

Automated Extraction and Analysis of Developer's Rationale in Open Source Software

MOUNA DHAOUADI, Université de Montréal, Canada

BENTLEY OAKES, Polytechnique Montréal, Canada

MICHALIS FAMELIS, Université de Montréal, Canada

Contributors to open source software must deeply understand a project's history to make coherent decisions which do not conflict with past reasoning. However, inspecting all related changes to a proposed contribution requires intensive manual effort, and previous research has not yet produced an automated mechanism to expose and analyze these conflicts. In this article, we propose such an automated approach for rationale analyses, based on an instantiation of Kantara, an existing high-level rationale extraction and management architecture. Our implementation leverages pre-trained models and Large Language Models, and includes structure-based mechanisms to detect reasoning conflicts and problems which could cause design erosion in a project over time. We show the feasibility of our extraction and analysis approach using the OOM-Killer module of the Linux Kernel project, and investigate the approach's generalization to five other highly active open source projects. The results confirm that our automated approach can support rationale analyses with reasonable performance, by finding interesting relationships and to detect potential conflicts and reasoning problems. We also show the effectiveness of the automated extraction of decision and rationale sentences and the prospects for generalizing this to other open source projects. This automated approach could therefore be used by open source software developers to proactively address hidden issues and to ensure that new changes do not conflict with past decisions.

CCS Concepts: • **Software and its engineering** → **Documentation; Maintaining software.**

Additional Key Words and Phrases: developer rationale, open source software, software maintenance, commit messages, rationale extraction

ACM Reference Format:

Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2025. Automated Extraction and Analysis of Developer's Rationale in Open Source Software. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE113 (July 2025), 23 pages. <https://doi.org/10.1145/3729383>

1 Introduction

One of the most valuable resources in the history of a software project is the rationale of developers, as it is a rich repository of knowledge and experience [4]. In collaborative and distributed development projects, commit messages are usually a main reservoir of such information for future developers, offering insights into the code alterations and the rationale behind them during everyday operations [8]. Such natural-text justifications in commit messages are particularly important for long-term open source projects, like the Linux kernel, because design information is subject to erosion [56] and evaporation [39] over time as contributors enter and leave the project.

Authors' Contact Information: Mouna Dhaouadi, Université de Montréal, DIRO, Montréal, Canada, mouna.dhaouadi@umontreal.ca; Bentley Oakes, Polytechnique Montréal, GIGL, Montréal, Canada, bentley.oakes@polymtl.ca; Michalis Famelis, Université de Montréal, DIRO, Montréal, Canada, famelis@iro.umontreal.ca.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2994-970X/2025/7-ARTFSE113

<https://doi.org/10.1145/3729383>

In fact, contributing to Open Source Software (OSS) often entails *manually* examining relevant commit messages and understanding the logic behind them to ensure that any proposed changes do not cause conflicts with previous decisions. However, going through the previous related design decisions is tedious, time-consuming and error-prone [7].

Although researchers have proposed frameworks to automatically extract and manage rationale information [20, 28, 31], to the best of our knowledge there is no prior work that has implemented *an automated mechanism to detect reasoning conflicts*. In this paper, we propose to fill this gap by proposing **an automated approach to analyze rationale information from commit messages**. Specifically, we expand on our previous work on Kantara, an automated and intelligent framework for rationale extraction and management [7]. Kantara is a high-level architecture that extracts decision and rationale information from text documents and structures them in a knowledge graph representation that can be leveraged for meaningful rationale analyses, such as detecting interesting relationships and reasoning about conflicts and inconsistencies.

In this work, we follow the definition of *rationale information* as introduced in our previous work [7], based on the seminal work by Burge et al. [4]: “sets of decision-rationale pairs”. For *reasoning conflicts* between decisions, we use the definition by Kruchten et al. [21]: “conflicting decisions are decisions that are mutually exclusive”. We show several examples of decision-rationale pairs and conflicting decisions in Tables 4 and 12. Furthermore, we also focus on the presence of rationale in the commit structure through time. Although further work is needed to further nuance the types and levels of rationale, we believe this is an essential first step.

Here, we describe a purpose-driven instance of Kantara based on state-of-the-art advances in Artificial Intelligence (AI) and Natural Language Processing . Assuming rationale is present in commits, our main research questions concern the *feasibility* (RQ.1) and *generalizability* (RQ.2) of an automated approach to support rationale analyses. Specifically, we utilize Semantic Role Labelling (SRL), pretrained models, and Large Language Models (LLMs) for decision rationale extraction, and pretrained models to detect reasoning conflicts. We do not present a technical contribution in AI, only an application of AI techniques to Software Engineering (AI4SE). We discuss the automation potential, the generalization capability, and limitations of these techniques vis-a-vis the subjectivity of rationale. To summarize, this research falls within a bounded scope: a) we assume the presence of rationale in commit messages, b) we focus on only the presence of rationale information, and c) we apply AI techniques without optimization. While the scope is constrained to manage complexity, the findings could be relevant to other contexts and to future research about rationale nuances or AI contributions for commit message rationale analysis.

We use the commit messages of the Out Of Memory (OOM)-Killer module of the Linux kernel project [9] to demonstrate the approach’s *feasibility* (RQ.1) , and its *generalization* (RQ.2) to two other Linux modules and five other OSS projects. This paper thus makes two key contributions: 1) a methodological contribution with **an approach to automatically extract, structure, and contextualize decisions and their rationale embedded in OSS commit messages**, and 2) a **knowledge contribution in the form of the rationale behind decisions** from these projects.

The rest of this paper is organized as follows. We present the Kantara architecture [7] and the OOM-Killer dataset [9] in Section 2. Then, we give an overview of the methodology and the evaluation in Section 3. We present the knowledge graph construction and the rationale analyses, and discuss the implementation performance in Section 4. We work on automating the extraction of the decision and rationale sentences, and investigate the generalization of our approach in Section 5. We discuss the implications of this work and the threats to validity in Section 6. Finally, we overview related research in Section 7 and conclude in Section 8.

2 Background

Kantara architecture. We introduced Kantara in 2022 [7], defining a high-level architecture that includes an *information inference* component to structure and extract rationale information, and an *analysis interface* to exploit and understand the extracted information. The architecture is sketched in the top part of Fig. 1 and has three key components:

- (1) The *information inference* component takes as input sentences that contain *both* decisions and the logic behind them, and outputs a knowledge graph of *decision* and *rationale* nodes, called the *Rationale and Decision Graph*.
- (2) The second component is a knowledge graph, named the *Rationale and Decision Graph*. In it, each *decision* node has a corresponding *rationale* node. These decision-rationale pairs make up the *rationale information* in the input text. Each pair corresponds to an input sentence that contains both a decision and the rationale behind it. The *decisions* are interconnected using relationships (e.g., *Similar* and *Contradicts*).
- (3) In the third component, the *analysis interface*, these relationships can be used for different analyses, such as the detection of inconsistencies and conflicts.

In our previous work, we only presented initial ideas about how to build the Kantara *information inference* component and did some preliminary evaluation on a small example of commits [7]. This lack of a concrete and fully automated implementation is addressed in the current article.

OOM Killer Dataset. The Linux Kernel project is a long-lived OSS project that involves many collaborators. As Linux culture explicitly promotes the capturing of rationale in commit messages¹, these messages usually have five to ten sentences providing an excellent explanation of decisions and their rationale [9]. The Linux project has several sub-projects, organized as sub-folders that focus on different areas of the operating system. For example, the 'mm' folder contains code that focuses on memory management, the 'fs' folder code for filesystems, etc. As these sub-projects have different concerns and committers, we treat each one as their own project in this article.

The OOM-Killer kernel component is invoked when the machine is running out of memory, and is responsible for selecting a task and killing it to free the task's memory². In another previous work, we constructed a dataset by manually (multi-) labelling 2234 sentences from 404 commit messages of the OOM Killer component with *Decision*, *Rationale* and *Supporting Facts* tags [9]. Here, we utilize this dataset as a manually extracted source of rationale to a) investigate how our implementation of Kantara can find conflicting decisions and reasoning problems, and b) as a training set for our extraction techniques.

In the dataset, the *Supporting Facts* tag was used to label sentences describing information about the state of system that was current when the commit was contributed. Thus, it captured a degree of temporal dependency between commits, as each was applied to the system at a particular time. In our work, we intentionally exclude the *Supporting Facts* category to remove temporal dependencies from our analysis. These dependencies could introduce bias related to the sequence of commit messages or system states, potentially affecting our evaluation.

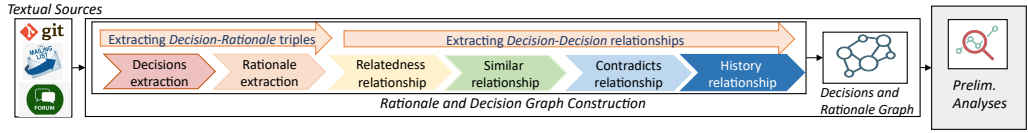
3 Methodology and Evaluation Overview

This section presents the methodology followed in this article, summarized in the bottom part of Fig 1, along with our evaluation plan for our research questions. We lay out our contribution across four components. In the first, we instantiate Kantara to build the decision and rationale graph used for rationale analysis. In the second component, we perform the rationale analyses themselves. In

¹<https://docs.kernel.org/process/submitting-patches.html#describe-your-changes>

²https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/mm/oom_kill.c

Our previous work: High-level Kantara architecture; Preliminary evaluation



This work: Concrete implementation of Kantara; Investigation of generalization

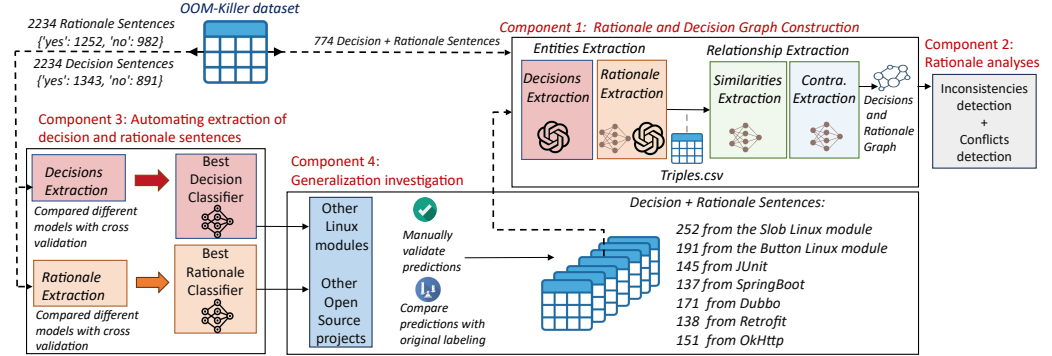


Fig. 1. Methodology overview, detailing our contributions to and evaluation of the Kantara approach.

the third component, we investigate whether the procedure can be effectively automated. These first three components allow us to answer different sub-questions of *RQ.1 (feasibility)*. In the fourth component we answer *RQ.2 (generalizability)* by applying the automated approach to previously unseen data. In the rest of this section, we describe the four components, as well as the breakdown of *RQ.1* in sub-questions, and the metrics we used to answer *RQ.1* and *RQ.2*. Finally, we summarize our correctness validation process.

Component 1: Rationale and Decision Graph Construction. In Section 4.1, we present and illustrate a concrete implementation of the Kantara architecture. Specifically, we construct the Decision and Rationale Graph that corresponds to the 774 sentences labelled as both *Decision* and *Rationale* in the OOM-Killer commit messages dataset. Graph construction has two steps: extracting the entities (*decision* and *rationale* nodes), and detecting the relationships (*Similar* and *Contradicts* edges).

To evaluate the *feasibility (RQ.1)* of our approach, we report on its ability to find interesting relationships and to detect potential conflicts. Thus, we define the sub-question *RQ.1.1: “ability to find similar and contradictory decisions”*. Detecting such decisions is interesting because it allows developers to reuse existing solutions and avoid conflicts. We answer *RQ.1.1* by reporting the number of *similar* and *contradictory* decisions that our approach was able to detect.

Component 2: Rationale Analyses. We implement structure-based mechanisms that leverage the obtained knowledge graph for rationale analyses. First, we define a more elaborate mechanism for conflict detection, that goes beyond the simple detection of a one-to-one *Contradicts* relationship. Also, to answer *RQ.1*, we report on our approach’s ability to detect reasoning problems in the development process and define the sub-question *RQ.1.2: “ability to detect inconsistencies”*. We define scenarios of inconsistencies in Section 4.2. These consider *Similarity* and *Contradiction* relationships between *rationale* nodes and between the *decision* nodes. They could help developers detect errors in the graph construction and reasoning problems in the development process. We answer *RQ.1.2* by reporting the number of inconsistencies that our approach was able to find.

Components 1 & 2: Performance. If our approach does not have reasonable performance, it would not be *feasible* (RQ.1). In Section 4.3, we answer the sub-question RQ.1.3: “*performance*” by reporting the time needed by our approach on the corpus from the OOM-Killer module.

To apply our approach beyond the OOM-Killer, we must automatically extract decision-containing and rationale-containing sentences from the commit messages of other projects, as the Kantara architecture *only works on sentences that embed both decision and rationale information*. Thus it does not handle scenarios where decisions and rationales span separate or multiple sentences.

Component 3: Automating the extraction of decision and rationale sentences. In the OOM-Killer dataset, the extraction of the decision and rationale sentences was done with manual labelling. To automate this extraction, we train two binary classifiers on the OOM-Killer dataset: a decision-sentences classifier and a rationale-sentences classifier. In Section 5.1, we experiment with different ML models and choose the best performing ones on the OOM-Killer dataset using cross-validation. As our goal is only to show the feasibility and initial generalizability of a reconstruction pipeline, we use default hyperparameter values for the ML models. Expanding the investigation scope to include hyperparameter optimization is left for future work, though it is likely that the results would improve. To complete our answer to RQ.1 (*feasibility*), we report on RQ.1.4: “*the effectiveness of the automatic extraction of decision and rationale sentences*”. Specifically, we report the performance of the best classifiers on the OOM-Killer dataset using standard evaluation metrics (accuracy, precision, recall, and F1-score), and also compare their performance to the human performance.

Component 4: Investigation of generalization. In Section 5.2 we answer RQ.2: (*generalizability*). Specifically, we validate the generalizability of the classifier's predictions to the two other Linux modules by manually validating a sample of the predictions and computing the inter-rater agreement. We validate the generalizability of the classifiers on the five other OSS projects by comparing their predictions to the original labelling and reporting standard metrics (accuracy, precision, recall, and F1-score). Finally, we extract, structure, and contextualize rationale information from these other projects, and present found inconsistencies.

Correctness validation of the extracted rationale information. Throughout this work, we extract different rationale information using different techniques: 1) decision-rationale pairs using ChatGPT, 2) *similar* and *contradictory* relationships using pre-trained models and 3) *Decision* and *Rationale* sentences using classification. Here we summarize how we answer the RQs and validate the correctness of the extracted information.

First, we show the *feasibility* (RQ.1) of our approach using the OOM-Killer dataset. To answer RQ.1.1, we report the number of similar and contradictory decisions detected. To validate the correctness of the extracted information, we manually validate a random sample of the entities extracted by ChatGPT (Table 2) and of the detected relationships (Table 3). To answer RQ.1.2, we report the number of inconsistencies detected and manually validate a random sample (Table 4). To answer RQ.1.4, we report the performance of the best classifiers on the OOM-Killer dataset using standard evaluation metrics (Table 7). We also compare it to the human performance (Table 8).

Second, to answer RQ.2 (*generalizability*), we manually validate the correctness of a sample of the labelling produced by the best performing classifiers on the other Linux modules and report inter-rater agreement (Table 9). For the other OSS projects, the correctness of the classification is determined by a comparison with the original labelling (Table 10) using standard classification metrics. Finally, to investigate the performance of the application of our inconsistency detection mechanisms in other contexts, we report the number of inconsistencies detected (Table 11). We also manually investigate a sample of detected inconsistencies (Table 12).

4 Our Instantiation of Kantara

4.1 Rationale and Decision Graph Construction

In this section, we instantiate the Kantara architecture to extract, structure, and contextualize rationale and decision information, by storing it into a knowledge graph representation. We reuse our schema of the Rationale and Decision Graph [7]. It has two types of nodes: *decision* entities and *rationale* entities, and two types of edges. The first type links decisions to their rationale. The second allows capturing relationships between decisions. To populate the graph, we use the sentences that were both labelled *Decision* and *Rationale* in our OOM-Killer dataset. After removing two nonsense entries, we ended up with 774 sentences. The graph construction then requires two sub-tasks. The first is extracting *decision* and *rationale* entities from the sentences, and linking entities extracted from the same sentence. The second sub-task is detecting the relationships between the decision entities.

4.1.1 Entities Extraction. In the following, we investigate two different methods (SRL and LLMs) for entities extraction and report our results and observations. To the best of our knowledge, this is the first work that has investigated the usage of SRL and LLMs for rationale extraction.

Semantic Role Labeling (SRL). Semantic Role Labeling, is a technique that uses tagging to extract the grammatical structure of a sentence [36]. SRL defines various tags for picking out parts of a sentence. As a first step towards entity extraction, we focused only on rationale by using the SRL tags “Cause” (CAU), “Purpose” (PRP). Specifically, we investigated whether we can rely on these tags to detect the sentences that contain the *why* information and extract the chunks of the sentence that correspond to the rationale information. We experimented with two pre-trained models: the *structured-prediction-srl-bert* model which is a BERT-based model with some modifications [46] and the *structured-prediction-srl* model which is a reimplementaion of a deep BiLSTM sequence prediction model [48] from the *allennlp_models* library [16].

The *structured-prediction-srl-bert* model applied the CAU or the PRP labels to substructures of 214 sentences (resp. 184 for *structured-prediction-srl*). The models attained 28% and 24% accuracy respectively, by considering label presence as an indicator that the sentence contains rationale.

SRL also includes the tag “Manner” (MNR). As rationale is often expressed in terms of “how” a decision should impact the system, we also investigated whether the inclusion of the MNR tag to the detection might improve accuracy. Indeed, if the presence of any of the tags CAU, PRP, MNR is taken to indicate rationale is present, the *structured-prediction-srl-bert* model detects 317 sentences (41% accuracy), and the *structured-prediction-srl* model detects 300 sentences (39% accuracy).

These improvements in accuracy notwithstanding, the SLR-based approach misses more than half the rationale-containing sentences. This can be explained by the subjectivity of rationale in free text; it is not always captured in an explicit manner that is grammatically clear.

Since the *structured-prediction-srl-bert* model when considering the “MNR” tag provided the best results, we used it for finer-grained analysis. We report examples of missed sentences in Table 1. The table shows several limitations. For instance, in sentences 1-5, the model did not output “PRP”, “CAU” or “MNR” tags because the rationale in these sentences was not explicit. On the other hand, the sentences 6-8 were successfully detected as the rationale was more clearly written.

Given the limitations of the SLR-based approach for rationale extraction, we did not attempt to extract decision entities with it. Instead, we investigated the use of LLMs.

GPT-based models . We used gpt-3.5-turbo, the public version of ChatGPT by OpenAI [35]. The performance of ChatGPT, as with all LLMs, depends heavily on the prompts used. Thus, we iteratively improved the prompt until we got reasonable results. For this stage, we randomly chose 25 sentences from the 774 in the OOM-Killer dataset. In a first iteration, we used the following

Table 1. Examples of sentence classification showing whether rationale was identified (✓) or missed (X).

N ^o	Sentence	Rationale Found?
1	This is an attempt to reduce the ballast that doesn't provide any relevant information for those allocation failures investigation	X
2	Remove the oom_reaper from exit_mmap which will make the code easier to read	X
3	mm/oom_kill.c: fix vm_oom_kill_table[] ifdeffery	X
4	There is still a need to update locked_vm accounting according to the munmapped vmas when munmapping: do that in detach_vmas_to_be_unmapped()	X
5	exit_mmap() does not need locked_vm updates, so delete unlock_range()	X
6	I have only compile tested it (in my default config setup) and I am throwing it mostly to see what people think about it.	✓
7	Munlock code is no longer a problem since [3] and there shouldn't be any blocking operation before the memory is unmapped by exit_mmap so the oom reaper invocation can be dropped	✓
8	This patch adds a new oom_group_kill event when memory.oom.group triggers to allow userspace to cleanly identify when an entire cgroup is oom killed.	✓

prompt: "Which chunks of this sentence indicate Decision and which chunks indicate Rationale? Answer Decision: XX \n Rationale: YY. Sentence:" However, the model did not perform well and would rephrase and analyze the given sentence to explain the decision and its rationale. Sometimes, it would output the whole sentence as decision and as rationale. Therefore, we added the definitions of Decision and Rationale in the prompt as defined in the Codebook used to create the OOM-Killer dataset [9]. We also added the following sentence in the prompt: "XX and YY must be chunks from the sentence in question." to prevent the model from rephrasing the sentence.

When the model could not find decision or rationale information, it outputs: "The sentence does not contain rationale information". This output was hard to process in an uniform way in order to extract the decision-rationale triples (i.e, a triple of the form ⟨decision, has_rationale, rationale⟩ [7]). Therefore, we added this requirement in the prompt: "If you can not find the decision or the rationale, only answer 'None' instead of XX or YY. Do not add any other text in your answer." We noticed that when given the *None* option, the model became "lazy" and used it excessively. In fact, it outputted *None* for at least one entity for 23 of the 25 sentences. For instance, for this clearly phrased sentence: "Also give the functions a kthread_ prefix to better document the use case.", the model output *None* for decision and *None* for rationale.

To solve this issue we added this sentence in the prompt: "All sentences express both a decision and the rationale behind it.". We noticed that the number of sentences with *None* entities reduced significantly to two. With these refinements, we noticed that the performance of the model improved. For example, given this sentence as input: "In addition to integration, this patch also renames high_zoneidx to highest_zoneidx since it represents more precise meaning.", the model produces: "Decision: Renames high_zoneidx to highest_zoneidx \n Rationale: Represents more precise meaning".

Finally, we noticed that the model had difficulty with terse value judgments (e.g., fix, clean-up, etc), producing *None* in these cases. In our Codebook [9], these are considered both decision and rationale. Thus, we used *few-shot learning* [57] and added some examples in our prompt to clarify this. The final prompt we used can be found in our replication package [11].

Given the 774 sentences in the dataset, ChatGPT produced *None*, *none*, *N/A* or *Null* at least for one entity (decision or rationale) for 247 sentences. The first part of Table 2 shows examples of these sentences. For instance, in example 1, the model failed to extract "describe task memory unit" as a decision. In example 2, the model failed to detect the terse value judgment verbs as rationale

Table 2. Limitations of ChatGPT with respect to the extraction of *Decision* and *Rationale* entities.

N ^o	Sentence	Decision	Rationale
1	mm, oom: describe task memory unit, larger PID pad	None	"describe task memory unit, larger PID pad"
2	mm/oom_kill: fix kernel-doc	"fix kernel-doc"	None
3	The first part can be done without a sleeping lock in most cases AFAICS	"The first part can be done without a sleeping lock in most cases"	"AFAICS" (As Far As I Can See)
4	mm,oom: speed up select_bad_process() loop	"speed up select_bad_process() loop"	"mm,oom"
5	It also matches the logic that the oom reaper currently uses for determining when to set [...] itself, so there's no new risk of excessive oom killing	There is no clear decision in this sentence	There is no clear rationale in this sentence

(although an example with "fix" was mentioned in the prompt). We leave as future work further prompt engineering and prompt refinement to reduce the number of these missed cases.

Some of the 527 non-null *Decision*, *Rationale* extracted tuples also do not make sense or contain noise. Examples of this limitation are presented in the second part of Table 2. In example 3, although we mentioned in the prompt that the decision and rationale should be chunks from the sentence, the model augmented the answer and explained the AFAICS acronym. In example 4, the model fails the rationale extraction. In example 5, although we specified in the prompt to output only "None" when the model can't find decision or rationale, the model still outputted other text. This validation is in line with the recommendations by Sallou et al. [43] that stress the importance of reviewing an LLM's output to ensure its applicability in software engineering research. However, since ChatGPT still outperformed SRL (527 > 317 non-missed sentences), we decided to continue utilizing ChatGPT in our approach.

4.1.2 Relationship extraction. The second sub-task needed to build the decision and rationale knowledge graph is to link decisions. We focus specifically on two types of relationships between decisions: *similarity* and *contradiction*. To detect them, we used pre-trained models. For similarity detection, we used the *distilbert-base-nli-mean-tokens* model from the *sentence_transformers* library [38]. For the contradiction detection, we used the *roberta-large-mnli* model from the *transformers* library [29]. These pre-trained models were chosen for prevalence in semantic similarity and contradiction detection in Natural Language Inference [22, 34, 38]. We computed these relationships between each pair of the 527 extracted decisions, and employed a threshold of 0.9 to have strong similarities/contradictions. Among these 138601 combinations, we found 29 similarities and 396 contradictions. Table 3 shows nine examples, while the full list is in our replication package [11].

Extracting these decision-decision relationships is useful as it can help reuse existing solutions. For instance, in the examples 1, 2 and 3 in Table 3, the decisions are practically the same, however, they concern different components (e.g, in the third example, the first patch concerns `exit_oom_victim()` while the second one concerns `oom_enable()`). Detecting these similarities can help developers reuse existing solutions, which will boost their productivity. Example 4 is interesting and worth-checking, as it could indicate an underlying contradiction: although it shows two highly similar decisions (one that moves the `oomkilladj` attribute from `task_struct` to `signal_struct`, and one that moves it from `task_struct` to `mm_struct`), we can not move the same attribute from `task_struct` twice to two different places.

Furthermore, detecting contradictory decisions can help in identifying and preventing conflicts and problems. For instance, in example 5, one decision adds the `try_oom_reaper()` function, while

Table 3. Examples of similar and contradictory decisions in the OOM-Killer module.

N ^o	Decision 1	Decision 2	Relationship
1	Prepare for new header dependencies before moving code to <linux/sched/coredump.h>	Prepare for new header dependencies before moving code to <linux/sched/mm.h>	Similar (0.99)
2	Create a trivial placeholder <linux/sched/coredump.h> file that just maps to <linux/sched.h> to make this patch obviously correct and bisectable	Create a trivial placeholder <linux/sched/mm.h> file to make this patch obviously correct and bisectable	Similar (0.94)
3	Remove unnecessary locking in exit_oom_victim()	remove unnecessary locking in oom_enable()	Similar (0.93)
4	moves oomkilladj (now more appropriately named oom_adj) from struct task_struct to struct signal_struct	Move oomkilladj (now more appropriately named oom_adj) from struct task_struct to struct mm_struct	Similar (0.97)
5	replace try_oom_reaper by wake_oom_reaper	This patch adds try_oom_reaper.	Contradicts (0.98)
6	push the re-check loop out of freeze_processes	We are not checking whether the task is frozen	Contradicts (0.96)
7	giving the dying task an even higher priority	to avoid boosting the dying task priority in case of mem_cgroup OOM	Contradicts (0.97)
8	Add the new helper, find_lock_task_mm()	Taking task_lock() in select_bad_process() to check for OOM_DISABLE and in oom_kill_task() to check for threads sharing the same memory will be removed in the next patch in this series	Contradicts (0.91)
9	Move oomkilladj (now more appropriately named oom_adj) from struct task_struct to struct mm_struct	make the magical value of -17 in /proc/<pid>/oom_adj defeat the oom-killer altogether	Contradicts (0.95)

the other replaces it by a different function. Having the second patch at hand, the author of the first patch can verify whether further changes in other files are needed. Example 6 also shows a potential conflict as the first decision concerns changing the behaviour of how to check frozen processes, while the second decision indicates the lack of such a check. Similarly, example 7 presents two contradictory patches (one that gives the dying task a higher priority and one that avoids boosting that priority). In example 8, the contradiction is that one decision adds a helper to find the task_lock() while the other one predicts its removal, which means it could be useless to add the helper. In example 9, the first decision moves and renames the oom_adj attribute while the second one changes the behaviour of a process with the same name. Although they are not truly contradictory, it is still interesting to detect this case as this naming similarity could lead to future errors or confusion.

RQ.1.1. – Ability to find *similar* and *contradictory* decisions. Our approach was able to find 29 *similar* and 396 *contradictory* decisions. We manually validated a portion of these detections as reasonable, and worthy of inspection by developers.

4.2 Rationale Analyses

In this section, we propose structure-based analyses that, given the decision and rationale knowledge graph, can detect conflicts and inconsistencies in development decisions. Conflict detection analysis may be triggered when a new decision is introduced, e.g., when a new patch is submitted to the project. This mechanism should be able to identify which previous decisions are being contradicted

Table 4. List of inconsistencies detected by M1 and M2 inconsistency-detection mechanisms in the OOM-Killer module.

<i>N</i> ^o	Decision 1	Rationale 1	Decision 2	Rationale 2	D/D Relation	R/R Relation
1	Add the new helper, <code>find_lock_task_mm()</code>	finds the live thread which uses the memory and takes <code>task_lock()</code> to <code>pin→mm</code> Decision: <code>change oom_badness()</code>	use the newly added <code>pidfd_get_task()</code> helper	Instead of duplicating the same code in two places	Similar (0.90)	Contradicts (0.6)
2	Upon returning to the page allocator, its allocation will hopefully succeed	so it can quickly exit and free its memory	The flag is set in <code>mark_oom_victim</code> and never cleared	so it is stable in the <code>exit_mmap</code> path	Contradicts (0.99)	Similar (0.63)
3	avoid unnecessary mm locking and scanning for <code>OOM_DISABLE</code>	unnecessary mm locking and scanning for <code>OOM_DISABLE</code>	We can tolerate a comm being printed that is in the middle of an update	to avoid taking the lock	Contradicts (0.91)	Similar (0.77)

by a decision introduced in the patch, assuming the respective commit messages are accurate. We show examples of contradictory decisions in Table 3, as the *Contradicts* relationship itself is a first conflict detection mechanism.

We also define an analysis mechanism to detect more complicated cases of potential conflicts. This second check should look for decisions that are similar to the new decision, and then for decisions that are contradictory with those similar decisions. It would thus raise a potential conflict between the new decisions and those that contradict its similar decisions. For instance, when Decision 1 in example 4 (E4.D1) in Table 3 is introduced, we should be able to compute a similarity of 0.97 between it and Decision 2 in example 4 (E4.D2). However, E4.D2 contradicts another decision: Decision 2 in example 9 (E9.D2). Thus, E4.D1 could cause conflict with E9.D2.

The goal of inconsistency-detection mechanisms is to verify the consistency of the extracted graph and of the development process, and ensure the graph is free from wrongly-added relationships. We thus implement a mechanism for detecting inconsistencies between rationales, and between decisions. Specifically, we implemented two analysis mechanisms. The first (M1) detects cases where two decisions have a *similar* relationship but their rationales have a *contradicts* relationship. The second (M2) does the opposite, i.e., detects cases where two decisions have a *contradicts* relationship but their rationales have a *similar* relationship. These can be used to validate the integrity of the knowledge graph and clean up nonsensical relations. They can also reveal miscommunications and reasoning problems among the development team in the project's development history. They could indicate for example that a contributor did not fully grasp the impact of a decision they propose. We show such examples in Section 5.2.2. This analysis could also help reveal inaccurate documentation, e.g., due to wrong description of a patch in a commit message or even malicious attempts to hide suspicious code contributions.

Table 4 shows some examples of the inconsistencies detected for the OOM-Killer example. We chose the threshold of 0.6 for the relationships between rationales, and 0.9 for the relationships between decisions. We found one pair of decisions with similar decisions but contradictory rationales, and 85 cases where contradictory decisions had similar rationales. The first row is the first pair: similar decisions but contradictory rationales. The other rows are cases of contradictory decisions with similar rationales. In the three cases, it would be pertinent to remove the relationships between the

decisions from the graph, as they do not make sense. Thus, the inconsistency-detection mechanisms can be used to clean the graph and double-check the soundness of the detected relationships.

RQ.1.2. – Ability to detect inconsistencies. Our approach was able to detect 86 inconsistencies, 85 of which were cases of contradictory decisions having similar rationale, indicating potential miscommunication or misunderstanding in the development process.

4.3 Performance

We have so far discussed how to instantiate the Kantara architecture by constructing the Decision and Rationale knowledge graph and implementing structure-based rationale analyses. Of these two, the graph construction is the most time-consuming with extracting a) the *decision* and *rationale* entities, and b) *Similar* and *Contradicts* relationships. To answer RQ.1.3 (*performance*), we report the time needed to create the graph for the OOM-Killer module (774 *Decision* and *Rationale* sentences).

The entities extraction depends mainly on the inference time of the chosen LLM and the number of queries. Since we initially used the public free version of ChatGPT hosted by OpenAI, this step took almost one hour as we had to deal with the free-version time limits. The relationships extraction also depends on the inference time of the pretrained models. This is a function of the available hardware, the volume of the input data, and batch sizes. For example, in order to extract the contradictory relationships, we used a server with an NVIDIA GeForce RTX 3090 GPU. We have 527 triples, thus 138601 possible combinations of two decisions. With a batch size of 2000, each relationship calculation had a runtime of about one hour. Thus, for a single Linux component (the OOM-Killer) we had a total runtime of three hours. For larger-scale projects, it is possible that the graph construction step may be considerably more time-consuming. In our approach, we assume that the knowledge graph should only be built once, then incrementally updated and maintained as a nightly task [7].

RQ.1.3. – Performance. The graph construction step can be time-consuming. It depends mainly on the inference time of the models and the hardware available. Given the assumption that it happens only once per project lifetime, the runtime of about three hours for our real-world example is reasonable. We surmise that our approach is reasonably performant for practical purposes.

5 Automation and Generalization

5.1 Automating the Extraction of Decision and Rationale Sentences

In this section, we explore the feasibility of automating our approach. Specifically, we evaluate and compare the performance of Machine Learning (ML) and Deep Learning (DL) classifiers and pre-trained models to automatically identify rationale and decision sentences in commit messages. To answer RQ.1.4 (*effectiveness of the automatic extraction of decision and rationale sentences*), we report the performance of the best decision and rationale binary classifiers on the OOM-Killer dataset and compare it to human performance. To the best of our knowledge, this is the first work that has investigated the usage of *pre-trained models* for rationale extraction.

The models were trained and evaluated using 10-fold cross-validation. We randomly divided the OOM-Killer dataset into ten equal splits; nine of them were used for training and one for the evaluation. We trained our models using this method ten times and reported the mean scores as in other works [26, 52]. We consider two binary classification tasks: detecting decision-containing sentences and rationale-containing sentences. Table 5 shows the metrics for the positive classification label. In the following paragraphs, we present relevant technical details about the models.

Table 5. Binary classification evaluation (with 10 folds).

Model	Decision				Rationale			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Logistic Regression	0.70	0.70	0.87	0.78	0.68	0.69	0.80	0.74
Decision Tree	0.63	0.70	0.69	0.69	0.62	0.66	0.65	0.66
SVM	0.70	0.70	0.87	0.78	0.69	0.69	0.80	0.74
LSTM	0.90	0.92	0.92	0.92	0.90	0.91	0.91	0.91
Bi-LSTM	0.92	0.93	0.94	0.93	0.90	0.92	0.90	0.91
DistilBERT	0.81	0.82	0.87	0.84	0.73	0.75	0.77	0.76
ALBERT (large)	0.80	0.82	0.85	0.83	0.73	0.75	0.80	0.77
BERT (large)	0.81	0.84	0.85	0.84	0.74	0.75	0.80	0.77
RoBERTa (large)	0.81	0.84	0.86	0.84	0.75	0.76	0.82	0.78

Table 6. Hyperparameter values for LSTM and Bi-LSTM models.

Hyperparameters	Values
max_words	1000
epochs	3
dropout rate	0.2
Embedding layer	50
Optimizer	Adam
Loss function	Mean Squared Error
Activation function	Sigmoid

Table 7. Bi-LSTM based classifier performance.

	Metrics	Decision	Rationale
Positive	Precision	0.97	0.98
	Recall	0.98	0.96
	F1-score	0.98	0.97
Negative	Precision	0.98	0.95
	Recall	0.96	0.97
	F1-score	0.97	0.96
	Accuracy	0.97	0.96

Classic AI models. We compare the performance of Logistic Regression (LR) [24], Decision Trees [47] and Support Vector Machines (SVM) [18] due to their popularity and good performance for rationale identification [3, 8]. We use the *TfidfVectorizer* with these extractors.

Deep Neural networks. We experiment with the Long Short-Term Memory (LSTM) [17] and the Bidirectional-LSTM (Bi-LSTM) [60] models, known for their good performance in text classification tasks in general. They also gave good results for rationale classification tasks [52, 55]. The various hyperparameters used to train the Deep Learning classifiers are listed in Table 6.

Pretrained Models. We fine-tuned four popular pre-trained models: DistilBERT [44], ALBERT [23], BERT [6] and RoBERTa [29] for five epochs on the OOM-Killer dataset. Specifically, we used the implementations from the *transformers* library [58]: *distilbert-base-uncased*, *albert-base-v2*, *bert-base-cased*, *roberta-base*, *albert-large-v2*, *bert-large-cased* and *roberta-large*. For BERT, ALBERT, and RoBERTa, we report the results of the *large* models in Table 5 as they gave the best results. We keep the default values for *learning_rate* (5e-05) and *optimizer* (*adamw*), and use a *weight_decay* of 0.01.

Best Model. The results in Table 5 indicate that the Bi-LSTM model has the best performance on the two categories, with an accuracy of 92% and 90% for the Decision and Rationale respectively. This result is consistent with prior findings [52, 55]. We thus chose the Bi-LSTM models as our best classifiers. We note that we did not optimize hyperparameters or investigate other models (e.g, pre-trained models). We leave this experimental investigation to future work. The performance of

Table 8. Human performance on the classification task.

Rater	Decision				Rationale			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
R1	0.94	1.0	0.91	0.95	0.83	1.0	0.69	0.82
R2	0.95	1.0	0.91	0.95	0.86	1.0	0.75	0.85
R3	0.83	1.0	0.73	0.84	0.86	1.0	0.76	0.86

the best classifiers for one specific execution run with 90% split is shown in Table 7. The precision for classifying a sentence as *A Decision* is 97% and the recall is 98%. In addition, the precision for labelling a sentence as *Not A Decision* is 98% and its recall is 96%. Similarly, the precision of classifying a sentence as *Rationale* is 98% and the recall is 96%, and the precision of labelling a sentence as *Not Rationale* is 95% and its recall is 97%.

Human Baselines. The dataset we used is a union of the labelling of three independent raters [9]. To compare our classifiers results to human performance, we compute the performance of each rater with regards to the final labelling. The metrics of the positive label are shown in Table 8. Overall, the performance of the classifiers is comparable to human performance. For the rationale category, our best classifier outperforms the best human performance.

RQ.1.4. – Effectiveness of the automatic extraction of decision and rationale sentences.

We proposed two classification models based on Bi-LSTM to automatically identify whether a commit message sentence contains Decision or Rationale information. Both of them performed well in our dataset (more than 90% accuracy) and reached human-like performance.

Given the answers to the sub-questions RQ.1.1 - RQ.1.4, we conclude:

RQ.1 – Feasibility. Instantiating the Kantara architecture to *automatically* support *effective rationale analyses*, such as finding similar and contradictory decisions and finding inconsistencies, is *feasible* and with *reasonable* performance.

5.2 Investigation of Generalization

In this section, we answer RQ.2 (*generalization*) of our approach. Specifically, we evaluate whether our best performing classifiers and the Kantara implementation can generalize to other a) memory management (*mm*) Linux modules, b) Linux sub-projects and c) OSS projects.

5.2.1 Classifiers Generalization.

Generalization to other Linux components and sub-projects. We consider the *slob.c*³ module of the *mm* component and the *button.c*⁴ module of the '*drivers/acpi*' subproject. The *slob* module was (until its removal in 2022) a memory allocator in Linux for embedded systems⁵. The *button* module is responsible for handling laptop lid and power or sleep buttons. Using the same pre-processing as for the OOM-Killer module [9], we obtain 146 commit messages and 833 sentences for *slob* and 110 commit messages and 576 sentences for *button*.

We ran our two best Bi-LSTM classifiers on these datasets. To validate these predictions, the three authors did post-label commit-based sampling. Specifically, we randomly chose 20 commits

³<https://api.github.com/repos/torvalds/linux/commits?path=mm/slob.c>

⁴<https://api.github.com/repos/torvalds/linux/commits?path=drivers/acpi/button.c>

⁵<https://lwn.net/Articles/918344/>

Table 9. Bi-LSTM inter-rater agreement.

Rater	mm/slob.c		drivers/acpi/button.c	
	Decision	Rationale	Decision	Rationale
Rater 1	70.3%	69.2%	73.6%	80.2%
Rater 2	73.6%	70.3%	71.4%	80.2%
Rater 3	73.6%	80.2%	65.9%	73.6%
Average	72.5%	73.6%	70.3%	78%
Inter-rater agreement	79.1%	65.9%	75.8%	60.4%

from each file and ended up with 91 sentences per file. Then, we individually indicated whether or not we agree with the predictions of the classifiers. Table 9 summarizes the agreement of the three raters. In average, the raters agree with 70%-72% of the predictions for the decision category and with 73%-78% for the rationale category. This indicates a good generalization to other Linux modules. Furthermore, we compute the inter-rater agreement using Fleiss Kappa [15] and report it in Table 9. As the values show that the three raters had a total agreement for about 60-79% sentences, we consider this to mean that the classifier is *about as accurate as a human annotator*.

Generalization to other OSS projects. There is no universally accepted definition of what constitutes rationale information and, to the best of our knowledge, the OOM-Killer dataset is the only extant dataset of labelled entities (decision, rationale) from commit messages in free-text. Thus, to evaluate the generalizability of our approach we use the dataset of Tian et al. [52], as it is the closest to our work. The dataset contains 1649 commit messages from five OSS projects (JUnit4, Apache Dubbo, Square Retrofit, Square OkHttp and Spring-boot). Commit messages are labelled as whether or not they contain “*what*” and “*why*” information. Specifically, four labels are used: *Why_and_What* (containing both), *No_What* (only Why, but no What), *No_Why* (only What information, but no Why), and *Neither_Why_nor_What*. After removing non-atomic commits we obtain 1597 commit messages. To make comparison possible, we interpret the *What* label to indicate the presence of a Decision and the *Why* label to indicate the presence of Rationale. Thus, for our binary decision classification task, messages labelled with *Why_and_What* and with *No_Why* are considered as messages having decision information, and the others as not. Respectively, messages labelled with *Why_and_What* and with *No_What* are considered as messages having rationale information, and the others as not. While our mapping of the labels is reasonable for comparison purposes, it may not be perfectly accurate due to potential mismatches or differing interpretations of the labels.

We test our classifiers on these already pre-processed commit messages [52], noting that they omit information about identifiers, which potentially impacts our analysis. We consider the commits of each project separately, then of all five projects altogether. Table 10 indicates reasonable generalization power to other OSS projects with an F1-score of 0.70 for decisions, and 0.75 for rationale overall.

Given the generalizability of our best binary classifiers to other Linux modules and OSS projects, we can use their predictions to extract sentences that are labelled both as Decision and Rationale. Then, we can apply our Kantara instantiation to structure these sentences as a knowledge graph and perform rationale analyses (see Fig. 1). In fact, this ML-based implementation of the Kantara framework is generalizable: we neither finetuned the LLM to extract decision and rationale tuples from sentences, nor finetuned the pre-trained models to extract the similar and contradictory relationships (Section 4.1). Thus, given these generalizable binary classifiers, and the rest of our

Table 10. Generalization results of the best classifiers on other open source projects

Project	Decision				Rationale			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
JUnit4	0.60	0.81	0.65	0.72	0.57	0.58	0.75	0.66
Apache Dubbo	0.57	0.72	0.70	0.71	0.66	0.77	0.78	0.78
Square Retrofit	0.64	0.92	0.66	0.77	0.57	0.55	0.80	0.65
Square OkHttp	0.57	0.87	0.59	0.71	0.62	0.71	0.77	0.74
Spring-boot	0.46	0.83	0.47	0.60	0.74	0.96	0.75	0.84
All commits	0.56	0.83	0.60	0.70	0.64	0.73	0.77	0.75

Table 11. Open source project decision/rationale information and inconsistencies.

Project	Number of sentences labelled both as decision and rationale	Number of decision- rationale triples without 'None' values	Number of inconsistencies (M1)	Number of inconsistencies (M2)
Slob.c	252	209	3	58
Button.c	191	156	0	12
JUnit	145	98	4	10
apache/dubbo	171	116	6	15
square/retrofit	138	100	14	11
square/okhttp	151	115	7	2
spring_boot	137	126	3	8

Kantara instantiation (entities extraction with LLMs and relationship extraction with pretrained models) is also reusable, this means that overall we were able to create an instance of Kantara that is independent of the particular project (the Linux OOM-Killer) that we used to introduce the architecture [7].

RQ2. – Generalization. Our best classifiers generalize well to other Linux modules (inter-rater agreement more than 60% for all cases) and other OSS projects (F1-score more than 70% for both categories). Thus, they can be reused. Furthermore, the usage of LLMs and pre-trained models allows for the generalizability of the approach.

5.2.2 Application and Inconsistency Detection in Other Contexts. The left side of Table 11 gives a summary of the number of sentences labelled both as decision and rationale as well as the number of decision-rationale triples (see Section 4.1) obtained after prompting ChatGPT and removing 'None' values (however, there is always noise, like producing '-' or '\nNone'). We then used the inconsistency detection mechanisms M1 (similar decisions, contradictory rationales) and M2 (contradictory decisions, similar rationales), defined in Section 4.2. The right side of Table 11 presents the number of inconsistencies detected by each of the two mechanisms when using a threshold of 0.8 for the relationships between decisions, and a threshold of 0.6 for the relationships between the rationales.

We show concrete examples of the inconsistencies and relationships detected in Table 12. The first example shows two similar decisions with contradictory rationales. The first tuple implies that the slab_mutex will be used in the future for the cache list, while the second one indicates it is useless for the kmem_cache_shrink. The second example illustrates contradictory decisions that

Table 12. Detected examples of inconsistencies in open-source projects.

Project	Decision 1	Rationale 1	Decision 2	Rationale 2	D/D Relation	R/R Relation
Slob.c	move it out of the slab_mutex	which we have to hold for iterating over the slab cache list	slab_mutex for kmem_cache _shrink is removed	after its applied, there is no need in taking the slab_mutex	Similar (0.806)	Contradicts (0.900)
Button.c	Setting lid_init_state to ACPI_BUTTON_LID_INIT_OPEN on the T200TA	fixes the unwanted behavior, adds a DMI based quirk	libinput (1.7.0+) can fix the state of the lid switch for us: you need to set the udev property [...] to write_open	can fix the state of the lid switch for us	Contradicts (0.860)	Similar (0.687)
Dubbo	fix typo:<iden> (<pr_link>)	typo	Support <iden> auto recognize in <file_name> (<pr_link>)	Enhancement Decision: Fix <issue_link>	Similar (0.8110)	Contradicts (0.627)
Retrofit	Propagate callback executor even when not explicitly specified	This lets custom <iden> implementations use the <file_name> default for their callbacks	Switch to a generic to/from <file_name> , explicit factory	This allows for specialization based on for what purpose the <file_name> is being created.	Contradicts (0.891)	Similar (0.776)

have similar rationales; the first decision indicates manually setting the lid state while the second one proposes using libinput. Although these are different approaches, they both aim at the same goal – fixing the lid state.

The third example highlights two decisions, one that fixes a typo in an identifier and one that supports the identifier’s auto recognition in order to fix an issue. It is thus interesting for the committer of the first decision to know that the identifier is used elsewhere and thus the typo might also need to be fixed elsewhere⁶. The last example shows a contradiction between a decision that suggests implicit callback executor propagation and a decision that suggests a switch to an explicit factory. Note that both these contradictory decisions are based on the same logic of enabling specific implementations. Although developer evaluation of whether these cases are truly problematic is necessary, our approach is able to automatically raise these potentially conflicting commits.

6 Discussion

6.1 Implications

For developers. The automated identification of rationale could be appropriately embedded in development tools [42] to encourage the writing of rationale-aware commit messages. For example, our CoMRAT tool recommends revising commit messages if they lack rationale information [10]. This is crucial in the context of rising adoption of AI-generated code, to help developers take ownership of the code and argue for its inclusion [54]. This would additionally help counter the *design evaporation* problem [39] by preserving crucial rationale information. In turn, this can foster

⁶Of course, this only makes sense if both decisions are about the same identifier. The pre-processing done by Tian et al. [52] (changing file names by <file_name>, identifiers by <iden>, etc) caused the loss of semantic information. A complete dataset would not suffer from this problem.

a shared understanding of the logic behind the decisions by developers, and reduce potential productivity barriers. The automated creation of a knowledge graph for rationale information in OSS projects also could offer significant and practical benefits, and is now feasible with reasonable performance because of recent AI advancements (e.g., LLMs and pre-trained models). For instance, detecting similarities between development decisions can help developers identify and re-use existing solutions to avoid duplication of effort. Detecting contradictory decisions, as well as inconsistencies can shed light on the existing collisions and hidden development issues, allowing developers to proactively address them. This can help counter the *design erosion* problem [56]. The conflicts detection mechanisms at the introduction of every new decision would also help developers take preventive actions and double-check their proposed changes against the relevant past commits before submitting them. This would help avoid problems and confusions, which should increase the sustainability and longevity of the project.

For researchers. Our approach can help create large repositories of decision and rationale data from open source repositories, that can then be used for further research and various analyses. For example, our tool, CoMRAT [10], replicates the rationale analysis on the OOM-Killer [9] for any GitHub module. Researchers can also study the correlation between rationale and other factors, such as project age, developer experience, causes of project failure, and contradictory decisions and re-engineering efforts. Our approach could thus be used to advance software development automation research, such as the automated generation of rationale-aware code comments [33] or commit messages [51], and could help enable automated on-demand documentation [40].

6.2 Threats to Validity

Threats to *external validity* relate to result generalizability. We showed the generalization of our approach by applying it to other Linux modules and other OSS projects. We also validated that the decision and rationale sentence classifiers generalize well to these other projects. This gives us confidence that our methodology can be applied to detect inconsistencies and conflicts in projects beyond those we studied. However, an external validity threat arises from the limited size of the English-only OOM-Killer dataset, on which the classifiers have been trained. It is thus possible that the extractors do not generalize well to other cultural and linguistic contexts. Our approach might generalize less well to contexts where the ways developers communicate, express themselves, and document their work differ significantly.

In the future, we plan to replicate our approach in other such contexts. We also plan to redo the experiments on the original commit messages of the OSS projects, as we suspect that the pre-processing applied [52] (removing identifiers, etc) may have wrongly influenced the extracted relationships, resulting in false conflicts. We also assume that rationale information is consistently present in commit messages and that it is limited to a single sentence. Consequently, our approach does not generalize to contexts where rationale is either absent or spans multiple sentences.

Threats to *internal validity* relate to experimental errors and biases.

Ground truth inclusion bias. The dataset that we used is the union of the labelling of three human raters. This presents a threat as we use it as a ground truth, and thus training a classifier on this data might make it biased for inclusion. However, since the results indicate that the classifiers generalized well and reached human-like performance, we believe this threat to be minor.

Hyperparameter and model selection. A further threat stems from the hyperparameter choices for our models, which was mitigated by keeping default values. Thus, results could be improved with hyperparameter optimization using search methods which tend to have high computational cost. For instance, pre-trained models like BERT could outperform Bi-LSTM if fine-tuned for more epochs. Similarly, we did not investigate other similarity and contradiction detection models, though

leveraging other pre-trained models could improve results. In fact, this aspect is not critical to our study as we have shown the feasibility of an end-to-end rationale extraction and management system for real-world projects with reasonable hyperparameters values and widely used Natural Language Inference models. We leave further experimental investigation for future work.

Threshold selection. We selected the thresholds heuristically, simply to illustrate our approach. In our replication package, we provide relationships extraction and rationale analysis results with only a threshold of 0.5 for D-D relationships. We intend to further investigate the threshold values as part of a planned user study to determine the thresholds which balance information overload and meaningful relationships. In a tool implementing our approach, the thresholds could also be selected according to user preference.

Temporal dependencies. As discussed in Section 2, we excluded the *Supporting Facts* category in the OOM-Killer dataset from our analysis. However, residual temporal dependencies in the dataset, even after disregarding this label, pose a threat to internal validity. They may introduce biases related to the sequence of commit messages or system states, potentially affecting our evaluation. We plan to study further the impact of the temporal dependencies and investigate the mitigation potential of rigorous temporal validation techniques such as time series splitting.

LLMs usage. Finally, threats to validity include the concerns of using LLMs such as closed-source models, data leakage, and reproducibility of findings [43]. For the purpose of review, we have made all ChatGPT responses publicly available [11]. We expect that other LLMs are also suitable for this task, and that their performance and cost characteristics will improve in the future.

Threats to *construct validity* include the choice of our evaluation metrics. We mitigate this threat by selecting metrics widely used in prior works to evaluate rationale detection tasks [3, 26, 52, 55].

7 Related Work

LLMs for Automating Software Engineering. Chaaben et al. proposed to automate model completion by mapping partial models to a semantically equivalent textual representation, and then using few-shot prompt learning in order to receive suggestions about related elements [5]. Huang et al. studied different aspects of fine-tuning LLMs for program repair, including code representations and evaluation metrics [19]. Sue et al. leveraged the reasoning capabilities of LLMs to generate feasible and coherent test scenarios [49]. Du et al. investigated the effectiveness of LLMs in class-level code generation [12]. To the best of our knowledge, our work is the first to leverage LLMs for developer's rationale extraction and management.

Linux Kernel. The Linux Kernel is one of the most elaborate open source development projects and thus has been the object of extensive research [9, 14, 37, 50, 53]. In [37], the authors conduct an empirical study on logging practices in the Linux kernel. Among others, they investigate the rationale behind the changes made to logging statements. They found that it mainly pertains to improving the precision, format and consistency of the statements. Patel et al. [53] tried to understand whether the motivation of OSS developer is influenced by the sense of a virtual community. To this end, they propose a theoretical model and test it on the Linux Kernel developer community. Ferreira et al. [14] analyzed Linux Kernel Mailing List (LKML) emails relating to rejected changes in order to study uncivility in OSS development communication. They found that more than half the emails contained uncivil communication. Tan et al. [50] studied the communication between Linux developers. They focused on the expressions used when submitting patches. After creating the rationale dataset from the commit messages of the OOM-Killer component, in our previous work, we also studied the characteristics of how the rationale information appears in these messages. Specifically, we analyzed the frequency of the presence of rationale, the factors that impact it, its temporal evolution, as well as the structure of commit messages [9].

This work is different from the above because we focus on Linux kernel commit messages as an object for an automated approach for rationale analyses, and as a training set for automatic rationale extraction.

Software Rationale. Recently, several researchers have studied the underlying rationale in end-users reviews [55] and designers documents [59]. Ullah et al. experimented with various ML and DL algorithms to capture and analyze end-user reviews containing rationale information, using the codes: *claim-attacking*, *claim-supporting*, *claim-neutral*, *decision*, and *issue*. For the *decision* category, the LSTM and BI-LSTM models gave the best results [55], which is in accordance with our findings. Yue et al. proposed a knowledge fusion method that builds a Design Rationale Knowledge Network and show its feasibility on patents documents and open access research articles [59]. The knowledge rationale network includes literature knowledge (author, affiliations, and title), artifact knowledge (the inheritance and composition relationship of design objects), rationale knowledge (issues, intents, pros/cons, and alternatives), and the relationships between them. The authors also computed semantic similarity to identify identical and similar relationship between the issues. Our research differs from the above in that we focus on software developers, as opposed to end-users.

Previous research also investigated the rationale behind developer's decisions [1, 27]. Alsafwan et al. conducted interviews and surveys, and found that developers decompose the rationale of code commits into 15 separate components [1]. Similarly, Liang et al. conducted surveys responses and interviews with professional developers about their decision processes for implementation design decisions. They found that these decisions are shaped by higher levels of design (e.g., requirements and architecture) [27]. These studies are different from us by the nature of the work.

Another line of research focuses on the automatic extraction of developers rationale [2, 25, 26, 32, 41, 45, 52]. Alkadhi et al. experimented with different ML models to extract rationale elements (decision, issue, alternative, pro-argument, con-argument) from developers' chat messages [2]. Sharma et al. tried to extract rationale from Python Enhancement Proposals (PEPs) using heuristics [45]. Rogers et al. investigated the usage of ontology and linguistic features to identifying rationale from Chrome bug reports [41]. Lester et al. investigated evolutionary algorithms for optimal features to extract and classify design rationale from Chrome bug reports and design discussion transcripts [25]. Mathur et al. investigated the generalizability of Lester's classifiers to other datasets [32]. They found that the classification models that were trained on Chrome Bug Reports are not generalizable to other datasets, as the training dataset contains specific Chrome-related terminology. Our work is different from the above in that we focus on commit messages specifically. Another major difference with previous work is that researchers previously did not examine developer rationale in its natural textual form, but as a pre-defined category (e.g. issue, alternative, pro-argument [2, 25, 32, 41] or transition from a PEP state to another [45]).

Tian et al. investigated the existence of *What* and the *Why* in the commit messages of five OSS projects [52]. Their experiments to automatically identify messages with *What* and *Why* information found that LSTM and Bi-LSTM gave the best performance, which is in accordance with our findings. Li et al. [26] continued this work but also considered the link content when identifying the *What* and *Why* information in commit messages. These lines of research are different from us as they only focus on the extraction and do not propose rationale management frameworks, i.e., they do not propose ways to leverage the extracted information. Furthermore, they do not investigate the usage of pre-trained models for rationale extraction like this work.

Other researchers proposed rationale extraction and management systems [8, 20, 28, 30, 31]. Kleebaum et al. proposed the Condec Tools to support documenting and managing decision knowledge for change impact analysis [20]. These tools build up and visualize a knowledge graph consisting of knowledge elements and trace links. Condec Tools include a limited automated rationale

extraction feature. In Condec Tools, rationale is represented as a model (i.e., *issue*, *alternatives*, *pro* and *con-arguments*, and *decision*). Liang et al. proposed a design rationale discovery, retrieval and management system for design patents [28]. They employed text analysis for discovery and created a semantic sentence graph to model sentence relations. They used the ISAL three-layered model (issue, solution and artifact) for rationale representation. In order to retrieve related sentences, they leveraged ranking algorithms. Bhat et al. proposed AdeX, a framework to extract and reuse architecture knowledge and to recommend alternative architectural solutions that could be considered during architectural design making [31]. The authors proposed a static and a dynamic architectural knowledge models. In AdeX, a decision's rationale refers to the quality attribute the decision addresses. AdeX automatically identifies architectural elements in design decisions. López et al. proposed the TREx (Toeska Rationale Extraction) approach to recover, represent and explore rationale information from text documents [30]. TREx proposes a software architecture ontology and a rationale ontology, used to create rationale-specific information extraction tools to recover software architecture and rationale information from documents. These lines of research differ from our work as to techniques for rationale extraction (they do not use LLMs and pre-trained models), rationale representation (they do not consider free-form textual rationale), and rationale management (they do not propose automated mechanisms to detect reasoning inconsistencies).

This work is also different from our previous implementation of Kantara [8]. On the one hand, we employ more advanced neural architectures for the decision and rationale extraction that were trained on the whole OOM-Killer dataset as well as LLMs, while our previous work only used basic ML models trained on parts of the dataset. On the other hand, we present an ML-empowered implementation while our earlier approach was ontology-based, using the openCAESAR framework [13]. Finally, our previous work did not investigate the generalizability of the approach.

8 Conclusion

We contributed an approach to automatically support effective rationale analyses based on commit messages. For this we created an efficient implementation of the Kantara architecture that leverages pre-trained models and LLMs. We showed how it can be used to find similar and contradictory decisions and to detect inconsistencies. We were able to detect 85 cases in the OOM-Killer history where contradictory decisions were made with similar rationales, which can help identify reasoning problems in the development history of the project. We further demonstrated that our approach is generalizable to other Linux modules and more generally to other open source projects. This opens up the possibility of applying our approach at a wide scale. Notably, it could be used to create tools to help developers better develop and document their rationale.

In the future, we plan to validate the usefulness of the extracted relationships and of the rationale analyses through a study with knowledgeable open-source developers, such as for the OOM-Killer module. We also plan to package our implementation as a GitHub bot that can assist developers directly during the development process. Developing our implementation into a research tool to create rationale datasets would facilitate research on how software rationale appears and develops in open source projects. Finally, we will use more powerful or more private/offline LLMs, better refine the prompt, and leverage other pre-trained models to provide even better results.

Data Availability

All the materials related to this work are available on Zenodo [11].

Acknowledgments

This research work is partially funded by the Fonds de Recherche du Québec – Nature et Technologies (FRQNT) Doctoral Research Scholarship (B2X).

References

- [1] Khadijah Al Safwan, Mohammed Elarnaoty, and Francisco Servant. 2022. Developers' Need for the Rationale of Code Commits: An in-Breadth and in-Depth Study. *Journal of Systems and Software* 189 (July 2022), 111320. <https://doi.org/10.1016/j.jss.2022.111320>
- [2] Rana Alkadhi, Teodora Lata, Emitza Guzman, and Bernd Bruegge. 2017. Rationale in development chat messages: an exploratory study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 436–446. <https://doi.org/10.1109/MSR.2017.43>
- [3] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. 2017. Automatic Extraction of Design Decisions from Issue Management Systems: A Machine Learning Based Approach. In *Software Architecture*, António Lopes and Rogério de Lemos (Eds.). Vol. 10475. Springer International Publishing, Cham, 138–154. https://doi.org/10.1007/978-3-319-65831-5_10
- [4] Janet E. Burge, John M. Carroll, Raymond McCall, and Ivan Mistrik. 2008. *Rationale-based software engineering*. Springer. <https://doi.org/10.1007/978-3-540-77583-6>
- [5] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. 2023. Towards using few-shot prompt learning for automating model completion. In *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 7–12. <https://doi.org/10.1109/ICSE-NIER58687.2023.00008>
- [6] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018). <https://doi.org/10.48550/arXiv.1810.04805>
- [7] Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2022. End-to-End Rationale Reconstruction. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–5. <https://doi.org/10.1145/3551349.3559547>
- [8] Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2023. Towards Understanding and Analyzing Rationale in Commit Messages using a Knowledge Graph Approach. In *2023 International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. <https://doi.org/10.1109/MODELS-C59198.2023.00101>
- [9] Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2024. Rationale Dataset and Analysis for the Commit Messages of the Linux Kernel Out-of-Memory Killer. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*. <https://doi.org/10.1145/3643916.3644413>
- [10] Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2025. CoMRAT: Commit Message Rationale Analysis Tool. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*.
- [11] Mouna Dhaouadi, Bentley Oakes, and Michalis Famelis. 2025. Replication package. <https://zenodo.org/records/13742395>.
- [12] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13. <https://doi.org/10.1145/3597503.3639219>
- [13] Maged Elaasar, Nicolas Rouquette, David Wagner, Bentley Oakes, Abdelwahab Hamou-Lhadj, and Mohammad Hamdaqa. 2023. openCAESAR: Balancing agility and rigor in model-based systems engineering. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 221–230. <https://doi.org/10.1109/MODELS-C59198.2023.00051>
- [14] Isabella Ferreira, Jinghui Cheng, and Bram Adams. 2021. The “shut the f** k up” phenomenon: Characterizing incivility in open source code review discussions. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–35. <https://doi.org/10.1145/3479497>
- [15] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378. <https://doi.org/10.1037/h0031619>
- [16] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A Deep Semantic Natural Language Processing Platform. <https://doi.org/10.48550/arXiv.1803.07640>
- [17] Alex Graves. 2012. Long short-term memory. *Supervised sequence labelling with recurrent neural networks* (2012), 37–45. https://doi.org/10.1007/978-3-642-24797-2_4
- [18] Marti A. Hearst, Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their applications* 13, 4 (1998), 18–28. <https://doi.org/10.1109/5254.708428>
- [19] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An empirical study on fine-tuning large language models of code for automated program repair. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1174. <https://doi.org/10.1109/ASE56229.2023.00181>
- [20] Anja Kleebaum. 2023. *Continuous Rationale Management*. Ph. D. Dissertation. Heidelberg University.
- [21] Philippe Kruchten, Patricia Lago, and Hans Van Vliet. 2006. Building up and reasoning about architectural knowledge. In *International conference on the quality of software architectures*. Springer, 43–58. https://doi.org/10.1007/11921998_8

- [22] Philippe Laban, Tobias Schnabel, Paul N Bennett, and Marti A Hearst. 2022. SummaC: Re-visiting NLI-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics* 10 (2022), 163–177. https://doi.org/10.1162/tacl_a_00453
- [23] Z Lan. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019). <https://doi.org/10.48550/arXiv.1909.11942>
- [24] Michael P. LaValley. 2008. Logistic regression. *Circulation* 117, 18 (2008), 2395–2399. <https://doi.org/10.1161/CIRCULATIONAHA.106.682658>
- [25] Miriam Lester and Janet E. Burge. 2019. Identifying design rationale using ant colony optimization. In *Design Computing and Cognition'18*. Springer, 537–554. https://doi.org/10.1007/978-3-030-05363-5_30
- [26] Jiawei Li and Iftekhhar Ahmed. 2023. Commit message matters: Investigating impact and evolution of commit message quality. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 806–817. <https://doi.org/10.1109/ICSE48619.2023.00076>
- [27] Jenny T. Liang, Maryam Arab, Minhuk Ko, Amy J. Ko, and Thomas D. LaToza. 2023. A qualitative study on the implementation design decisions of developers. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 435–447. <https://doi.org/10.1109/ICSE48619.2023.00047>
- [28] Yan Liang, Ying Liu, Chun Kit Kwong, and Wing Bun Lee. 2012. Learning the “Whys”: Discovering Design Rationale Using Text Mining — An Algorithm Perspective. *Computer-Aided Design* 44, 10 (Oct. 2012), 916–930. <https://doi.org/10.1016/j.cad.2011.08.002>
- [29] Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019). <https://doi.org/10.48550/arXiv.1907.11692>
- [30] Claudia López, Víctor Codoceo, Hernán Astudillo, and Luiz Marcio Cysneiros. 2012. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. *Science of Computer Programming* 77, 1 (2012), 66–80. <https://doi.org/10.1016/j.scico.2010.06.009>
- [31] Manoj Mahabaleshwar. 2020. *Tool support for architectural decision making in large software intensive projects*. Ph.D. Dissertation. Technische Universität München.
- [32] Tanmay Mathur. 2015. *Improving classification results using class imbalance solutions & evaluating the generalizability of rationale extraction techniques*. Master’s thesis. Miami University.
- [33] Fangwen Mu, Xiao Chen, Lin Shi, Song Wang, and Qing Wang. 2023. Developer-intent driven code comment generation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 768–780. <https://doi.org/10.1109/ICSE48619.2023.00073>
- [34] MultiNLI Benchmark. 2024. Papers With Code <https://paperswithcode.com/sota/natural-language-inference-on-multinli>.
- [35] OpenAI. 2024. ChatGPT. <https://chat.openai.com/chat>. [Large language model].
- [36] Martha Stone Palmer, Daniel Gildea, and Nianwen Xue. 2010. *Semantic role labeling*. Vol. 6. Morgan & Claypool Publishers. <https://doi.org/10.2200/S00239ED1V01Y200912HLT006>
- [37] Keyur Patel, João Faccin, Abdelwahab Hamou-Lhadj, and Ingrid Nunes. 2022. The Sense of Logging in the Linux Kernel. *Empirical Software Engineering* 27, 6 (Nov. 2022), 153. <https://doi.org/10.1007/s10664-022-10136-3>
- [38] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <https://doi.org/10.48550/arXiv.1908.10084>
- [39] Martin P. Robillard. 2016. Sustainable software design. In *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*. 920–923. <https://doi.org/10.1145/2950290.2983983>
- [40] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurelio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vasquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. 2017. On-Demand Developer Documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Shanghai, 479–483. <https://doi.org/10.1109/ICSME.2017.17>
- [41] Benjamin Rogers, Yechen Qiao, James Gung, Tanmay Mathur, and Janet E. Burge. 2015. Using text mining techniques to extract rationale from existing documentation. In *Design computing and cognition'14*. Springer, 457–474. https://doi.org/10.1007/978-3-319-14956-1_26
- [42] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspán, Emma Soderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 598–608. <https://doi.org/10.1109/ICSE.2015.76>
- [43] June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the Silence: the Threats of Using LLMs in Software Engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (Lisbon, Portugal) (ICSE-NIER'24)*. Association for Computing Machinery, New York, NY, USA, 102–106. <https://doi.org/10.1145/3639476.3639764>

- [44] V Sanh. 2019. DistilBERT, A Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. *arXiv preprint arXiv:1910.01108* (2019). <https://doi.org/10.48550/arXiv.1910.01108>
- [45] Pankajeshwara Nand Sharma, Bastin Tony Roy Savarimuthu, and Nigel Stanger. 2021. Extracting Rationale for Open Source Software Development Decisions — A Study of Python Email Archives. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, Madrid, ES, 1008–1019. <https://doi.org/10.1109/ICSE43902.2021.00095>
- [46] Peng Shi and Jimmy Lin. 2019. Simple BERT Models for Relation Extraction and Semantic Role Labeling. *ArXiv abs/1904.05255* (2019). <https://doi.org/10.48550/arXiv.1904.05255>
- [47] Yan-Yan Song and LU Ying. 2015. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry* 27, 2 (2015), 130. <https://doi.org/10.11919/j.issn.1002-0829.215044>
- [48] Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and I. Dagan. 2018. Supervised Open Information Extraction. In *NAACL-HLT*. <https://doi.org/10.18653/v1/N18-1081>
- [49] Yanqi Su, Dianshu Liao, Zhenchang Xing, Qing Huang, Mulong Xie, Qinghua Lu, and Xiwei Xu. 2024. Enhancing Exploratory Testing by Large Language Model and Knowledge Graph. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 98, 12 pages. <https://doi.org/10.1145/3597503.3639157>
- [50] Xin Tan and Minghui Zhou. 2019. How to communicate when submitting patches: An empirical study of the Linux kernel. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–26. <https://doi.org/10.1145/3359210>
- [51] Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2022. A large-scale empirical study of commit message generation: models, datasets and evaluation. *Empirical Software Engineering* 27, 7 (2022), 198. <https://doi.org/10.1007/s10664-022-10219-1>
- [52] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering (Pittsburgh, Pennsylvania) (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2389–2401. <https://doi.org/10.1145/3510003.3510205>
- [53] Bianca Trinkenreich, Klaas-Jan Stol, Anita Sarma, Daniel M. German, Marco A. Gerosa, and Igor Steinmacher. 2023. Do I Belong? Modeling Sense of Virtual Community Among Linux Kernel Contributors. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 319–331. <https://doi.org/10.1109/ICSE48619.2023.00038>
- [54] Rosalia Tufano, Antonio Mastropaolo, Federica Pepe, Ozren Dabić, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling ChatGPT's Usage in Open Source Projects: A Mining-based Study. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. 571–583. <https://doi.org/10.1145/3643991.3644918>
- [55] Tahir Ullah, Javed Ali Khan, Nek Dil Khan, Affan Yasin, and Hasna Arshad. 2023. Exploring and mining rationale information for low-rating software applications. *Soft Computing* (2023), 1–26. <https://doi.org/10.1007/s00500-023-09054-3>
- [56] Jilles Van Gurp and Jan Bosch. 2002. Design erosion: problems and causes. *Journal of systems and software* 61, 2 (2002), 105–119. [https://doi.org/10.1016/S0164-1212\(01\)00152-2](https://doi.org/10.1016/S0164-1212(01)00152-2)
- [57] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34. <https://doi.org/10.1145/3386252>
- [58] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019). <https://doi.org/10.18653/v1/2020.emnlp-demos.6>
- [59] Gaofeng Yue, Jihong Liu, Qiang Zhang, and Yongzhu Hou. 2023. Building a Design-Rationale-Centric Knowledge Network to Realize the Internalization of Explicit Knowledge. *Applied Sciences* 13, 3 (2023), 1539. <https://doi.org/10.3390/app13031539>
- [60] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 207–212. <https://doi.org/10.18653/v1/P16-2034>

Received 2024-09-12; accepted 2025-04-01