the prompt without the search_mode consistently achieves slightly higher average values than the search_mode prompt.

			target-to-target					source-to-source				
	n	0	1	2	3	4	5	1	2	3	4	5
GANNT	P	.590	.610	**.638**	.632	.600	.610	.621	.596	.593	.583	.574
	R	.529	.529	**.544**	.529	.529	.529	.529	.500	.515	.515	.515
	F_1	.558	.567	**.587**	.576	.563	.567	.571	.544	.551	.547	.543
	F_2	.541	.544	**.561**	.547	.542	.544	.545	.517	.529	.527	.526
CM1-NASA	P	**.475**	.474	.452	.452	.422	.443	.444	.444	.456	.423	.429
	R	**.622**	.600	**.622**	**.622**	.600	.600	.533	.533	.578	.489	.533
	F_1	**.538**	.529	.523	.523	.495	.509	.485	.485	.510	.454	.475
	F_2	**.586**	.570	.579	.579	.553	.560	.513	.513	.549	.474	.508
dronology	P	.510	.549	.556	.564	.546	.533	.538	**.568**	.543	.549	.557
	R	**.668**	.632	.627	.641	.591	.623	.609	.591	.600	.632	.618
	F_1	.579	.588	**.590**	.600	.568	.574	.571	.579	.570	.588	.586
	F_2	**.629**	.613	.612	.624	.581	.602	.593	.586	.588	.613	.605
MODIS	P	.533	.600	.400	.500	.417	.385	**.636**	.556	.545	.600	.545
	R	**.195**	.146	.098	.146	.122	.122	.171	.122	.146	.146	.146
	F_1	**.286**	.235	.157	.226	.189	.185	.269	.200	.231	.235	.231
	F_2	**.223**	.172	.115	.170	.142	.141	.200	.145	.171	.172	.171
WARC	P	.524	.597	.583	.593	.569	.559	**.635**	.628	.591	.586	.577
	R	**.640**	.610	.618	.610	.603	.596	.588	.596	.596	.603	.603
	F_1	.576	.604	.600	.601	.586	.577	**.611**	**.611**	.593	.594	.590
	F_2	**.613**	.608	.610	.607	.596	.588	.597	.602	.595	.599	.598
Average	P	.527	.566	.526	.548	.511	.506	**.575**	.558	.546	.548	.537
	R	**.531**	.504	.502	.510	.489	.494	.486	.468	.487	.477	.483
	F_1	**.507**	.505	.491	.505	.480	.482	.502	.484	.491	.484	.485
	F_2	**.518**	.501	.495	.505	.483	.487	.490	.472	.486	.477	.482

Table 6.4.: Precision (P), Recall (R), F_1 and F_2 score for LiSSA when *n* additional context requirements are supplied at k=4 and grouped by selection mode without the search mode in the prompt. n=0 is the baseline configuration with no context requirements.

In source-to-source selection mode, the non-search_mode prompt achieves slightly higher precision values than the search_mode prompt. The search_mode prompt achieves higher scores when three or five context requirements are used. The strongest differences can be seen when one context requirement is used on the MODIS dataset, where the non-search_mode prompt achieves an 18 percentage point higher precision score than the search_mode prompt. When three requirements are used, however, the search_mode prompt beats the non-search_mode prompt by nine percentage points on the MODIS dataset. In recall, the results of both prompts are very similar, with the search_mode prompt achieving slightly higher average values at two, three, and five context elements. At two and three context elements, the difference is less than one percentage point. At five context elements,

however, the search_mode prompt achieves a recall value that is three percentage points higher than the value achieved by the non-seach_mode prompt. The recall results are transferred to the F_1 and F_2 score results in source-to-source mode, with the search_mode prompt slightly outperforming the non-search_mode prompt at two and three context elements and achieving higher average scores when five context elements are used.

In general, the prompt not including the search_mode achieves higher results than the prompt with search_mode in target-to-target mode, especially for smaller values of n. In source-to-source mode, results are more mixed, with the search_mode prompt performing slightly better than the non-search_mode prompt. Neither prompt is able to consistently outperform the baseline configuration.

Across all conducted experiments, the approach using additional project internal requirements as context generally achieved lower values for recall than the baseline. This means that the approach using context actually discovers fewer of the trace links than the baseline, answering Question 1 of the GQM-plan (see Table 6.1). In precision, however, the context requirements generally lead to increased values. This means, that fewer false positives are generated, answering Question 2 of the GQM-plan.

7. External Trace Links as Context

The third and final approach explored by this thesis uses external context in combination with a few-shot prompt. Previous work on LLMs has shown that few-shot prompts can improve LLM performance for complex tasks [4, 9, 13]. It uses known existing trace links from external projects. The LiSSA framework is built as a tool for traceability link recovery (TLR). As the term *recovery* indicates, the LiSSA tool is used in projects that do not possess any trace link. Using internal trace links as context is therefore impossible. Using external trace links as context allows for the approach to be directly applied to projects without existing trace links and without requiring an initial manual tracing effort.

7.1. Prompt Generation

The prompt used for the classification of two requirements with external trace links is generated dynamically for each classification. As a template, a combination of the CoT prompt suggested by Fuchß et al. [11] and the recommendations from Geng et al. [13] is created. The prompt consists of two templates which are used in two subsequent steps. For each example trace link that is added to the prompt, an example template is filled with information about that trace link. The example template can be seen in Prompt 5.

Prompt 5: Example template for individual external trace links
({i}) Source {source_type}: '"{source_content}"'
Target {target_type}: '"{target_content}"'
Link: {link}

The context trace links are numbered starting from one. As the example template is applied to each context trace link individually, the *{i}* placeholder is replaced with the number corresponding to the trace link's position in the list of examples. The *type* placeholders are replaced with the type of the source and target elements constituting the trace link. In the requirements-to-requirements application discussed in this thesis, the value for this placeholder is always "requirement". The *content* placeholders in the example template are always replaced with the content of the requirements from the external trace link. Ideally, the LLM can use this information to create parallels to the requirements being classified. The *link* placeholder is replaced with either "yes" or "no", depending on if the two requirements inserted into the template are connected by a trace link.

In a second step, the pre-filled example templates are used to generate the final classification prompt. The template for this prompt can be seen in Prompt 6.

Prompt 6: Template for external few-shot context
Below are artifacts from multiple software systems.
Considering the examples, is there a traceability link between the source and target requirement ({i}) ?
Answer with 'yes' or 'no'.

{negative_examples}
{positive_examples}
({i}) Source {source_type}: '"{source_content}"'
Target {target_type}: '"{target_content}"'
Link:

Just as the example trace links, the requirements under classification are assigned a number. The *{i}* placeholder is replaced with that number in both the instruction text and the final section of the template containing the requirements that are being classified. This way, the instructions can reference the requirements to be classified and make it explicit to the LLM that only those two requirements should be considered for the classification.

The template also contains placeholders for positive and negative examples. The *positive_examples* placeholder is replaced by a numbered list of positive trace link examples that all are filled out representations of Prompt 5. Positive examples are all examples that contain a valid trace link from an external project. Similarly, the *negative_examples* placeholder is replaced by a numbered list of negative trace link examples. Negative trace link examples are examples containing two requirements from an external project that are not connected by a trace link. The actual elements to be classified follow the example segments. Just as with the example template, the *source_type* and *target_type* placeholders are always replaced by the text "requirement" for the experiments conducted as part of this thesis. The *source_content* and *target_content* placeholders are replaced with the content of the element pair that is to be classified. In contrast to the example templates, the *link* field of the template is left empty to allow for the LLM to provide the classification result. This way, the LLM only has to continue the prompt text to provide its answer, a format suggested by Geng et al. [13].

7.2. Context Design

Compared to requirement paraphrases and using requirements as context, external trace links are a more complex type of context. This thesis does not explore how the external trace links are obtained and does not make consider the quality of the data from external projects. For the implementation of the external trace link context, preexisting data in the structure of

the test datasets is assumed. This section describes how external trace links are loaded into the LiSSA context and how that context can then be used to assemble the classification prompt.

7.2.1. Loading External Trace Links

In order to add external trace links to the prompt, different pieces of information from the external project are required. First, a list or set containing all trace links is required. Additionally, the set of source and target requirements of the external project is needed, so that their content can be accessed by the system and added to the classification prompt. Also required are the embeddings of the project internal requirements so that they can be used as queries for a similarity search.

To maintain the reusability of individual context loaders and to contain complexity, more than one context loader is used to load such a complex set of context information. Each context loader is only responsible for loading a specific kind of artifact and providing it to the context store. The context configuration allows configuring multiple context loaders that are differentiated by their context ID. Each context loader can then be individually configured as described in Section 6.2.2. By using the context store to hold multiple individual contexts, a more complex representation can then be constructed by the classifier and used to insert the context information into the prompt. Two different implementations of context loaders are required to load all the required parts to use external trace links as context.

Figure 7.1 shows how the context is structured and which context loaders are used to load which part of the context. The embeddings context loader introduced in Section 6.2.3 can be reused to load all of the needed requirements. As multiple sets of requirements are needed, one instance of the embeddings context loader can be used for each individual set. Each instance is then configured to load the requirements from a different path on the file system. Element stores containing all of the involved requirements are needed, in addition to trace link information, to access the embeddings and content of the requirements.

To load trace link information, a new trace link context loader is implemented. It uses LiSSA's gold standard module to load trace links from a file. This module is normally used in LiSSA's evaluation component. The evaluation system allows LiSSA to immediately compare the trace links it predicts with a known gold standard and self-report metrics such as precision, recall, and F_1 score. Using the gold standard module, the context loader can take a CSV file as input. The CSV file needs to contain one trace link per line, where each trace link is represented by the artifact identifier of the source requirement followed by the artifact identifier of the target requirement. The individual trace links are passed to a gold standard object that is added to the context store. The gold standard object can be retrieved from the context store and be used to either get all the trace links it contains or to get only those trace links that contain a specific element.

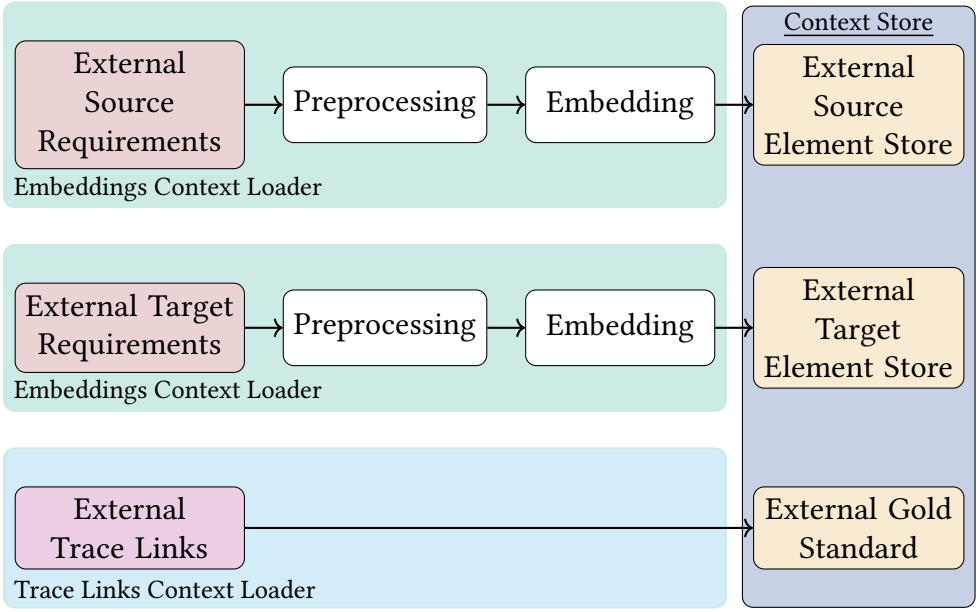

Figure 7.1.: Overview of the design of the contexts needed to use external trace links as context and the interaction of the individual contexts with their context loaders.

7.2.2. Example construction

After the external trace links and the external and internal requirements are loaded into the context, the classifier can use the different contexts to construct the few-shot examples used in the prompt. For this purpose, a new trace-links-as-context-classifier is implemented. This classifier needs to be configured with the same context IDs used to load the different kinds of artifacts. Specifically, the classifier needs the context ID of the gold standard containing the external trace links as well as the context IDs of the element stores containing the internal source and target requirements and the element stores containing the external source and target requirements. A full example configuration for using LiSSA with external trace links as context can be found in Section A.1.

For positive examples, the construction process is straightforward. Once a trace link is selected to be included in the prompt, it can be retrieved from the gold standard in the context store. Based on the source requirement ID saved in the trace link, the external source requirement can be retrieved from the external source requirements context. Similarly, the external target requirement can be retrieved from the external target requirements context using its ID that is saved in the trace link. With both requirements available, the example template (see Prompt 5) can be filled.

Constructing negative examples is more difficult, as no set of known incorrect trace links is available that can simply be loaded into the context. Technically, negative examples can be generated by arbitrarily generating trace links between any pair of random source and

target elements and checking that the link is not part of the gold standard. This approach has multiple weaknesses, though:

1. Generating negative examples randomly comes at the risk of using a certain subset of requirements excessively, or even creating duplicates.
2. The generated negative examples may not be relevant to the current classification or be of general low informational value to the LLM. Examples should allow the LLM to correctly identify edge cases or project specific terminology. If two requirements that are obviously not related are used as an example, the LLM does not gain any information.
3. Constructing examples randomly during classification makes it difficult to repeat individual experiments. In order to be able to evaluate the effect project external trace links as context can have on the automated TLR task, a deterministic system for example construction should be used.

For these reasons, a simple randomized approach is unsuitable for example construction. Instead, the classifier uses the external source requirements context and the external target requirements context to simulate LiSSA's IR step. Construction of negative examples happens only after positive examples have been constructed. In order to generate relevant negative examples, the starting point for each negative example is the source requirement of a positive example. Using this external source requirement, a similarity search is conducted in the element store containing the external target requirements. All external target requirements are ranked based on their similarity score to the external source requirement. Starting with the most similar target requirement, a trace link is created between it and the external source requirement used for the search. If this trace link is part of the external trace links context, it is discarded and the next target element is tested. These steps repeat until a trace link is created which is not part of the external trace links context. This means, that the link was not created when trace links were created in the external project, meaning it must be incorrect. The link is finally added to the set of negative examples, and the example template can be filled with the requirements' information. The process is then repeated for the source requirements of all remaining positive examples.

7.2.3. Context Selection

Few-shot prompting, while promising improvements in reasoning performance, is sensitive to the ordering and selection of items that are used as examples [13]. Experiments conducted by Geng et al. [13] and Liu et al. [27] showed that examples should best be selected and ordered based on their similarity to the actual task the LLM should be solving.

The amount of context elements (n) to be used can be configured. Based on this value, equal amounts of positive and negative examples are added to the prompt. In order to choose semantically related trace links, positive examples are chosen based on the source requirement that is being classified. Using the source requirement as the query, a similarity search is conducted in the external source requirements context. All external trace links

containing this external source requirement are retrieved from the external trace links context. If the amount of retrieved trace links is lower than the configured n, the trace links containing the next most similar external source requirement are retrieved. When an amount of trace links >= n has been retrieved from the context, the trace links are ordered by the similarity of their target requirement to the target requirement of the current classification. Any excess trace link above the configured amount are finally discarded.

Since the construction of positive and negative examples is tightly coupled, as explained in Section 7.2.2, negative examples do not need to be selected, but are constructed based on their similarity to the positive examples. Just like the positive examples, they are ordered based on the similarity of their target requirement to the target requirement of the classification. This way, the negative requirements are equally semantically relevant to the current classification as the positive examples. Positive and negative examples are treated as two separate blocks when filling the classification prompt template (see Prompt 6). The order of the two blocks can be modified by changing the position of the positive_examples and negative_examples placeholders, respectively.

7.3. Evaluation

The approach of using external trace links as context is evaluated according to the goals laid out in the GQM plan in Table 7.1. The goal of the proposed approach is to improve the overall performance of LLM-based requirements-to-requirements TLR. To evaluate if the goal is achieved, Question 1 is aiming to determine if the proposed approach manages to find more of the known trace links than the baseline. Since the approach would be less useful if it increases the amount of mistakes made by the system, Question 2 asks how many of the found trace links are actually correct and if the few-shot approach produces more correct results than the baseline. Question 3 aims to determine if the overall reliability of LiSSA increases when the proposed external context is used. The F_1 can be used to quantify if the system is overall more reliable for automated TLR with the F_2 score serving as a measure of usefulness in a recommender system. Intuitively, external context trace links are more useful if they are drawn from a project that is using similar terminology or solving similar problems. Question 4 therefore aims to determine if context is more useful if projects are sharing the same domain. To compare the effect of context on different generations of LLMs, Question 5 asks to compare the overall performance of the TLR task using OpenAI's GPT-4o, GPT-4o-mini, GPT-5, GPT-5-mini, and GPT-5.1. As external projects are required so that context can be loaded, individual test datasets are not sufficient for the evaluation of this kind of context. During the execution of each test run, the gold standard of another test dataset is used as a source of external trace links. Generally, multiple TLR runs are executed for each dataset, with each run using a different test dataset as a source for context. All of the conducted experiments use a value of $n = 4$, meaning that four positive examples and four negative examples are added to each classification prompt.

	Goal	
	Improving the performance of LLM-based requirements-to-requirements TLR by using external trace links as context.	
	Question	**Metric**
1	Does the LLM identify more of the existing trace links when trace link examples from external projects are provided?	Recall
2	Are fewer false positives generated when external trace links are used as context?	Precision
3	Is the system overall more reliable for automated requirements-to-requirements TLR tasks when external trace link examples are added to the prompt?	F_1 score, F_2 score
4	Do external trace link examples from the same general domain have a different influence on overall TLR performance than examples from a different domain?	F_1 score, F_2 score
5	Which one of the current and last generation LLMs can profit the most when external trace links are provided as context?	F_1 score, F_2 score

Table 7.1.: GQM plan for using project external trace links as context

7.3.1. Example Ordering

Geng et al. [13] note that the ordering of examples has an impact on few-shot performance. Examples can be ordered in either positive-first or negative-first order. In the positive-first example order, the positive examples are provided before the negative examples. In the negative-first example order, the negative examples are provided before the positive examples. Using OpenAI's GPT-5-mini, experiments are conducted in which both orders of context examples are tested. The results of the experiments can be seen in Table 7.2.

On the GANNT dataset, higher precision values are achieved with negative-first ordering. Though both ordering methods using external trace links as context outperform the baseline with between five and thirteen percentage points higher precision values. The negative-first ordering also achieves higher recall values on the GANNT dataset than the positive-first ordering. Neither ordering method outperforms the baseline, however. In F_1 score, the approach with trace links as context outperforms the baseline. With negative-first ordering, this is true regardless of which dataset is used as the source for external trace links. In contrast, with positive-first ordering, the baseline is only outperformed when the dronology and MODIS datasets are used to supply the context. In F_2 score, the context approach does not outperform the baseline on the GANNT dataset, no matter the ordering of the examples. Slightly higher F_2 scores are achieved with negative-first ordering compared to positive-first ordering.

On the CM1-NASA dataset, higher precision values are achieved when using positive-first ordering compared to negative-first ordering. Both ordering methods consistently outperform the baseline in precision values, with increases between nine and 23 percentage points. The baseline achieves higher recall values than the approach using context. Generally,

		positive-first				negative-first			
	Context source	P	R	F_1	F_2	P	R	F_1	F_2
GANNT	**Baseline**	.571	**.529**	.550	**.537**	.571	**.529**	.550	**.537**
	CM1-NASA	.627	.471	.538	.495	.636	.515	.569	.535
	dronology	**.642**	.500	.562	.523	.648	.515	**.574**	**.537**
	MODIS	.625	.515	**.565**	.534	.636	.515	.569	.535
	WARC	.620	.456	.525	.481	**.654**	.500	.567	.525
	Average	.628	.485	.547	.508	.644	.511	.570	.533
CM1-NASA	**Baseline**	.409	**.600**	.486	.549	.409	**.600**	.486	**.549**
	GANNT	**.531**	.578	**.553**	**.568**	**.500**	.556	**.526**	.543
	dronology	.469	.511	.489	.502	.490	.556	.521	.541
	MODIS	.490	.556	.521	.541	.471	.533	.500	.519
	WARC	.500	.533	.516	.526	.490	.533	.511	.524
	Average	.498	.544	.520	.534	.488	.544	.514	.532
dronology	**Baseline**	.435	**.686**	.533	.615	.435	**.686**	.533	.615
	GANNT	**.559**	.645	**.599**	**.626**	.522	.655	.581	**.623**
	CM1-NASA	.421	.195	.267	.219	**.570**	.609	**.589**	.601
	MODIS	.543	.636	.586	.615	.524	.641	.577	.614
	WARC	.540	.614	.574	.597	.545	.636	.587	.616
	Average	.557	.625	.589	.610	.540	.635	.583	.613
MODIS	**Baseline**	.313	**.244**	.274	**.255**	.313	**.244**	.274	.255
	GANNT	**.474**	.220	**.300**	.246	.381	.195	.258	.216
	CM1-NASA	.421	.195	.267	.219	.444	.195	.271	.220
	dronology	.429	.220	.290	.243	**.474**	.220	.300	.246
	WARC	.391	.220	.281	.241	.435	**.244**	**.313**	**.267**
	Average	.429	.213	.285	.237	.433	.213	.285	.237
WARC	**Baseline**	.445	**.691**	.542	.623	.445	**.691**	.542	.623
	GANNT	**.558**	.676	**.611**	**.649**	.517	.684	.589	.642
	CM1-NASA	.551	.640	.592	.620	**.551**	.676	**.607**	**.647**
	dronology	.511	.662	.577	.625	.508	.684	.583	.640
	MODIS	.492	.684	.572	.634	.506	.669	.576	.628
	Average	.528	.665	.588	.632	.520	.678	.589	.639

Table 7.2.: Precision (P), Recall (R), F_1, and F_2 score for LiSSA when using external trace links as context. GPT-5-mini is used for classification at $k = 4$ with 4 positive and 4 negative examples. The baseline is created using LiSSA with GPT-5-mini without any context. For the baseline, the CoT prompt that can be seen in Prompt 2 is used. Averages are calculated across context sources.

the positive-first ordering achieves higher values than the negative-first ordering. Similarly, higher F_1 scores are achieved by the positive-first ordering, with both ordering methods consistently outperforming the baseline. In F_2 score, the approach using context only manages to outperform the baseline with positive-first ordering and when using the GANNT

dataset as the source for the context information. The negative-first ordering does not outperform the baseline in F_2 score and generally achieves worse scores on the CM1-NASA dataset than the positive-first ordering.

The approach using external trace links as context outperforms the baseline in precision on the dronology dataset. Contrary to the previously investigated datasets, the positive-first ordering does not consistently outperform the baseline in precision. When using the CM1-NASA dataset as the trace link source, the approach is unable to outperform the baseline with positive-first ordering. When ordering the examples with negative-first ordering, the approach using context consistently outperforms the baseline in precision values. Neither ordering approach can outperform the baseline in recall values when external trace links are used as context. Higher values in recall on the dronology dataset are generally achieved when using negative-first ordering. While both ordering methods manage to outperform the baseline in F_1 and F_2 score, better results are again achieved with negative-first ordering. In F_1 score, the positive-first ordering again does not outperform the baseline when the CM1-NASA dataset is used as the source for the external trace links. In F_2 score, positive-first ordering only outperforms the baseline when the GANNT dataset is used as external context source. The negative-first ordering consistently outperforms the baseline on dronology in F_1 score and only achieves lower scores than the baseline in F_2 score when the CM1-NASA or MODIS datasets are used as sources for context.

On the MODIS dataset, both ordering methods are able to consistently outperform the baseline in precision values. The negative-first ordering generally achieves higher precision values than positive-first ordering. The only exception is when the GANNT dataset is used as the source for context, in which case the positive-first ordering achieves a higher value by almost ten percentage points. Neither ordering method outperforms the baseline in recall values. Only the negative-first ordering is able to at least achieve the same value as the baseline when the WARC dataset is used as a context source. In general, recall values are very similar between both ordering methods, with the negative-first ordering only slightly outperforming positive-first ordering with WARC as a context source. Both ordering methods are able to outperform the baseline in F_1 score. Both methods achieve higher scores than the baseline on MODIS when the dronology and WARC datasets are used as context. The positive-first ordering additionally achieves a higher F_1 score than the baseline when the GANNT dataset is used as the source for external trace links. Just as with the precision metric, better values in F_1 score are generally achieved by the negative-first ordering, with the exception of when the GANNT dataset is used as context. The positive-first ordering is unable to outperform the baseline in F_2 score. The negative-first ordering only outperforms the baseline in F_2 score when WARC is used as the context source. With the exception of the GANNT dataset, better results are again generally achieved by the negative-first ordering.

When LiSSA is applied to the WARC dataset, both ordering methods consistently outperform the baseline in precision values. The positive-first ordering generally achieves slightly higher precision values compared to the negative-first ordering. The only exception is when the MODIS dataset is used as a source for the external trace links. In recall values, the approach using external trace links as context does not manage to outperform the baseline on WARC

no matter the ordering method. Generally, higher results for recall are achieved using negative-first ordering compared to positive-first ordering, again except for when MODIS is used as a context source, in which case the positive-first ordering achieves better results. The approach using context consistently outperforms the baseline in both F_1 and F_2 score. The only exception is when positive-first ordering is used and the external context is drawn from the CM1-NASA dataset, in which case the baseline achieves a higher F_2 score than the approach using external trace links. F_1 scores are very similar, differing at most by two percentage points between the two ordering modes. Generally, higher results for F_1 score are achieved by negative-first ordering. Only when GANNT is used as the source of external context does the positive-first ordering achieve a higher F_1 score. Similar results can be seen in F_2 score with the positive-first ordering outperforming the negative-first ordering when the external context is drawn from the GANNT or MODIS datasets.

Overall, the performance of the two ordering modes is dependent on the dataset on which the TLR task is conducted as well as which dataset is used to provide the context trace links. While negative-first ordering generally performs better than positive-first ordering on the GANNT, dronology, MODIS, and WARC datasets, positive-first ordering generally outperforms negative-first ordering on the CM1-NASA dataset. This indicates that across the tested datasets, negative-first ordering can more often achieve better TLR performance than positive-first ordering.

7.3.2. Differences Between LLMs

In addition to the experiments using GPT-5-mini, comprehensive experiments using different datasets as context sources are conducted on OpenAI's GPT-4o-mini. The results of these experiments can be seen in Table 7.3. To get a broader picture of the variations between different state-of-the-art LLMs, experiments are also conducted using OpenAI's GPT-4o, GPT-5 and GPT-5.1. These experiments are limited to the three datasets from the aerospace domain, which are the CM1-NASA, dronology, and MODIS datasets. Their results can be seen in Table 7.4. All of the experiments using other LLMs than GPT-5-mini are conducted using negative-first ordering. Therefore, only the results using negative-first ordering with GPT-5-mini are considered for comparison. Experiments for the GANNT and WARC datasets are only carried out using GPT-4o-mini and GPT-5-mini. This section is strictly concerned with the effects of external trace links on the LLMs' TLR performance. While tests have been carried out using internal trace links as context with GPT-5-mini, these findings are addressed in Section 7.3.3.

In baseline performance, GPT-5-mini achieves higher precision than GPT-4o-mini on the GANNT dataset. GPT-4o-mini does, however, achieve higher recall values. This leads to GPT-4o-mini achieving higher baseline F_1 and F_2 scores than GPT-5-mini. When looking at the differences between the baseline and the approach using external context, precision results for GPT-5-mini improve by between 8 and 6.5 percentage points when context is used. If GPT-4o-mini is used, precision on the GANNT dataset compared to the baseline fluctuates between 7.5 percentage points and 3 percentage points when context is used, depending on the source of the context elements. In recall, GPT-5-mini loses between 7

			context source				
		Baseline	GANNT	CM1-NASA	dronology	MODIS	WARC
GANNT	P	.561		.576	**.636**	.531	.576
	R	**.544**		.279	.206	.250	.279
	F_1	**.552**		.376	.311	.340	.376
	F_2	**.544**		.311	.238	.280	.311
CM1-NASA	P	.367	.769		**1.00**	.875	.889
	R	**.644**	.222		.067	.156	.178
	F_1	**.468**	.345		.125	.264	.296
	F_2	**.560**	.259		.082	.186	.212
dronology	P	.510	.624	**.672**		.627	.639
	R	**.668**	.491	.409		.482	.482
	F_1	**.579**	.550	.508		.545	.549
	F_2	**.629**	.513	.444		.505	.507
MODIS	P	.314	.500	.500	**1.00**		.667
	R	**.268**	.049	.049	.049		.049
	F_1	**.289**	.089	.089	.093		.091
	F_2	**.276**	.060	.060	.060		.060
WARC	P	.433	.857	**.871**	.843	.810	
	R	**.669**	.397	.397	.316	.375	
	F_1	.526	.543	**.545**	.460	.513	
	F_2	**.603**	.445	.446	.361	.420	

Table 7.3.: Precision (P), Recall(R), F_1, and F_2 score for LiSSA when using external trace links as context. GPT-4o-mini is used for classification at $k = 4$ with 4 positive and 4 negative examples. The baseline is created using LiSSA with GPT-4o-mini without any context. For the baseline, the CoT prompt that can be seen in Prompt 2 is used.

and 1.5 percentage points when context is used compared to its baseline values. GPT-4o-mini with external trace links as context achieves recall values on the GANNT dataset that are between 33 and 26.5 percentage points worse compared to its baseline values. In the combined F_1 score, GPT-5-mini with context achieves 1.7 to 2.4 percentage points higher scores than the baseline. With context, GPT-4o-mini actually achieves F_1 scores that are 24 to 17.5 percentage points lower than its baseline on the GANNT dataset, owing to its poor recall performance. This continues in F_2 scores where GPT-5-mini with context matches the baseline or loses up to 1 percentage point. GPT-4o-mini loses between 30.5 and 23 percentage points in F_2 score compared to the baseline on the GANNT dataset. The increase in both scores when using GPT-5-mini paired with the decrease when using GPT-4o-mini results in GPT-5-mini outperforming GPT-4o-mini on the GANNT dataset when external trace links are used as context.

					context source								
		Baseline			CM1-NASA			dronology			MODIS		
	GPT-	4o	5	5.1	4o	5	5.1	4o	5	5.1	4o	5	5.1
CM1-NASA	P	.475	.482	**.632**				**.720**	.551	.545	**.600**	.556	.482
	R	**.622**	.600	.533				.400	**.600**	.533	.333	.556	**.600**
	F_1	.538	.535	**.578**				.514	**.574**	.539	.429	**.556**	.535
	F_2	**.586**	.572	.550				.439	**.590**	.536	.366	.556	**.572**
dronology	P	.510	.455	**.636**	**.821**	.638	.665				**.703**	.597	.665
	R	.668	**.695**	.605	.291	**.577**	**.577**				.409	**.645**	.577
	F_1	.579	.550	**.620**	.430	.606	**.618**				.517	**.620**	.618
	F_2	**.629**	**.629**	.611	.334	.589	**.593**				.446	**.635**	.593
MODIS	P	**.533**	.400	.455	**.667**	.500	.563	**.800**	.474	.500			
	R	.195	**.244**	.122	.098	.195	**.220**	.098	**.220**	**.220**			
	F_1	.286	**.303**	.192	.170	.281	**.316**	.174	.300	**.305**			
	F_2	.223	**.265**	.143	.118	.222	**.250**	.118	.246	**.247**			

Table 7.4.: Precision (P), Recall (R), F_1, and F_2 score for LiSSA when using external trace links as context. Grouped by context source at $k = 4$ with 4 positive and 4 negative examples. The baseline is created using LiSSA with the respective LLM without any context. For the baselines, the CoT prompt that can be seen in Prompt 2 is used.

On the WARC dataset, baseline results in all metrics for GPT-5-mini are already higher than for GPT-4o-mini. In F_1 score, GPT-5-mini with context achieves improvements over the baseline score between 3 to 6.5 percentage points. GPT-4o-mini also generally achieves higher F_1 scores of up to 2 percentage points when context is used. In one case, however, GPT-4o-mini with context loses 6.5 percentage points compared to the baseline. With GPT-5-mini achieving bigger improvements in F_1 score than GPT-4o-mini on the WARC dataset when context is used, it manages to keep outperforming GPT-4o-mini in this metric. In F_2 score, GPT-4o-mini with context loses between 15.5 to 24 percentage points to the baseline WARC results. GPT-5-mini with context, in contrast, achieves F_2 scores that are 0.5 and 2.5 percentage points higher than the baseline values. Equally to the F_1 score results, this means that GPT-5-mini outperforms GPT-4o-mini on the WARC dataset both in absolute values and by achieving bigger improvements than GPT-4o-mini.

Since the experiments using the GPT-4o, GPT-5 and GPT-5.1 LLMs are only conducted on the aerospace related datasets, and only the aerospace related datasets are used as context sources, all further comparisons in this section will only consider results obtained when aerospace datasets are used as context sources. On the CM1-NASA dataset, the highest baseline F_1 score is achieved by GPT-5.1, followed by GPT-4o, GPT-5, GPT-5-mini, and GPT-4o-mini in order. When external trace links are added as context to the CM1-NASA dataset, GPT-5.1 loses approximately 4 percentage points compared to the baseline. GPT-4o also achieves between 2.5 and 11 percentage points lower F_1 scores on CM1-NASA when context is added. In contrast, F_1 scores on CM1-NASA increase by 2 to 4 percentage points

when context is used with GPT-5. The same can be observed with GPT-5-mini, which, when using context, achieves F_1 scores that are between 1.5 and 3.5 percentage points higher than the baseline result. GPT-4o-mini achieves drastically lower F_1 scores when context is used. It achieves results that are 34 and 20 percentage points lower than the baseline. This is particularly interesting as GPT-4o-mini has the best precision value for the CM1-NASA dataset across all tested models, with a value of 1.0. For F_2 scores, the best baseline result on the CM1-NASA dataset across all models is achieved by GPT-4o, followed by GPT-5, GPT-4o-mini, GPT-5.1, and GPT-5-mini in order. Similar to its behavior in F_1 score, GPT-4o loses 14.5 and 22 percentage points when context is added compared to the baseline. With GPT-5, the F_2 score improves by 18 percentage points compared to the baseline when dronology is used as a context source. On the other hand, the score decreases by 1.6 percentage points when examples are drawn from MODIS. Similar to GPT-4o and its own F_1 score results, GPT-4o-mini loses approximately 37.5 and 49 percentage points in F_2 score when external trace links are added as context due to its poor recall values. Like GPT-5, GPT-5.1 achieves both higher and lower F_2 scores when context is used. The external context sources are flipped for GPT-5.1 compared to GPT-5 though with context from dronology decreasing the F_2 score by 1.5 percentage points and context from MODIS increasing the score by 2 percentage points compared to the baseline. GPT-5-mini's F_2 score decreases by 0.8 and 3 percentage points compared to the baseline when context is added. Overall, GPT-5 and GPT-5-mini see the biggest gains in F_1 score on CM1-NASA compared to the baseline when external trace links are added as context. Together with its stronger baseline score compared to most models, the increase in performance of GPT-5 means that it outperforms all other models in absolute F_1 score. In F_2 score, the largest improvements can be seen with GPT-5 and GPT-5.1 with all other models achieving lower scores when context is added. Unlike in F_1 score, however, GPT-5 cannot consistently outperform all other models. Since its increase in F_2 score is higher than that seen with GPT-5.1 and its decrease in F_2 score is lower than that seen with GPT-5.1, GPT-5 is still the model that profits the most from trace links as context on the CM1-NASA dataset.

The highest baseline F_1 score on the dronology dataset is achieved by GPT-5.1. It is followed by GPT-4o and GPT-4o-mini, which both achieve the same baseline F_1 and F_2 scores. They are followed again by GPT-5 with GPT-5-mini achieving the lowest baseline F_1 score on dronology. When external trace link context is added to the prompt, GPT-5.1's F_1 scores on dronology stay relatively consistent with a slight decrease of 0.2 percentage points compared to the baseline. GPT-5.1 actually achieves the exact same results on dronology for all metrics, regardless of which dataset is used as the context source. Compared to its baseline result, GPT-4o's F_1 scores decrease by 6 and 15 percentage points. This observation is in line with the behavior seen for GPT-4o on other datasets. Similar to the non-mini variant, GPT-4o-mini achieves F_1 scores that are 3.5 and 7 percentage points lower when context is used compared to the baseline. GPT-5 by contrast, achieves higher F_1 scores on dronology when external trace links are used as context. It can achieve scores that are 5.5 and 7 percentage points higher than the baseline with the external context. GPT-5-mini can also achieve higher F_1 scores when external context is added compared to the baseline. With context, it achieves F_1 scores that are 4.5 and 5.5 percentage points higher than the baseline result. In F_2 score, the highest baseline score is shared by GPT-4o, GPT-4o-mini,

and GPT-5. They are closely followed by GPT-5-mini and GPT-5.1 with a difference of less than 2 percentage points between the models. Just as in F_1 score, the F_2 scores achieved by GPT-4o on dronology decrease when context is added. It achieves F_2 scores that are approximately 14.5 and 22 percentage points lower when context is used compared to the baseline. Similarly, the F_2 scores achieved on dronology with GPT-4o-mini and context trace link are 12.5 and 18.5 percentage points lower than the baseline. GPT-5 is the only of the top performing baseline models on dronology that can increase the achieved F_2 score when context is added. It achieves a 0.5 percentage points higher score when the MODIS dataset is used as the context source. When CM1-NASA is used as a context source, the F_2 score achieved with GPT-5 actually decreases by 4 percentage points. Contrary to previous datasets, the F_2 score achieved when using GPT-5-mini decreases by up to 1.5 percentage points when using external trace links as context compared to the baseline. The F_2 scores achieved using GPT-5.1 on dronology with context also decrease when compared to the baseline. The decrease in F_2 is comparatively large at 18 percentage points. While they do not achieve the highest absolute F_1 scores on dronology for all context sources, the F_1 scores of the GPT-5 and GPT-5-mini models profit the most from the added context. The GPT-4o series models, in contrast, achieve worse F_1 scores on dronology with external trace links as context. All models are experiencing decreases in F_2 scores, with the smallest decrease in score seen when using GPT-5-mini. With GPT-5 being the only model that can achieve an increased F_2 score when context is added, the two models are able to profit the most from added external trace links out of the tested models on dronology.

On the MODIS dataset, the highest baseline result for the F_1 score is achieved when using GPT-5. Its results are followed by GPT-4o-mini, GPT-4o, GPT-5-mini, and GPT-5.1 in order. When external trace links are added as context, the F_1 score achieved with GPT-5 decreases slightly by 0.3 and 2 percentage points compared to the baseline. When using GPT-4o-mini on MODIS with context, the achieved F_1 scores are 19.5 and 20 percentage points lower than the baseline result due to a strong decrease in recall values. Interestingly, similar to when the TLR task is applied to the CM1-NASA dataset, GPT-4o-mini is able to achieve a precision value of 1.0 when dronology is used as the context source for MODIS. Due to the strong decline in recall of over 20 percentage points, this still leads to a reduced F_1 and F_2 score. GPT-4o achieves approximately 11 and 11.5 percentage points worse F_1 scores when using context compared to the baseline. In contrast, GPT-5-mini is able to achieve a higher F_1 score on MODIS than the baseline when the dronology dataset is used as the context source. In this case, the achieved F_1 score is 2.5 percentage points higher than that of the baseline. When CM1-NASA is used as the context source, however, the F_1 score decreases slightly by 0.3 percentage points compared to the baseline. The model with the worst baseline F_1 score, GPT-5.1, is the only model that is able to achieve a small increase in F_1 score of 1.3 and 0.2 percentage points on MODIS compared to the baseline. In F_2 score, the best baseline result is achieved by GPT-4o-mini. It is followed by GPT-5, GPT-5-mini, GPT-4o and GPT-5.1 in order. Just as in F_1 score, GPT-4o-mini's F_2 score results when using context are approximately 20 percentage points worse than the baseline F_2 score at 21.5 percentage points. When using GPT-5 with context, the achieved F_2 scores on MODIS are 4 and 2 percentage points worse than the baseline. The same is true for GPT-5-mini, which achieves F_2 scores that are 3.5 and 1 percentage points below the baseline result. For GPT-4o

F_2 scores decrease as well, with scores being 10.5 percentage points lower than the baseline. Just as in F_1 score GPT-5.1, though it has the worst baseline results, is able to improve its performance in F_2 score on MODIS when external trace links are added as context. When context is used, it achieves F_2 scores that are approximately 10.5 percentage points higher than its baseline result. With these results, GPT-5.1 achieves the absolute highest F_2 scores when context is used. Together, with it being the only model that achieves higher scores in both F_1 and F_2 score when external trace links are used as context, this shows that GPT-5.1 is the model the profits the most from the added context on the MODIS dataset.

Across all datasets, the GPT-5 family of LLMs including GPT-5, GPT-5.1, and GPT-5-mini sees the largest gains in TLR performance when external trace links are added to the prompt as context information. Specifically, GPT-5 seems to see the largest performance increases on most of the datasets it is tested on. For the datasets not tested with GPT-5 and GPT-5.1, GPT-5-mini sees the biggest improvements. The scores achieved by the GPT-4o family of models seem to consistently decrease when context is added to the prompt compared to their baseline results. When using GPT-4o-mini for example, the baseline scores for F_1 and F_2 score are never outperformed when context is added to the prompt. Thus, Question 5 of the GQM-plan for this chapter can be answered by saying that GPT-5 can profit the most when external trace links are provided as context. This tendency can also be seen for GPT-5-mini which can also achieve higher performance when examples are provided.

7.3.3. Variation from Different Context Sources

In addition to the ordering methods used or the LLM that is chosen for the classification task, the results achieved by the approach using external trace links as context is affected by the datasets serving as sources for the external trace links. To compare the influence of different datasets serving as context sources, the results from the experiments using GPT-5-mini (see Table 7.2) and GPT-4o-mini (see Table 7.3) are considered. The results for both models, when using negative-first ordering, are compared between the individual context datasets.

The best results when using external trace links as context are often achieved when the GANNT dataset is used as the context source. When using GPT-4o-mini, the best results for both the CM1-NASA and dronology datasets in F_1 and F_2 scores are achieved when GANNT is used as the context source. The same is true when GPT-5-mini is used on CM1-NASA. On the dronology dataset, with GPT-5-mini using context from GANNT only achieves the third best F_1 score. But using GANNT as context source on dronology does achieve the best F_2 score with GPT-5-mini. On the WARC dataset, using GANNT as the context source achieves the second best result for both LLMs. On the MODIS dataset, using context trace links from GANNT consistently achieves the worst results compared to the other datasets.

The best results in F_1 and F_2 scores on the WARC dataset is achieved when context is drawn from the CM1-NASA dataset. This holds true for both tested models. When GPT-4o-mini is used, context from the CM1-NASA dataset can also achieve the best results for the GANNT dataset. This result is shared with context from the WARC dataset. When using GPT-5-mini,

context from CM1-NASA can also be used to achieve the highest F_1 score for the dronology dataset. When GPT-4o-mini is used on dronology, the CM1-NASA trace links achieve the worst results in F_1 and F_2 score across all tested datasets. On MODIS, scores achieved with the help of context from CM1-NASA are the worst, when using GPT-4o-mini and comparatively low when using GPT-5-mini. Especially on MODIS with GPT-4o-mini it is noteworthy that context drawn from GANNT and CM1-NASA results in equally bad F_1 scores with all external trace link context resulting in large drops in both F_1 and F_2 scores. When GPT-5-mini is used, scores achieved with context from CM1-NASA are worse than all other context sources except for GANNT. A possible reason for CM1-NASA's low performance as a context source could be its relatively low coverage of low-level requirements by the gold standard. A bit more than 40% of low-level requirements in the dataset do not appear in the gold standard. If the reason for this is an incomplete gold standard, it may very well be that the negative examples generated for the prompt should actually be marked as positive examples and were simply not found in the gold standard. If such false negative link examples are shown to the LLM as context, it could be directed in a wrong direction.

External trace links drawn from the dronology dataset are used to achieve the best results in F_1 and F_2 score with GPT-5-mini on the GANNT dataset. In contrast, GPT-4o-mini with context from dronology achieves the worst results on the GANNT dataset. With GPT-4o-mini, the best results when using trace links from dronology are achieved on the MODIS dataset, where the highest F_1 score is achieved. When dronology is used as context with GPT-4o-mini on the GANNT, CM1-NASA, and WARC datasets, it consistently achieves the worst F_1 and F_2 scores. With GPT-5-mini, context from dronology can be used to achieve the second best results on the CM1-NASA and MODIS datasets.

Context trace links from MODIS are never used to achieve the highest F_1 or F_2 scores on any of the datasets. At the same time, the experiments using trace links from MODIS also never achieve the worst results when compared with other context sources. Given that MODIS has a low coverage of its requirements by trace links in its gold standard as well as a comparatively low absolute number of trace links, it is interesting that its usage as context does not result in similar low performance as with CM1-NASA.

When the WARC dataset is used as the source for external trace links as context, the best results with GPT-5-mini for the MODIS dataset can be achieved. When used with GPT-4o-mini, WARC as context also achieves the highest performance for the GANNT dataset. When GPT-5-mini is used on all other datasets with context from WARC, it generally achieves the 3rd or 2nd best results in F_1 or F_2 score placing it in the middle of the used context sources. When GPT-4o-mini is used with context examples drawn from WARC, it generally achieves the 2nd best scores across all tested data sources.

Intuitively, the TLR task should benefit the most from external trace links as context if the context and the dataset to which it is applied share the same domain. However, even though the GANNT dataset does not share a domain with any of the other datasets, it is involved in most of the highest scores across all datasets. Equally, results for the WARC dataset are consistently best when the CM1-NASA dataset is used as context, which also does not share a domain. Only when using GPT-5-mini with CM1-NASA as context on dronology or

when using GPT-4o-mini with dronology on MODIS there is a connection between a shared domain and the best results achieved on a dataset. Even when considering the second best result on the datasets from the aerospace domain, the context source is not always from the same domain when GPT-4o-mini is used. When GPT-5-mini is used, aerospace datasets used as context sources more often achieve higher scores on other aerospace datasets. To answer Question 4 of the GQM-plan (see Table 7.1), external trace links from the same domain do not seem to have a stronger positive effect on TLR performance than external trace links from unrelated domains.

7.3.4. Internal Trace Links as Context

In addition to considering external datasets as sources for the context, one can use internal trace links originating from the very dataset the TLR task is carried out on. This is not always practical due to requiring existing internal links and reintroducing the drawbacks of other automated TLR systems that LiSSA aims to avoid. Using internal trace links can easily be tested using the existing trace links as context loader and classifier. Using the context configuration LiSSA can simply be configured to load the context from the same dataset as the source and target requirements for the regular pipeline.

Experiments using internal trace links as context are conducted using GPT-5-mini only. Since external trace links are drawn from the context based on the similarity between the source requirement of the current classification and the source requirement of the external trace link and then ordered based on the similarity of the involved target requirements, each classification run that is using internal trace links as context is guaranteed to contain the gold standard answer to its own classification as context. To prevent the LLM from seeing the answer to the question it is being asked as context, an additional check is introduced in the classifier when the context is assembled. This check prevents the inclusion of a trace link in the context if the IDs of the source and target requirements in the trace link are equivalent to the IDs of the source and target requirements that are currently being classified. The results of the experiment can be seen in Table 7.5.

Using internal trace links as context on the dronology and MODIS datasets yields the best values across all tested datasets and the baseline for all metrics. On dronology, the baseline precision value is almost doubled from 43.5% to 82.4%. On the MODIS dataset, precision with internal context trace links increases by 27.5 percentage points. While improvements in recall on both datasets are not as extreme, the F_1 and F_2 scores for both datasets outperform the baseline and all external context.

On the WARC dataset, the approach with internal trace links outperforms external context and the baseline in precision, F_1 and F_2 score. Compared to the best performing external context source, the internal context achieves a 5 percentage points higher precision score and an approximately 3 percentage points increase in F_1 score. Similarly, on the GANNT and CM1-NASA datasets, higher values in precision and F_1 score are achieved with internal trace links as context compared to the baseline and external trace links as context. On all three datasets, WARC, GANNT, and CM1-NASA, the configuration with internal context

		P	R	F_1	F_2
GANNT	Baseline	.571	**.529**	.550	**.537**
	Internal links	**.702**	.485	**.574**	.517
CM1-NASA	Baseline	.409	**.600**	.486	**.549**
	Internal links	**.579**	.489	**.530**	.505
dronology	Baseline	.435	.686	.533	.615
	Internal links	**.824**	**.700**	**.757**	**.722**
MODIS	Baseline	.313	**.244**	.274	.255
	Internal links	**.588**	**.244**	**.345**	**.276**
WARC	Baseline	.445	**.691**	.542	.623
	Internal links	**.608**	.662	**.634**	**.650**

Table 7.5.: Precision (P), Recall (R), F_1 and F_2 score for LiSSA when using project-internal trace links as context. GPT-5-mini is used for classification at $k = 4$ with 4 positive and 4 negative examples. The baseline is created without any context. The baseline uses the CoT prompt (see Prompt 2), internal links are evaluated with Prompt 6 and negative-first ordering.

does not outperform the baseline in recall. On the GANNT and CM1-NASA datasets, the increase in precision in combination with the decrease in recall leads to lower F_2 scores with internal context compared to the baseline. On both datasets, recall is worse than when external trace links are used, leading to lower F_2 scores compared to external contexts as well. Recall is also worse with internal context compared to external context on the WARC dataset. In contrast to GANNT and CM1-NASA however, the highest F_2 score on this dataset is still achieved when using internal context.

With the exception of the MODIS and dronology datasets which achieve higher recall values than the baseline, using internal trace links as context seems to have the same but amplified effect as using external trace links as context.

When using trace links as context, precision values generally increase across all datasets. This means that fewer false positives are generated when external trace links are used as context, answering Question 2 of the GQM-plan (see Table 7.1). At the same time, recall values generally decrease compared to the baseline. The amount of decrease in recall differs between LLMs, with the GPT-5 family of models generally seeing lower decreases compared to the GPT-4o family. This answers Question 1 of the GQM-plan. Generally, when external trace links are used as context, the LLMs are not able to identify more of the existing trace links in the datasets. Overall, the system is more reliable for automated TLR tasks when a model from the GPT-5 family is used, as can be seen by the F_1 scores achieved with external trace links as context that are generally higher than the scores achieved by the baseline. In F_2 score however, the baseline generally performs better than when trace links are used as context due to the decreases in recall. This means that the system using external trace links as context is overall less suitable as a recommender system, that suggests potential trace link candidates for human review. To answer Question 3 of the

GQM-plan, it can be said that the system, when using a GPT-5 family model, is overall more reliable for fully automated TLR when external trace link context is used but less reliable than the baseline as a recommender system.

8. Discussion

This chapter discusses the results reported in the previous chapters in relation to the research questions introduced at the beginning of this thesis. For each of the research questions, the results collected for the performance of the three types of context are interpreted. Finally, threats to the validity of the findings of this thesis are discussed.

8.1. Effect of Context on Traceability Link Recovery

RQ1: How does the use of context effect LLM-based requirements-to-requirements TLR performance?

Regarding RQ1, different behavior can be observed for all three kinds of context that are implemented with LiSSA. The first implemented approach of paraphrasing requirements has a limited impact on the overall TLR performance of the LiSSA framework. Since the paraphrased requirements are injected into the element store and not added to the prompt, this kind of context is mainly targeted at the IR step in the LiSSA pipeline. On average, an increase in performance can be seen in recall values as well as in F_1 and F_2 score. The increase in recall makes sense, as this type of context is affecting the IR step of the LiSSA pipeline. As was shown in the preliminary analysis in Section 5.2.1, the composition of the vector space does change visibly. It seems that through the diversification of the embeddings of individual requirements, the LLM is actually presented with more relevant trace link candidates compared to the baseline that does not use paraphrased requirements.

The average improvement in recall is relatively low, though, between 0.5 and 2 percentage points depending on the amount of paraphrases being generated. These low improvements in average values are, however, in part created by the high fluctuation in improvements between datasets. For some datasets, improvements in recall of up to 10 percentage points can be seen, while others actually achieve lower recall values when the paraphrases are added to the element store. This indicates that the performance of paraphrased requirements as context is highly dependent on the specific original requirements they are generated from. Interestingly, sometimes improvements are also achieved in precision when paraphrased requirements are used. In most cases, however, precision actually decreases. This is most likely a result of the fact that when a requirement paraphrase is picked as a trace link candidate, the paraphrased text is passed to the LLM instead of the original requirement the paraphrase is based on. The paraphrased requirement might be a less accurate description than the original target requirement or might use terminology that is too different from the source requirement of the classification, which causes the LLM to miss the link. In

some cases, the paraphrase might be a more fitting description of the requirement than the original in the dataset. This hypothesis requires further research though.

Along with the increase in recall, the values for F_1 and F_2 score increase slightly when paraphrased requirements are present in the element store. In F_1 score, the average increase is even lower than that of recall values, at up to 0.6 percentage points. The average gain in F_2 score is slightly higher, at up to 1.5 percentage points, which is expected as this score puts a stronger emphasis on recall results. The fact that both scores increase on average and for most of the tested datasets shows that the approach has a slightly positive effect on overall TLR performance even when the decreased precision is considered.

The second investigated approach of using other requirements than the trace link candidate pair as context does not generally have a positive impact on automated TLR. Improvements when using requirements as context compared to the baseline can generally be seen in precision values. At the same time, recall values are generally lower when requirements are used as context compared to the baseline. Both metrics together seem to suggest that the LLM is more cautious before classifying two requirements as being linked when context requirements are present.

Best results are generally achieved in source-to-source mode, meaning that the LLM is presented with more high-level requirements that are semantically similar to the high-level requirement of the classification. Overall, context selection modes that add high-level requirements to the prompt perform better than selection modes that add low-level target requirements. Through the high-level requirements, the LLM is getting a broader overview of the system than if low-level requirements are added as context simply due to the larger scope of high-level requirements. This suggests that broader knowledge of the overall system and its requirements is more important to precision than presenting the LLM with alternative options for target requirements. The achieved increase in precision is, however, unable to outweigh the drop in recall values. This means that the TLR performance of the system decreases when this kind of context is used. When using the best performing source-to-source selection mode, however, the drop in F_1 score is only between 0.2 and 1.4 percentage points on average compared to the baseline. In scenarios in which precision is more important, such as when applied to safety-critical systems, this tradeoff may be worthwhile. In applications where recall is important, such as systems that recommend trace links for human review, using requirements as context does not appear to be helpful.

The third and final investigated approach for using context for LLM-based TLR is the use of project-external trace links. On initial testing of the trace links as context classifier, high increases in precision along with decreases in recall are noticed. At this point in testing, positive examples appear before negative examples in the prompt. This suggests that instead of actually classifying the requirement pair, the LLM is continuing the pattern of placing “Link: no” below the requirements text from the examples. When negative-first example ordering is tested, higher average recall values than with positive-first sorting can be observed. The decrease in recall compared to the baseline does, however, persist. The difference in recall values between the two ordering methods is, however, relatively small at around approximately 0 to 2 percentage points. Similarly, the differences in F_1 and F_2 score between both ordering modes are marginal, below 1 percentage point, for most datasets.

While example ordering does seem to have a slight effect on TLR performance, it is unlikely that the LLM is continuing a pattern from the example order instead of performing the classification.

Similar to the behavior observed when requirements are used as context, recall is consistently lower when external trace links are used as context compared to the baseline. Even when internal trace links are used as context, recall decreases for most tested datasets compared to the baseline. Precision results, in contrast, are consistently increasing when context is used. When models from the GPT-4 family are used, the decrease in recall is too high to be compensated by the increase in precision, leading to worse overall TLR performance with trace links as context when compared to the baseline. On GPT-5 family models, however, the decrease in recall performance compared to the baseline is not as high. In most cases, the increase in precision can compensate for the loss in recall and leads to F_1 scores that are higher than the baseline. This observation might be a result of the increased reasoning ability of the GPT-5 family models compared to their GPT-4 predecessors, as advertised by the vendor. This is an opportunity for future research to examine if similar differences between the two generations of models exist for the use of requirements as context.

Overall, using external trace links as examples for a few-shot prompt does seem to have a positive effect on TLR-performance as can be seen by the improvement in F_1 score of, on average, 2 to 5 percentage points compared to the baseline. Combined with the fact that the results presented in Section 7.3.3 show that the external trace links that are used do not need to be from a project sharing a domain with the project the TLR task is conducted on, this means that external trace links are a viable form of context to improve TLR performance in almost any project.

In addition to the three kinds of context discussed so far, the small experiment using internal trace link shows interesting results. On all datasets, precision increases by a large amount compared to the baseline. When internal trace links are used, improvements in precision between 13 and 39 percentage points are achieved. Given that the project internal context is most likely to contain information explaining how different components of the system under review relate to each other, it makes sense that the added information allows the LLM to more accurately judge which requirements should be linked. Even still, for most of the datasets, recall values decrease slightly compared to the baseline. On the dronology dataset, recall increases slightly when internal trace links are used. On MODIS it stays the same. Since recall measures how many of the known trace links are found by the system, this might be a barrier imposed by the fundamental design of the LiSSA framework. The model is only presented a fixed amount of target candidates for each source requirement following the retrieval step. If a dataset contains source requirements that are traced to more target requirements than the configured k for the retrieval step, the system cannot find all trace links, no matter how much additional context is included. This is especially obvious on the MODIS dataset, where one high-level requirement exists that is traced to 22 different low-level requirements. The recall value does not change between the baseline and the experiment using project internal trace links as context because it might be the maximum achievable result when only 4 targets can be considered per source. Similarly, the slight increase on dronology may represent a ceiling in achievable recall at $k = 4$ since

some of the high-level requirements in that dataset are also traced to more than 4 low-level requirements.

To answer RQ1, it can be said that the effect context has on LLM-based requirements-to-requirements TLR varies for each type of context. Paraphrasing requirements has a small positive overall effect on TLR performance, with improvements especially visible in recall. Adding context to the prompt seems to generally decrease the recall achieved by the system while increasing its precision. When requirements are added as context, the overall TLR performance decreases slightly, which may be a worthwhile tradeoff for the increase in precision. With trace links as context, the increase in precision leads to an overall increase in TLR performance.

8.2. Best Type of Context for Traceability Link Recovery

RQ2: Which one of the investigated contexts results in the highest benefit to automated TLR tasks?

The answer to RQ2 depends on the context in which automated TLR is used. In situations in which recall is important, improvements can be achieved when using paraphrased requirements. The results indicate that the context type can yield improvements in both recall and F_2 score over a baseline without context. In contrast, neither requirements nor external trace links should be used as context when recall is the most important metric. Results for both context types show that recall values are generally lower when context is used compared to the baseline. Even when precision is additionally considered, neither context type is able to achieve higher average F_2 scores than the baseline. While requirements and external trace links are not suitable contexts to increase recall values, the positive effect of paraphrases on the metric does not guarantee success. The difference in recall between the baseline and using paraphrases is comparatively small at 2 percentage points on average. This needs to be placed in the context of the additional costs incurred for paraphrase generation. While the additional costs are not studied as part of this thesis, it must be noted that all potential target requirements need to be paraphrased multiple times for this approach to work. Depending on the amount of requirement text that exists in the project, this may amount to a significant cost factor. The decision if the increase in recall is worth these costs must be made for each use case and project individually.

In scenarios in which precision and recall are equally important, improvements in TLR performance can be achieved when using paraphrases or when using external trace links as examples. When using requirements as context, a decrease in F_1 score compared to the baseline can be observed. Given that the F_1 score is a combined metric that values precision and recall equally, using requirements as context is not beneficial when both values are equally important.

While adding requirement paraphrases to the vector store can yield improvements in F_1 score, these improvements are comparatively low at below 1 percentage point on average. When external trace links are used as context, however, improvements compared to the

baseline in F_1 score between 1 and 5 percentage points can be achieved. The results when testing the use of external trace links as context indicate that there is no benefit to drawing the external trace link from projects sharing a domain with the project TLR is used on. This means that this type of context is accessible to any project, as datasets with trace link information are openly available.

In addition to the performance benefits, using external trace links as context may be cheaper than using requirement paraphrases. Most LLMs that are offered as a service are priced depending on the amount of text being sent to the LLM and the size of the model's output text. When using OpenAI's LLMs, the input to the model is cheaper than the model's output [33]. When external trace links are used as context, only the amount of required input text is increased as the prompt is lengthened by the content of the requirements from the external trace links. The length of the classification response and, therefore, the more costly output text remains unchanged. When requirement paraphrases are used, the paraphrases are generated by an LLM. For each individual paraphrase, additional costs are incurred in the form of the more expensive model output. As the effect on cost is not measured as part of this thesis, this remains an assumption.

For applications in which precision is more important than recall, improvements can be achieved by using requirements or external trace links as context. Requirements paraphrasing is not a suitable kind of context for this kind of scenario, as the average precision when using the context decreases compared to the baseline. When using requirements as context, the precision increases by 5 percentage points on average compared to the baseline. Higher increases in precision can be achieved when using external trace links as context. This kind of context achieves an average increase in precision between 7 and 12 percentage points compared to the baseline. It must be noted, however, that using trace links as context is likely to incur higher costs than using requirements as context. This is due to the fact that the size of the prompt is increased by the context. When n requirements are used as context, the content of all n requirements is added to the prompt. When trace links are used as context and the system is configured to add n examples, the content of two requirements per example needs to be added to the prompt. If equal amounts of positive and negative examples are desired, this number is doubled again by the need to add one negative example for each positive example. This generally results in longer prompts when trace links are used compared to when requirements are used as context, which in turn leads to higher costs for external trace links. Fully substituted prompts can be seen in A.2.

To answer RQ2, it can be said that the context to use for the highest benefit to automated TLR tasks depends on the application. If recall is more important than precision, using requirement paraphrases can be a viable approach. If precision is more important than recall, benefits can be observed when using requirements or external trace links as context. If both metrics are equally important, the largest benefit is obtained when using external trace links as context.

8.3. Amount of Context Information

RQ3: How much context information should be used to increase LLM-based TLR performance?

All of the investigated methods for the inclusion of context in LLM-based automated TLR can supply varying, configurable amounts of context to the system. In the case of using external trace links or requirements as context, the amount of context controls how many of these context elements are added to the prompt. In the case of using requirement paraphrases, the amount of context is the number of paraphrases generated for each original requirement. Results for using varying amounts of context have been collected for requirements paraphrasing and requirements as context. When using requirements paraphrasing, the best average results are achieved when three paraphrases are generated per requirement. The difference in F_1 score between configurations using varying amounts of paraphrases is very small, though, as soon as more than two paraphrases are generated per requirement. Results of the preliminary analysis of paraphrasing requirements indicate that even when few paraphrases are generated per requirement, the amount of duplicates increases quickly. This fact could be a potential reason why adding more than a few paraphrases does not increase TLR performance. Given that the costs for generating paraphrases increase the more paraphrases are generated, it makes sense to maintain a low number of paraphrases per requirement.

Best results for requirements as context can also be observed when two or three requirements are added as context. When more requirements are added, precision decreases by approximately 3 percentage points depending on the project. Similarly, precision is lower when fewer requirements are added as context. This indicates that there is a balance to consider between adding enough additional information so that the LLM can have the biggest possible view of the project and not adding too much additional information so the LLM is not distracted from the task at hand. Given that most tested projects only contain around 20 high-level requirements, the LLM can already see a sizeable amount of the available information when two or three are added to the prompt. On the projects with more high-level requirements, the best results are achieved with more context elements than on those with fewer requirements. This suggests that the amount of context requirements that is presented to the LLM should be scaled depending on the amount of high-level requirements in the project. Proving this assumption requires larger datasets with an available gold standard. Those datasets are currently not available, leaving this as a question for future research.

To answer RQ3, it can be said that low amounts of context information seem to generally be sufficient to achieve improvements in TLR performance. Adding too much context information may actually be detrimental to TLR performance. Exact numbers for each context type are difficult to determine because all available test projects are small in size compared to very large software projects.

8.4. Threats to Validity

The threats to the validity of the findings reported by this thesis are contextualized based on the recommendations by Runeson et al. [39]. Concerning construct validity, three threats are identified. Firstly, no extensive prompt engineering is done as part of this work, meaning that the ability of this thesis to judge LLM performance is limited by the tested prompts. To combat this threat, prompts that are drawn from previous works by Hey et al. [19] or derived from work done by Rodriguez et al. [37] and Geng et al. [13]. Secondly, the terms used as part of the prompt are not defined for the LLM and may be interpreted differently by it. Especially the phrasing of "considering" the supplied context may be open for interpretation by the LLM. Thirdly, the correctness of the gold standards supplied with the test datasets is not verified as part of this work. This carries the risk of miscalculating the metrics since the LLM may be able to find correct links that are not captured by the standard. Correctness of individual links is also open to interpretation. To address this risk, the same test datasets are used that are commonly used by the traceability community and other researchers in their works.

A threat regarding the internal validity of this thesis is the potential for data leakage. Since the LLMs that are used are closed source, it is possible that the test datasets that are used for this evaluation are part of the training data that was used to train the models. Another threat regarding internal validity affects the investigation of the performance of external trace links from similar domains. Since some of the used datasets from the aerospace domain have a low coverage of their requirements by the gold standards, the observed results might not be a product of domain similarity but instead of gold standard completeness. Since no other datasets with a matching domain were available, this threat can not be eliminated.

Three threats are also identified concerning the external validity of the results. Firstly, the testing done as part of the evaluation is confined to a small selection of test datasets. As variations in results already occurred in this limited selection, it is unknown if the observed results can be generalized to other projects. To combat this threat, datasets that are commonly used for the evaluation of TLR tasks are used. Additionally, testing is performed using only a few models, all supplied by the same vendor. Results obtained for these models might not carry over to other LLMs. To address this threat, state-of-the-art models are used. Lastly, the non-determinism of LLM responses is a threat to validity. To minimize this threat, the model temperature is set to 0 wherever possible, and a fixed seed is used across experiments. Additionally, caches are used and published as part of the replication package [3] to make the results of this thesis reproducible.

9. Conclusion and Future Work

This thesis presents three independent approaches to augment LLM-based automated requirements-to-requirements TLR with context information. With the goal of improving the performance of automated TLR, the LiSSA framework is extended through the implementation of components for the use of the three kinds of context. A method for adding paraphrased requirements to the vector store of the IR step is proposed to increase the likelihood of suggesting relevant trace link candidates to the LLM for classification. It is shown that the generation of paraphrases does affect the composition of the vector store. Additionally, a method to systematically load and use various context is introduced for the LiSSA framework. Along with this, a format to configure the use of different sources of context information is developed to allow for the construction of complex contexts sourced from multiple data sources. These capabilities are then used for the implementation of two further types of context. In one approach, the LLM is provided with additional requirements from the project to which the TLR task is applied. This way the model has access to project-specific internal information during the classification. As a final third type of context, project-external trace links are added as context to assist the LLM's reasoning by showing relevant examples of trace links. This is the first time that RAG-based few-shot prompting is applied, using the LiSSA framework.

All three types of context are evaluated for their effect on the automated TLR task. It is found that the requirements paraphrasing and external trace links approaches can achieve better TLR performance than the unmodified LiSSA framework without any context. While the overall improvement achieved by the paraphrasing approach is comparatively small and dependent on the project under investigation, this is the only investigated context type that can be used to increase the recall performance of the system. By comparison, using external trace links as context yields higher improvements in overall TLR performance. The increase in performance achieved through the use of this context is still small at approximately 3 percentage points in F_1 score on average across the tested projects. Given the right source of external requirements, however, the difference to the baseline can double to up to 6 percentage points. This thesis finds that a shared domain between the project under investigation and the project used to source the trace link examples does not seem to lead to improved results and that examples from the GANNT dataset often result in best performance. It is additionally found that the newer GPT-5 family of LLMs benefits more from the context type than older GPT-4 family models. When constructing the few-shot prompt, higher performance is observed when negative trace link examples are placed before the trace links drawn from external projects. In contrast to the other two context types, using additional requirements as context does not have a positive effect on overall TLR performance. The context does increase precision results over a system without context,

but this gain does not make up for the drop in recall. Requirements work best as context when high-level requirements are used that are selected based on their semantic similarity to the source requirement of the respective classification.

Overall, context only has a small effect on the performance of an automated LLM-based TLR system on the requirements-to-requirements task.

Many further opportunities can be identified to expand on the work done as part of this thesis. It would be interesting to test the usefulness of context on larger scale software projects than the used test datasets. This is especially true for closed-source software that definitely is not part of the training data used by the LLM provider. In this environment, tests could be conducted to determine if the optimal amount of supplied context scales along with the size of the project or if a small amount of context is already sufficient even for large-scale software development. Another avenue for future research is the development of other context selection strategies. Even when only the context types presented as part of this thesis are considered, there are countless possibilities to define other processes or criteria that can be used to select which elements exactly are supplied to the LLM as context. Additionally, different kinds of contexts could be combined. It would be interesting to see if the combination of a context that generally increases recall results and a context that generally increases precision values could lead to better overall performance by boosting both metrics. On a smaller scale, the concept of external trace links as context could be expanded. In this thesis, external trace links are only drawn from a single project at a time. Potentially, results could be improved if trace links from several projects are combined into one larger context. Finally, the world of LLMs is still constantly evolving. During the creation of this thesis alone, two new generations of LLMs have been released by OpenAI. With continually improving reasoning abilities, future models and models from other providers might see increased benefits from the use of context compared to the models tested here.

Bibliography

[1] Syed Juned Ali, Varun Naganathan, and Dominik Bork. "Establishing Traceability Between Natural Language Requirements and Software Artifacts by Combining RAG and LLMs". en. In: *Conceptual Modeling*. Ed. by Wolfgang Maass et al. Cham: Springer Nature Switzerland, 2025, pp. 295–314. ISBN: 978-3-031-75872-0. DOI: 10.1007/978-3-031-75872-0_16.

[2] Victor R. Basili and David M. Weiss. "A Methodology for Collecting Valid Software Engineering Data". In: *IEEE Transactions on Software Engineering* SE-10.6 (Nov. 1984), pp. 728–738. ISSN: 1939-3520. DOI: 10.1109/TSE.1984.5010301.

[3] David Bauch. *Replication Package for: Context in Automated Requirements-to-Requirements Traceability Link Recovery*. DOI: 10.5281/ZENODO.18010485.

[4] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. ISBN: 978-1-7138-2954-6.

[5] Lei Chen et al. "Enhancing Unsupervised Requirements Traceability with Sequential Semantics". In: *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. ISSN: 2640-0715. Dec. 2019, pp. 23–30. DOI: 10.1109/APSEC48747.2019.00013.

[6] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, eds. *Software and Systems Traceability*. en. London: Springer, 2012. ISBN: 978-1-4471-2238-8. DOI: 10.1007/978-1-4471-2239-5.

[7] Jane Cleland-Huang, Michael Vierhauser, and Sean Bayley. "Dronology: an incubator for cyber-physical systems research". In: *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 109–112. ISBN: 978-1-4503-5662-6. DOI: 10.1145/3183399.3183408.

[8] Scott Deerwester et al. "Indexing by latent semantic analysis". en. In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. ISSN: 1097-4571. DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[9] Qingxiu Dong et al. "A Survey on In-context Learning". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 1107–1128. DOI: 10.18653/v1/2024.emnlp-main.64.

[10] Dominik Fuchß et al. “Beyond Retrieval: A Study of Using LLM Ensembles for Candidate Filtering in Requirements Traceability”. de. In: *2025 IEEE 33rd International Requirements Engineering Conference Workshops (RE)*. Institute of Electrical and Electronics Engineers (IEEE), 2025. DOI: 10.5445/IR/1000183058.

[11] Dominik Fuchß et al. “LiSSA: Toward Generic Traceability Link Recovery through Retrieval-Augmented Generation”. In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. 2025, p. 723. DOI: 10.1109/ICSE55347.2025.00186.

[12] Chuyan Ge et al. “Cross-Level Requirements Tracing Based on Large Language Models”. In: *IEEE Transactions on Software Engineering* (2025), pp. 1–23. ISSN: 1939-3520. DOI: 10.1109/TSE.2025.3572094.

[13] Mingyang Geng et al. “Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE ’24. New York, NY, USA: Association for Computing Machinery, Feb. 2024, pp. 1–13. ISBN: 979-8-4007-0217-4. DOI: 10.1145/3597503.3608134.

[14] O.C.Z. Gotel and C.W. Finkelstein. “An analysis of the requirements traceability problem”. In: *Proceedings of IEEE International Conference on Requirements Engineering*. Apr. 1994, pp. 94–101. DOI: 10.1109/ICRE.1994.292398.

[15] Jin L. C. Guo et al. “Natural Language Processing for Requirements Traceability”. In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Cham: Springer Nature Switzerland, 2025, pp. 89–116. ISBN: 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_4.

[16] J.H. Hayes, A. Dekhtyar, and J. Osborne. “Improving requirements tracing via information retrieval”. In: *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.* ISSN: 1090-705X. Sept. 2003, pp. 138–147. DOI: 10.1109/ICRE.2003.1232745.

[17] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. “Advancing candidate link generation for requirements tracing: the study of methods”. In: *IEEE Transactions on Software Engineering* 32.1 (Jan. 2006), pp. 4–19. ISSN: 1939-3520. DOI: 10.1109/TSE.2006.3.

[18] Tobias Hey, Jan Keim, and Sophie Corallo. “Requirements Classification for Traceability Link Recovery”. In: *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. ISSN: 2332-6441. June 2024, pp. 155–167. DOI: 10.1109/RE59067.2024.00024.

[19] Tobias Hey et al. “Requirements Traceability Link Recovery via Retrieval-Augmented Generation”. de. In: *International Working Conference on Requirements Engineering (REFSQ 2025)*. 2025, p. 381. ISBN: 978-3-031-88530-3. DOI: 10.1007/978-3-031-88531-0_27.

[20] E. Ashlee Holbrook, Jane Huffman Hayes, and Alex Dekhtyar. “Toward Automating Requirements Satisfaction Assessment”. In: *2009 17th IEEE International Requirements Engineering Conference*. ISSN: 2332-6441. Aug. 2009, pp. 149–158. DOI: 10.1109/RE.2009.10.

[21] Jan Keim et al. “Recovering Trace Links Between Software Documentation And Code”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE ’24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1–13. ISBN: 979-8-4007-0217-4. DOI: 10.1145/3597503.3639130.

[22] Wei-Keat Kong et al. “Process improvement for traceability: A study of human fallibility”. In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. ISSN: 2332-6441. Sept. 2012, pp. 31–40. DOI: 10.1109/RE.2012.6345824.

[23] Quoc Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. en. In: *Proceedings of the 31st International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, June 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html (visited on 01/06/2026).

[24] Brian Lester, Rami Al-Rfou, and Noah Constant. “The power of scale for parameter-efficient prompt tuning”. In: *Proceedings of the 2021 conference on empirical methods in natural language processing*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3045–3059. DOI: 10.18653/v1/2021.emnlp-main.243.

[25] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html (visited on 07/04/2025).

[26] 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)*. ISSN: 1558-1225. May 2021, pp. 324–335. DOI: 10.1109/ICSE43902.2021.00040.

[27] Jiachang Liu et al. “What Makes Good In-Context Examples for GPT-3?” In: *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Ed. by Eneko Agirre, Marianna Apidianaki, and Ivan Vulić. Dublin, Ireland and Online: Association for Computational Linguistics, May 2022, pp. 100–114. DOI: 10.18653/v1/2022.deelio-1.10.

[28] Pengfei Liu et al. “Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. In: *ACM Comput. Surv.* 55.9 (Jan. 2023), 195:1–195:35. ISSN: 0360-0300. DOI: 10.1145/3560815.

[29] Xiao Liu et al. “P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks”. In: *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 2: Short papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 61–68. DOI: 10.18653/v1/2022.acl-short.8.

[30] Patrick Mäder and Alexander Egyed. “Assessing the effect of requirements traceability for software maintenance”. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. ISSN: 1063-6773. Sept. 2012, pp. 171–180. DOI: 10.1109/ICSM.2012.6405269.

[31] A. Marcus and J.I. Maletic. "Recovering documentation-to-source-code traceability links using latent semantic indexing". In: *25th International Conference on Software Engineering, 2003. Proceedings.* May 2003, pp. 125–135. DOI: 10.1109/ICSE.2003.1201194.

[32] Feifei Niu et al. *TVR: Automotive System Requirement Traceability Validation and Recovery Through Retrieval-Augmented Generation.* arXiv:2504.15427 [cs]. June 2025. DOI: 10.48550/arXiv.2504.15427.

[33] OpenAI. *Pricing*. en. URL: https://platform.openai.com/docs/pricing (visited on 01/10/2026).

[34] Michael Rath, David Lo, and Patrick Mäder. "Analyzing requirements and traceability information to improve bug localization". In: *Proceedings of the 15th International Conference on Mining Software Repositories*. MSR '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 442–453. ISBN: 978-1-4503-5716-6. DOI: 10.1145/3196398.3196415.

[35] Laria Reynolds and Kyle McDonell. "Prompt programming for large language models: Beyond the few-shot paradigm". In: *Extended abstracts of the 2021 CHI conference on human factors in computing systems*. Chi ea '21. Yokohama, Japan and New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 978-1-4503-8095-9. DOI: 10.1145/3411763.3451760.

[36] Leanna Rierson. *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. Boca Raton: CRC Press, Dec. 2017. ISBN: 978-1-315-21816-8. DOI: 10.1201/9781315218168.

[37] Alberto D. Rodriguez, Katherine R. Dearstyne, and Jane Cleland-Huang. "Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability". In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. ISSN: 2770-6834. Sept. 2023, pp. 455–464. DOI: 10.1109/REW57809.2023.00087.

[38] Marcela Ruiz, Jin Yang Hu, and Fabiano Dalpiaz. "Why don't we trace? A study on the barriers to software traceability in practice". en. In: *Requirements Engineering* 28.4 (Dec. 2023), pp. 619–637. ISSN: 1432-010X. DOI: 10.1007/s00766-023-00408-9.

[39] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". en. In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8.

[40] Yonghee Shin, Jane Huffman Hayes, and Jane Cleland-Huang. "Guidelines for Benchmarking Automated Software Traceability Techniques". In: *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*. ISSN: 2157-2194. May 2015, pp. 61–67. DOI: 10.1109/SST.2015.13.

[41] Fangchao Tian et al. "The impact of traceability on software maintenance and evolution: A mapping study". en. In: *Journal of Software: Evolution and Process* 33.10 (2021). ISSN: 2047-7481. DOI: 10.1002/smr.2374.

[42] Shenghui Wang and Rob Koopman. “Semantic embedding for information retrieval”. English. In: *BIR 2017: 5th Workshop on Bibliometric-enhanced Information Retrieval 2017*. CEUR, 2017, pp. 122–132. URL: https://research.utwente.nl/en/publications/semantic-embedding-for-information-retrieval/ (visited on 01/06/2026).

[43] Teng Zhao, Qinghua Cao, and Qing Sun. “An Improved Approach to Traceability Recovery Based on Word Embeddings”. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. Dec. 2017, pp. 81–89. DOI: 10.1109/APSEC.2017.14.

A. Appendix

A.1. Sample LiSSA Configurations

LiSSA's behavior can be modified through the use of a configuration file (see Section 2.5.2). This appendix section contains sample configurations that can be used to execute the LiSSA framework.

A.1.1. Requirements-to-requirements with Reasoning Classifier

```
{
  "cache_dir": "./cache-r2r/GANNT-ensemble",
  "gold_standard_configuration": {
    "path": "./datasets/GANNT/answer.csv",
    "hasHeader": "true"
  },
  "source_artifact_provider": {
    "name": "text",
    "args": {
      "artifact_type": "requirement",
      "path": "./datasets/GANNT/high"
    }
  },
  "target_artifact_provider": {
    "name": "text",
    "args": {
      "artifact_type": "requirement",
      "path": "./datasets/GANNT/low"
    }
  },
  "source_preprocessor": {
    "name": "artifact",
    "args": {}
  },
  "target_preprocessor": {
    "name": "artifact",
    "args": {}
```

```
  },
  "embedding_creator": {
    "name": "openai",
    "args": {
      "model": "text-embedding-3-large"
    }
  },
  "source_store": {
    "name": "custom",
    "args": {}
  },
  "target_store": {
    "name": "cosine_similarity",
    "args": {
      "max_results": "4"
    }
  },
  "classifier": {
    "name": "reasoning_openai",
    "args": {
      "model": "gpt-4o-2024-08-06"
    }
  },
  "result_aggregator": {
    "name": "any_connection",
    "args": {
    }
  },
  "tracelinkid_postprocessor": {
    "name": "req2req",
    "args": {}
  }
}
```

This configuration can be used to execute LiSSA on the GANNT dataset. It uses simple *text* artifact providers which load text files from the specified disk location and create an artifact from the file contents. To convert the artifacts into elements, the *artifact* preprocessor is used, which transforms each artifact into one element preserving the exact content. OpenAI's text-embedding-3-large model is used as the embedding model and the vector store for target elements uses the *cosine similarity* to calculate similarity scores. OpenAI's GPT 4o model with the CoT prompt presented by Fuchß et al. [11] is used for the classification step. The result aggregator *any_connection* specifies that a trace link between elements should be counted as a trace link between their respective artifacts. The req2req postprocessor cuts of file extensions before generating the result trace link set.

A.1.2. Configuration for Requirements as Context

```
{
  "cache_dir" : "./cache-r2r/GANNT-ensemble",
  "gold_standard_configuration" : {
    "path" : "./datasets/GANNT/answer.csv",
    "hasHeader" : true,
    "swap_columns" : false,
    "name" : "gold_standard"
  },
  "context_configurations" : [ {
    "name" : "embeddings",
    "context_id" : "target_requirements",
    "modules" : [ {
      "ARTIFACT_PROVIDER" : {
        "name" : "text",
        "args" : {
          "artifact_type" : "requirement",
          "path" : "./datasets/GANNT/low"
        }
      },
      "PREPROCESSOR" : {
        "name" : "artifact",
        "args" : { }
      },
      "EMBEDDING_CREATOR" : {
        "name" : "openai",
        "args" : {
          "model" : "text-embedding-3-large"
        }
      },
      "ELEMENT_STORE" : {
        "name" : "custom",
        "args" : {
          "max_results" : "2"
        }
      }
    } ]
  }, {
    "name" : "embeddings",
    "context_id" : "source_requirements",
    "modules" : [ {
      "ARTIFACT_PROVIDER" : {
        "name" : "text",
        "args" : {
```

```
          "artifact_type" : "requirement",
          "path" : "./datasets/GANNT/high"
        }
      },
      "PREPROCESSOR" : {
        "name" : "artifact",
        "args" : { }
      },
      "EMBEDDING_CREATOR" : {
        "name" : "openai",
        "args" : {
          "model" : "text-embedding-3-large"
        }
      },
      "ELEMENT_STORE" : {
        "name" : "custom",
        "args" : {
          "max_results" : "5"
        }
      }
    } ]
  } ],
  "source_artifact_provider" : {
    "name" : "text",
    "args" : {
      "artifact_type" : "requirement",
      "path" : "./datasets/GANNT/high"
    }
  },
  "target_artifact_provider" : {
    "name" : "text",
    "args" : {
      "artifact_type" : "requirement",
      "path" : "./datasets/GANNT/low"
    }
  },
  "source_preprocessor" : {
    "name" : "artifact",
    "args" : { }
  },
  "target_preprocessor" : {
    "name" : "artifact",
    "args" : { }
  },
  "embedding_creator" : {
```

```
    "name" : "openai",
    "args" : {
      "model" : "text-embedding-3-large"
    }
  },
  "source_store" : {
    "name" : "custom",
    "args" : { }
  },
  "target_store" : {
    "name" : "cosine_similarity",
    "args" : {
      "max_results" : "4"
    }
  },
  "classifier" : {
    "name" : "embeddings-as-context_openai",
    "args" : {
      "model" : "gpt-4o-2024-08-06",
      "query_context" : "source_requirements",
      "search_context" : "target_requirements",
      "search_mode" : "TARGET",
      "seed" : "133742243",
      "temperature" : "0.0",
      "template" : "Below are artifacts from the same software system. Is
      ↪  there a traceability link between (a) and (b)?\nConsider that the
      ↪  numbered {search_mode} {context_type} also exist in the same
      ↪  software system.\nGive your reasoning, then answer with 'yes' or
      ↪  'no' enclosed in <trace></trace>.\n\n(a) {source_type}:
      ↪  ```{source_content}```\n(b) {target_type}:
      ↪  ```{target_content}```\n\nContext:\n{context_content}\n"
    }
  },
  "result_aggregator" : {
    "name" : "any_connection",
    "args" : {
      "source_granularity" : "0",
      "target_granularity" : "0"
    }
  },
  "tracelinkid_postprocessor" : {
    "name" : "req2req",
    "args" : { }
  },
  "name" : "LiSSA"
```

}

This configuration can be used to execute LiSSA on the GANNT dataset with requirements as context. Context requirements are drawn from the target requirements and are selected based on their similarity to the source requirement of the current classification. GPT-4o is used as the LLM for the classification step.

A.1.3. Configuration for External Trace Links as Context

```
{
  "cache_dir" : "./cache-ext-tl/dronology-ensemble",
  "gold_standard_configuration" : {
    "path" : "./datasets/dronology/answer.csv",
    "hasHeader" : true,
    "swap_columns" : false,
    "name" : "gold_standard"
  },
  "context_configurations" : [ {
    "name" : "embeddings",
    "context_id" : "internal_source_requirements",
    "modules" : [ {
      "ARTIFACT_PROVIDER" : {
        "name" : "text",
        "args" : {
          "artifact_type" : "requirement",
          "path" : "./datasets/dronology/high"
        }
      },
      "PREPROCESSOR" : {
        "name" : "artifact",
        "args" : { }
      },
      "EMBEDDING_CREATOR" : {
        "name" : "openai",
        "args" : {
          "model" : "text-embedding-3-large"
        }
      },
      "ELEMENT_STORE" : {
        "name" : "custom",
        "args" : {
          "max_results" : "infinity"
        }
      }
    } ]
```

```
}, {
  "name" : "embeddings",
  "context_id" : "internal_target_requirements",
  "modules" : [ {
    "ARTIFACT_PROVIDER" : {
      "name" : "text",
      "args" : {
        "artifact_type" : "requirement",
        "path" : "./datasets/dronology/low"
      }
    },
    "PREPROCESSOR" : {
      "name" : "artifact",
      "args" : { }
    },
    "EMBEDDING_CREATOR" : {
      "name" : "openai",
      "args" : {
        "model" : "text-embedding-3-large"
      }
    },
    "ELEMENT_STORE" : {
      "name" : "custom",
      "args" : {
        "max_results" : "infinity"
      }
    }
  } ]
}, {
  "name" : "embeddings",
  "context_id" : "external_source_requirements",
  "modules" : [ {
    "ARTIFACT_PROVIDER" : {
      "name" : "text",
      "args" : {
        "artifact_type" : "requirement",
        "path" : "./datasets/CM1-NASA/high"
      }
    },
    "PREPROCESSOR" : {
      "name" : "artifact",
      "args" : { }
    },
    "EMBEDDING_CREATOR" : {
      "name" : "openai",
```

```
        "args" : {
          "model" : "text-embedding-3-large"
        }
      },
      "ELEMENT_STORE" : {
        "name" : "custom",
        "args" : {
          "max_results" : "infinity"
        }
      }
    } ]
  }, {
    "name" : "embeddings",
    "context_id" : "external_target_requirements",
    "modules" : [ {
      "ARTIFACT_PROVIDER" : {
        "name" : "text",
        "args" : {
          "artifact_type" : "requirement",
          "path" : "./datasets/CM1-NASA/low"
        }
      },
      "PREPROCESSOR" : {
        "name" : "artifact",
        "args" : { }
      },
      "EMBEDDING_CREATOR" : {
        "name" : "openai",
        "args" : {
          "model" : "text-embedding-3-large"
        }
      },
      "ELEMENT_STORE" : {
        "name" : "custom",
        "args" : {
          "max_results" : "infinity"
        }
      }
    } ]
  }, {
    "name" : "external_trace_links",
    "context_id" : "external_trace_links",
    "modules" : [ {
      "GOLD_STANDARD" : {
        "args" : {
```

```
        "path" : "./datasets/CM1-NASA/answer.csv",
        "hasHeader" : "true",
        "swap_columns" : "false"
      }
    }
  } ]
} ],
"source_artifact_provider" : {
  "name" : "text",
  "args" : {
    "artifact_type" : "requirement",
    "path" : "./datasets/dronology/high"
  }
},
"target_artifact_provider" : {
  "name" : "text",
  "args" : {
    "artifact_type" : "requirement",
    "path" : "./datasets/dronology/low"
  }
},
"source_preprocessor" : {
  "name" : "artifact",
  "args" : { }
},
"target_preprocessor" : {
  "name" : "artifact",
  "args" : { }
},
"embedding_creator" : {
  "name" : "openai",
  "args" : {
    "model" : "text-embedding-3-large"
  }
},
"source_store" : {
  "name" : "custom",
  "args" : { }
},
"target_store" : {
  "name" : "cosine_similarity",
  "args" : {
    "max_results" : "4"
  }
},
```

```
  "classifier" : {
    "name" : "external-trace-links-as-context_openai",
    "args" : {
      "model" : "gpt-5-2025-08-07",
      "temperature" : "1",
      "internal_source_context_id" : "internal_source_requirements",
      "internal_target_context_id" : "internal_target_requirements",
      "external_source_context_id" : "external_source_requirements",
      "external_target_context_id" : "external_target_requirements",
      "external_trace_links_context_id" : "external_trace_links",
      "number_of_examples" : "4",
      "seed" : "133742243",
      "template" : "Below are artifacts from multiple software
      ↪  systems.\nConsidering the examples, is there a traceability link
      ↪  between the source and target requirement ({i}) ?\nAnswer with
      ↪  'yes' or 'no'.\n\n{negative_examples}\n{positive_examples}\n({i})
      ↪  Source {source_type}: '''{source_content}'''\nTarget {target_type}:
      ↪  '''{target_content}'''\nLink:\n",
      "example_template" : "({i}) Source {source_type}:
      ↪  '''{source_content}'''\nTarget {target_type}:
      ↪  '''{target_content}'''\nLink: {link}\n"
    }
  },
  "result_aggregator" : {
    "name" : "any_connection",
    "args" : {
      "source_granularity" : "0",
      "target_granularity" : "0"
    }
  },
  "tracelinkid_postprocessor" : {
    "name" : "identity",
    "args" : { }
  },
  "name" : "LiSSA"
}
```

This configuration can be used to execute LiSSA on the dronology dataset with external trace links as context. It uses CM1-NASA as the source for external trace links and GPT-5 for classification. Context element stores are set to retrieve infinite similar elements so that the selection can be made within the classifier, which is configured with the context IDs of all involved contexts. Four positive and four negative examples are provided for each classification, and the prompt from Prompt 6 is used with negative-first ordering.

A.2. Complete Example Prompts

This appendix section contains full example prompts which have been generated and used by LiSSA in the evaluation.

A.2.1. Requirements as Context

Prompt 101 was generated by LiSSA during evaluation on the Dronology dataset. It shows the classification of high-level requirement RE-510 (source) and low-level requirement DD-515 (target) with 5 dynamically selected context requirements from the pool of potential targets.

A.2.2. External Trace Links as Context

A fully substituted prompt when trace links are used as context can be seen in Prompt 102. This prompt was generated by LiSSA using GPT-5 during the evaluation on the dronology dataset with the CM1-NASA dataset serving as the context source.

Prompt 101: Generated requirements as context prompt for dronology
Below are artifacts from the same software system. Is there a traceability link between (a) and (b)?
Consider that the numbered TARGET requirement also exist in the same software system.
Give your reasoning, then answer with 'yes' or 'no' enclosed in <trace></trace>.

(a) requirement: "'Coordinate system conversion

When requested the _CoordinateSystem_ shall transform coordinates to alternate formats."'
(b) requirement: "'Coordinate system conversion from p-vector

When requested coordinates shall be transformed from p-vectors to the default representation."'

Context:
(1): "'Coordinate system conversion to n-vector

When requested the _CoordinateSystem_ shall transform coordinates transformed from the default representation to n-vectors."'
(2): "'Coordinate system conversion to p-vector

When requested the _CoordinateSystem_ shall be transform coordinates from the default representation to p-vectors."'
(3): "'Coordinate system conversion from n-vector

When requested coordinates shall be transformed from n-vectors to the default representation."'
(4): "'Default coordinate system

The default coordinate representation shall use LLA (longitude latitude altitude) format."'
(5): "'Forward commands to UAV

The _GCS_ shall transform commands into a vehicle specific format."'

Prompt 102: Generated trace links as context prompt for dronology
Below are artifacts from multiple software systems.
Considering the examples, is there a traceability link between the source and target requirement (9) ?
Answer with 'yes' or 'no'.

(1) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Control and Monitoring Every time the CCM Control executes, it calls ccmPerProcess() to handle periodic processing responsibilities. Such responsibilities include analog to digital conversion updates, DPU task monitoring, ICU heartbeat message production, and watchdog strobe. The ccmHealthChk() function, called by ccmPerProcess() verifies the execution of other tasks by monitoring the amount of time that has elapsed since each task last reported. Other tasks report their execution to the CCM Control Task by calling the function, ccmTaskReport(), providing their task index. Each task has an expected execution frequency, and if a task does not execute as expected, an error is reported in DPU housekeeping. If the Command Dispatch Task fails to report for an extended period, the DPU will execute a reboot, since it is impossible to command the DPU if this task is not executing, otherwise it will strobe the watchdog"'
Link: no
(2) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Command and Control CSC The Command and Control (CCM) CSC is a Level 2 reuse component from the INSTRUMENT Y project. The Command and Control CSC includes the following components:
* A Control Task, ccmCtrlTask(), which initializes the DPU FSW and spawns other tasks at bootup, schedules the production of DPU housekeeping data packets, monitors the execution of other tasks, and schedules the execution of other periodic tasks such as the heartbeat message and the watchdog strobe; and
* A Command Dispatch Task, ccmCmdTask(), which receives and dispatches real-time commands received from the SCU or the ICU.
Major data structures include:
* A Static Data table that keeps track of the operational state. Data included in this table includes housekeeping production rates, CCM specific flags, and the number of commands executed.
* A Command Queue into which commands are placed when they arrive, via interrupt, from the ICU or the SCU.
* An Error/Event Queue which accumulates error and event codes which are reported by the DPU FSW. These error and event codes are removed from the queue and placed into a telemetry packet at a given interval and included in DPU housekeeping data."'
Link: no

(3) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Error Collection and Reporting The S_ccm_ERR_REPEAT error encodes the count of the last repeated error in its low order byte. If a new error is reported as discussed above, ccmErrEnq() will enqueue a S_ccm_ERR_REPEAT for any previously repeated error, along with the newly reported error. In order to keep the original error codes and their repeated counts together in the same error packet, ccmMkHkErr(), enqueues a special error code, S_ccm_ERRQ_FLUSH, as a special signal to ccmErrEnq() that it needs to clear its error tracking mechanism and enqueue any repeated error counts associated with a particular error."'
Link: no
(4) Source requirement: '"The DPU-TMALI shall provide TMALI_HK to the DPU-CCM on request."'
Target requirement: '"Normal Data Exchange Sequence The TMALI CSC serves as an intermediate manager of EVENT data supplied by the DCI Driver CSC and eventually delivered to the DPA CSC. The TMALI CSC waits for notification from the DCI CSC that a frame limit (or data timeout) has been reached in the Ping-Pong buffer indicating the EVENT data is ready to be served to TMALI. TMALI reads all EVENT data from the DCI and notifies the DCI that it can swap Ping-Pong buffers when ready. TMALI gives a semaphore to unblock the tmaliWait() call from the DPA."'
Link: no

(5) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Control and Monitoring The DPU produces eight types of housekeeping packets."'
Link: yes
(6) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Error Collection and Reporting The DPU-CCM CSC provides a centralized error reporting interface, ccmErrEnq(), that other FSW tasks use to report errors. Each time it wakes, ccmTask() checks to see if it is time to form an error/event packet for transmission to the ground. If so, ccmTask() calls ccmHkMkError() to actually create the packet and forward it to DPU-SCUI for transmission to the ground."'
Link: yes
(7) Source requirement: '"The DPU-CCM shall provide a mechanism for other CSCs to report errors for inclusion in the DPU_HK."'
Target requirement: '"Public Functions This routine is called by any CSC in order to report an error or event that should be included in DPU housekeeping. If this routine is called from interrupt context a static global variable, ccmISRError, is set so that the error can be enqueued later (see ccmCtrlTask()). This is done since the

error/event queue is semaphore-protected and a semaphore cannot be taken in an ISR. The error queue semaphore has priority inversion set to reduce conflicts between multiple callers should a priority inversion situation arise. This routine also replaces frequently occuring errors with a special repeat error code. The repeat error code is a special error code that follows a normally reported error code to indicate that the normally reported error code previously reported has occurred more than once in the last high rate reporting period'''
Link: yes
(8) Source requirement: '''The DPU-TMALI shall provide TMALI_HK to the DPU-CCM on request.'''
Target requirement: '''Public Functions This routine gets housekeeping data stored for the TMALI CSC including some DCI parameters and resets the tmali internal counters to zero. When the caller supplied pointer to a TMALI_HK structure is NIL no data is returned but the tmali internal counters are still reset to zero.Two of the four error counters are updated within an ISR context, this requires a task switch safe implementation of these counters. Within VxWorks this could be solved using a counting semaphore. In the TMALI CSC design a faster method was used by using a free running error counter and a careful update of the reported number of errors, using an extra temporary variable. As the increment and assignment of the 32 bit unsigned integers themselves are 'atomic' operations the resulting counters are task safe and no error events are lost. This means that the sum of the errors reported by the tmaliHkGet() function is equal to the total numbers of errors that occurred, no error reports are lost due to the update of the internal structures.'''
Link: yes

(9) Source requirement: '''Coordinate system conversion

When requested the _CoordinateSystem_ shall transform coordinates to alternate formats.'''
Target requirement: '''Coordinate system conversion from p-vector

When requested coordinates shall be transformed from p-vectors to the default representation.'''
Link: