2023 IEEE 31st International Requirements Engineering Conference Workshops (REW) | 979-8-3503-2691-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/REW57809.2023.00087

Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability

Alberto D. Rodriguez
College of Engineering
University of Notre Dame
Notre Dame, Indiana
arodri39@nd.edu

Katherine R. Dearstyne
College of Engineering
University of Notre Dame
Notre Dame, Indiana
kdearsty@nd.edu

Jane Cleland-Huang
College of Engineering
University of Notre Dame
Notre Dame, Indiana
JaneClelandHuang@nd.edu

***Abstract*—Large Language Models (LLMs) have the potential to revolutionize automated traceability by overcoming the challenges faced by previous methods and introducing new possibilities. However, the optimal utilization of LLMs for automated traceability remains unclear. This paper explores the process of prompt engineering to extract link predictions from an LLM. We provide detailed insights into our approach for constructing effective prompts, offering our lessons learned. Additionally, we propose multiple strategies for leveraging LLMs to generate traceability links, improving upon previous zero-shot methods on the ranking of candidate links after prompt refinement. The primary objective of this paper is to inspire and assist future researchers and engineers by highlighting the process of constructing traceability prompts to effectively harness LLMs for advancing automatic traceability.**

***Index Terms*—automated software traceability, large language models, prompt engineering**

I. Introduction

The challenges of automating traceability have been well documented over the past two decades [1], [15], [18], [25]; however, achieving satisfactory degrees of accuracy across diverse datasets has been an ongoing challenge [9], [13] that has inhibited its adoption in industry. The release of the Google's BERT model [11] in 2018 introduced new possibilities for the field, transforming the once far off dream of automatic traceability into a reality for projects in common domains [16], [17]. However, despite these improvements, challenges such as highly-technical domain-specific terminology, low data availability for training, and lack of interpretability meant that automated tracing continued to under-perform in many projects and domains where trace links were still delivered at low degrees of accuracy [8], [19]. In the present day, large language models (LLMs), such as GPT3 and Claude [2], [4], offer the promise of further transformation in automated traceability, eliminating many of these problems and introducing new possibilities for the field. However, as of yet, there is no clear direction on how best to utilize LLMs for automated traceability.

When we began the work for this paper, our initial aspiration was to discover the "silver bullet" prompt for automated traceability. Similar to previous approaches [1], [16], [17], the "silver bullet" would discern true candidate links from false ones across all projects and circumstances. While we identified a prompting approach that performed well across multiple projects, we concluded that the optimal prompting strategy depends on factors like available resources, the model being used, and the targeted usage scenario. Different LLMs exhibit distinct strengths and weaknesses and may require different prompts to achieve desired outcomes on the same data sets; compounding this, variance across versions of the same base model can alter performance on the same task [5]. Moreover, top-performing models can be cost-prohibitive to many engineers and researchers. Despite LLMs' capabilities, high variability persists across projects, prompts, and parameters.

Therefore, by bringing attention to some of the obstacles we encountered while crafting out prompts, we hope to make researchers and practitioners aware of potential pitfalls when employing the models for traceability related tasks. Rather than merely showcase top results, we have chosen to elaborate on the process we followed to construct our prompts with the goal of inspiring other engineers who may wish to identify a prompt that best suits their needs.

In this paper, we seek to shed light on the following questions:

1) Do LLMs possess knowledge necessary for tracing projects with technical domain-specific vocabulary?
2) Can LLMs provide reasonable explanations for their decisions?
3) If so, can these explanations be utilized to improve prompts?
4) Can reasoning be used to improve responses?
5) How can LLMs be leveraged to generate software traceability links?

While much future work is needed in this area, we hope to aid future researchers and engineers by highlighting the process of constructing traceability prompts for leveraging LLMs effectively to advance automatic traceability.

II. Related Work

Effective automated software traceability has many benefits for software engineering, and several approaches have therefore been proposed to address its challenges. In recent years, the emergence of LLMs, such as GPT-3 and Claude, has shown promise for automating software traceability and mitigating the limitations of previous methods. In this section, we discuss

2770-6834/23/$31.00 ©2023 IEEE
DOI 10.1109/REW57809.2023.00087

TABLE I
DATASETS

Project Name	Description	Artifacts	Children	Candidates	True
CM1 [15]	The requirements for an instrument a part of NASA's Metric Data Program (MDP).	High-Level Requirements → Low-Level Requirements	53	265	13
iTrust [21]	Open-source electronic health record system. Created at North Carolina State University as a part of a software engineering course.	Requirements → Java Classes	227	1135	13
Dronology [10]	A system for managing the navigation of UAVs and their communication to the ground control station.	**NL**: Requirements → Design Definitions	99	495	4
		PL: Design Definitions → Java Classes	458	2290	48

Describes the artifact types in each dataset, the number of children per query, the resulting candidate links across all queries, and how many of those candidates were true links. Dronology is split into two datasets, DronologyNL and DronologyPL, to focus on traces between natural language artifacts and between natural language and programming language artifacts respectively.

the relevant works that have explored the use of large language models and the subjectivity of trace establishment in the context of software traceability.

Early work in automated traceability relied on classical natural language processing (NLP) techniques such as the vector space model (VSM) and latent semantic indexing (LSI) to establish traceability links between software artifacts based on their textual similarity [1], [3]. In the 2010s, deep learning techniques such as long short-term memory networks (LSTMs) and gated recurrent units (GRUs) were applied to improve traceability performance. Researchers used these neural networks to learn distributed representations of software artifacts and match them based on semantic similarity [13]. Around 2018, pretrained language models and transformers revolutionized the field. Models like Google's BERT allowed researchers to generate contextualized embeddings of software artifacts and achieve state-of-the-art results in automated traceability tasks [16], [17]. Transformer language models then grew exponentially larger and more powerful, culminating in GPT-3 and models with hundreds of billions of parameters. GPT-3 demonstrated human-level language understanding with 175 billion parameters, achieving startling fluency and few-shot learning capabilities [4], [7], [24]. GPT-4 continues to push the limits of LLMs, scoring in the top 10% of the BAR exam [22].

In the domain of software engineering, efforts have been made to leverage large language models for various software engineering tasks including code generation, summarization, and enhancement [6], [27]. Although prompt-engineering is a relatively new area of exploration, some prior work has been done on how best to instruct models for various tasks. Researchers have identified different prompt patterns and techniques that tend to produce the best results - many of which are employed in this paper [12], [29]. Additionally, prompt engineers have crafted prompts for a variety of tasks, including classification [14], [20] and ranking [23], both of which we utilize in this paper.

However, there has not been extensive evaluation of the potential for large language models in automated software traceability. To address this gap, we conducted a preliminary investigation using Claude, an LLM developed by Anthropic, to predict trace links between software artifacts. We outlined our two approaches for trace link prediction: classification and ranking. The evaluation of our approaches will be discussed in the following section.

III. EXPERIMENTAL SETUP

For the preliminary investigation reported in this paper, we analyzed three software engineering datasets: CM1, iTrust, and Dronology. We selected these datasets to span natural language and programming language artifacts as well as diverse application domains (embedded systems, healthcare, UAVs).

For each dataset, we selected only a subset of its data to use in our study in order to increase the depth of our analysis, reduce run-time, and decrease cost. To select the links, we first calculated the number of children artifacts traced to each parent and then identified the minimum, maximum and median number of links. Using these categories, we identified five parent artifacts: one with the fewest child links, three with the median number of child links, and one with the maximum number of child links. In cases where multiple parent artifacts tied for the minimum, median, or maximum, we randomly sampled from those tied parents. This allowed us to create a set of trace queries that were representative of the project's link distribution of its trace queries. Table I describes the selected queries for each system noting the parent and child types, the number of potential trace links (candidates), and the number of those links that were actually true.

Prior to the start of our experiments, we tested OpenAI's *text-davinci-003* model for predicting trace links, and found that, while it required slightly different prompts, it had comparable capabilities to Anthropic's Claude instant model (*claude-instant-v1*). Due to its lower cost and increased speed, we selected Claude for the remainder of our experiments. We also explored utilizing embeddings to compute similarity scores between artifacts, similar to the original Vector Space Model (VSM) approaches [1]. We examined the ada-embedding model developed by OpenAI (*text-embedding-ada-002*), however, the results obtained from this investigation did not show a significant advantage over VSM. Therefore, we decided to leverage the generative capabilities of the models for trace link predictions within this paper. Nevertheless, we acknowledge the need for future endeavors to conduct a more comprehensive analysis of the advantages and disadvantages associated with utilizing embeddings for generating trace links.

Additionally, we obtained summaries of all code artifacts to use in our experiments. We accomplished this by prompting the model to provide a several sentences focusing on the high-level functionality of the code. Although this removed some information, the resulting summaries contained most of the relevant details and reduced the number of tokens required for each tracing prompt.

For our first approach, we prompted the model to classify every source and target artifact pair. Each prompt followed a similar format, consisting primarily of a question and instructions to answer 'yes' or 'no', followed by the content of the source artifact numbered as '1' and the target artifact numbered as '2'. When a prompt directly referenced the source or target in the question, it used (1) to indicate the source or (2) to indicate the target, corresponding to the numbers of the artifact content (e.g., "Is (1) related to (2)?"). Each question was posed such that an answer of 'yes' was indicative of a link between the answers, while 'no' indicated that the artifacts were not linked. The resulting candidate links are then evaluated against the ground truth links using common classification metrics such as precision and recall.

Precision is the ratio of the number of correctly identified relevant trace links to the total number of trace links identified by the system. Recall, on the other hand, measures the ratio of the correctly identified relevant trace links to the total number of relevant trace links in the system. This is shown below where TP is the true positives, FP is false positives, and FN is false negatives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

For our ranking approach, we prompted the model to rank all targets for each source artifact. In this case, the model was given the content of the source artifact and the ID and body of each target, separated by newlines. The model was instructed to return the artifact IDs in the order of relevance (from most to least) in a comma delimited list. Given the non-deterministic nature of responses from each model, there were times when the model neglected to include some artifact IDs. This problem was unique to the ranking task, as the model correctly output yes and no each time for the classification task. In these cases we randomly appended the missing ids to the end of the list for our evaluation. We calculate the Mean Average Precision of these rankings to showcase their performance. It provides a measure of the effectiveness of the ranking algorithm in identifying relevant trace links between software artifacts. To calculate MAP, the precision is computed at different levels of recall. The average precision is then calculated as the average of the precision values at each recall level. Finally, the mean of the average precision values across trace queries is taken to obtain the MAP score. The equation for MAP is obtained by taking the mean of the average precision values across different queries or datasets:

$$\text{MAP} = \frac{1}{N} \sum_{i=1}^{N} \text{Average Precision}_i$$

where N is the number of queries or datasets.

Throughout our process of generating trace-links, we have several conversations with the model to test its prior knowledge, understand its responses, and to brainstorm potential prompts and improvements to prompts. We include many of these in our paper. It is important to note that these exchanges occurred independently - the model could not reference previous conversations when responding to subsequent questions or when making new traceability classifications. Each conversation represented an isolated context, rather than a continuing thread. This methodology enabled us to systematically probe the model's knowledge without reliance on prior memorization.

IV. Analysis and Prompt Discovery

Due to space constraints, we focus our analysis on our prompt engineering process for the CM1 dataset. While we followed a similar prompting process with other datasets, we found that many of the challenges we faced were exemplified in CM1. We specifically chose CM1 to focus on because it has posed a unique challenge to researchers in the past [16], [17]. In the end, we show the performance of our best approach on all the datasets to highlight that many of the lessons we learned from CM1 were transferable to the other datasets as well.

A. Domain-specific Knowledge

Before running trace-link prediction with the models, we first wanted to understand what prior knowledge the model had about the system being evaluated. Previously, BERT models often required extensive pre-training on domain-specific documentation to interpret the vocabulary within a project [16], [17]. CM1, in particular, includes a myriad of jargon and acronyms that have long presented a challenge to previous methods of automatic traceability [15], [26]. Since one strength of LLMs is their ability to perform well without any training [4], [22], we wanted to explore whether Claude possessed enough prior knowledge to succeed on the CM1 dataset. To do this, we had the following conversation with it about one of the CM1 artifacts containing a large number of acronyms.

> **Human**: What does this artifact mean?
> The DPU-TMALI shall utilize SCM-DCI-SR, along with ERRNO provided by DPU-DCI to decode errors and place them on an error queue for DPU-CCM.

Assistant: ... Some guesses at the meaning:

DPU may stand for Data Processing Unit ... SCM-DCI-SR seems to be some kind of protocol or interface used for communication between DPU-TMALI and DPU-DCI.

ERRNO is a standard way for software components to return error codes. So DPU-DCI seems to be providing error codes to DPU-TMALI ... DPU-TMALI is receiving error data from DPU-DCI, decoding or interpreting those errors, and then queuing them up to be sent to DPU-CCM.

DPU-CCM seems to be a component that is receiving error data, perhaps for monitoring purposes...

As shown in the conversation above, the model was able to leverage its general knowledge to reason about what each component did from only the artifact content. Although we did not extensively investigate the model's knowledge of each acronym or vocabulary word in the dataset, this example shows that Claude can interpret the meaning of complex acronyms in some cases where previous models could not without fine-tuning [16], [17].

B. Classification Prompt Discovery

Next, we began the process of creating a prompt to use for our trace link prediction. As discussed previously, we provide highlights only for our experiments with CM1 but performed a similar analysis across datasets. Results for each prompt can be found in Table II. Our initial prompt for the model was general:

Human: Is (1) related to (2)? Answer 'yes' or 'no'.

The content of the two artifacts was included in the prompt following the format explained in Section III. On CM1, this prompt resulted in the model correctly identifying 6 out of 13 links and mispredicting 27 of the 225 true negative links as positive links for precision of 18% and recall of 46%.

In order to understand why the prompt failed on over half of the true links, we randomly selected one of the false negatives and prompted the model to explain why it responded 'no' for the link. For reference, the artifacts in the link are shown in Table IV in the row for Link #1.

Assistant: ...The first paragraph is very high level, while the second paragraph goes into more technical implementation details. If they were related, you would expect a more consistent level of technical depth...

Inspired by this response, we inserted "Ignoring the different abstraction levels of the artifacts..." before our question. Interestingly, this link was correctly predicted when we used the modified prompt. Throughout our experiments, this would be a re-occurring problem where the model often failed to predict links between artifacts with different levels of details/abstractions. We therefore began instructing it to ignore abstraction levels in many of our following prompts. When we re-ran this modified prompt on the dataset, it correctly identified 7 out of the 13 links (one more than before) but also predicted 86 false positives. We again prompted it to explain its decision for one of the remaining false negatives (Link #2 in IV). Its response and our subsequent questioning are shown below:

Assistant: ...while there is a small overlap in the error handling mention, the main substance and purposes of the artifacts are distinct. I would still assess them as not strongly related...

Human: Would there be a traceability link between artifact 1 and 2?

Assistant: There could potentially be a weak traceability link between the two artifacts based on the minor overlap in error handling/queueing...

We found that asking it directly if the artifacts were linked received a better response, and further prompting revealed that Claude had a relatively robust understanding of traceability, thus we decided to update our prompt as follows:

Human: Is there a traceability link between (1) and (2)?

We suspected that this prompt would be more specific, potentially eliminating some of the false positives, and this was indeed the case with Claude only predicting one false positive. However, it now correctly identified only 4 of the 13 links. This time when questioned about why it incorrectly believed Link #3 (Table IV) was un-traced, Claude's confusion seemed to come from it not realizing the artifacts were talking about the same system:

Assistant: ...The first requirement mentions specific tasks...that need to report heartbeats, while the second requirement talks about tasks in general without specifying which tasks. So it's not clear if the tasks mentioned in the first requirement are the same as those referred to in the second...

We found this to be another common point of confusion for Claude so we updated our prompt to include "Below are artifacts from the same software system" at the start of our prompt.

Human: Below are artifacts from the same software system. Is there a traceabilty link between (1) and (2)?

With this additional information, Claude identified all of the project trace-links but wrongly predicted 98 negative links to be positive. Since Claude now mistakenly believed many artifacts were linked, we randomly selected a false positive (Link #4 in Table IV) to investigate further. This lead us to gain the following insight into the misprediction:

Assistant: Yes, there is a traceability link between (1) and (2). (1) mentions that the DPU-TMALI places errors on an error queue for the DPU-CCM.

(2) describes the error collection and reporting mechanism of the DPU-CCM, specifically referring to enqueueing errors.

So (2) provides an implementation detail for the error queueing functionality briefly mentioned in (1). Therefore, there is a traceability link between these two artifacts in the software system.

We found nothing in Claude's response to be inherently wrong, so we speculated that the mistake came not from a misunderstanding but rather from a misalignment between Claude and the tracers of CM1 on what should be considered a trace-link. Thus, we began to generate prompts that focused on the specific relationships between the requirements and regulations of CM1.

> **Human**: Below are artifacts from the same software system. Ignoring different levels of abstraction, can (2) be decomposed from (1)? Answer 'yes' or 'no'.

This led to the model identifying 8 of the trace links with only 14 false positives. These missed trace links seemed to result from a confusion surrounding the model's understanding of decomposition, where the model believed that no new information could be added to a decomposed requirement. This was indicated by its response below for Link #1:

> **Assistant**: ... So (2) contains new information that cannot be derived or decomposed from (1) alone. The answer is 'no'.

Replacing the question with "does (2) fulfill (1)?" resulted in the model missing 7 links for a similar reason, including Link #1 again:

> **Assistant**: ...(2) lacks critical details and scope needed to conclusively fulfill (1)...

Despite missing 7 links, the model was able to correctly rule out all but 3 true negatives. Although both of these prompts were too restrictive (i.e. not all the true links were captured), they seemed to align Claude with at least some of the relationships established in CM1 as indicated by its success at eliminating false links.

Some other prompts we tried included:

1. *are (1) and (2) part of the same feature and share functionality?*
2. *does (2) implement system functionality from (1)?*
3. *are (1) and (2) dependent on the same system capability?*

where each prompt began with "Below are artifacts from the same software system. Ignoring different levels of abstraction" and concluded with "Answer 'yes' or 'no'." Each subsequent prompt successfully predicted one more positive link than the one before, but led to an increasing number of false positives ranging from 28 all the way to 48 for #3.

Our final prompt of this structure was a combination of several previous prompts that had shown promise:

> **Human**: Below are artifacts from the same software system. Ignoring different levels of abstraction, can (2) be one of the hierarchical decompositions of (1)? Consider whether (2) implements a part of (1).

By adding "one of the hierarchical" and "a part of", we hoped to make the prompt less restrictive so that the model would allow new information to be added to decomposed artifacts. This time the model correctly predicted all but one true link. Its 44 false positives still fell short of our goal of less than 20 but were significantly less than the 98 returned in the previous approach that achieved these high levels of recall.

C. Introduction of Chain-of-Thought Reasoning

As our prompt evolved, we saw improvements in the model's link-prediction abilities, but the results were still falling short of our goals. It seemed evident that we needed to approach the task differently. As a result, we decided to break down the task into steps to aid the model in reasoning about the relationship between the artifacts before arriving at a final decision. This process, known as "chain of thought", has previously shown to be effective in helping LLMs solve complex tasks [28]. To test the impact of incorporating reasoning into the process, we reran our original prompt with this additional step at the beginning of the prompt:

> **Human**: You are working on a software system and are given a high-level requirement and a low-level requirement. Your job is to determine if they are traced. Give one reason why (1) might be related to (2) enclosed in <related> </related> and one reason why (1) might be un-related to (2) enclosed in <unrelated> </unrelated> but ignore levels of abstraction. Then based on these reasons, do you think (1) is related to (2)? Answer should be 'yes' or 'no' enclosed in <answer> </answer>.

Having it enclose each separate response in xml tags allowed us to easily parse its answer despite the added complexity. An example of the model's response for link #5 in Table IV is shown below:

> **Assistant**: <related> Both specify requirements related to processing commands within a certain time </related> <unrelated> Requirement 1 specifies a time constraint while requirement 2 provides a general description of a command and control subsystem </unrelated> <answer>yes </answer>

Recall increased by nearly 50% while precision rose by 14%. With the added reasoning step, the model missed only 1 true link and misidentified just 25 false positives. With both questions (1) and (3) from Table II, the introduction of incremental reasoning helped the model find a higher portion of the true connections in the data. Due to time constraints, we were unable to test the reasoning on the remaining questions but we believe this is an interesting avenue for future work.

Encouraged by this initial success, we decided to have the model answer each of our questions as intermediate steps before finally determining whether the artifacts were related. We hoped this approach would help the model explore different ways in which the artifacts could be connected. It also allowed us to use a simple ranking system in which more 'yes' responses would increase the likelihood that the artifacts were linked. By quantifying the model's degree of support for a relationship through the ranking system, we could evaluate not just whether it predicted a link but also how confident it was in that prediction based on the reasoning exhibited in its responses.

> **Human**: I am giving you two software artifacts from a system. Your job is to determine if there is a traceability link. Answer whether (2) implements a part of (1) with yes or no enclosed in <implements> </implements>. Answer whether (2) is a hierarchical decomposition of (1) with yes or no enclosed in <decomposed> </decomposed>. Answer whether (2) fulfills (1) with yes or no enclosed in <fulfills> </fulfills>. Answer whether (2) and (1) are part of the same feature and shares functionality with yes or no enclosed in <feature> </feature>. Answer whether (2) and (1) are dependent on the same system capability with yes or no enclosed in <capability> </capability>. Use your answers to give one reason why (1) might be related to (2) enclosed in <related> </related> and one reason why (1) might be un-related to (2) enclosed in <unrelated> </unrelated> Now answer is (1) related to (2) with yes or no enclosed in <traced> </traced>.

D. Ranking Prompt Discovery

Despite not outperforming other classification prompts, ranking the artifacts by the number of 'yes' and 'no' answers, did provide the opportunity to establish a threshold retrospectively, allowing us to categorize items based on the strength of the model's prediction instead of relying on a single yes/no choice. This, combined with Claude's new 100k context window, inspired us to experiment with an entirely new strategy.

For our next experiment, we gave Claude the following instructions:

> **Human**: # Task
> Rank all related artifacts from most to least related to the source.
>
> Source: [SOURCE ARTIFACT]
>
> # Artifacts
>
> <artifact>
> <id>... </id>
> <body>... </body>
> </artifact>
>
> # Instructions
> Rank the artifact bodies from most to least relevant to the source. Provide the ranked artifacts as comma delimited list of artifact ids where the first element relates to the source the most and the last element does so the least.

By providing the model with more context about the system in the prompt and allowing it to compare all targets when making its decision, we hoped to see a performance boost. Unfortunately, the task was not as simple as we had hoped, and we, like previous researchers, identified another nuance with the prompts - order matters [23]. When we presented the target artifacts in a random order, performance was barely above random; however, ordering artifacts that were more likely to be linked at the top, delivered significantly higher performance. It seemed that unless there was some pattern already established, the task would overwhelm the model. Because of this, we decided to rank the target artifacts based on their VSM similarity to the source. Then, we presented the model with targets in this order. With this initialization, the model improved upon the original VSM ranking. Furthermore, While discussions throughout the paper have focused on the CM1 dataset, we applied this approach to the three other datasets presented in Table I and report results for all four datasets in Table III.

E. Summary of Results

Overall, our results demonstrated that the ranking task could be a useful approach to automated traceability, but it may require additional steps and further prompt refinement to reach the necessary performance. In the future, we plan to explore ways of decomposing the overall task into simpler, incremental steps to reduce complexity for the model as we did for the classification task. It should also be noted that the ranking task necessitated a large context window, which may pose a challenge for certain open-source models. Consequently, classification remains a valuable alternative when ranking is infeasible. Furthermore, classification opens up avenues for diverse applications of traceability, such as "trace views" that we discuss further in Section VI.

V. Threats to Validity

While this initial study provides promising evidence that prompt engineering can enhance LLMs for software traceability tasks, several threats could limit the validity of our findings. First, we evaluated only three open-source projects and only provide a detailed analysis of one, limiting the generalization of our findings. However, we selected projects that spanned multiple domains, artifact types, and sizes to improve generalizability. We also constructed trace queries that were representative of their parent distribution. Second, existing traceability datasets are typically incomplete, as truly considering every candidate link in a project grows $\mathcal{O}(n^2)$ with the number of artifacts. The LLMs identified potential missing traces, but we could not fully validate their accuracy without a project expert. Third, our study used a limited set of LLMs which may not represent the full space of the current state-of-the-art. However, we chose the leading LLMs from our initial explorations with publicly available commercial models. Clearly, there are many extension to this study considering more datasets, different LLMs, and other prompt engineering methods. We leave the full exploration of the problem space to future work and focus on showing the potential these models have towards advancing automated software traceability.

VI. Conclusions and Future Directions

Throughout our experiments, we addressed multiple questions and derived several key takeaways regarding using LLMs for trace-link prediction.

A. Key Takeaways

- Small modifications to prompts can lead to significant differences in model outputs, emphasizing the importance of carefully crafting prompts.
- The performance of a given prompt in comparison to alternative phrasings can vary across datasets and models, though some general techniques like chain-of-thought reasoning tend to produce a more consistent performance.
- LLMs frequently identify different artifact relationships by than those selected by human tracers. Prompts should specify the targeted usage of the traceability links (e.g.

TABLE II
CLASSIFICATION METRICS FOR CM1 PROMPTS

ID	Prompt	Precision	Recall	TP	TN	FP	FN
1	**Is (1) related to (2)?**	18%	46%	6	225	27	7
		32.4%	92.3%	12	227	25	1
2	**Ignoring the different abstraction levels of the artifacts, is (1) related to (2)?**	17%	54%	7	218	34	6
3	**Is there a traceability link between (1) and (2)?**	80%	31%	4	251	1	9
		40%	46.2%	6	243	9	7
4	**Below are artifacts from the same software system, is there a traceability link between (1) and (2)?**	12%	100%	13	154	98	0
5	**Below are artifacts from the same software system. Ignoring different levels of abstraction, can (2) be decomposed from (1)?**	36%	62%	8	238	14	5
6	**Below are artifacts from the same software system. Ignoring different levels of abstraction, does (2) fulfill (1)?**	67%	46%	6	249	3	7
7	**Below are artifacts from the same software system. Ignoring different levels of abstraction, are (1) and (2) part of the same feature and share functionality?**	32%	54%	7	237	15	6
8	**Below are artifacts from the same software system. Ignoring different levels of abstraction, does (2) implement system functionality from (1)?**	22%	77%	10	216	36	3
9	**Below are artifacts from the same software system. Ignoring different levels of abstraction, are (1) and (2) dependent on the same system capability?**	19%	85%	11	204	48	2
10	**Below are artifact from the same software system. Ignoring different levels of abstraction, can (2) be one of the hierarchical decompositions of (1)? Consider whether (2) implements a part of (1).**	22%	92%	12	208	44	1
11	**Combining all questions and chain-of-thought reasoning.**	37.9%	84.6%	11	234	18	2

Rows in gray use chain-of-thought to make their final trace classifications.

change impact analysis, hierarchical composition) to better align the model's output with the desired outcome.

- Specifying the targeted usage has the additional benefit of opening the door for creating different trace views - a possible advantage over purely similarity-based methods.
- Requiring models to show intermediate reasoning steps boosts performance on some tasks and builds in explanations into the decision making process. This is useful to both to those establishing the trace links and those using them.
- List ranking style prompts are highly sensitive to the order of artifacts presented in the prompt. This variability was mitigated by pre-sorting by VSM scores.
- Overall, carefully tailored prompts are needed to harness the versatility of LLMs for the task of traceability and to produce outputs that are consistent with the goals of traceability engineers and researchers.

Throughout this process, one of our biggest takeaways was how minor adjustments to prompts could have dramatic impacts on the results. Subtle changes, such as pluralizing words, interchanging prepositions, or reordering phrases, could alter the outcomes. These findings underscore the inherent challenge of engineering robust prompts. In future research, we aim to explore strategies that mitigate such variability and delve into the effectiveness of different prompts across different models.

Further, due to the limited number of trace queries we analyzed per dataset as well as our integration of chain-of-thought, we were able to review trace predictions in depth. Interestingly, we were often surprised by the strength of many false positives, forcing us to re-think the accurate and complete nature of these datasets. Reviewing predictions for even our smallest subset (265 combinations) became an arduous task. In reality, industrial projects range from 50K to 500K potential trace links, making it extremely challenging to have complete and standardized tracing practices. However, examining the predictions of a few selected trace links may still provide traceability experts with the insights they need to refine prompts in a way that improves performance across the project.

TABLE III
MEAN AVERAGE PRECISION OF RANKED TARGET ARTIFACTS

Dataset	Ranking by VSM	Ranking by VSM + LLM
CM1	70.7%	79.4%
iTrust	44.5%	44.2%
Dronology (NL)	82.9%	100%
Dronology (PL)	23.2%	30.8%

B. Do LLMs possess knowledge necessary for tracing projects with domain-specific vocabulary?

Our conversations with Claude revealed that it contained sufficient knowledge to draw many correct conclusions about the CM1 system, irrespective of the acronyms or jargon used. Furthermore, we were able to obtain high MAP scores without performing any additional pre-training. Nevertheless, we plan to experiment with pre-training in the future to see if it can provide a performance boost. Additionally, we hope to test the model's knowledge on a wider range of datasets. It is important to note that since the datasets in this paper were all publicly available at the time of the model's creation, we cannot eliminate the possibility that the model had previous exposure to them. Thus, we are particularly interested to see how the model performs on an entirely new dataset.

C. Can LLMs provide reasonable explanations for their decisions?

By probing the model to elicit explanations for many of its mispredictions, we found that it could provide an in-depth analysis of its decision. Whether or not these explanations are accurate reflections of the reasoning behind the model's decision is beyond the scope of this paper, but we did find that when we adjusted the prompts based on the model's explanation, we were often able to change its answer.

D. If so, can these explanations be utilized to improve prompts?

The ability to alter the model's decision by using its explanations proved to be a useful tool for improving prompts. Engaging in conversations with the model enabled an increased understanding of its interpretation of a given prompt, facilitating an iterative approach to refine prompts. Gradually adjusting the prompts in this way can be used to find a prompt that better aligns the model's understanding with the objectives of the tracer.

E. Can reasoning be used to improve responses?

By asking the model to formally articulate its thinking in response to probing questions, the model was able to make a more well-informed final judgment about the relationship between the artifacts in the classification. This also offers the advantage of allowing the task to be broken down into smaller pieces, where the model first evaluates the relationship between the artifacts and then makes a final decision. Further, chain-of-thought reasoning has the potential to improve the ranking task and should be evaluated in future work.

F. How can LLMs be leveraged to generate software traceability links?

In our experiments, we explored two different tasks which could be used to predict trace links from pairs of software artifacts: classification and ranking. While ranking allows for a nuanced expression of confidence in a prediction, classification offers the advantage of needing a smaller context window and enables the discovery of diverse relationship types. By adapting our prompts to describe various relationships, we captured distinct links. For instance, when inquiring whether two artifacts were part of the same feature, we discovered different links than when asking if they shared functionality. This can be used to present multiple "views" of traceability, where each view highlights different relationships within the system. This may be particularly valuable for change propagation where the prompt can focus on determining whether a modification to one artifact necessitates a change in the other. Additionally, multiple prompts may be combined to capture the many different relationships present in the project. This presents an avenue for future investigation.

An alternative way in which LLMs can be used for trace link prediction is by comparing the similarity of artifact embeddings. As mentioned previously, we opted not to explore this method in this paper, but future works might benefit from comparing this approach to those discussed in this paper.

G. Concluding Remarks

Overall, our experiments demonstrated that large language models show promise for tracing software systems. As opposed to previous approaches for automated traceability, LLMs can perform well without pre-training and are able to offer detailed explanations of their decisions. These explanations are not only useful for helping an engineer make an informed decision about a trace-link but can guide the process of selecting an appropriate prompt for the tracing task. Through iterative prompt refinement, the models can be used to classify trace links and establish a diverse set of relationships between project artifacts. The models are also capable of ranking target artifacts based on how related they are to a source artifact, albeit with aid from VSM. Ranking can allow engineers to sift through a prioritized list of candidate links and potentially reducing the review time required.

While this paper showcases the power of LLMs for traceability, it also highlights many of the lingering challenges in engineering effective prompts for the models. Careful tailoring of prompts can help to reach high performance for each project but this was ultimately a time-consuming task that may not always be feasible. Although the community might one day discover a "silver bullet" prompt, a more practical path forward may be to identify common patterns that make prompts most effective for certain projects and tracing objectives. Discovering such patterns could enable partially automating this process so that it can be seamlessly integrated into current traceability workflows. There remains much future work that must be done to gain a comprehensive understanding of how LLMs can best be utilized to enhance the field of traceability.

Acknowledgement

The work in this paper has been partially funded by USA National Science Foundation Grants # SHF-1901059, SHF-1909007, and PFI-TT-2122689.

TABLE IV
INVESTIGATED LINKS

ID	Source	Target
1	The DPU-CCM shall implement a mechanism whereby large memory loads and dumps can be accomplished incrementally.	Memory Upload and Download Handling Data can be uploaded to several types of locations, including: • DRAM • EEPROM • Hardware registers • EEPROM filesystem The D-MEM-DAT-UPLD command specifies the target location. If the destination is the EEPROM filesystem, a "block number" is provided in lieu of a memory address, which is used by the DPU FSW to formulate a filename of the form *eefs1:DPU_blk.##*, where ## is the block number. In this case, once the entirety of the uploaded data is received by the DPU FSW, the uploaded data is then written to that file in the EEPROM filesystem. If a file already exists with that name, it is overwritten. The EEPROM filesystem can be reinitialized using the command D-MEM-DISK-INIT.
2	The DPU-TMALI shall utilize SCM-DCI-SR, along with ERRNO provided by DPU-DCI to decode errors and place them on an error queue for DPU-CCM.	Control and Monitoring the CCM Control Task initializes the DPU FSW. It is the responsibility of the CCM Control Task to establish a successful boot. It does so by blocking on temporary semaphores, each with a 5 second timeout, after spawning the SCU Interface Task and the CCM Command Task. If both of these tasks report a successful initialization by giving the semaphore, the CCM Control Task toggles the BC_INDEX parameter in EEPROM to indicate a successful boot. If either task does not report a successful initialization, the CCM Control Task disables the watchdog strobe to effect a reboot of the DPU. The rationale for selecting the successful initialization of these two tasks as the definition of a successful boot is that the DPU FSW requires these tasks, as a minimum, to establish ground contact and provide commandability. Once this initialization is complete, the task blocks on a binary semaphore which is given by the SCUI Command ISR upon arrival of the 1 Hz Clock Message. In the event a Clock Message does not arrive, the semaphore will time out after 1.5 seconds. The CCM Control Task remains alive to create and transmit DPU housekeeping at the appropriate intervals, perform various periodic processing tasks, and to process memory dump commands. The final call to ccmErrEnq() is performed in order that if an error occurs in an interrupt service routine, a global variable is set to the value of the errno which is then enqueued into the Error/Event Queue as part of this task's normal processing. 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.
3	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.	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.
4	The DPU-TMALI shall utilize SCM_DCI_SR, along with ERRNO provided by DPU-DCI to decode errors and place them on an error queue for DPU-CCM.	Error Collection and Reporting The ccmErrEnq() function tracks the last error reported and its frequency of occurrence. Once an error code has been reported it becomes the previously reported error code maintained by ccmErrEnq(). A repetition count is then incremented for each subsequent, consecutively reported, identical instance of this previously reported error. If this error code is reported more than once in one high-rate housekeeping reporting period, then a special error, S_ccm_ERR_REPEAT is enqueued with the repetition count for the error encoded in the least significant byte. This mechanism effectively reduces the potential for housekeeping telemetry to become flooded with a single repeated error.
5	The DPU-CCM shall process real-time non-deferred commands within B ms of receipt from the ICU or the SCU.	The Command and Control CSC provides the core command and control functionality for the system. It includes tasks for initializing the system at bootup, scheduling housekeeping data generation, monitoring other tasks, executing periodic tasks, and receiving and dispatching real-time commands. It maintains data structures for system state, commands, errors and events.

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, p. 970–983, Oct 2002.

[2] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, J. Kernion, K. Ndousse, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, and J. Kaplan, "A general language assistant as a laboratory for alignment," 2021.

[3] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, vol. 1. Cape Town, South Africa: ACM Press, 2010, p. 95. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1806799.1806817

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] L. Chen, M. Zaharia, and J. Zou, "How is ChatGPT's behavior changing over time?" Jul. 2023, arXiv:2307.09009 [cs]. [Online]. Available: http://arxiv.org/abs/2307.09009

[6] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[7] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[8] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, no. 6, pp. 27–35, 2007. [Online]. Available: https://doi.org/10.1109/MC.2007.195

[9] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, J. D. Herbsleb and M. B. Dwyer, Eds. ACM, 2014, pp. 55–69. [Online]. Available: https://doi.org/10.1145/2593882.2593891

[10] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: An incubator for cyber-physical system research," *CoRR*, vol. abs/1804.02423, 2018. [Online]. Available: http://arxiv.org/abs/1804.02423

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[12] S. Ekin, "Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practices," 5 2023. [Online]. Available: https://www.techrxiv.org/articles/preprint/Prompt_Engineering_For_ChatGPT_A_Quick_Guide_To_Techniques_Tips_And_Best_Practices/22683919

[13] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. Buenos Aires: IEEE, May 2017, p. 3–14. [Online]. Available: http://ieeexplore.ieee.org/document/7985645/

[14] X. Han, W. Zhao, N. Ding, Z. Liu, and M. Sun, "PTR: Prompt Tuning with Rules for Text Classification," Sep. 2021, arXiv:2105.11259 [cs]. [Online]. Available: http://arxiv.org/abs/2105.11259

[15] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng.*, vol. 32, no. 1, pp. 4–19, 2006. [Online]. Available: http://dx.doi.org/10.1109/TSE.2006.3

[16] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," *arXiv:2102.04411 [cs]*, Feb 2021, arXiv: 2102.04411. [Online]. Available: http://arxiv.org/abs/2102.04411

[17] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Enhancing automated software traceability by transfer learning from open-world data," no. arXiv:2207.01084, Jul 2022, arXiv:2207.01084 [cs]. [Online]. Available: http://arxiv.org/abs/2207.01084

[18] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," in *Software and Systems Traceability*. Springer, 2012, pp. 71–98.

[19] S. Maro, J.-P. Steghöfer, and M. Staron, "Software traceability in the automotive domain: Challenges and solutions," *Journal of Systems and Software*, vol. 141, pp. 85–110, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300608

[20] C. W. F. Mayer, S. Ludwig, and S. Brandt, "Prompt text classifications with transformer models! An exemplary introduction to prompt-based learning with large language models," *Journal of Research on Technology in Education*, vol. 55, no. 1, pp. 125–141, Jan. 2023, publisher: Routledge _eprint: https://doi.org/10.1080/15391523.2022.2142872. [Online]. Available: https://doi.org/10.1080/15391523.2022.2142872

[21] A. Meneely, B. Smith, and L. Williams, *iTrust Electronic Health Care System: A Case Study*, Software System Traceability. Ed. Jane Cleland-Huang and O. Gotel, Eds. Springer, 2011.

[22] OpenAI, "GPT-4 Technical Report," Mar. 2023, arXiv:2303.08774 [cs]. [Online]. Available: http://arxiv.org/abs/2303.08774

[23] Z. Qin, R. Jagerman, K. Hui, H. Zhuang, J. Wu, J. Shen, T. Liu, J. Liu, D. Metzler, X. Wang, and M. Bendersky, "Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting," Jun. 2023, arXiv:2306.17563 [cs]. [Online]. Available: http://arxiv.org/abs/2306.17563

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[25] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 834–845.

[26] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Understanding the Challenges of Deploying Live-Traceability Solutions," Jun. 2023, arXiv:2306.10972 [cs]. [Online]. Available: http://arxiv.org/abs/2306.10972

[27] G. Sridhara, R. H. G., and S. Mazumdar, "Chatgpt: A study on its utility for ubiquitous software engineering tasks," no. arXiv:2305.16837, May 2023, arXiv:2305.16837 [cs]. [Online]. Available: http://arxiv.org/abs/2305.16837

[28] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Jan. 2023, arXiv:2201.11903 [cs]. [Online]. Available: http://arxiv.org/abs/2201.11903

[29] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," Feb. 2023, arXiv:2302.11382 [cs]. [Online]. Available: http://arxiv.org/abs/2302.11382