Karlsruhe Institute of Technology

Automated Prompt Engineering for Traceability Link Recovery

Bachelor's Thesis of

Daniel Schwab

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

23. June 2025 – 23. October 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Contents

List of Figures

1 Introduction

During software development, many artifacts are created, ranging from the actual project code through documentation to a multitude of formal and informal diagrams. Trace link recovery aims to link these artifacts across different domains or versions. Challenges such as inconsistent naming are frequent issues in software projects [19]. Trace link recovery approaches need to be able to deal with such hardships. fig. 1.1 provides an overview of how these artifacts might look like.

Large language models (LLM) have made rapid advancements in recent years. They are becoming increasingly popular to deal with trace link recovery tasks and have shown very promising results. As they are still unable to reason and think on their own [18], prompt engineering is required to extract good results. Manually determining prompts suited for each specific problem is a quite tedious and time intensive task.

To address this shortcoming, automated prompt engineering tools can be used. They will typically use the LLM itself to determine suitable adjustments or generate new prompts based on a description or training data of the problem [16].

The LiSSA framework [5] .... I will contribute an automatic prompt refinement algorithm, hoping to improve trace recovery, especially for larger projects in the requirements to requirements domain.

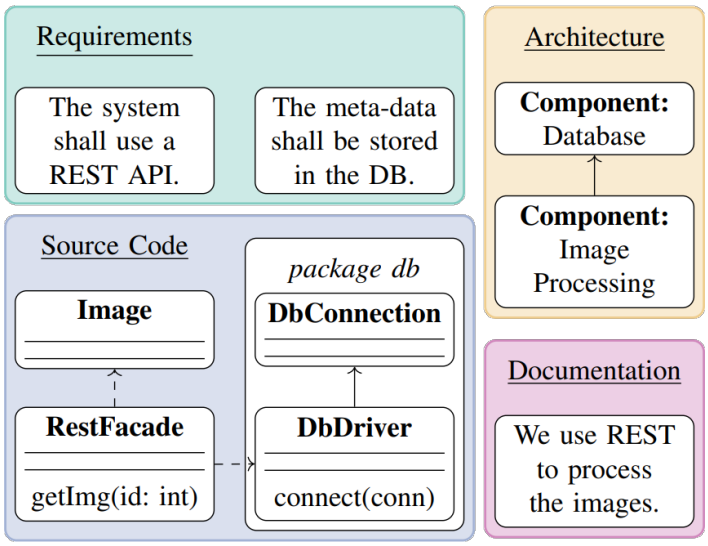

Figure 1.1: Overview of different artifacts during software development by Fuchß et al. [5]

2 Foundations

In order to understand the application of automated prompt engineering for trace link recovery, some key concepts are quite important. They will be explained in the following subsections.

2.1 Definition of Trace Link Recovery

Traceability is the ability for something to be traced. Meaning, there is evidence of some past occurrence. The task of trace link recovery (TLR) in software engineering is to find instances of the same element across different artifacts and link them for further processing. These traceability links help with tracking relationships between for example: code, requirements, diagrams, documentation, ...

This becomes especially important when inconsistencies are introduced into the project's artifacts.

As shown by Wohlrab et al. [19] inconsistency in wording and language is quite common during different stages of development. For example, naming conventions for architectural components may not be followed during implementation. They find the impact of these inconsistencies to be quite insignificant. However, for trace link recovery, this means that simple string comparisons by name are insufficient.

2.2 Definition of Automated Prompt Engineering

Prompt engineering is the process of refining a prompt for the specific use case. This is usually done in a non-systematic manual way. Automated Prompt Engineering (APE) enables Large Language Models to refine the initial prompt in order to optimize its performance [22].

Typically, automatic prompt engineering processes are iterated to improve previous results further[15, 22, 23, 14]. The performance of prompts can be measured using a set of training data, which includes input/output pairs.

The initial prompt will be the origin for the refinement process. It needs to be selected firstly before any optimization can occur. This prompt can be chosen manually. This approach

is very intuitive, however manual human interaction is still required to design this initial prompt.

Zhou et al. [23] propose using the LLM to generate the initial prompt. They prompt the model to generate a likely set of instructions, also called candidates, that will achieve good results with a batch of input/output pairs.

2.3 Automatic Prompt Optimization Using Gradient Descent

Based on the work of Pryzant et al. [15] I will implement a more sophisticated prompt optimization algorithm into the LiSSA framework. They propose the Prompt Optimization with Textual Gradients (ProTeGi) algorithm. This entire section is based on their work.

2.3.1 Initial Prompt

The ProTeGi algorithm takes an initial prompt p_0 and training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ consisting of input and output. They «assume access to a black box LLM API [...] which returns a likely text continuation y of the prompt formed by concatenating p and x» [15, sec. 2]. They then iteratively optimize the initial prompt p_0 to produce an approximation of the most optimized prompt for the given task. In order to optimize the prompt, a function is required, to compute deviance between the actual output y and expected output y_i as a numeric value.

2.3.2 Evaluation and Expansion

To evaluate the output of the current prompt p_i, they use a loss signal prompt ∇. In addition to the prompt errors that were not correctly classified when testing p_i are also provided. The result summarizes the flaws of p_i in natural language. This summary is called the textual gradient g.

The second prompt δ they use, is required to adjust p_i in the opposite direction of g to minimize the loss signal. They differ from other implementations by generating multiple gradients and refinements that might improve p_i and select good candidates to optimize further in the next iterative step.

In addition, they broaden their candidates further by paraphrasing them into semantically similar prompts, which are worded different. Generating new prompts with g and broadening them is considered candidate expansion.

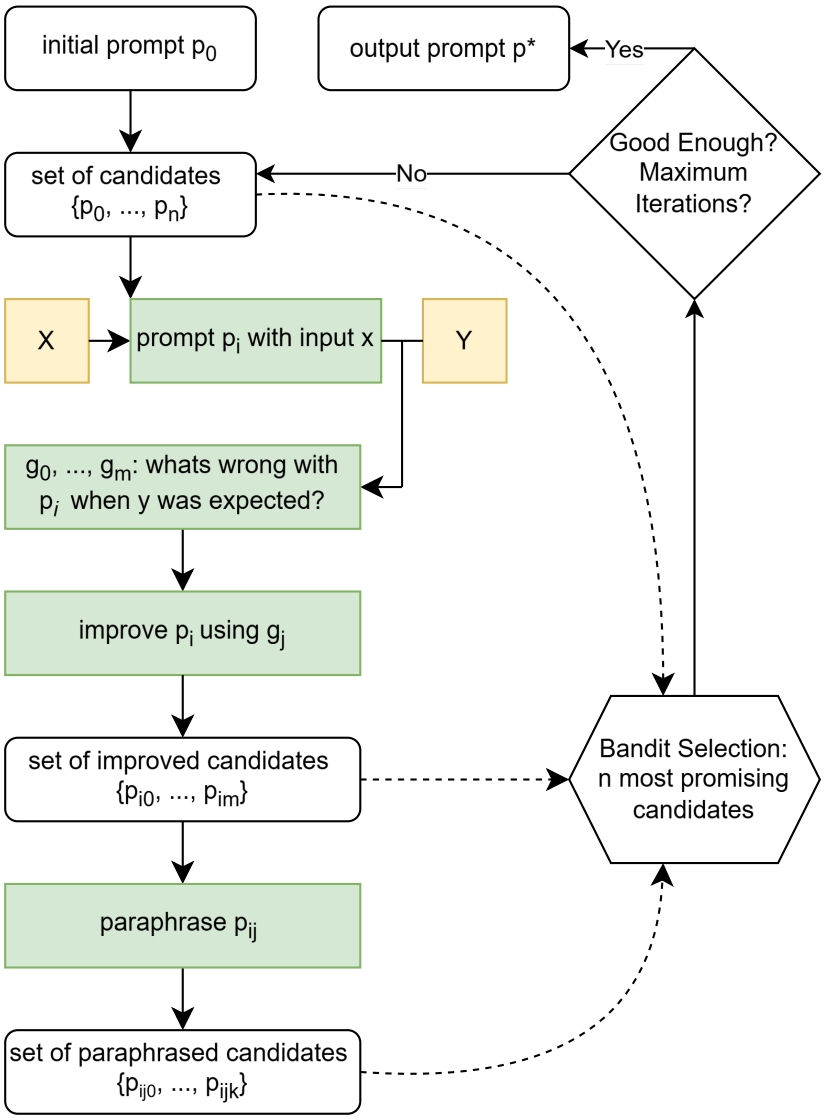

Figure 2.1: Overview of the iterative optimization loop in [15]
TODO:

- add legend
- symbolise core loop

2.3.3 Candidate Selection

In order to select candidates for the next iteration, they apply a beam search algorithm. Beam search is a heuristic best first graph search algorithm to select a fixed number of promising paths. The remaining paths will be discarded to allocate resources on more promising paths instead. [2]

The selection of the most promising candidates is quite expensive on the training set. As the problem is quite similar to the best arm identification in bandit optimization by Audibert and Bubeck [1], they rely on this well studied problem instead.

The problem consists of N stochastically independent probability distributions $\{D_1, \ldots, D_N\}$ with corresponding expected values and variances. These distributions are unknown to the user initially. The goal is to maximize the sum of rewards for each pull from the set of probability distributions, utilizing the knowledge gained through previous pulls. [12]

It is an analogy to multiple one-armed bandit slot machines. Pulling a lever on one of these slot machines does not affect the others but costs noticeable resources. The optimization aims to find the best performing arms with as few pulls as possible.

Pryzant et al. have used the successive rejects algorithm by Audibert and Bubeck [1] to select the best prompt candidates for the next iteration step.

The set of current candidates S is therefore initialized with all k candidate prompts in the iteration step. Depending on the budget for sampling candidates and remaining candidates in S will be prompted with n_k pairs of the training data. The results are evaluated with a metric function m to quantify the performance. The lowest scoring prompt is discarded from S and sampling repeated with the remainder until the set S only contains the desired amount of candidates.

The remaining candidates will be used as initial prompts for the next expansion step.

TODO: Explain successive rejects

Algorithm 1 «$Select(\cdot)$ with Successive Rejects» [15, Algorithm 4, p. 4]

1: Initialize: $S_0 \leftarrow \{p_1, \ldots, p_n\}$
2: **for** $k = 1, \ldots, n-1$ **do**
3: Sample $D_{sample} \subset D_{tr}, |D_{sample}| = n_k$
4: Evaluate $p_i \in S_{k-1}$ with $m(p_i, D_{sample})$
5: $S_k \leftarrow S_{k-1}$, excluding the prompt with the lowest score from the previous step
6: **end for**
7: **return** Best prompt $p^* \in S_{n-1}$

3 Related Work

Trace link recovery through large language models is an active field of work to apply rapid advancements in LLM performance.

3.1 Trace Link Recovery

Keim et al. [10] have first proposed cross domain trace link recovery using agent like algorithms. *Todo: Maybe read the paper?*

In previous work by Fuchß et al. [5] simple handwritten prompts by Ewald [3] were used to recover trace links between software documentation and architectural diagrams. They have proven, that they can outperform other state-of-the-art for code tasks.

Hey et al. [8] have adapted these prompts into trace link recovery in the domain of requirements to requirements. While they were also able to outperform state-of-the-art approaches, the recall rate for larger data sets still has room for improvement.

Rodriguez, Dearstyne, and Cleland-Huang [17] have also evaluated LLM usage for trace link recovery. Their main takeaway was, that even minor adjustments, «such as pluralizing words,interchanging prepositions,or reordering phrases» [17, sec. VI] lead to major differences in the outcome. They were unable to find a singular generalized optimal prompt to cover all TLR tasks. They manually adjusted prompts for each single task instead to greatly improve recovery rates.

3.2 Automatic Prompt Engineering

Many prompt optimization *todo find fitting word*: *papers* require an initial prompt to start the refinement process by using training data consisting of input / output pairs [16].

The Automatic Promot Engineer by Zhou et al. [23] can generate prompts for tasks which are specified only by input / output pairs. This eliminates the need for the initial prompt to seed the optimization process.

Self-Refine by Madaan et al. [14] uses feedback from the same large language model that generated the prompt, to improve the prompt further. This imitates human behavior, when initial drafts are adjusted rapidly.

ProTeGi by Pryzant et al. [15] utilizes a gradient descent algorithm to find the minimal deviance between an optimized prompt and the expected outputs. More details about their work can be found in 2.3.

Yang et al. [20] have taken a slightly different approach in their OPRO optimizer. Instead of adjusting the current iteration of prompts to steer them in an improved direction like Pryzant et al., they generate new independent prompts instead.

In order to reduce uncertainty and improve reproducibility, the Declarative Self-Improving Python (DSPy) Framework by Khattab et al. [11] proposes a composite like structure in python to program prompts. They are also generated and refined the prompts in a pipeline like structure, not unlike other LLM based prompt optimization algorithms.

3.3 Training Data

Trace Link Recovery tasks can be applied across many domains. Often authors will focus on one or few domains instead of attempting to cover everything. Domain training data includes the artifacts and ideally also a gold standard to compare results to.

Software architecture documentation to software architecture models for BigBlueButton, MediaStore, Teammates, Teastore are publicly available by Fuchß et al. [4] on GitHub [1].

For the domain of requirements to requirements, which my work will also focus on, a compilation of training data can be found in the replication package[7] for recent work of Hey et al.

[1] https://github.com/ArDoCo/Benchmark

4 Approach

In this chapter, I will outline how my thesis will be realized and provide source code that can be merged into the LiSSA project.

4.1 Simple Tree-of-Thought Classifier

The LiSSA framework is publicly accessible in a repository using the MIT license. Existing classifiers can be used as an inspiration for development.

The class `classifier` is an abstraction of all classifiers. It provides basic functionality to include further classifiers. Integration for different large language model (LLM) providers, such as Chat-GPT, Ollama, and Blablador, already exists in the framework. The new classifier can be plugged in directly with little more effort than writing the request prompt and implementing the `classify` function to extract the classification result out of the LLM output.

tree-of-thought (ToT) is a prompting technique exploring more different branches of thoughts than typical chain-of-thought (CoT) prompts. As illustrated in Figure 4.1

The prompts by Hulbert [9] aim to simplify ToT prompting into a singular request, instead of chaining multiple calls. I will use one of their prompts. The benchmark data to compare ToT prompting with the existing zero-shot and CoT classifiers also exists in a public repository [4].

This work package has two major goals. Foremost, to get confident and set up with the general project structure and benchmark data. This will be a prerequisite for all future work in the LiSSA framework. Second, this will broaden the baseline to compare later results and possible improvements against.

4.2 Naive Iterative Optimizing Classifier

How: Another quick classifier implementation will be the simplest automatic iterative prompt optimization algorithm. The same existing abstract `classifier` class can be used as in 5.1.1 to provide basic functionality for the iteratively optimized classifier.

Next, a function f will be required to quantify the result of the current prompt p iteration, called to a large language model llm. The simple approach here is to use the successful

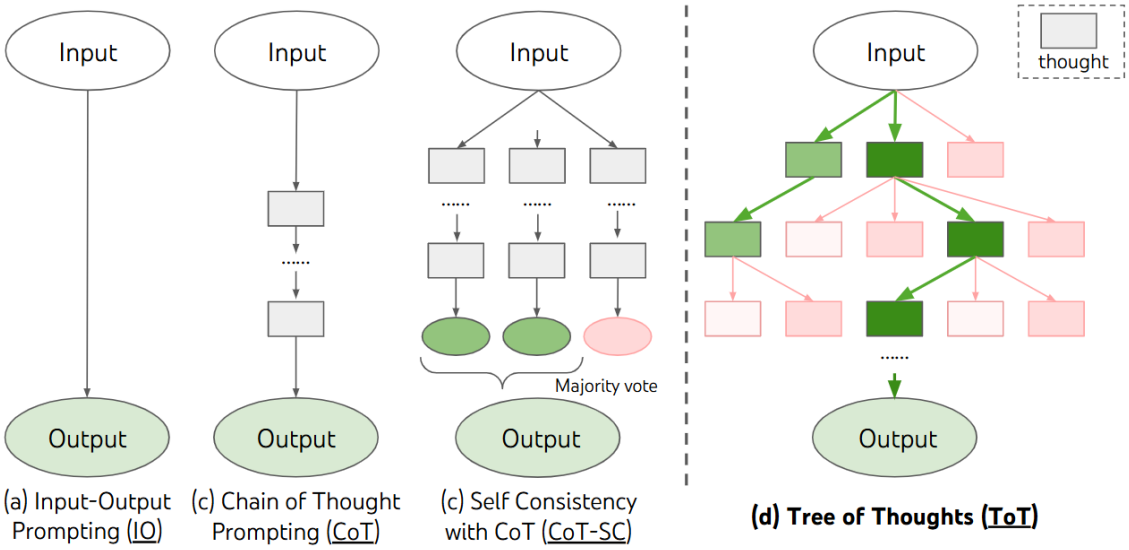

Figure 4.1: Tree of Thought visualization by Yao et al. [21]

retrieval rate, by dividing the amount of correctly classified trace links by the total amount of recoverable trace links. The function f will need to have the same range for any set of training data to ensure the threshold value t will always be viable.

$$f : llm(p) \rightarrow [0, 1]$$
$$t \in [0, 1]$$

The benchmark data by Fuchß et al. [4] also provides the gold standard for each given problem. The total amount of recoverable trace links thus is already available. *to-do: how can the task set be divided into training and testing data? Use one entire TLR problem for training, the remaining for testing?*

To improve the performance, an optimization prompt is used. The naive approach is to simply ask the LLM to improve the prompt without providing further direction or information about why the prompt did not meet the requirements yet. The hopefully improved prompt will be taken into the next iteration. The act of evaluation and improving will be repeated until a threshold value is passed, meaning the prompt is considered good enough. To limit resource usage, a hard limit will also be included to put an upper bound on the optimization attempts.

Why: Implementing this naive iteratively optimized classifier will yield fast results to evaluate later, even if more complicated implementations might fail. The core concepts also apply for 5.1.3 and can be reused.

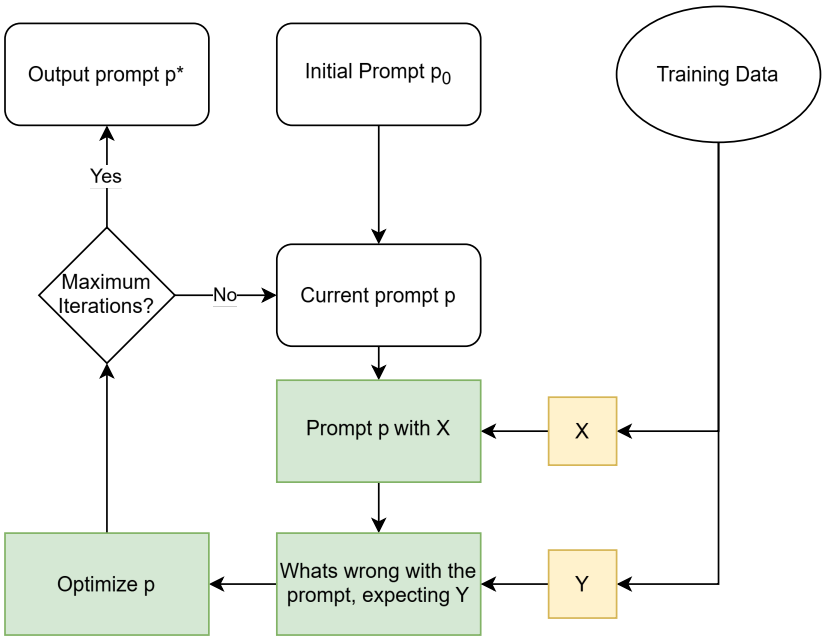

Figure 4.2: Visualization of a simple iterative optimization algorithm

4.3 Automatic Prompt Optimization with Gradient Descent

How: The authors have provided their python source code in a publicly accessible repository[1] under the MIT licensing. The project code can be of great help when adapting their algorithm to the LiSSA java framework.

The existing iterative optimization implementation from 5.1.2 can be used as foundations for the implementation, as the core concepts remain the same. Refer to 2.3 for details of the gradient descent based optimization algorithm. The actual implementation of gradient descent will be the main focus here, in order to steer optimized prompts in the right direction.

Why: The primary idea of this thesis is to explore automated optimization. The naive approach may yield good results already. Feeding more information and processing logic into the optimization system is expected to present better results. Especially suitable results may be reached faster by doing more processing on the machine instead of expensive API calls to the large language model.

4.4 Evaluation

How: Precision and recall are key measures for information retrieval tasks [6].

[1] https://github.com/microsoft/LMOps/tree/main/prompt_optimization

TODO: Explain these metricize and how they are calculated instead of direct quoting and format formulas «Precision measures the ratio of correctly predicted links and all predicted links. Recall measures the ratio of correctly predicted links and all relevant documents. It shows an approach's ability to provide correct trace links. The F1 score is the harmonic mean of precision and recall. The Metrics are calculated in the following way, where TP is true positives, FP is false positives, and FN is false negatives.» [3, sec. 7.1]

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$f_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Depending on how long each phase will take, it is possible to generate a bunch of different data and perform comparisons. A simple but interesting approach is to compare different LLMs for the task. Many variations can be achieved by comparing for example how a prompt optimized by one system performs on the others. We can also compare how well each system manages to optimize the initial prompt. Another interesting thought is to take optimized prompts from a different system as the initial prompt.

Why: Evaluating the experimental results of prompt optimization is key to making them accessible and comparable.

5 Work Plan

In this chapter, I will elaborate on my plan to realize the thesis topic and mitigate risks. This includes the individual work packages outlined in section 5.1. Different artifacts that will be produced through the thesis are mentioned in section 5.2. Expected risks during the project and their mitigation will be addressed in section 5.3. Lastly, a giant chart will be provided to visualize the temporal planning for my thesis project in section 5.4.

5.1 Phases

My work will be split into four distinct phases, with only a few dependencies in between. Each of the first three implementation packages can be evaluated separately. However, each phase provides building blocks used by the next with increasing complexity. Even though Evaluation and Buffer is placed as the last phase, some progress will already be made after the first implementation in subsection 5.1.1 is finished.

5.1.1 Initial Overview

To become familiar with the LiSSA framework [5], I will implement another basic classifier with a singular request variation of the simple prompting technique ToT [13] explained in section 4.1. Currently, a zero-shot and CoT-style reasoning classifier is implemented. However, as Long has shown, ToT-style prompts can improve performance compared to CoT approaches in solving logic puzzles. They have tested their approach to ToT prompting with different-sized Sudoku puzzles and compared their results against zero-shot prompting as well as CoT styled one- and few-shot prompting.

5.1.2 Naive iterative Optimization

Next, I plan to implement a naive iterative approach to prompt optimization. Many automatic prompt optimization algorithms [15, 22, 23, 14] depend on an iterative core loop (see Figure 4.2). This loop will be repeated until the optimized prompt performs sufficiently good or a maximum number of iterations has been reached. My plan to realize this is explained in section 4.2.

5.1.3 Automatic Prompt Optimization Based on Gradient Descent

The major implementation of this thesis will be an adaptation of the gradient-descent-based automatic prompt optimization (APO) algorithm by Pryzant et al. [15]. Refer to section 4.3 for details about the implementation. The APO algorithm will be trained and tested on the requirements-to-requirements benchmark data by Hey et al. It is available in their replication package [7]. The optimized prompt will be compared against previous prompts used by Fuchß et al. and Hey et al. in the greater LiSSA framework.

5.1.4 Evaluation and Buffer

As this work can be seen as an expansion on the recent work of Hey et al. [8], the same metrics will be used to evaluate the different prompts used and optimized in this work. These are the precision, recall, F_1-Score, and F_2-Score. This enables an easy comparison, especially with the manual prompts designed by Ewald [3] included in the work of Hey et al.

For further evaluation, aside from the performance of the optimized prompt, different LLMs will be used, as explained in section 4.4.

5.2 Artifacts

This work will yield code to be merged into the LiSSA repository[1]. The implementation of 5.1.2 and 5.1.3 will provide a base to implement further automatic prompt optimization techniques with similar building blocks.

A written report will be created to document, summarize, and evaluate the results of my (automatic) prompt optimization work to retrieve trace links.

5.3 Risk Management

The moajor risk of my work would be failing to implement the automatic prompt optimization in 5.1.3. By choosing an optimization algorithm that has Python ode publicly available, this risk is already greatly reduced. Furthermore, all three work packages with implementation content will yield a classifier that can be tested and evaluated independently against the existing classifiers in the LiSSA Framework. Thus, early working examples will be available throughout the thesis.

It is hard to accurately estimate the required time for each phase. Frequent buffer slots are planned in order to dynamically allocate more time if needed.

[1] https://github.com/ArDoCo/LiSSA-RATLR/

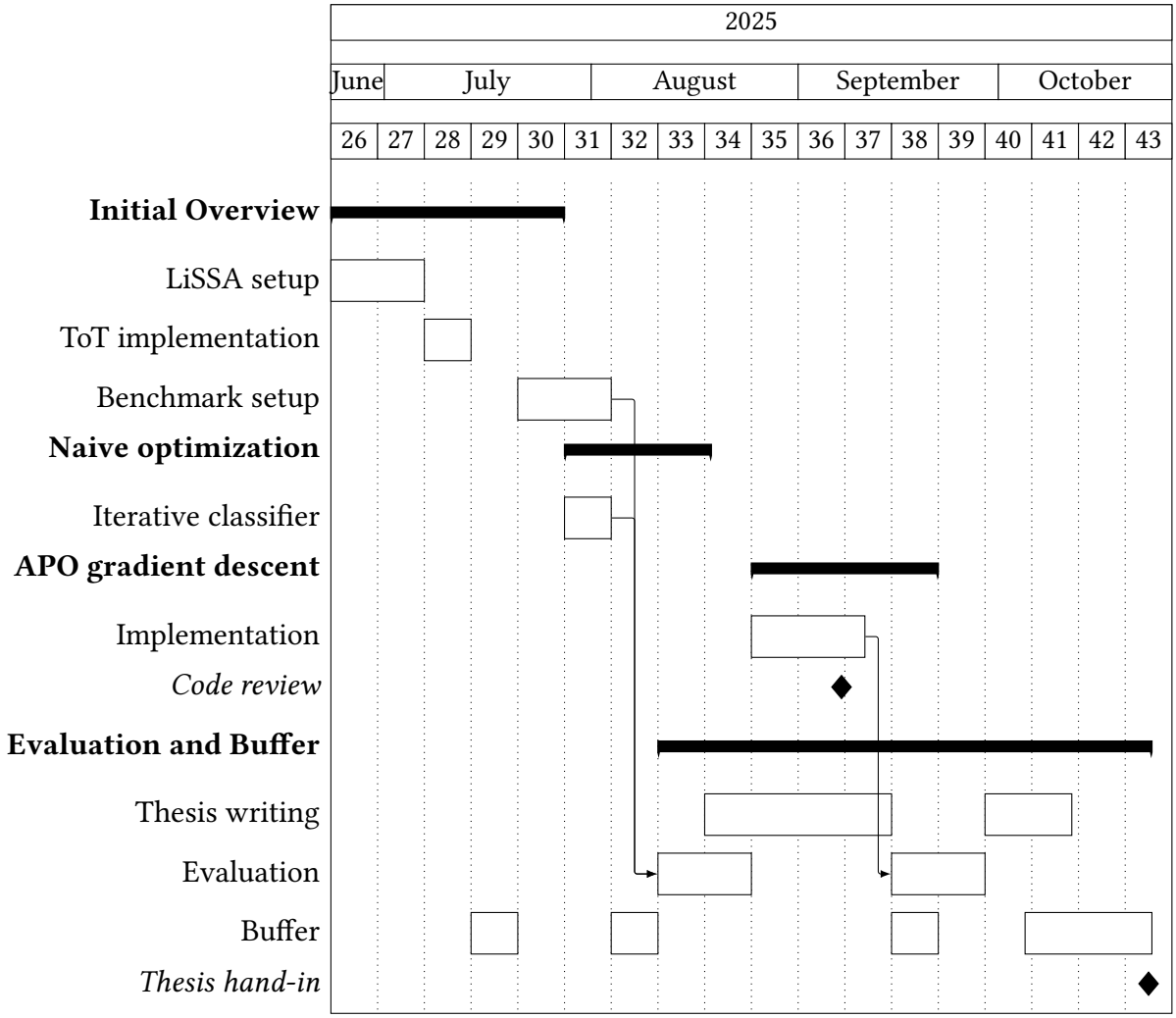

Figure 5.1: Gantt chart for thesis plan

5.4 Schedule

The Gantt chart in section 5.4 illustrates how I plan to allocate my time. The early months will probably take up less time than planned, which can be used to reduce congestion in later phases.

Bibliography

[1] Jean-Yves Audibert and Sébastien Bubeck. “Best Arm Identification in Multi-Armed Bandits”. In: COLT - 23th Conference on Learning Theory - 2010. June 27, 2010, 13 p. URL: https://enpc.hal.science/hal-00654404 (visited on 05/20/2025).

[2] *Beam Search from FOLDOC*. URL: https://foldoc.org/beam+search (visited on 06/12/2025).

[3] Niklas Ewald. *Retrieval-Augmented Large Language Models for Traceability Link Recovery*. 2024. DOI: 10.5445/IR/1000178218. URL: https://publikationen.bibliothek.kit.edu/1000178218 (visited on 05/20/2025).

[4] Dominik Fuchß et al. *ArDoCo/Benchmark*. Version ecsa22-msr4sa. Zenodo, Aug. 5, 2022. DOI: 10.5281/zenodo.6966832. URL: https://zenodo.org/records/6966832 (visited on 05/20/2025).

[5] 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)*. International Conference on Software Engineering (ICSE 2025), Ottawa, Kanada, 27.04.2025 – 03.05.2025. 2025, p. 723. DOI: 10.1109/ICSE55347.2025.00186. URL: https://publikationen.bibliothek.kit.edu/1000179816 (visited on 05/20/2025).

[6] 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. URL: https://ieeexplore.ieee.org/abstract/document/1583599 (visited on 05/26/2025).

[7] Tobias Hey et al. *Replication Package for "Requirements Traceability Link Recovery via Retrieval-Augmented Generation"*. Zenodo, Jan. 31, 2025. DOI: 10.5281/zenodo.14779458. URL: https://zenodo.org/records/14779458 (visited on 06/15/2025).

[8] Tobias Hey et al. “Requirements Traceability Link Recovery via Retrieval-Augmented Generation”. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Anne Hess and Angelo Susi. Cham: Springer Nature Switzerland, 2025, pp. 381–397. ISBN: 978-3-031-88531-0. DOI: 10.1007/978-3-031-88531-0_27.

[9] Dave Hulbert. *Using Tree-of-Thought Prompting to Boost ChatGPT’s Reasoning*. Version 0.1. May 2023. URL: https://github.com/dave1010/tree-of-thought-prompting (visited on 06/02/2025).

[10] Jan Keim et al. “Trace Link Recovery for Software Architecture Documentation”. In: *Software Architecture*. Ed. by Stefan Biffl et al. Cham: Springer International Publishing, 2021, pp. 101–116. ISBN: 978-3-030-86044-8. DOI: 10.1007/978-3-030-86044-8_7.

[11] Omar Khattab et al. "DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines". In: The Twelfth International Conference on Learning Representations. Oct. 13, 2023. URL: https://openreview.net/forum?id=sY5N0zY5Od (visited on 06/15/2025).

[12] Volodymyr Kuleshov and Doina Precup. *Algorithms for Multi-Armed Bandit Problems*. Feb. 25, 2014. DOI: 10.48550/arXiv.1402.6028. arXiv: 1402.6028 [cs]. URL: http://arxiv.org/abs/1402.6028 (visited on 06/12/2025). Pre-published.

[13] Jieyi Long. *Large Language Model Guided Tree-of-Thought*. May 15, 2023. DOI: 10.48550/arXiv.2305.08291. arXiv: 2305.08291 [cs]. URL: http://arxiv.org/abs/2305.08291 (visited on 06/02/2025). Pre-published.

[14] Aman Madaan et al. "Self-Refine: Iterative Refinement with Self-Feedback". In: *Advances in Neural Information Processing Systems* 36 (Dec. 15, 2023), pp. 46534–46594. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f Abstract-Conference.html (visited on 05/20/2025).

[15] Reid Pryzant et al. "Automatic Prompt Optimization with "Gradient Descent" and Beam Search". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2023. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7957–7968. DOI: 10.18653/v1/2023.emnlp-main.494. URL: https://aclanthology.org/2023.emnlp-main.494/ (visited on 06/02/2025).

[16] Kiran Ramnath et al. *A Systematic Survey of Automatic Prompt Optimization Techniques*. Apr. 2, 2025. DOI: 10.48550/arXiv.2502.16923. arXiv: 2502.16923 [cs]. URL: http://arxiv.org/abs/2502.16923 (visited on 06/14/2025). Pre-published.

[17] 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)*. 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). Sept. 2023, pp. 455–464. DOI: 10.1109/REW57809.2023.00087. URL: https://ieeexplore.ieee.org/abstract/document/10260721 (visited on 05/13/2025).

[18] Parshin Shojaee et al. *The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity*. June 1, 2025. DOI: 10.48550/arXiv.2506.06941. URL: https://ui.adsabs.harvard.edu/abs/2025arXiv250606941S (visited on 06/15/2025). Pre-published.

[19] Rebekka Wohlrab et al. "Improving the Consistency and Usefulness of Architecture Descriptions: Guidelines for Architects". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. 2019 IEEE International Conference on Software Architecture (ICSA). Mar. 2019, pp. 151–160. DOI: 10.1109/ICSA.2019.00024. URL: https://ieeexplore.ieee.org/abstract/document/8703919 (visited on 05/26/2025).

[20] Chengrun Yang et al. *Large Language Models as Optimizers*. Apr. 15, 2024. DOI: 10.48550/arXiv.2309.03409. arXiv: 2309.03409 [cs]. URL: http://arxiv.org/abs/2309.03409 (visited on 05/20/2025). Pre-published.

[21] Shunyu Yao et al. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. Dec. 3, 2023. DOI: 10.48550/arXiv.2305.10601. arXiv: 2305.10601 [cs]. URL: http://arxiv.org/abs/2305.10601 (visited on 06/17/2025). Pre-published.

[22] Mohammad Amin Zadenoori et al. "Automatic Prompt Engineering: The Case of Requirements Classification". In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Anne Hess and Angelo Susi. Cham: Springer Nature Switzerland, 2025, pp. 217–225. ISBN: 978-3-031-88531-0. DOI: 10.1007/978-3-031-88531-0_15.

[23] Yongchao Zhou et al. *Large Language Models Are Human-Level Prompt Engineers*. Mar. 10, 2023. DOI: 10.48550/arXiv.2211.01910. arXiv: 2211.01910 [cs]. URL: http://arxiv.org/abs/2211.01910 (visited on 05/13/2025). Pre-published.