CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Semantic Similarity of LLM Explanations in Software Engineering Tasks

Master's Thesis in Computer science and engineering

Simon Andersson, Qianyuan Wang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

Semantic Similarity of LLM Explanations in Software Engineering Tasks

Simon Andersson, Qianyuan Wang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Semantic Similarity of LLM Explanations in Software Engineering Tasks
Simon Andersson, Qianyuan Wang

Supervisors: Sabina Akbarova, Department of Computer Science and Engineering,
Robert Feldt, Department of Computer Science and Engineering
Examiner: Farnaz Fotrousi, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Simon Andersson, Qianyuan Wang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Large language models (LLMs) are increasingly used to generate natural-language explanations for software engineering (SE) tasks. However, it is still unclear how similar these explanations are across models when the task remains the same. We address this gap in two SE contexts: Python code comprehension and explaining requirements classifications. We use a controlled experimental design to compare GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash under two prompt types: high-level and detailed. In total, the experiment produces 240 explanations. We analyze these explanations from three complementary perspectives: embedding-based semantic similarity, manual reasoning pattern analysis using a predefined codebook, and perceived similarity and quality through a human survey and an LLM-as-a-judge evaluation. Our results show that explanations from different models are often close in meaning, but their similarity varies across model pairs, task types, and prompt conditions. The reasoning pattern analysis shows that models also differ in how they structure their explanations, and these differences depend on the task and prompt type. The quality evaluation of the selected explanation pairs shows that no model is consistently better than the others across all dimensions, and that human and LLM-as-a-judge judgments do not always agree. Therefore, evaluating LLM explanations in SE should combine semantic similarity measures, reasoning-structure analysis, and human or LLM-based quality judgments to support trustworthy use of these models in practice.

Keywords: Large language models, semantic similarity, explanation quality, explanation structure, LLM-as-a-judge

Acknowledgements

We would like to thank our supervisors Sabina Akbarova and Robert Feldt, and our examiner, Farnaz Fotrousi, for their dedication, helpfulness, and guidance for our thesis.

Simon Andersson, Qianyuan Wang, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

Large language models (LLMs) have gained significant global attention due to their strong performance in text generation and reasoning tasks [1]. Within software engineering (SE), LLMs have become a major research focus, as they are increasingly studied and applied to various tasks [1, 2]. In addition to generating outputs, LLMs are increasingly used in SE to provide natural-language explanations for those outputs. For example, they can describe the purpose and behavior of a code snippet in plain language [3, 4]. Although explanations have long been studied in philosophy and explainable artificial intelligence (XAI), what counts as a "good" explanation is still debated, and depends on the context [5, 6]. Since LLMs operate as highly complex black-box systems, their internal reasoning remains non-transparent, requiring users to critically assess whether the explanations they produce are reliable, as they may introduce risks [7]. Work in explainable AI and LLM explainability highlights that explanations play a key role in establishing user trust, especially in professional and safety-critical environments [6, 7].

Getting stable and comparable explanations from LLMs, however, is not trivial. Outputs can vary substantially depending on how prompts are formulated, and poorly designed prompts may lead to poor responses [8]. Research further indicates that LLM behavior can differ across model versions and that outputs may lack reproducibility even when the same prompt is repeated, leading to different responses, or no response at all, despite identical inputs [2]. In some cases, LLM outputs for SE tasks may vary across runs and may include inconsistent reasoning or hallucinations, increasing uncertainty for practitioners who rely on these explanations [9]. These observations highlight that LLM explanations are not stable or reliable by default, indicating the need to investigate the extent to which different LLMs produce similar versus divergent explanations under controlled conditions.

1.1 Problem Description

Most existing studies of LLMs in the SE domain evaluate task-level outcomes such as functional correctness and other metrics, including code quality or runtime performance, rather than comparing similarity across models in their explanations [1, 10, 11]. There is little research about whether different LLMs, when given the same SE tasks and prompts, actually generate explanations that are semantically

similar. In particular, no empirical studies have been found on cross-model explanation similarity and reasoning patterns for SE tasks.

This lack of knowledge affects both trust and the responsible use of LLMs in SE. If LLMs produce inconsistent or opposite explanations across models or repeated runs, it can result in overtrust, undertrust, or confusion, decreasing confidence in using LLMs for SE tasks, which can potentially lead to incorrect or poorly justified changes [9, 12]. Since there is limited evidence-based understanding of whether explanations from different LLMs align or diverge under controlled conditions, developers and researchers cannot reliably evaluate the trustworthiness of these tools, nor create appropriate prompting strategies and usage guidelines. The proposed study addresses this problem by empirically analyzing explanations generated by three different LLMs for the same SE tasks from three complementary perspectives: semantic similarity, reasoning patterns, and perceived similarity and quality.

1.2 Purpose of the Study

The purpose of this study is to provide empirical evidence on whether different LLM families produce explanations that are consistent in meaning, reasoning structure, and perceived quality and similarity when given the same SE task. The study compares explanations generated by three LLM families, namely GPT, Claude, and Gemini, when solving the same SE tasks. The study examines explanation similarity from three complementary perspectives: quantitative semantic similarity based on embedding measures, qualitative reasoning patterns identified through manual coding, and perceived similarity and quality based on human and LLM-as-a-judge evaluations. Together, these perspectives provide a more comprehensive view of how the models compare than any single measure on its own.

1.3 Significance of the Study

This study is useful for both researchers and practitioners who rely on LLMs in SE and care about using them in a responsible way. As these LLMs become a bigger part of everyday development work, the consistency of their explanations can shape how much developers actually trust them [1, 2]. If the reasoning changes from one response to another or feels contradictory, that can affect decisions during tasks such as debugging, testing, and code analysis [9, 12]. By comparing how GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash explain the same SE tasks under the same prompts, and by looking at the results through semantic, structural, and perceived-quality lenses, this study gives a better sense of how these models reason in practice and how consistent their explanations appear from different evaluation angles. The main contribution of this thesis is to examine whether different LLMs explain the same SE tasks in similar or different ways under controlled conditions. For research, this provides empirical evidence on cross-model explanation similarity in SE and can serve as a basis for future studies on explanation reliability. For practice, the findings can help developers be more cautious when interpreting explanations from different

models, rather than assuming that explanations that look similar are necessarily equivalent.

1.4 Research Questions and Hypotheses

In this study, reasoning patterns refer to the structural organization of explanations. This includes whether an explanation builds up to a conclusion through intermediate reasoning steps (inductive) or states the conclusion first and then justifies it (deductive), is grounded in the given input (grounded) or remains more general (generic), and compares the selected answer with alternatives (contrastive) or explains the answer without such comparison (non-contrastive). High-level prompts provide only the basic task instruction without additional guidance, whereas detailed prompts include extra elements such as role descriptions, definitions, examples, structural directions, etc.

This study is designed to thoroughly investigate the semantic similarity, structural reasoning patterns, and perceived similarity and quality of explanations generated by three LLMs: GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash. The investigation focuses on three specific research questions covering different dimensions of similarity:

RQ1: To what extent do different LLMs generate explanations that are semantically similar when solving the same SE tasks?

- **H1.** Semantic similarity differs across model pairs, with some pairs consistently producing higher similarity scores than others under the same task and prompt condition.
- **H2.** Semantic similarity varies across task types, with differences in both overall similarity levels and the distribution across model pairs.

RQ2: What kinds of reasoning patterns (e.g., inductive vs. deductive, grounded vs. generic, contrastive vs. non-contrastive) appear in explanations from different LLMs, and how do these patterns differ when the task remains the same?

RQ3: How do different prompting strategies (high-level vs. detailed) affect the perceived similarity and quality of selected explanation pairs generated by different LLMs?

The three research questions are designed to evaluate cross-model explanation similarity from complementary perspectives. RQ1 examines semantic similarity between explanations under controlled experimental conditions using quantitative embedding-based measures, allowing us to assess the consistency of meaning across models per task and prompt condition. RQ2 focuses on structural differences in reasoning patterns, identifying how each model constructs and organizes its explanations and whether those patterns differ across models under the conditions. RQ3 shifts the focus to perceived similarity and quality in a subset of selected explanation pairs

evaluated through the human survey and LLM-as-a-judge assessment. Since these pairs were a selected subset of the full explanation dataset rather than the complete set, RQ3 is treated as an exploratory analysis of perceived differences across prompt conditions. Together, the three RQs examine explanation similarity from three complementary angles, quantitative, structural, and perceptual, providing a broader picture than any single measure could on its own.

2

Background

This chapter introduces the key concepts that form the foundation of the study. It starts with an overview of large language models and their core capabilities and limitations in Section 2.1. Prompt engineering strategies are discussed in Section 2.2. The nature of explanations and explanation quality is presented in Section 2.3, and the different patterns and structures in explanations are presented in Section 2.4. Section 2.5 covers concepts for evaluating explanations. Finally, the two SE task contexts used to generate the explanations are introduced in Section 2.6 (requirements engineering) and Section 2.7 (code comprehension).

2.1 Large Language Models

2.1.1 What are LLMs?

LLMs are neural language models that are trained on a very large collection of text to learn patterns in natural language and then generate text based on that learned data [13]. A simple way to understand LLMs is as models that process a sequence of tokens. A token can be a single word like “fruit”, part of a word “fr”, “u”, “it”, or just a single character. Given the input token, the model then tries to predict which token is most likely to come next, given the previous context.

The Transformer architecture is the most widely used in today’s LLMs. They are designed to effectively process relationships between words in a sequence. The key mechanism for these transformers is self-attention, which lets the model focus on the most relevant parts of the provided input when producing the next part of the text sequence [14, 15]. Transformer models can be used for a variety of tasks, where architectures such as encoder-only, encoder-decoder, and decoder-only are common [14]. The most relevant architecture for this study is the decoder-only since it handles text generation tasks. It works as such: to predict the next token, it generates the text step-by-step by only looking at the token that has already appeared in context, which is how modern LLMs such as GPT work [14, 15].

2.1.2 Core Capabilities and Limitations of LLMs

LLMs are capable of performing a vast range of language tasks from text prompts alone. In a few-shot setting, GPT-3 has been shown to handle tasks such as answering questions, translating, and filling in missing words in a sentence without the need for frequent updates or task-specific fine-tuning [13]. This suggests that a core capability of these models is in-context learning, meaning adapting to a given task from the instructions or examples that are provided in the prompt. Transformer-based language models can generate more fluent and readable natural language for tasks such as summarizing and dialogue generation [16]. Newer models, in contrast to GPT-3, have shown increased capabilities in language, reasoning, and code tasks [17, 18]. GPT-4 outperforms GPT-3.5 and other LLMs on human-level performance and on NLP benchmarks [17]. MMLU is a common exam benchmark, and Gemini was the first model to reach performance equivalent to human experts [18]. These models also perform well on tasks such as code understanding, a central capability to the explanation tasks examined in this study [17, 18]. A survey on LLM evaluation has further found that the Claude family can be competitive with GPT models on dialogue tasks [19], indicating that the Claude family performs well across general language tasks.

A central limitation of LLMs is hallucination, which is content that is not faithful to the provided source [16]. Truthfulness is also something that is viewed as a concern since language models can provide false statements and mimic common human misconceptions, and unfortunately, just scaling up the system to solve the problem is not the way to make it go away [20]. Earlier models, such as GPT-3, have weaknesses in longer text generation tasks, where outputs can be semantically repetitive, coherence can be lost, earlier statements can be contradicted, or sentences or paragraphs may not logically flow with the rest of the output [13]. These limitations are well worth noting since LLM outputs can seem plausible at first, making them appear trustworthy even though the content is incorrect.

Further limitations of LLMs include being sensitive to the way prompts are written and constructed. In TruthfulQA, they highlight that prompts can change how truthful a model is [20, 21]. Other prompting strategies, such as chain-of-thought, show a clear shift in model performance across different examples and annotations.

2.2 Prompt Engineering

Clear instructions, explicit directions, and contextual examples can improve the structure, content, and consistency of LLM outputs [8]. Prompts are therefore not only task instructions. They also influence how responses are presented and what aspects are emphasized. This matters in this study, since differences in prompt wording may affect how explanations are produced across models.

Common prompting strategies include zero-shot prompting, few-shot prompting, and persona/role prompting [22, 23]. Prompts may also include structural directions, such as required headings or length limits [24, 25]. These strategies influence model

behavior in different ways. Some seem to affect correctness more strongly, while others mainly shape output form and phrasing [26].

In this study, prompt specificity is examined through a comparison between high-level and detailed prompts. This distinction is important because the amount of guidance in a prompt can affect both what is included in an explanation and how it is presented. High-level prompts give only the basic instruction and give the model more freedom in how to respond. In contrast, detailed prompts include extra guidance such as definitions, examples, role assignments, or structural directions, which can make the explanations more explicit, focused, and consistent. The exact prompt templates used in this study are described in the Methods section.

2.3 Explanations and Explanation Quality

In philosophy of science and in AI research, an explanation is usually not seen as just repeating a result. Instead, it is understood as something that shows the reasons or causes behind that result [5, 6]. In AI systems, explanations are valued not only for being correct, but also for making the result easier to understand [7].

Explanation quality can be looked at from two sides: semantic quality and syntactic quality. Semantic quality is about meaning. Based on earlier work on explanations and LLM evaluation, it includes dimensions such as clarity, correctness, completeness, coherence, parsimony, and groundedness [27, 28]. In practice, this means that an explanation should be easy to understand, accurate, logically connected, and focused on what is relevant. It should also stay close to the given context instead of making unsupported claims. Syntactic quality is about form. It concerns whether the explanation stays within the expected length. Together, these perspectives give a broader way of evaluating explanations than correctness alone.

LLM-generated explanations make this distinction even more important. In many cases, LLMs produce an explanation after generating an answer, rather than directly showing the reasoning process behind it [29]. This means that even if an explanation sounds fluent and well-organized, it may not fully reflect what the model based its answer on. Because of this, the quality of the explanation also depends on whether it remains connected to the provided context [28, 30].

2.4 Explanation Patterns and Structures

Explanations can differ in content but also in how they are organized. The same conclusion can be presented in different ways. Two explanations may be equally correct or complete, but still differ a lot in how the reasoning is presented. Earlier studies have suggested several structural patterns that can be used to analyze explanations [31, 32, 33, 34, 35]. One common distinction is between *inductive (Bottom-Up)* explanations and *deductive (Top-Down)* explanations. Inductive explanations show the justification before giving the final conclusion, while deductive explanations give the conclusion first and then explain it [31, 32]. Another distinction is how closely

the explanation is tied to the specific case. *Grounded* explanations refer to concrete evidence from the case, such as a particular code snippet or requirement. *Generic* explanations are broader and could apply to many similar situations [33]. Explanations can also be contrastive or non-contrastive. A *contrastive* explanation justifies an answer by comparing it with an alternative, while a *non-contrastive* explanation only explains the chosen answer [34, 35].

These patterns matter because they may vary across models, tasks, and prompt designs. They may also affect how people perceive an explanation, even when the meaning is similar. It is also important to note that these patterns are not mutually exclusive across all dimensions. For example, a single explanation can be deductive, grounded, and non-contrastive at the same time. It may start with the conclusion, refer directly to specific details in the given code or requirement, and justify the answer without comparing it to an alternative. Looking at structural patterns, therefore, gives another way to understand differences between LLM-generated explanations, beyond correctness or completeness alone.

2.5 Semantic Similarity and Rubric-Based Evaluation

Semantic similarity measures how similar two sentences are in meaning, regardless of the exact words used [36]. Reimers and Gurevych introduced the sentence-transformers framework, which makes it possible to compare sentence meaning efficiently by using cosine similarity between sentence embeddings [37]. To produce the embeddings for this study, it uses the `all-mpnet-base-v2`, which is a sentence transformer that is built on an MPNet backbone [38]. Cosine similarity scores range from -1 to 1, where a score of -1 indicates opposite meaning, while a score of 1 indicates almost identical meaning [39]. In the case of LLM-generated explanations, semantic similarity is useful because two explanations may use different wording while still showing the same understanding of a task. For this reason, a meaning-based measure can capture agreement between explanations better than a surface-level comparison alone. However, no literature has been found that identifies a specific threshold for what counts as a high or low semantic similarity score. Additionally, a meaningful difference between values can be interpreted in many ways and depends a lot on what models, domain, and pieces of text are being compared.

Automated evaluation alone may not fully reflect how human readers judge text quality. Hashemi et al. highlight that human raters often do not completely agree with each other, and even LLM-based evaluators do not always align very well with human judgment when they are asked to give only a single overall quality score [40]. The study continues by showing that evaluation becomes closer to human judgment when it is broken down into several specific dimensions and calibrated against human annotations. For this reason, rubric-based assessment offers an important complement to automated measures when evaluating explanations.

2.6 Task Context: Requirements Engineering

One of the tasks used in this study is requirements engineering, where the models are asked to explain why a given software requirement is classified as either functional or non-functional. Requirements engineering is an essential activity in software development, including collecting, structuring, specifying, and modeling requirements of a system [41]. The key role of requirements engineering is that it produces a project specification, which ideally is a complete set of requirements. This is typically done in the earlier stages of the project and forms the foundation for subsequent development work. In requirements engineering, software requirements are commonly divided into functional requirements (FRs), which define what a system should do, including its functions, services, and behaviors, and non-functional requirements (NFRs), which describe how a system should perform, for example, in terms of performance, security, usability, or other relevant quality attributes [22]. Since FRs and NFRs serve different purposes in requirements engineering, accurately classifying requirements is important for project success. Poor requirements management has been recognized as one of the main causes of software failure, cost overruns, and extended development cycles [42].

FR/NFR classification is also a well-established task in requirements engineering research. The PROMISE NFR dataset provides a benchmark dataset of labeled functional and non-functional requirements, containing 255 functional requirements and 370 non-functional requirements, and has therefore been widely used in empirical studies [22, 43]. In requirements engineering research, FR/NFR classification has also been used to evaluate prompt-based large language models, and previous studies suggest that different prompting strategies can affect model performance on this task [22, 23].

2.7 Task Context: Code Comprehension

The second task used in this study is code comprehension, where the models explain the purpose and behavior of a Python code snippet in natural language. Program comprehension refers to the process of reading, interpreting, and building an understanding of source code, and it is one of the most time-consuming activities in software development. Developers spend a large amount of time on activities related to understanding code [44, 45]. This process involves two broad strategies: bottom-up comprehension, which builds understanding incrementally from individual lines of code, and top-down comprehension, which starts from domain knowledge and connects it to specific code segments [45].

With the rise of LLMs, researchers have started to examine whether automatically generated explanations can support code understanding. Nam et al. found that LLM-based information support can help developers make better progress on code-related tasks than traditional web search, although questions about the quality and reliability of such explanations remain [46]. Code explanation is also an important skill in computing education. Being able to explain code in natural language

is closely related to other programming skills, such as tracing code execution and writing code [3]. This can be seen in "Explain in Plain English" (EiPE) tasks, where students are asked to describe the overall purpose and behavior of a code snippet at a higher level, rather than focusing on implementation details [3, 25]. LLMs have also shown potential as learning support tools in this area. Studies comparing LLM-generated code explanations with student-written explanations have found that LLM explanations often score higher in terms of clarity and accuracy, suggesting that they may be useful as examples to support students in developing their code understanding [3].

Although LLMs can produce fluent and well-structured explanations, this does not mean that the explanations are always correct or truly based on the code itself. Prior work shows that LLM outputs can sound convincing even when they contain logical mistakes, unsupported statements, or small inaccuracies [9, 29]. In SE, this is a real concern because such explanations can affect how users understand, maintain, or check code. For beginners, the risk may be even greater, since they often find it hard to tell whether an LLM-generated explanation is correct [47]. This can make them trust the model too much and think they understand the code better than they do.

3

Related Work

This chapter reviews the related work that provides context for this study and against which our findings are later compared. Section 3.1 reviews previous work on the use of LLMs in SE, the challenges related to the consistency and reliability of their outputs, and broader work on cross-model text generation. Section 3.2 examines how prompt engineering strategies influence LLM performance on SE tasks. Section 3.3 presents common reasoning patterns observed in LLM-generated explanations. Finally, Section 3.4 covers prior work on evaluating explanation quality through both human and LLM-based approaches, including their limitations and reported agreement. The chapter closes with a short discussion of the overall gap in the literature that motivates this study.

3.1 LLMs in SE and Reliability Challenges

Zheng et al. conduct a systematic review of 123 empirical studies and identify several major application areas of LLMs in SE, including code generation, testing, repair, and developer support [1]. Their review suggests that LLMs are now studied and used across a broad range of SE tasks, although most existing studies focus on task performance rather than explanation-level comparison across models.

Reliability and reproducibility remain important concerns in LLM-assisted SE. Jahi'c and Sami report that LLM outputs may vary substantially even under the same setup, raising concerns about output consistency and reproducibility [2]. Sallou et al. similarly argue that LLMs can be non-deterministic and that repeated runs with the same prompt may still produce different outputs [9]. They also highlight the issue of time-based drift, where model behavior may change across versions or over time.

These reliability and reproducibility concerns are especially relevant for explanation generation. Hao et al. highlight that LLM-generated explanations can be wrong even though they appear reasonable and well-structured, and that explanations are often generated after the answer has already been produced, meaning they may not faithfully represent the actual basis of the output [29]. Zhao et al. further emphasize that clear and grounded explanations are important for trust and transparency, especially in professional or safety-critical SE settings [7]. Together, these studies

suggest that reliability in SE depends not only on the generated output itself, but also on the trustworthiness and consistency of the accompanying explanations.

Cross-model text-generation has also been studied outside the SE domain. Smith et al. conducted a large-scale study where they used roughly 5000 different prompts generated by 12 different models, yielding roughly 3 million text outputs, where they measured the inter-LLM similarity using embedding-based measurements such as cosine similarity and word-level edit distance [48]. Their results were mixed, where some model pairs produced similar outputs while others differed substantially. Moreover, their embeddings-based measurements also told different stories about which models were close in style in comparison to meaning. Although their study is more focused on general text generation and not on SE explanation tasks, their study is still closely related to ours. For example, their findings provide a solid and useful baseline to which we can compare our cross-model similarity, which is in a more constrained setting of SE explanation generation.

3.2 Prompt Engineering Strategies

Prior work shows that prompts can influence how LLMs respond to SE tasks. Ekin provides a broad guide to prompt engineering techniques and emphasizes the value of clear instructions, explicit constraints, and contextual examples [8]. The guide suggests that well-structured prompts can improve the structure, content, and consistency of LLM outputs by reducing the gap between user intent and model understanding. Although this work is not limited to SE, it provides a useful starting point for understanding why prompt formulation may affect explanation generation.

Similar findings appear in requirements engineering research. Binkhonain and Alfayez evaluate prompt-based LLMs on several requirements classification tasks and show that prompt strategy can influence model performance [22]. Their results suggest that few-shot prompting generally performs better than zero-shot prompting, and that adding persona prompting or combining persona with chain-of-thought can improve performance in some settings. Ronanki et al. also show that prompt design affects RE tasks, although they focus on prompt patterns rather than traditional prompting techniques [23]. Their findings suggest that performance can vary depending on how the prompt is written.

Prompt effects have also been reported in code explanation settings. Denny et al. highlight how prompt constraints affect explanation quality in "Explain in Plain English" (EiPE) tasks, where students explain code snippets in natural language [25]. Their results suggest that word limits encourage more concise and high-level explanations that better capture the intent of the code. Together, these studies suggest that prompt design affects not only task performance but also the structure and content of generated explanations.

3.3 Reasoning Patterns in Explanations

Besides the semantic quality of explanations, some prior work has looked at how explanations are structured and presented. El-Assady et al. proposed a conceptual model of explanation processes in XAI based on strategies drawn from several different domains [31]. They grouped explanation strategies into inductive, deductive, and contrastive forms, and argued that explanations can be organized differently depending on the level of detail, the target audience, and the goal of the explanation. Their work suggests that explanations can vary not only in what they say, but also in how the reasoning is organized and communicated.

Contrastive explanations have received particular attention in model interpretability research. Jacovi et al. proposed a framework for generating contrastive explanations for neural classifiers by turning input representations into a contrastive subspace that captures only the features that distinguish one decision from another [35]. Using two NLP classification benchmarks, they found that contrastive explanations can offer a more precise view of model decisions. They also argue that non-contrastive explanations may include too many causal factors at once. In contrast, contrastive explanations reduce cognitive load by leaving out factors that do not help distinguish the fact from the foil, the alternative scenario used to clarify the concept.

Wang et al. developed a theory-driven, user-centric XAI framework linking explanation design to human reasoning [32]. They distinguish between causal attribution and causal explanation, and argue that more useful causal explanations are often contrastive, focusing on what happened in relation to an alternative outcome. Their framework also links different forms of reasoning, such as inductive and deductive exploration, to explanation design. Taken together, these studies suggest that reasoning structure matters in how explanations are constructed.

3.4 Human and LLM-Based Evaluation of Explanations

Evaluating the quality of natural language explanations requires more than a single overall judgment. Van der Lee et al. argue that generated text should be evaluated using separate and clearly defined quality criteria rather than a single overall measure, since overall quality is often too abstract to measure directly [49]. Howcroft et al. further show that human evaluation in natural language generation lacks consistent terminology and explicit criterion definitions across studies [50]. Together, these studies highlight the importance of structured, multi-dimensional rubrics when evaluating explanation quality.

To address the scalability limitations of human evaluation, recent work has explored the use of LLMs as evaluators. Zheng et al. study the LLM-as-a-judge paradigm and show that strong LLMs such as GPT-4 can provide scalable approximations of human raters [51]. At the same time, they identify limitations including position

bias, verbosity bias, and weaker reliability on tasks requiring precise reasoning. Hashemi et al. extend this work through LLM-RUBRIC, which evaluates outputs using explicit multi-dimensional rubrics rather than a single overall score [40]. Their results show that combining multiple rubric dimensions and calibrating LLM outputs to individual human judges improves alignment by more than twofold compared with uncalibrated LLM scoring. Together, these studies suggest that LLM-based evaluation is more reliable when it is guided by clearly defined criteria rather than overall impressions.

However, LLM-as-a-judge evaluation remains vulnerable to systematic biases. Wang et al. show that changing the order of two explanations being evaluated can influence the evaluation outcome, highlighting that LLM-based evaluators have a position bias in which explanations they prefer [52]. Panickssery et al. further identify self-preference effects, meaning LLM evaluators tend to favor outputs generated by themselves over outputs from other models [53]. These findings indicate that LLM-based evaluation can be misinterpreted by biases that do not necessarily reflect actual output quality.

Evaluation challenges are especially important in SE tasks. Roy et al. show that automatic metrics commonly used for code summarization are unreliable indicators for human judgment when score differences are small [54]. Lubos et al. further demonstrate that although LLMs can identify quality flaws in software requirements and generate plausible explanations, human verification is still necessary because of false positives and inconsistent assessments [55]. Together, these studies suggest that evaluating explanations in SE requires both domain awareness and careful evaluation design.

Taken together, the studies reviewed in this chapter show that LLMs are widely studied in SE, especially in relation to model performance, reliability, prompting strategies, and evaluation methods. However, less attention has been given to whether different models produce semantically similar explanations when they explain the same SE task, or to how those explanations compare in reasoning structure and perceived quality. This study addresses this gap by comparing explanations generated by three different LLMs for the same SE tasks and analyzing them from three perspectives: semantic similarity, reasoning patterns, and perceived similarity and quality.

4

Methods

The method chapter explains the design and execution of the study. The overall research design and experimental structure are highlighted in Section 4.1. Section 4.2 outlines the experimental setup, including the task types, the rationale behind their prompt templates, and the instruments used to generate, code, and evaluate the explanations. Additionally, Section 4.3 explains how the data were collected for each of the research questions, while Section 4.4 describes how the data were analyzed.

4.1 Research Design

This study followed the laboratory experiment strategy as defined in the ABC framework for SE [56], using a controlled experimental design to examine the semantic similarity, reasoning patterns, and perceived quality and similarity of explanations generated by different LLMs. Figure 4.1 provides an overview of the full research process, from defining the initial version of the research questions through to the final report.

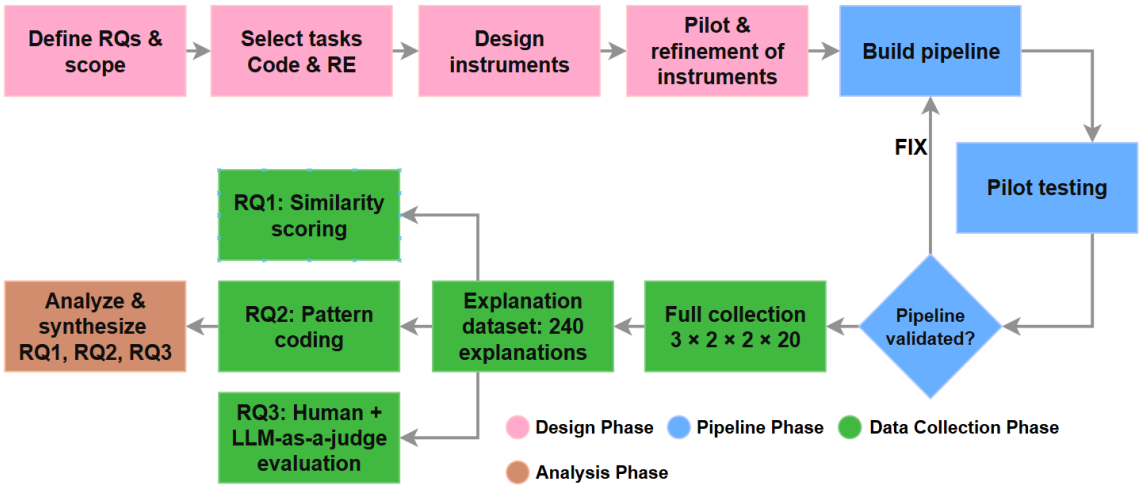

Figure 4.1: Overview of the research methodology

The process was divided into four phases. First, the design phase, where the research questions and scope were defined, the two SE tasks were selected, and the four

instruments (prompt templates, codebook, rubric, and survey) were designed and refined through pilot testing. Second, the pipeline phase, where the explanation generation pipeline was built and validated through pilot testing, with iterative fixes applied until the pipeline produced reliable outputs. Third, the data collection phase, where the validated pipeline was used to generate the full explanation dataset of 240 explanations, followed by RQ-specific data activities: similarity scoring for RQ1, pattern coding for RQ2, and human and AI judgment collection for RQ3. Lastly, the analysis phase, where the results from all three research questions were analyzed and synthesized to answer the research questions.

A controlled experiment was suitable for this research because it allowed us to manipulate the model, the prompt type, and the task type while keeping other conditions, such as the task instances, generation parameters (temperature, max output tokens), and preprocessing procedures, constant.

The experiment followed a $3 \times 2 \times 2$ factorial design. Three independent variables were manipulated: the LLM model (GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash), the prompt type (high-level and detailed), and the task type (code explanation and requirements explanation). The three model families, GPT, Claude, and Gemini, were chosen because they are widely used general-purpose LLMs that are evaluated and applied across a broad range of different tasks [57]. They are also among the model families most commonly studied and applied in SE [1, 2, 57]. This setup enabled direct comparison between models under the same conditions. It also allowed us to examine whether prompt effects differ across models.

The dependent variables corresponded to the research questions. For RQ1, semantic similarity scores were computed using embedding-based cosine similarity and compared across models and tasks. For RQ2, reasoning patterns were identified and manually labeled using a predefined codebook, and their distribution was compared across models and task types. For RQ3, the effect of prompt type was examined by comparing perceived explanation quality and similarity across high-level and detailed prompt conditions. These judgments were collected from a human survey and an LLM-as-a-judge evaluation, both applying a shared quality rubric.

To strengthen internal validity, strict experimental controls were applied. The same task instances were used in all conditions, where an instance refers to a single input item, such as one code snippet in the code task or one requirement statement in the RE task. Generation parameters were aligned as closely as possible. Identical preprocessing procedures were applied before analysis. These controls helped to ensure that observed differences in similarity, structure, or quality were attributable to model behavior, task differences, or prompt variation rather than inconsistencies in the experimental setup.

4.2 Experimental Setup

This section describes the instruments and materials used in the experiments: the two SE tasks and their prompt templates (4.2.1), the explanation length constraints applied to each prompt condition (4.2.2), the codebook for RQ2 (4.2.3), the quality rubric used as the shared evaluation framework for RQ3 (4.2.4), and the survey and LLM-as-a-judge setups that serve as the two primary data sources for RQ3 (4.2.5 – 4.2.6).

4.2.1 Tasks and Prompt Templates

The RE task used the PROMISE NFR dataset [43], from which 20 labeled requirements were selected, covering both functional and non-functional requirements. The selection balanced FRs and NFRs while preserving variation in application domain, requirement type, and length, avoiding the overrepresentation of any single category or domain. In this task, the models were not asked to carry out the classification itself, but instead, asked to explain why the given label was appropriate. This matched the main focus of the study, which was on explanation generation. Two prompt variants were used: a high-level prompt and a detailed prompt. The high-level prompt followed a zero-shot pattern, giving only a short instruction to explain the classification, and served as a baseline consistent with prior work on prompt-based requirements classification [22]. The detailed prompt added a role description ("an experienced requirements analyst"), definitions of FRs and NFRs, a few demonstration examples, and an explicit instruction not to reclassify the requirement. It also allowed the model to choose whatever output structure it judged clearest for the task. This follows the finding that few-shot prompting combined with a persona performs best on FR/NFR classification tasks [22].

The code explanation task used 20 Python instances collected from PyNative, an openly available collection of Python exercises with provided solutions [58]. Instances were picked from the intermediate section to match the intended participant level (entry-intermediate programmers) and to avoid snippets that are either too simple or too advanced, providing code that is both meaningful and understandable to the target audience. As in the RE task, two prompt variants were used. The high-level prompt was intentionally minimal, pairing a short instruction with the code snippet, and served as a baseline, following prior work showing that simple instructions can be a good baseline for SE tasks [24]. The detailed prompt added a no-code-rewriting constraint, an instruction to use language suited to the target audience, and a coverage checklist (purpose, inputs and return value, main logic, and edge cases or assumptions), aligning with findings that effective prompts often include structured guidance matched to the task objective [24]. As in the RE task, the detailed prompt also allowed the model to choose whatever output structure it judged clearest for the task. Both variants targeted high-level code comprehension rather than line-by-line description, consistent with the "Explain in Plain English" (EiPE) approach [25].

The full prompt templates for both tasks and prompt conditions are provided in

Appendix A.1, together with a link to an online document containing the complete set of task instances.

4.2.2 Explanation Length Constraints

Without length constraints, early pilot generations were often long and contained repeated content, so a word range was set for each task to keep explanations comparable and in a natural format. To determine an appropriate range, a small pilot using ChatGPT-5.2 was conducted on five representative instances per task, generating one high-level and one detailed explanation per instance and recording the suggested word range. The ranges were averaged across prompt variants and rounded to a practical interval, resulting in a shared target of 135–150 words for the code task and 100-120 words for the RE task. Smaller pilot runs across models confirmed that the outputs generally followed these constraints without overlap, formatting issues, or obvious misunderstandings.

4.2.3 Reasoning Pattern: Codebook Design

To identify what structural patterns appear in explanations, a codebook was developed based on patterns described in prior literature on LLM-generated explanations. The purpose of the codebook was to label visible structural patterns in the explanations that are difficult to capture through embedding-based similarity measures. A codebook approach was chosen because it fit the manual coding style of RQ2 and provided a clear and familiar framework for systematic labeling.

The codebook consists of 6 patterns, organized into 3 conceptual pairs: inductive and deductive, grounded and generic, and contrastive and non-contrastive. Because explanations could contain multiple structural characteristics at the same time, a single explanation could receive more than one pattern label. The patterns themselves are described in Section 2.4. Each pattern has a short definition and a decision rule, where the decision rule is a one-sentence criterion for when an explanation should be labeled with that pattern. To help with consistent labeling, the codebook also included, specific to each dimension, shorter examples from both the code and RE task. Smaller refinements were made after the pilot test described in Section 4.3.2, mainly to the decision rules for contrastive and non-contrastive, since these dimensions were less clear to apply during the pilot. The full codebook is provided in Appendix A.2.

4.2.4 Explanation Quality: Rubric Design

To evaluate explanation quality in a structured way, a rubric-based scoring framework was developed, covering both semantic and syntactic qualities. Unlike the codebook for RQ2, which focused on structural patterns, the rubric measured perceived explanation quality and was applied in both the survey (Section 4.2.5) and the LLM-as-a-judge assessment (Section 4.2.6).

The rubric consists of 7 quality dimensions:

- **Clarity**: The extent to which the explanation is clear and easy to understand.
- **Correctness**: The extent to which the explanation accurately covers the behavior of the given task.
- **Completeness**: The extent to which the explanation covers key conditions or reasoning elements to justify the output.
- **Coherence**: The extent to which the explanation is logically structured, and the reasoning should flow naturally.
- **Parsimony**: The extent to which the explanation focuses on the relevant steps, while avoiding unnecessary detail, repetition, etc.
- **Groundedness**: The extent to which the explanation is anchored in the provided context rather than relying on unsupported claims.
- **Length compliance**: The extent to which the explanation stays within the given word range.

4.2.5 Human Evaluation Survey Design

The design of the survey followed established guidelines for conducting surveys in SE [59]. Following these guidelines, we first defined the survey objective and the constructs to be measured before designing the questionnaire items. The goal of the survey was to collect human judgments on explanation quality and perceived similarity for outputs produced by different LLMs using different prompting strategies. These judgments were used to address RQ3.

The survey was administered in two task-specific versions, one covering the code explanation task and one covering the RE task. The surveys were designed to be conducted anonymously, and no personally identifiable information was collected. Participation was voluntary, and participants were free to stop the surveys at any time. The target population consisted of individuals with a background in software engineering or requirements engineering who could evaluate LLM-generated explanations. We recruited two groups of participants, students and industry professionals, where the survey included a total of 14 students and six industry professionals. The inclusion of both groups was intended to provide complementary perspectives, combining formal academic training with practical industry experience. The participants were recruited through university channels, student networks, and professional contacts. The questionnaire was designed to allow participants from different backgrounds to complete the evaluation at a convenient time and place.

Students provided perspectives shaped by formal training in programming and requirements engineering, while industry professionals contributed practical experience with source code, design, and requirements documentation. Both groups were familiar enough with Python and the FR/NFR distinction to evaluate the explanations meaningfully.

Participants were balanced across the two task-specific survey versions: 7 students and 3 industry professionals per version, for a total of 10 participants each. The structure, judgment format, and rubric dimensions were identical across versions; only the task content and the domain-experience question differed.

Participants were not restricted to a single experience level so that the sample would not reflect only one professional perspective. The survey began with background questions about the participants' roles (student or industry professional), fields of study, years of experience in their field, years of experience in requirement engineering, software development, or using generative AI, and lastly, experience with the given task, either Python programming or software requirements engineering. This helped us to understand their expertise and overall background within the field of software development and RE.

Each survey version consisted of 6 pairwise comparison items, covering 3 pairs from the high-level prompt condition and 3 from the detailed prompt condition. Each model appeared at least once within those 6 pairs, and pairs were selected based on low semantic similarity scores from RQ1, since these pairs were more likely to highlight noticeable differences in meaning, structure, or quality, in contrast to near-identical pairs that would give participants little to distinguish. Where the lowest-scoring pairs shared the same task instance, the next lowest-scoring pair was used instead, preventing participants from seeing the same input twice. In each item, participants saw the task input alongside two anonymized explanations, labeled as "Explanation A" and "Explanation B", where each pair consisted of explanations generated by two different models for the same instance under the same prompt condition. The generation model and prompt condition were not disclosed in order to prevent potential bias toward a particular model or prompt variant. The code survey presented participants with a Python code snippet alongside the two explanations, while the RE survey showed the requirement statement, its pre-assigned classification (FR or NFR), and the two explanations of that classification. In both versions, explanations were shown in a standardized format, and participants were instructed to judge each explanation solely based on what was presented and how well it matched the task input.

For each explanation pair, participants were provided with 4 types of judgments. First, for each of the 6 rubric dimensions, including clarity, correctness, completeness, coherence, parsimony, and groundedness, participants selected either "Explanation A is better", "Explanation A is slightly better", "Explanation B is better", "Explanation B is slightly better", or "About the same". Each quality dimension had a short definition to support participant understanding. To improve readability for survey respondents, three dimensions were relabeled: coherence as logical flow, parsimony as conciseness, and groundedness as faithfulness to the input. The underlying constructs were unchanged. Second, they rate the overall similarity of the two explanations on a 1–5 scale, ranging from very different to almost identical. The 1–5 scale was chosen because it provides sufficient range to capture variation while remaining simple enough to reduce response fatigue [60, 61]. Third, they indicate an overall preference by selecting "Explanation A", "Explanation B", or "No clear

preference". Finally, a 1–5 scale to indicate whether the explanations provided in the survey are helpful or not.

Each survey version ended with an optional open-ended question asking participants to reflect on what factors shaped their quality assessments, similarity ratings, or overall preferences. The responses complemented the rubric-based comparisons when interpreting the results. The complete survey for each task version is provided as online documents in Appendix A.3.

4.2.6 LLM-as-a-judge Setup

The LLM-as-a-judge evaluation was designed to mirror the two survey versions as closely as possible. The prompt structure followed the same format as the human survey: the initial instruction prompt contained all task instructions and quality criteria, and the model was asked to confirm when it was ready before any pairs were presented. The next six evaluation prompts each contained the task input, either a Python code snippet or a requirement statement with its classification, alongside "Explanation A" and ´´Explanation B" for that model pair. The final reflection prompt posed the same open-ended reflection question included at the end of the human survey. The code and RE evaluations were conducted in two separate sessions to mirror the task-specific structure of the survey. To prevent any carry-over between sessions, the model's memory and access to previous conversations were disabled, ensuring each session started from a clean state. The evaluation model used was Claude Opus 4.7 with adaptive thinking mode enabled, as it represents one of the most capable models at the time of the study [62]. This approach acted as a complement to the survey and assessed whether the quality patterns observed in human judgments were consistent with those produced by an AI evaluator. The full prompt templates for both tasks used for the LLM-as-a-judge evaluation are provided as online documents in Appendix A.4.

4.3 Data Collection

4.3.1 Explanation Generation (RQ1)

A Python-based pipeline was constructed to produce explanations for all combinations of tasks, prompt variants, and models. For the code task, the source code of each selected instance was inserted into the corresponding prompt template; for the RE task, the requirement statement and its pre-assigned classification were inserted. This ensured that, within each experimental condition, all models received the same content. The explanations were generated in late February to early March.

Explanation generation was controlled through the task type, prompt variant, model, and the generation parameters (temperature = 0.0, maximum_output_tokens = 450). The models that were used to generate the explanations were GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash. A shared interface was created so the same experimental procedure could be applied across all model runs since all of the models

were accessed through different providers. To improve robustness, outputs were appended incrementally to JSONL files. This reduced the risk of losing previously generated records if a later generation would fail, supporting traceability, reproducibility, and error inspection. The pipeline retried failed API calls up to three times before recording the generation as unsuccessful and moving on to the next instance.

For each of the generated explanations, the pipeline stored both the output text and associated metadata, including run identifier, task, prompt variant, model, provider, temperature, maximum output length, instance identifier, response status, provider response identifier, and SHA-256 fingerprint of the rendered prompt. For readability, the explanations were also stored in JSON format.

4.3.1.1 Preprocessing

Preprocessing was intentionally limited to preserve the original content of the model-generated explanations as closely as possible. Only the outputs that were generated successfully were included in the semantic similarity analysis. Before similarity computation, each of the texts was normalized by trimming leading and trailing whitespace. Additionally, edge cases such as empty text were handled explicitly, so if both texts were empty, similarity was treated as maximal, whereas for cases where only one text was empty, the similarity was treated as zero.

No further normalization or preprocessing was applied (no lowercasing, lemmatization, stop-word removal, stemming, or manual rewriting), so that the semantic similarity analysis reflected the explanations as originally generated. A separate HTML-based viewer supported manual inspection of the outputs, but was not part of the preprocessing used for RQ1.

4.3.2 Pattern Coding (RQ2)

Before labeling started, a pilot test was conducted to check whether the pattern definitions and decision rules were sufficiently clear and whether any refinements were needed. Each explanation could receive one label from each conceptual pair (e.g., “inductive, grounded, contrastive”), and contradictory labels within the same pair were not allowed.

The pilot set consisted of 16 explanations, distributed evenly across task type, prompt type, and model. The two authors labeled the pilot explanations independently and then compared the results in order to identify unclear cases and revise the codebook where necessary.

The full dataset contained 240 explanations. Due to the manual effort required for coding, a subset of 120 explanations was selected for labeling. The subset was constructed to maintain balanced coverage across model, task type, and prompt type. More specifically, 40 explanations were selected from each model, with each set of 40 consisting of 20 code explanations and 20 RE explanations. Within each

task type, 10 explanations were selected from the high-level prompt condition and 10 from the detailed prompt condition. To reduce manual selection bias, the specific explanations within each subgroup were chosen using an AI-assisted random selection procedure rather than being manually selected.

The codebook labeling was carried out in early April, during which the two authors labeled the explanations independently and then compared the coding results. Disagreements were recorded for each coding dimension before discussion. Some disagreements reflected missed details during the initial coding, while others required additional discussion because certain explanations could reasonably be interpreted in more than one way. After discussion, agreement was reached on all contested cases, and the finalized labels were then used for analysis. These disagreement patterns were documented during the coding process to show which dimensions were easier to apply consistently and which required more interpretive judgment.

4.3.3 Human and LLM-as-a-judge Evaluation (RQ3)

The data collection for RQ3 revolved around two complementary instruments: a survey and an LLM-as-a-judge evaluation. The instruments were applied to the same subset of explanation pairs of low semantic similarity generated by the pipeline, which ensured that the results could be compared directly (see Section 4.2.5). This subset was selected manually and consisted of 12 explanation pairs, 6 for the code task and 6 for the RE task, with each task-specific survey version designed for a completion time of roughly 20–25 minutes.

Both the survey and the LLM-as-a-judge evaluation used the rubric (defined in Section 4.2.4) as the shared evaluation framework. 6 out of 7 rubric dimensions were used to collect the judgments: clarity, correctness, completeness, coherence, parsimony, and groundedness. Length compliance was excluded from both approaches because the prompts were not shared, so this dimension could not be scored without the prompt's context. Instead, this dimension was directly measured against the prompt specifications. A pairwise comparison format was used for both the survey and the LLM-as-a-judge evaluation, where two explanations from different models under the same instance and prompt condition were compared.

The survey was published in mid-April and was open for roughly one week. It was administered through Google Forms, where each of the participants received a link to the task-specific version they had been assigned to. All the responses were directly stored within Google Forms. The full set of respondent answers for both task versions is available as online documents in Appendix A.3.

The LLM-as-a-judge evaluation was run during the same period the survey was open. It was run once per task, resulting in two separate sessions as described in Section 4.2.6. For each evaluated pair, the LLM-as-a-judge's responses were stored in a text document across the six rubric dimensions, the similarity rating, the overall preference, and the helpfulness rating, along with the answer to the open-ended reflection question presented in the final prompt message of the session. The

answers from the LLM-as-a-judge for both tasks are available as online documents in Appendix A.4.

4.4 Data Analysis

4.4.1 Semantic Similarity Analysis (RQ1)

The purpose of the analysis for RQ1 was to determine how similar the generated explanations were from different LLMs in meaning when solving the same SE task instance, and how similarity varied across task types. The analysis used the generated explanations from the pipeline (Section 4.3.1), covering all task instances, models, and prompt conditions.

The analysis focused on pairwise cosine similarity between explanations where the model was the varying factor, while task instance and prompt template were held constant. The results were grouped by task type, prompt condition, and model pair, and summarized using mean and standard deviation to capture overall similarity levels and their variation across conditions.

Before any statistical tests were chosen, the similarity data needed to be checked for normality. This was done by using the Shapiro-Wilk test [63]. Normality was rejected ($p < 0.05$) in 7 of the 12 individual model-pair distributions, and in all 4 task–prompt conditions when the scores from all model pairs were combined (see Table A.1 in Section A.5). This, combined with the small sample (20 instances per condition), justified the use of a non-parametric test such as the Friedman test for cross-model comparison and the Mann-Whitney U test for cross-task comparison.

The descriptive patterns found were then evaluated statistically against the two hypotheses (H1 and H2) for RQ1 introduced in Section 1.4. The statistical evaluation of H1 used the Friedman test, a non-parametric test for comparing three or more related groups [63]. The test was separately applied within each of the four experimental conditions, including code high-level, code detailed, RE high-level, and RE detailed, where similarity scores were compared across the three models and the 20 task instances. The chosen significance level for the tests was $\alpha = 0.05$, the most common threshold in empirical SE research [64]. Post-hoc analysis was used for the conditions that reached significance for the Friedman tests. The post-hoc analysis used pairwise Wilcoxon signed-rank tests to determine which specific model pairs differed. Bonferroni correction was applied to control the family-wise error rate across the three pairwise comparisons. The significance threshold was lowered to $\alpha_{corrected} = 0.0167$. Effect sizes were reported as Kendall's W for the Friedman test and rank-biserial correlation r for the Wilcoxon tests. Conventional interpretation thresholds were also applied of 0.1, 0.3, and 0.5 for small, medium, and large effects.

H2 was evaluated using the Mann-Whitney U test [64]. It compares the distributions of similarity scores between the code and RE task within each prompt condition. The same significance level was applied here, $\alpha = 0.05$. Effect size used rank-biserial

correlation (r), following the same thresholds of 0.1 (small), 0.3 (medium), and 0.5 (large) as above.

The results were presented using summary tables, bar plots with error bars, and example pairs illustrating low and high similarity scores (see Figure A.1 and Figure A.2 in Appendix A.1.5.1). The interpretation focused on which model pairs produced more similar explanations, how the patterns differed between the two tasks, and whether these patterns remained stable across high-level and detailed prompts.

4.4.2 Reasoning Pattern Analysis (RQ2)

The purpose of the analysis was to examine how structural output patterns in explanations differed across tasks and models, and whether these patterns were stable across prompt conditions.

Unlike the semantic similarity analysis in RQ1, the reasoning pattern analysis was based on manual coding using the predefined codebook described in Section 4.2.3, rather than embedding-based measures.

After the explanations had been labeled, inter-rater agreement between the two authors was computed within each of the three dimensions using Cohen's kappa, based on the labels that were assigned before any disagreement discussion took place. Kappa was computed over all 120 labeled explanations per dimension rather than separately by model, tasks, or prompt condition, since the per-condition subsets would be too small to produce a stable estimate. Raw percentage agreement was also reported. The finalized labels were then processed in Python to aggregate the results into counts and percentages. This made it possible to examine how often each pattern appeared and to compare these distributions across models, task types, and prompt conditions.

The results were presented using frequency tables, a task-level visualization of pattern distribution, and selected example explanations to illustrate the most relevant patterns (see Figure A.3 and Figure A.4 in Appendix A.1.5.1). In addition, a short mismatch summary was included in the results to report where disagreement occurred before discussion and final agreement.

4.4.3 Perceived Explanation Similarity and Quality Analysis (RQ3)

The purpose of this analysis was to examine how different prompting strategies (high-level vs. detailed) influenced perceived quality and similarity of explanations and whether human judgments agreed with LLM-as-a-judge judgments. The analysis drew from the response collected in the surveys and LLM-as-a-judge evaluation as described in Section 4.3.3. The analysis was divided into four parts: survey analysis, LLM-as-a-judge analysis, the agreement comparison between the respondents in the survey and the LLM-as-a-judge, and length compliance.

In all result figures and tables, the anonymized "Explanation A" and "Explanation B" labels used during the survey and the LLM-as-a-judge evaluation are replaced with the name of the model that produced each explanation.

For the survey, criterion-level responses across the six rubric dimensions were visualized as stacked horizontal bar charts, showing the share of responses in each of the five options for each pair, grouped by task type and prompt condition. Perceived similarity and helpfulness were summarized using descriptive statistics (mean and standard deviation (SD)) and presented in a table. Overall preference was visualized as stacked horizontal bar charts showing the majority preference within each pair. Since the prompt condition was held constant within each pair, the prompt effect was identified by comparing preference and rating patterns across the high-level and detailed pairs. Since explanation pairs were selected based on low similarity rather than balanced model coverage, model effect was observed and noted for each pair and viewed when analyzing the results.

The LLM-as-a-judge evaluation followed the same four judgment types as the survey. Since the judge produced one rating per pair, criterion-level preference was visualized as a heatmap showing the judge's rating for each pair and rubric dimension, while similarity, preference, and helpfulness ratings were reported in a table directly from the ratings given by the judge, rather than as a proportion across raters.

For the agreement analysis, the survey results and the LLM-as-a-judge results were compared across the six rubric dimensions and visualized as a side-by-side heatmap showing matches and mismatches for each pair and dimension. Agreement was counted across the six rubric dimensions for each of the six pairs per task, giving 36 comparisons per task. Similarity, preference, and helpfulness ratings were also compared in parallel between the two sources in a table. The purpose was not to validate one source over the other, but rather to examine where human and automated evaluations agree and where they disagree.

Length compliance was analyzed for each task separately. For each model and prompt condition, the mean and standard deviation of explanation word counts were computed relative to the task-specific target range defined in Section 4.2.2. Length compliance was reported in a table as the counts of explanations within, below, and above the range.

5

Results

This chapter presents the findings for each research question. Section 5.1 reports the semantic similarity results for RQ1, organized by model pair and task type differences. Section 5.2 reports the reasoning pattern results for RQ2, organized by task type, model differences, and coding mismatches. Section 5.3 reports the RQ3 results from the survey, the LLM-as-a-judge evaluation, length compliance, and the agreement between the human and LLM-as-a-judge judgments.

5.1 RQ1: Semantic Similarity

This section presents the pairwise semantic similarity scores between explanations generated by the three models across all experimental conditions. Semantic similarity is calculated through cosine similarity on sentence embeddings from the model `all-mpnet-base-v2`. Similarity is computed for each pair of explanations generated for the same task instance and prompt type. Section 5.1.1 highlights the differences between model pairs, while Section 5.1.2 focuses on the differences in task type.

As mentioned in Section 2.5, cosine similarity does not have a fixed threshold for what counts as high or low scores. The scores in this study are therefore interpreted relative to the observed range rather than to a specific threshold. Across all conditions, the scores are fairly close to each other, ranging from 0.609 for the lowest-scoring pair to 0.948 in one of the highest. Most of the conditions fall in a range between 0.85 and 0.92. The pairs within each condition vary by about 0.05 on average (the standard deviation), so a small gap between two mean similarities, such as 0.89 vs. 0.91, is smaller than this normal variation and is therefore treated as descriptive rather than meaningful. The differences that this study counts as meaningful are therefore the ones the statistical tests confirm as significant, with the accompanying effect sizes indicating how large each difference is (see Table 5.2). To illustrate what high and low similarity scores look like in practice, examples of the lowest- and highest-scoring pairs are shown in Figure A.1 and Figure A.2 in Appendix A.1.5.1.

5.1.1 Similarity Across Model Pairs

Table 5.1: Pairwise cosine similarity scores across all model pairs, tasks, and prompt conditions. Values are reported as mean (SD) across 20 instances per condition. HL = high-level prompt; Det. = detailed prompt.

Model Pair	Code HL	Code Det.	RE HL	RE Det.
GPT vs. Claude	0.897 (0.048)	0.913 (0.046)	0.919 (0.028)	0.881 (0.055)
GPT vs. Gemini	0.899 (0.055)	0.917 (0.033)	0.875 (0.079)	0.854 (0.078)
Claude vs. Gemini	0.868 (0.068)	0.885 (0.059)	0.887 (0.059)	0.816 (0.096)
Overall	**0.888 (0.058)**	**0.905 (0.049)**	**0.894 (0.061)**	**0.850 (0.081)**

Table 5.2: Summary of the statistical analysis for RQ1 hypotheses. Friedman tests evaluate H1 within each of the four experimental conditions (Code HL, Code Det., RE HL, RE Det.; HL = high-level prompt, Det. = detailed prompt), with post-hoc Wilcoxon signed-rank tests using Bonferroni correction. Mann-Whitney U tests evaluate H2 by comparing the code and RE tasks within each prompt condition. Friedman and Mann–Whitney p-values are evaluated against $\alpha = 0.05$; post-hoc Wilcoxon p-values are evaluated against the Bonferroni-corrected threshold $\alpha_{corrected} = 0.0167$. Reported p-values are uncorrected (raw). SSD = statistically significant difference; GPT = GPT-4.1; C = Claude Sonnet 4; G = Gemini 2.5 Flash.

Hyp.	Condition	Comparison	Test	Stat.	p-value	Effect size	SSD*
H1	Code HL	All pairs	Friedman	$\chi^2 = 2.10$	0.350	W = 0.05 (S)	NO
		C×GPT vs. C×G	Wilcoxon	$W = 57.0$	0.076	r = 0.46 (M)	NO
		C×GPT vs. GPT×G	Wilcoxon	$W = 83.0$	0.430	r = −0.21 (S)	NO
		C×G vs. G×GPT	Wilcoxon	$W = 58.0$	0.083	r = −0.45 (M)	NO
	Code Det.	All pairs	Friedman	$\chi^2 = 7.50$	0.024	W = 0.19 (S)	YES
		C×GPT vs. C×G	Wilcoxon	$W = 41.0$	0.015	r = 0.61 (L)	YES
		C×GPT vs. GPT×G	Wilcoxon	$W = 102.0$	0.927	r = −0.03 (S)	NO
		C×G vs. G×GPT	Wilcoxon	$W = 26.0$	0.002	r = −0.75 (L)	YES
	RE HL	All pairs	Friedman	$\chi^2 = 12.90$	0.002	W = 0.32 (M)	YES
		C×GPT vs. C×G	Wilcoxon	$W = 17.0$	<0.001	r = 0.84 (L)	YES
		C×GPT vs. GPT×G	Wilcoxon	$W = 27.0$	0.002	r = 0.74 (L)	YES
		C×G vs. G×GPT	Wilcoxon	$W = 96.0$	0.756	r = 0.09 (S)	NO
	RE Det.	All pairs	Friedman	$\chi^2 = 7.30$	0.026	W = 0.18 (S)	YES
		C×GPT vs. C×G	Wilcoxon	$W = 18.0$	<0.001	r = 0.83 (L)	YES
		C×GPT vs. GPT×G	Wilcoxon	$W = 73.0$	0.245	r = 0.31 (M)	NO
		C×G vs. G×GPT	Wilcoxon	$W = 48.0$	0.033	r = −0.54 (L)	NO
H2	HL prompt	Code × RE	Mann-Whitney	$U = 1670.0$	0.497	r = 0.07 (S)	NO
	Det. prompt	Code × RE	Mann-Whitney	$U = 2605.0$	<0.001	r = −0.45 (M)	YES

S = Small; M = Medium; L = Large.

Table 5.1 shows that pairs involving GPT are the strongest performers throughout, while Claude vs. Gemini is consistently the weakest, except for the RE high-level condition. Among the two GPT-involved pairs, no clear dominance is evident, with means ranging from 0.854 to 0.919 across conditions. The main exception to their

closeness is distributional, where GPT vs. Claude tend to show tighter variance, particularly for the RE high-level condition, where it reaches its peak of 0.919 (SD = 0.028). Claude vs. Gemini, by contrast, not only sits consistently below the other pairs but also shows the widest spread, suggesting that the distance between these two models is less stable across instances.

Table 5.2 reports the statistical evaluation of the two hypotheses. For H1, each Friedman test reveals whether the three tested model pairs differ in similarity within a single condition. A p-value below 0.05 indicates that at least one pair behaves differently. This holds for code detailed ($p = 0.024$), RE high-level ($p = 0.002$), and RE detailed ($p = 0.026$), but not for the code high-level ($p = 0.350$), where the three pairs cannot be distinguished, hence not statistically significant. The Kendall's W values show this separation is small for code detailed and RE detailed ($W = 0.19$ and 0.18) and medium for RE high-level ($W = 0.32$). The post-hoc Wilcoxon reveal which pairs differ by comparing the three model-pair similarity columns against one another. Their p-values are evaluated against the Bonferroni-corrected threshold of 0.0167, deciding if they are statistically significant or not. In code detailed, the Claude vs. Gemini pair shows lower similarity than both GPT-involved pairs, while the two GPT-involved pairs do not differ from each other ($p = 0.927$) (see Table 5.1). Under both conditions for the RE task, the Claude vs. GPT pair separates the most from the other pairs, with large effect sizes ($r = 0.84$ and 0.83). H1 is therefore supported in three of the four conditions, with no significant pair differences under the code high-level prompt.

5.1.2 Similarity Across Task Types

While the model pair ordering described above holds across both tasks, the two tasks differ in two important ways: the direction of the prompt effect and the stability of that ordering under the detailed prompt condition. In the code task, similarity is modestly higher under the detailed prompt in comparison to the high-level prompt (0.905 vs. 0.888), suggesting that the relative ranking of model pairs remains stable across both conditions. However, the RE task behaves differently, where the overall similarity is lower under the detailed prompt than under the high-level prompt (0.850 vs. 0.894), hence the ranking of model pairs becomes less stable. The sharpest shift comes from Claude vs. Gemini, which drops from 0.887 to 0.816, with the lowest mean and highest SD (0.096) observed anywhere in the dataset, moving it from second to last place under the detailed prompt. GPT vs. Gemini, by contrast, is more stable at 0.875 and 0.854. The RE task also shows higher variance throughout compared to the code task, indicating that model agreement is more instance-dependent in this condition.

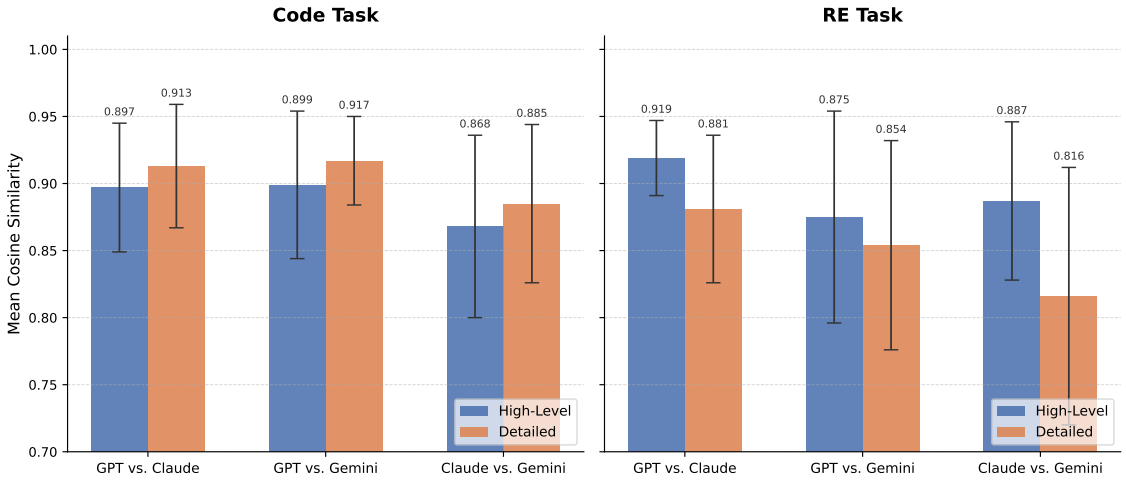

Figure 5.1: Pairwise cosine similarity across the three model pairs for code task (left) and RE task (right). Values are mean (SD) across 20 instances per condition; error bars show ±1 SD. The y-axis is zoomed into the high-similarity range for readability.

Figure 5.1 visualizes the results from Table 5.1. The error bars are generally wider for the RE task, especially under the detailed prompt, suggesting greater variation in similarity in that condition.

For H2, Table 5.2 highlights that the Mann-Whitney test indicates a significant difference between the two tasks under the detailed prompt ($p < 0.001$, $r = -0.45$). The high-level prompt does not follow this pattern ($p = 0.497$, $r = 0.07$). H2 is therefore supported by the detailed prompt, but not for the high-level condition, where overall similarity levels cannot be statistically distinguished between the two tasks. However, the variance and ranking instability described above are present under both prompt conditions, indicating that task-level differences extend beyond the mean similarity captured by the Mann-Whitney test.

5.2 RQ2: Reasoning Patterns

This section presents the distribution of reasoning patterns identified in the labeled 120 explanations, coded along three conceptual pairs: inductive vs. deductive, grounded vs. generic, and contrastive vs. non-contrastive. Labeling was done manually using the predefined codebook described in Section 4.2.3. Section 5.2.1 reports the differences between models, Section 5.2.2 reports the differences in task types, and lastly, Section 5.2.3 highlights the coding disagreements between the two coders observed during the labeling process.

All 120 explanations are labeled as deductive, where every RE and code task explanation begins by stating the classification before justifying it (e.g., “This requirement is appropriately classified as a Non-functional Requirement (NFR)”...), confirming the deductive labeling shown in Figure 5.2. Full examples of RE explanations with a

deductive pattern can be viewed in Figure A.3 and Figure A.4 in Appendix A.1.5.1.

5.2.1 Reasoning Patterns Across Models

Table 5.3: Reasoning pattern distribution by task, prompt condition, and model in the labeled subset of 120 explanations. Values are reported as n count.

Task	Prompt	Model	n	Deductive	Inductive	Grounded	Generic	Contrastive	Non-contrastive
Code	High-level	GPT-4.1	10	10	0	10	0	0	10
		Claude Sonnet 4	10	10	0	10	0	0	10
		Gemini 2.5 Flash	10	10	0	10	0	0	10
	Detailed	GPT-4.1	10	10	0	10	0	0	10
		Claude Sonnet 4	10	10	0	10	0	0	10
		Gemini 2.5 Flash	10	10	0	10	0	0	10
RE	High-level	GPT-4.1	10	10	0	8	2	9	1
		Claude Sonnet 4	10	10	0	9	1	9	1
		Gemini 2.5 Flash	10	10	0	8	2	8	2
	Detailed	GPT-4.1	10	10	0	8	2	10	0
		Claude Sonnet 4	10	10	0	9	1	10	0
		Gemini 2.5 Flash	10	10	0	8	2	10	0

As Table 5.3 highlights, in the code task, the pattern distribution is completely consistent across all models and both prompt conditions. GPT, Claude, and Gemini all remain deductive, grounded, and non-contrastive, indicating highly stable reasoning patterns in this task.

In contrast, Table 5.3 shows that the RE task has some variation, mainly in the grounded and generic dimension and the contrastive and non-contrastive dimension. In the grounded and generic, the distribution remains unchanged across the two prompt conditions: Claude is grounded in 9 of the cases, while GPT and Gemini are grounded in 8 out of 10. Generic explanations appear only in the RE task, with 1 case for Claude and 2 for both GPT and Gemini under both prompt conditions. In the contrastive/non-contrastive dimension, the RE task already shows a clear contrastive tendency under the high-level prompt, with 9 cases for Claude and GPT and 8 for Gemini. Under the detailed prompt, this tendency becomes even stronger, as all three models reach 10 contrastive explanations and 0 non-contrastive explanations. Overall, in comparison to the code task, the RE task not only includes a certain proportion of generic explanations but also relies much more on contrastive explanations. Contrastive patterns are dominant under both prompt conditions, and this tendency is even stronger under the detailed prompt.

5.2.2 Reasoning Patterns Across Task Types

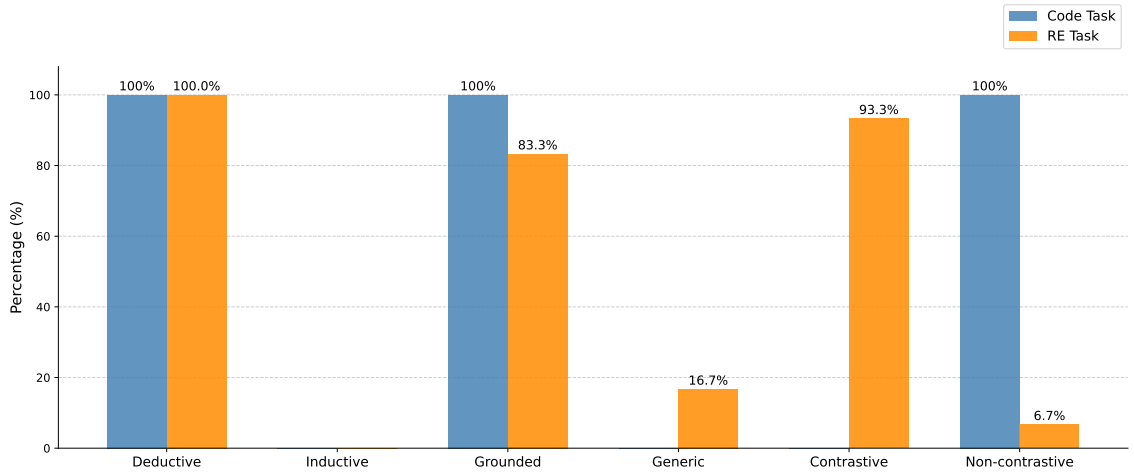

Figure 5.2: Reasoning pattern distribution by task type in the labeled subset of 120 explanations. Values show the percentage of explanations assigned to each pattern dimension in the Code and RE tasks.

From the task perspective, Figure 5.2 shows the clear structural difference. In the code task, all explanations are labeled as grounded and non-contrastive. In the RE task, by contrast, all explanations are labeled as deductive, grounded, and non-contrastive, but the distribution on the other two dimensions is clearly different. The proportion of grounded explanations drops to 83.3%, while 16.7% are labeled as generic. A contrast between a grounded and generic explanation can be viewed in Figure A.3 where the left grounded explanation from GPT anchors its reasoning in “standard virus protection software” and “compliance with organizational security policies”, while the right generic explanation from Gemini reframes the same requirement more abstractly (e.g., “a quality attribute” and “a behind-the-scenes operational characteristic”), with no clear anchor in the given requirement. More notably, contrastive explanations account for 93.3% of the RE task, whereas only 6.7% are non-contrastive. A clear distinction between contrastive and non-contrastive explanations can be viewed in Figure A.4. GPT’s contrastive explanation justifies the NFR label by setting it against the alternative ("...rather than a specific feature or function the system must perform"), whereas Gemini’s non-contrastive explanation for a different instance only explains why the given requirement is functional, without comparing it to the NFR alternative.

5.2.3 Coding Disagreements

Overall, inter-rater agreement is high, but not consistent across the three dimensions. The inductive vs. deductive dimension produces no mismatches, where all 120 explanations are labeled as deductive, giving 100% agreement. Cohen’s kappa is therefore undefined for this dimension since there is no variation. The grounded vs. generic dimension produces 10 mismatches, all in the RE task. This corresponds to a 91.7% agreement and a Cohen’s kappa of 0.54. Despite high agreement, the lower

kappa suggests a skewed distribution of labels, where the majority of explanations are labeled grounded. The contrastive vs. non-contrastive dimension produces 4 mismatches (3 in code, 1 in RE). This corresponds to a near-perfect 96.7% agreement between authors and a Cohen's kappa of 0.93, substantially higher than in the grounded vs. generic dimension. For the contrastive vs. non-contrastive dimension, the labels are more evenly split, meaning less agreement is expected by chance, hence producing a higher kappa than the skewed label distribution seen in the grounded vs. generic dimension, even though they have a similar level of raw agreement. Overall, coding difficulty is not consistent across dimensions: the grounded vs. generic dimensions are the least clear to apply consistently, whereas the inductive vs. deductive dimensions are the most stable.

5.3 RQ3: Perceived Explanation Similarity and Quality

This section presents the results on how high-level and detailed prompting affect perceived explanation quality and perceived similarity across the three models. The results are based on two complementary data sources: human evaluation through two task-specific surveys and an LLM-as-a-judge evaluation. Both evaluations use the same setup, including the same rubric dimensions and the same subset of explanation pairs.

Section 5.3.1 reports the survey results collected from students and industry professionals. Section 5.3.2 reports the corresponding LLM-as-a-judge results. Section 5.3.3 compares the human and LLM-as-a-judge results, highlighting where the two sets of judgments align and where they differ. Finally, Section 5.3.4 presents length compliance, measured against the word range specified in each task-specific prompt, to show how closely each model follows the given limit across both tasks and prompt conditions.

5.3.1 Survey: Human Evaluation

The survey includes 20 valid responses, including 14 students and 6 industry professionals. The students come from several computing-related programmes: Computer Science (5), Software Engineering and Management (4), Software Engineering and Technology (2), Game Design (2), and Game Design and Technology (1). Most of them are master's students (11 out of 14). Overall, the student group has some background in requirements engineering, software development, and generative AI. Most students report 1–3 years of experience in requirements engineering and software development, and generative AI use is also common, with 9 students reporting 1–3 years of experience. Among the 7 students who completed the code survey, 4 rated their Python programming experience as beginner and 3 as intermediate. Among the 7 students who completed the RE survey, 6 rated their software requirements engineering experience as beginner and 1 as intermediate.

All 6 industry participants are software developers. Compared with the students,

they generally have more software development experience, with 4 reporting 4–6 years. All industry participants have some experience with generative AI, most commonly 1–3 years. Their requirements engineering experience is more mixed, ranging from no experience to 4–6 years. Among the 3 industry professionals who completed the code survey, all rate their Python programming experience as intermediate. Among the 3 who completed the RE survey, 2 rated their software requirements engineering experience as beginner, and 1 as intermediate.

This section first reports the human survey results for perceived quality, overall preference, similarity, and helpfulness in the code and RE tasks, comparing the high-level and detailed prompt conditions.

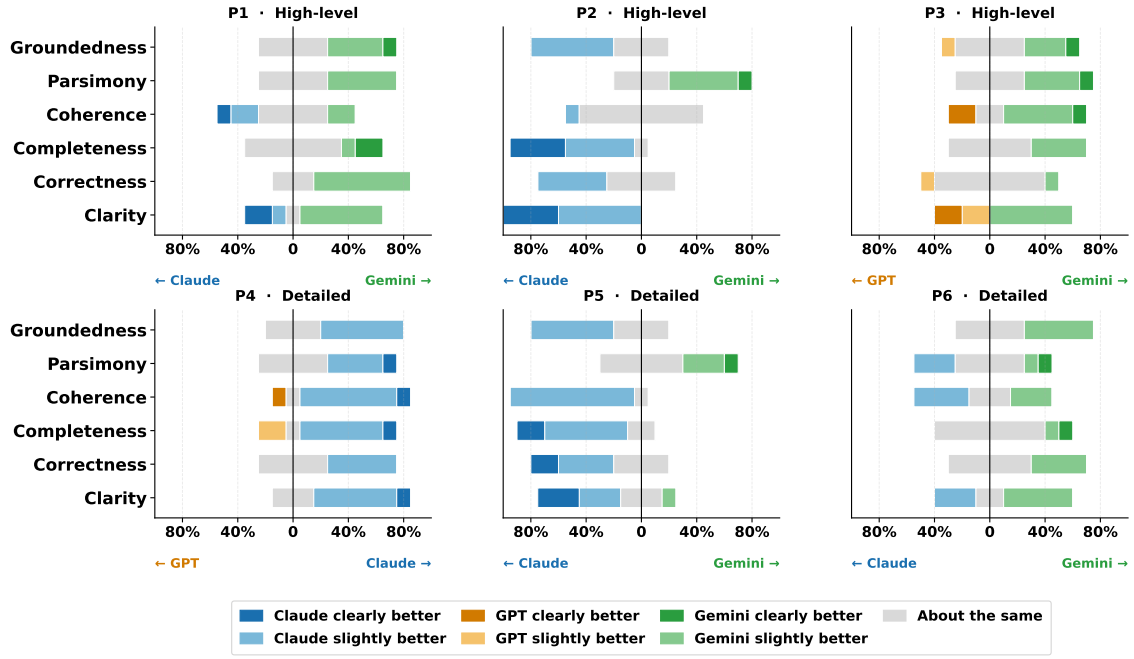

Figure 5.3: Quality dimension judgments from the code survey. Each subplot shows one explanation pair, with high-level prompts on the top row and detailed prompts on the bottom row. P = Explanation Pair

Figure 5.3 shows that under the high-level prompt, Gemini receives more support in two of the three comparisons. In Pair 1 and Pair 3, participants favor Gemini across several criteria, particularly parsimony, completeness, and groundedness. However, this pattern is not consistent across all high-level comparisons. In Pair 2, Claude receives more support for clarity, correctness, completeness, and groundedness, while Gemini is mainly favored for parsimony. Thus, Gemini tends to receive stronger support under the high-level prompt, but this does not apply uniformly across all model pairs.

Under the detailed prompt, the pattern shifts more toward Claude (see Figure 5.3). Pair 4 and Pair 5 show clear support for Claude across several criteria, especially clarity, completeness, coherence, and groundedness. However, the detailed prompt

also does not produce a consistent preference for one model. Pair 6 is more mixed: participants favor Gemini for clarity, correctness, and groundedness, while Claude receives more support for parsimony and coherence.

Across both prompt conditions, parsimony appears as one of the more variable criteria. Gemini is often favored for parsimony, even in comparisons where another model receives stronger support on other quality criteria. By contrast, clarity, coherence, and groundedness more often contribute to support for Claude in the detailed-prompt condition. Overall, the code survey results indicate that participants judge explanation quality by specific criteria rather than consistently favoring a single model across all dimensions.

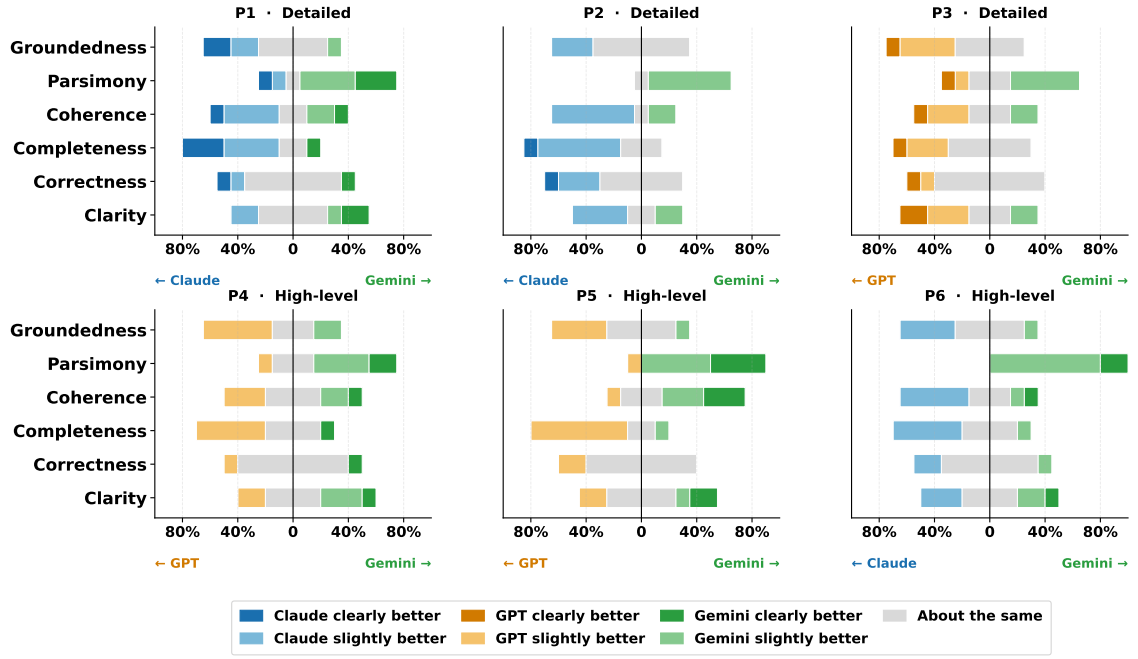

Figure 5.4: Quality dimension judgments from the RE survey. Each subplot shows one explanation pair, with detailed prompts on the top row and high-level prompts on the bottom row. P = Explanation Pair

Figure 5.4 highlights that under the detailed prompt, the results do not indicate a consistent preference for one model across all criteria. Instead, the judgments differ by quality dimension. In the two Claude–Gemini comparisons, Claude receives more support for completeness and coherence, while Gemini is more often favored for parsimony. The GPT–Gemini comparison is more mixed: GPT receives more support for clarity and groundedness, while Gemini again receives more support for parsimony. Across the detailed-prompt pairs, correctness remains close to neutral, with many "About the same" judgments.

Figure 5.4 further shows that under the high-level prompt, parsimony again stands out as the criterion most strongly associated with Gemini. Gemini receives clear support for parsimony across all three high-level comparisons. By contrast, com-

pleteness tends to favor the non-Gemini model, with stronger support for GPT in Pair 4 and Pair 5, and for Claude in Pair 6. Correctness is again mostly judged as "About the same", indicating that participants often view the explanations as similarly accurate even when they differ on other quality criteria. Overall, the RE survey results show a criterion-specific pattern rather than a single overall model preference.

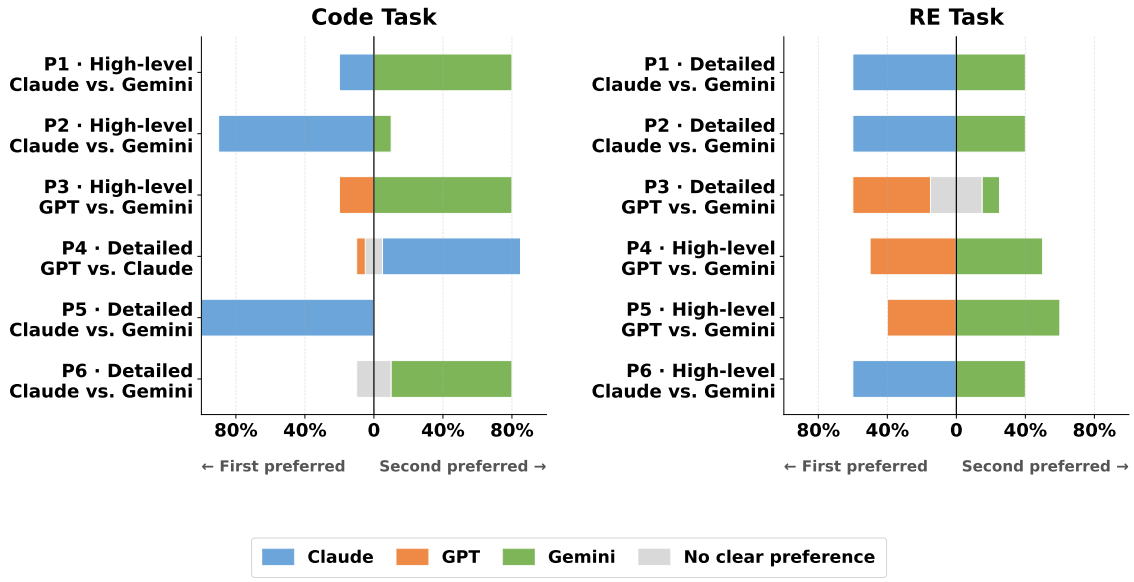

Figure 5.5: Overall preference judgments from the human survey for the code and RE tasks. P = Explanation Pair

Figure 5.5 illustrates that the preference pattern is mixed for the code task. Gemini receives strong support in Pair 1, Pair 3, and Pair 6, while Claude receives strong support in Pair 2, Pair 4, and Pair 5. This results in a balanced overall pattern, where both Claude and Gemini are clearly preferred in some comparisons. The detailed-prompt condition also does not produce a stable preference for one model, as the two Claude–Gemini comparisons point in opposite directions.

For the RE task, Gemini receives more support in most comparisons, especially in the high-level GPT–Gemini pairs. However, Claude is preferred in the detailed Claude–Gemini comparisons, and the detailed GPT–Gemini comparison is more divided. Overall, in both tasks, the results vary by model pair and prompt condition, and no model is consistently preferred across all comparisons.

Table 5.4: Mean and standard deviation for similarity and helpfulness ratings from the survey, broken down by task and prompt condition.

Pair	Models	Similarity Mean (SD)	Helpfulness Mean (SD)
Code Task – High-level Prompt			
Pair 1	Claude vs. Gemini	3.50 (0.85)	3.50 (0.71)
Pair 2	Claude vs. Gemini	3.20 (0.79)	3.60 (0.70)
Pair 3	GPT vs. Gemini	3.40 (0.70)	3.90 (0.88)
Overall		**3.37 (0.76)**	**3.67 (0.76)**
Code Task – Detailed Prompt			
Pair 4	GPT vs. Claude	3.00 (1.05)	3.80 (0.63)
Pair 5	Claude vs. Gemini	3.60 (0.70)	3.80 (0.79)
Pair 6	Claude vs. Gemini	3.50 (0.97)	3.70 (0.67)
Overall		**3.37 (0.93)**	**3.77 (0.68)**
RE Task – Detailed Prompt			
Pair 1	Claude vs. Gemini	3.60 (0.84)	3.70 (0.67)
Pair 2	Claude vs. Gemini	3.50 (0.71)	3.60 (0.70)
Pair 3	GPT vs. Gemini	3.90 (0.74)	4.00 (0.82)
Overall		**3.67 (0.76)**	**3.77 (0.73)**
RE Task – High-level Prompt			
Pair 4	GPT vs. Gemini	3.50 (0.71)	3.80 (0.79)
Pair 5	GPT vs. Gemini	3.80 (0.42)	3.80 (0.79)
Pair 6	Claude vs. Gemini	3.30 (0.95)	3.80 (0.79)
Overall		**3.53 (0.73)**	**3.80 (0.76)**

Table 5.4 shows that helpfulness ratings are moderately positive across all tasks and prompt conditions, with condition-level means ranging from 3.67 to 3.80 on a 1–5 scale. In the code task, the mean helpfulness score increases slightly from 3.67 under the high-level prompt to 3.77 under the detailed prompt. In the RE task, the differences are smaller, with means of 3.77 under the detailed prompt and 3.80 under the high-level prompt. These differences are small relative to the within-condition variability (SDs between 0.63 and 0.88 at the pair level), suggesting that prompt condition does not have a clear effect on perceived helpfulness.

Similarity scores vary more across individual model pairs, ranging from 3.00 to 3.90. In the code task, both prompt conditions have the same mean of 3.37, suggesting that the detailed prompt does not make the code explanations appear more similar overall. In the RE task, perceived similarity is slightly higher under the detailed prompt than under the high-level prompt, with means of 3.67 and 3.53, respectively. Pair-level SDs for similarity range from 0.42 to 1.05, with the largest spread observed in Pair 4 in the code task. Given the small sample size (n=10 per pair) and the larger numbers of these standard deviations, the differences between condition means should be interpreted as descriptive trends rather than strong effects.

The open-ended question at the end of each survey version provides additional insight into how participants evaluate the explanations. For the code task, one participant states: “None of the explanations were wrong. So my judgments were based on the tiny details in each explanation.” Another participant prefers explanations with clearer structure, such as sections or bullet points, because they are easier to

read and follow than wall-of-text explanations.

For the RE survey, participants similarly emphasize clarity and structure. One participant writes: "Mostly clarity, how closely the explanation stuck to the requirements. A lot of them were very similar, so I usually leaned toward the one that was a bit clearer." Another participant bases their judgments on how well the explanation matches the original input, while another prefers explanations that are concise, non-repetitive, and clearly structured.

5.3.2 Survey: LLM-as-a-judge Evaluation

It is worth noting that even though the LLM-as-a-judge was given the same five-option scale as the survey participants, it only used three of the five options across all 12 pairs and six rubric dimensions: the two "slightly better" options and "About the same". The two stronger "clearly better" options were never used. Figure 5.6 reflects this by only including "slightly better" options. This pattern is discussed further in Chapter 6.

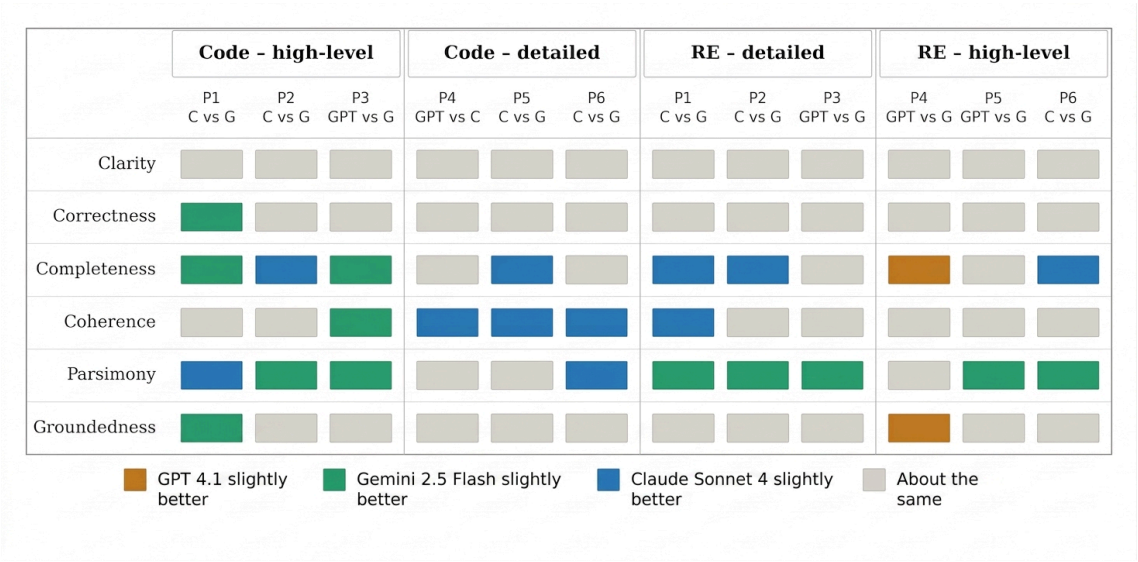

Figure 5.6: Presents the LLM-as-a-judge ratings across quality criteria, task types, and prompt conditions. GPT = GPT 4.1; C = Claude Sonnet 4; G = Gemini 2.5 Flash; P = Explanation Pair

For the code task, Figure 5.6 highlights that the high-level prompt shows more variation across criteria, with Gemini receiving support in several comparisons, especially for completeness and parsimony. Under the detailed prompt, the pattern shifts toward Claude, most clearly for coherence, where Claude is preferred in all three detailed-prompt comparisons. The other criteria under the detailed prompt are mostly rated as "About the same".

Figure 5.6 indicates that for the RE task, parsimony shows the clearest criterion-level pattern. Gemini is preferred for parsimony in all three detailed-prompt comparisons

and in two of the three high-level comparisons. In contrast, completeness tends to favor the non-Gemini explanation, with Claude or GPT preferred in several comparisons. The remaining RE criteria are mostly rated as "About the same", particularly clarity and correctness.

Overall, the judge most often rates paired explanations as "About the same", especially for clarity and correctness. Clarity is rated as "About the same" in all pairs, and correctness differs only once, in Code Pair 1, where Gemini is preferred over Claude.

Table 5.5: Similarity, preference, and helpfulness ratings for each pair from the LLM-as-a-judge evaluation

Pair	Models	Similarity	Preference	Helpfulness
Code task – high-level prompt				
Pair 1	Claude vs. Gemini	4	Gemini	4
Pair 2	Claude vs. Gemini	4	Claude	4
Pair 3	GPT vs. Gemini	4	Gemini	4
Mean		**4.00**	–	**4.00**
Code task – detailed prompt				
Pair 4	GPT vs. Claude	4	Claude	4
Pair 5	Claude vs. Gemini	4	Claude	4
Pair 6	Claude vs. Gemini	4	Claude	4
Mean		**4.00**	–	**4.00**
RE task – detailed prompt				
Pair 1	Claude vs. Gemini	4	Claude	4
Pair 2	Claude vs. Gemini	4	Claude	4
Pair 3	GPT vs. Gemini	5	No clear preference	4
Mean		**4.33**	–	**4.00**
RE task – high-level prompt				
Pair 4	GPT vs. Gemini	4	GPT	4
Pair 5	GPT vs. Gemini	5	No clear preference	4
Pair 6	Claude vs. Gemini	4	Claude	4
Mean		**4.33**	–	**4.00**

Table 5.5 shows that helpfulness receives a score of 4 for every pair across both tasks and prompt conditions, giving a mean of 4.00 throughout. This shows that all compared outputs are judged to be similarly helpful, with no pair receiving a lower or higher helpfulness rating.

Similarity scores are also consistently high, clustering around 4 on the 1–5 scale. The mean similarity score is 4.00 for both code task conditions and 4.33 for both RE task conditions. The two pairs that receive the maximum similarity score of 5 are also the pairs labeled "No clear preference". With only two such cases, this is not a strong pattern, but it is consistent with the intuition that more similar outputs are harder to differentiate on preference.

In terms of preference, Claude is selected most often overall, especially under the detailed prompt condition. Gemini is preferred in some cases, mainly in the code task under the high-level prompt, while GPT is selected only once, in the RE task under the high-level prompt. Overall, the LLM-as-a-judge results show consistent helpfulness and similarity ratings across all pairs, while preference varies by model pair and, in the code task, by prompt condition. The absence of variation in helpfulness ratings is discussed further in Chapter 6.

The final reflections from the LLM-as-a-judge revealed several factors that influenced its evaluations. For the code task, structure and readability played a major role, as explanations with clear sections or bullet points were easier to follow than large text blocks. The judge also preferred explanations that balanced detail with parsimony. Since both explanations generally stayed close to the code, differences in groundedness were minimal and mainly related to wording, structure, and small interpretive differences. Overall, many evaluations were close calls where minor formatting or phrasing differences influenced the preference.

For the RE task, the reflections showed that the explanations were generally very similar in both reasoning and conclusions. Explanations from Claude and GPT were often more detailed and engaged more directly with the wording of the requirement, while explanations from Gemini tended to be more concise and efficient. The judge also noted that most requirements were relatively clear-cut, meaning the evaluations did not fully reveal how the explanations would perform in more ambiguous classification cases where larger quality differences might emerge.

5.3.3 Agreement Between Human and LLM-as-a-judge

As noted in Section 5.3.2, the LLM-as-a-judge only uses three of the five available options across all pairs and dimensions, while the survey uses the full five-option scale. To enable a direct side-by-side comparison, the five-option survey responses are collapsed into three labels: the first-listed model preferred, the second-listed model preferred, and “About the same”. Both “slightly better” and “clearly better” responses in favor of a given model are grouped as that model being preferred, and “About the same” responses are kept as is. This collapse means that the agreement comparison no longer distinguishes between slight and clear preferences on the survey side.

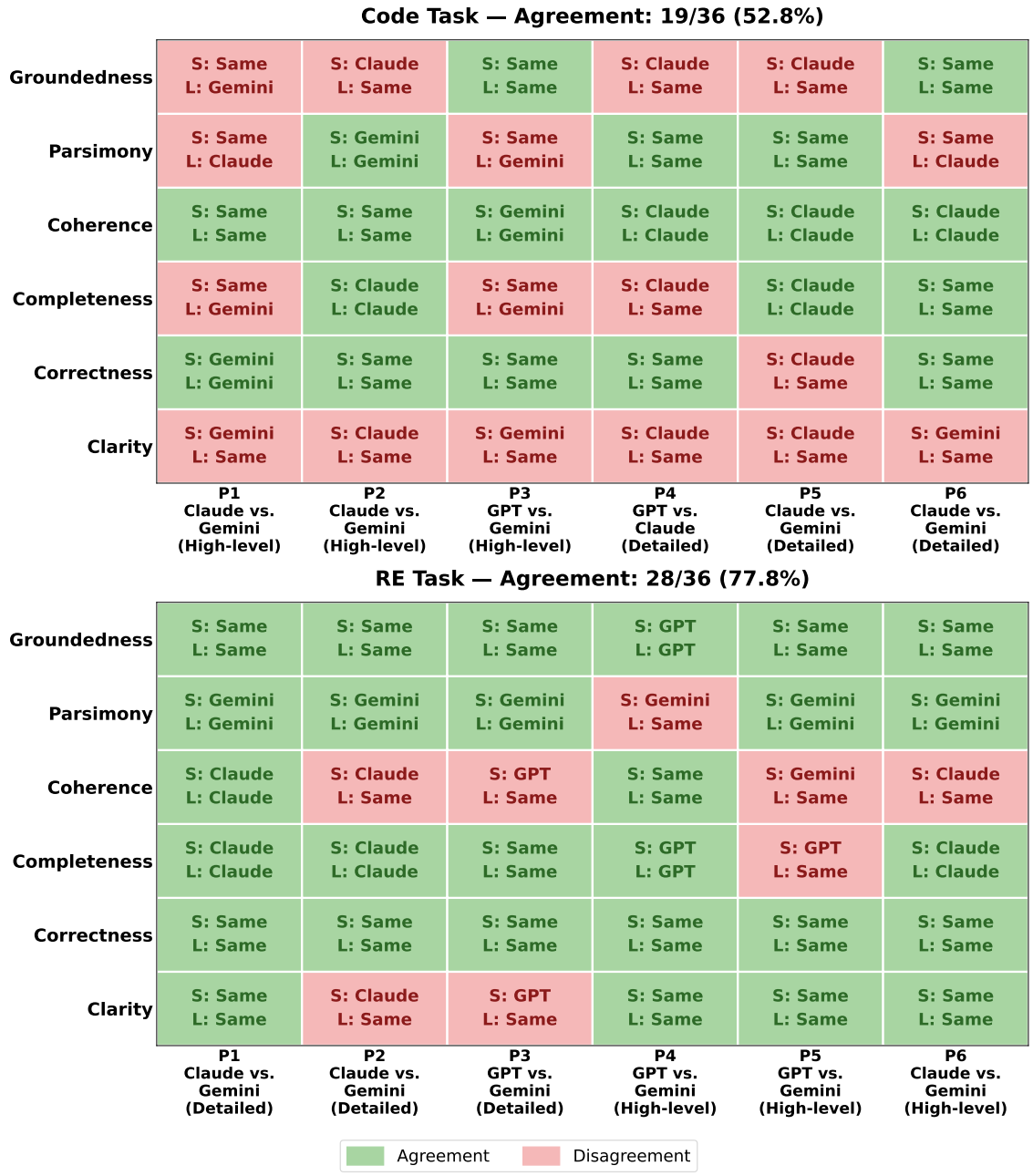

Figure 5.7: Compares the agreement between the survey majority directions with the LLM-as-a-judge judgments across the six quality criteria; Human = H; LLM-as-a-judge = L; P = Explanation Pair

Figure 5.7 indicates that overall agreement is higher for the RE task compared to the code task. In the code task, the two evaluations agree in 19 out of 36 cases (52.8%), whereas in the RE task they agree in 28 out of 36 cases (77.8%).

For the code task, agreement is strongest for coherence, where all six pairs match, and for correctness, where five out of six pairs match. The clearest disagreement appears for clarity, where none of the six pairs match. In all clarity cases, the LLM-

as-a-judge rates the explanations as the same, while the survey majority favors either Gemini or Claude.

For the RE task, agreement is more consistent across the criteria. Correctness and groundedness show full agreement across all six pairs, while completeness and parsimony each match in five out of six cases. The main exception is coherence, where only two out of six pairs match. This contrasts with the code task, where coherence shows the strongest agreement.

Table 5.6: Comparison of human evaluation results and LLM-as-a-judge results for perceived similarity, overall preference, and helpfulness. In the human results, similarity and helpfulness are reported as participant mean (SD) scores, while preference is reported as the majority preference. The LLM-as-a-judge columns report single ratings and therefore do not include standard deviations.

Models	H.Sim (SD)	L.Sim	H.Pref	L.Pref	H.Help (SD)	L.Help
Code task – high-level prompt						
Claude vs. Gemini	3.50 (0.85)	4	Gemini	Gemini	3.50 (0.71)	4
Claude vs. Gemini	3.20 (0.79)	4	Claude	Claude	3.60 (0.70)	4
GPT vs. Gemini	3.40 (0.70)	4	Gemini	Gemini	3.90 (0.88)	4
Mean	**3.37 (0.76)**	**4.00**	–	–	**3.67 (0.76)**	**4.00**
Code task – detailed prompt						
GPT vs. Claude	3.00 (1.05)	4	Claude	Claude	3.80 (0.63)	4
Claude vs. Gemini	3.60 (0.70)	4	Claude	Claude	3.80 (0.79)	4
Claude vs. Gemini	3.50 (0.97)	4	Gemini	Claude	3.70 (0.67)	4
Mean	**3.37 (0.93)**	**4.00**	–	–	**3.77 (0.68)**	**4.00**
RE task – detailed prompt						
Claude vs. Gemini	3.60 (0.84)	4	Claude	Claude	3.70 (0.67)	4
Claude vs. Gemini	3.50 (0.71)	4	Claude	Claude	3.60 (0.70)	4
GPT vs. Gemini	3.90 (0.74)	5	GPT	No pref.	4.00 (0.82)	4
Mean	**3.67 (0.76)**	**4.33**	–	–	**3.77 (0.73)**	**4.00**
RE task – high-level prompt						
GPT vs. Gemini	3.50 (0.71)	4	No pref.	GPT	3.80 (0.79)	4
GPT vs. Gemini	3.80 (0.42)	5	Gemini	No pref.	3.80 (0.79)	4
Claude vs. Gemini	3.30 (0.95)	4	Claude	Claude	3.80 (0.79)	4
Mean	**3.53 (0.73)**	**4.33**	–	–	**3.80 (0.76)**	**4.00**

Note: Human = H; LLM-as-a-judge = L; Similarity = Sim; Preference = Pref; Helpfulness = Help; Human preference is the majority preference; No pref. = No clear preference.

Overall, the LLM-as-a-judge gives more uniform judgments than the survey participants, which can be seen in Table 5.6. This is clearest for helpfulness: the LLM-as-a-judge gives the same score across all task–prompt conditions, while the survey results show moderate variation around similarly positive mean scores. This suggests that both evaluation sources generally view the explanations as helpful, but the LLM-as-a-judge gives less differentiated helpfulness ratings across tasks and prompt conditions.

Perceived similarity shows a similar pattern. The LLM-as-a-judge consistently rates the explanation pairs as more similar than the survey participants do. This difference is especially visible in the RE task, where the LLM similarity scores are higher than

the corresponding survey means under both prompt conditions. By contrast, the survey scores show more spread, as indicated by the standard deviations. This suggests that human participants are more sensitive to differences between paired explanations, while the LLM-as-a-judge tends to treat them as semantically closer.

For overall preference, the comparison shows a more mixed pattern. The survey and LLM-as-a-judge align more often in the code task than in the RE task. In the code task, the only mismatch appears in the detailed Claude–Gemini comparison, where the survey majority favors Gemini while the LLM-as-a-judge favors Claude. In the RE task, mismatches are more common, especially under the high-level prompt condition. Overall, Table 5.6 shows that the LLM-as-a-judge aligns with the survey in some preference judgments, but gives more uniform similarity and helpfulness ratings than the human participants.

The open-ended reflections from both sources converge on the same factors: structure and readability, conciseness, and faithfulness to the input. Both also note that many pairs are very similar in content and correctness, so judgments rest on smaller differences in wording and organization. This suggests that humans and the LLM-as-a-judge rely on broadly similar criteria when evaluating explanations, even where their final ratings differ.

5.3.4 Length Compliance

Table 5.7: Word count compliance for all models under both prompt conditions for the code task (target range: 135–150 words).

Model	Mean (SD)	In Range	Below Range	Above Range
High-level Prompt				
GPT-4.1	137.0 (8.0)	15/20	5	0
Claude Sonnet 4	153.4 (9.6)	9/20	0	11
Gemini 2.5 Flash	126.3 (17.7)	6/20	12	2
Detailed Prompt				
GPT-4.1	191.4 (25.0)	0/20	0	20
Claude Sonnet 4	162.0 (9.1)	1/20	0	19
Gemini 2.5 Flash	167.3 (24.5)	1/20	5	14

Table 5.7 reports that under the high-level prompt, compliance varies considerably across models. GPT is the most compliant, with 15/20 explanations within the range and a mean of 137.0 (SD = 8.0), with the five non-compliant explanations falling just below the range. Claude and Gemini both struggle, though in opposite directions; Claude consistently overshoots with a mean of 153.4 (SD = 9.6), while Gemini consistently undershoots with a mean of 126.3 (SD = 17.7).

Under the detailed prompt, Table 5.7 highlights that compliance collapses across all three models, with none achieving more than 1/20 explanations within the range, suggesting that the additional content requirements consistently override the word

count instruction. All three models overshoot on average, but differ in how much and how consistently. GPT shows the largest overshoots with a mean of 191.4 (SD = 25.0), indicating both the highest deviation from the target and the widest spread. Claude overshoots more moderately at 162.0 (SD = 9.1), maintaining a tighter distribution than the other two models despite the non-compliance. Gemini has a similar mean to Claude at 167.3, but with a much wider spread (SD = 24.5), with 14 explanations above and five below the range, indicating inconsistent behavior under the detailed prompt.

Table 5.8: Word count compliance for each model under both prompt conditions for the RE task (target range: 100–120 words).

Model	Mean (SD)	In Range	Below Range	Above Range
High-level Prompt				
GPT-4.1	108.3 (3.7)	20/20	0	0
Claude Sonnet 4	116.2 (6.4)	15/20	0	5
Gemini 2.5 Flash	95.5 (5.7)	4/20	16	0
Detailed Prompt				
GPT-4.1	111.0 (5.1)	19/20	0	1
Claude Sonnet 4	117.2 (6.5)	14/20	0	6
Gemini 2.5 Flash	98.2 (4.0)	9/20	11	0

Table 5.8 shows that under the high-level prompt, compliance is strong for GPT and Claude, while Gemini consistently undershoots the target. GPT is the most compliant model with 20/20 explanations within the range, a mean of 108.3, and a tight spread of SD = 3.7. Claude achieves 15/20 within the range, with five above, and a mean of 116.2 (SD = 6.4), indicating that overshooting explanations are close to the upper limit. Gemini undershoots 16/20 explanations with only 4/20 within the range, producing a mean of 95.5 (SD = 5.7), suggesting consistent but below-range output.

Under the detailed prompt, the compliance pattern remains largely stable across all three models (see Table 5.8). GPT maintains near-perfect compliance with 19/20 explanations within the range and a mean of 111.0 (SD = 5.1). Claude produces 14/20 within the range with six above, and a mean of 117.2 (SD = 6.5), consistent with its high-level performance. Gemini improves slightly to 9/20 within the range but continues to undershoot most explanations, with 11 below the target and a mean of 98.2 (SD = 4.0).

Overall, the RE task shows considerably more stable length compliance than the code task, and switching from the high-level to the detailed prompt has little effect on compliance for any of the three models.

6

Discussion

Semantic similarity. The semantic similarity results suggest that different LLMs often give explanations with very similar meanings for the same SE task. However, the explanations are not equally similar across all model pairs and task conditions. Table 5.2 shown in Section 5.1 indicates that differences in semantic similarity across model pairs appear in three out of four experimental conditions (H1), namely code detailed, RE high-level, and RE detailed. The exception is the code high-level condition, where the three model pairs are statistically indistinguishable. Differences between task types appear only under the detailed prompt condition (H2), suggesting that task-level variation in similarity depends on prompt design rather than task type alone. However, as described in the results, the higher variance and ranking instability for the RE tasks under both prompt conditions suggest that these patterns may reflect task-level effects more consistently than differences in mean similarity captured by the Mann–Whitney test.

One possible explanation is that the two tasks place different constraints on the explanation. In the code task, program behavior and input-output relations have relatively clear boundaries, which limit the possible directions of the explanation to some extent. In the RE task, however, the model has more freedom in explaining why a requirement is labeled as an FR or an NFR. The explanation can be explained from different angles, such as functional scope, testability, or user impact. This difference in the space for justification may be one reason why cross-model similarity follows different patterns across the two tasks. At the same time, the semantic similarity should not be overinterpreted. A high score of semantic similarity does not necessarily mean that different models reason in the same way internally. Some of the similarity may also come from shared training data, common explanation templates, or relatively fixed ways of expressing SE principles. The semantic similarity analysis, therefore, suggests that different models often produce explanations with similar meanings in their outputs, but this does not by itself show that their internal reasoning processes are also the same.

The semantic similarity findings also indicate that different prompt conditions affect cross-model consistency, but not in the same way across tasks. In the code task, the detailed prompt is associated with higher similarity across models, which may suggest that stronger structural guidance narrows the range of possible explanation paths. In the RE task, however, the detailed prompt is associated with lower and less

stable similarity. This suggests that adding more prompt detail does not necessarily reduce differences across models, and that the effect of prompt specificity depends on the task context.

This finding also makes for an interesting comparison with earlier work on reliability and reproducibility. Jahić and Sami point out that LLM outputs can be difficult to reproduce, even under the same setup, and that repeated runs may lead to substantially different results [2]. Our findings do not contradict that observation. Instead, they add a more fine-grained perspective: at the semantic level, different LLMs often converge on similar meanings when explaining the same task. At the same time, this does not happen in the same way across all tasks and prompting conditions. For our study, similarity is generally lower and varies more in the RE task, especially under the detailed prompt condition. This suggests that explanation consistency is still shaped by both task type and prompting condition.

Our findings can also be related to broader cross-model similarity. A study by Smith et al. highlights that GPT-4 was the least similar to other models from their inter-LLM analysis [48]. Furthermore, the study shows that GPT-4 and GPT-3.5 are the least similar pairs despite being in the same model family. Our results show the opposite pattern, where the two GPT-involved pairs are consistently among the highest similarity pairs across all four conditions, while Claude vs. Gemini is generally the lowest-scoring pair. Some clear differences between our study and theirs are the model generation, model families included, and prompt scope. For example, our prompts are much more restricted to the SE explanation tasks, while their study covers text-generation more broadly. This might suggest that cross-model similarity depends on the task, the domain, and how much the prompts constrain the output. Smith et al. used word-level edit distance and embedding-based cosine similarity as analytical tools, and observed that the two measures could disagree about which models were close overall [48]. This approach supports our choice of semantic similarity since explanations expressing the same reasoning may differ considerably on the surface.

Several methodological choices also shape these results. Semantic similarity is measured here using the sentence transformer `all-mpnet-base-v2` for cosine similarity. This captures how close the explanations are in meaning at the output level, but it does not show whether the models arrive at similar conclusions through similar reasoning. The results may also be shaped by the task design, the way the prompts are written, and the similarity measure itself. The different patterns found under the prompt conditions in the code task and the RE task make this especially clear. They suggest that cross-model similarity in explanations varies with the research design. The semantic similarity findings should therefore be interpreted with the task type, prompt design, and similarity measure in mind, rather than assumed to hold across different settings.

For practitioners, cross-model semantic consistency can be seen as an early sign of explanation stability, but it should not be taken as evidence that the explanations are fully reliable or based on the same reasoning process. The generally high cross-

model similarity in the code explanation task suggests that, for well-defined tasks with concrete inputs, different models are likely to express a similar understanding of the same code behavior, reducing instances where practitioners encounter clear contradictions when comparing explanations from multiple models. However, in the RE task, the lower and more variable similarity under the detailed prompt condition suggests that practitioners should be more careful when using explanations from different models and should not treat them as interchangeable. When consistency is important, such as in classification or justification tasks, both prompt design and model choice can affect the explanations that users receive. For researchers, cross-model semantic similarity can be a useful way to study explanation consistency, but the task-dependent nature of the findings shows that it is important to report the task type and prompt condition. It also shows the need to combine similarity measures with other forms of analysis, such as pattern analysis or human evaluation, to avoid overinterpreting embedding-based similarity scores.

Reasoning patterns. The reasoning pattern results suggest that differences across LLMs are not only about the content of the explanation, but also about how the explanation is organized. These patterns do not appear equally across tasks. The biggest differences are in the grounded vs. generic and contrastive vs. non-contrastive dimensions. This suggests that, in SE tasks, the way explanations are organized varies with task type.

One possible explanation is that the tasks differ in both input form and reasoning target, which affects how explanations are constructed. In the grounded vs. generic dimension, 16.7% of the explanations in the RE task are generic, compared with 0% in the code task. In the code task, the model works with a concrete code snippet and can directly refer to variables, statements, or execution logic. This makes grounded explanations easier to produce. In the RE task, the input is a natural-language requirement. When justifying a classification, the model can either stay close to the specific requirement or explain the decision in terms of general properties of FRs and NFRs. This makes generic patterns more likely to appear. This flexibility may also explain why the boundary between grounded and generic is harder to draw in the RE task. This is reflected in the annotation disagreements: there are 10 mismatches for the grounded vs. generic dimension, all in the RE task. Some cases fall into a gray area. The explanation still refers to the requirement under discussion, but the reasoning is based more on general descriptions of FRs or NFRs than on the specific wording of the requirement. As a result, these cases can plausibly be read as either grounded or generic (see Appendix Figure A.3 for an illustrative contrast). This difference may reflect not only the task itself, but also differences between models. Gemini is the only model that explicitly uses quotation marks to indicate the requirement details it refers to in the RE task. This makes its grounded explanations easier to recognize during annotation, and it also shows that models can differ in subtle ways even within the same explanation pattern.

Another notable finding in the reasoning pattern analysis is the clear difference in contrastive patterns between the two tasks. In the code task, all explanations are non-contrastive. In the RE task, 93.3% of the explanations are contrastive, and this

even reaches 100% under the detailed prompt condition. This is likely related to the kind of decision the model has to explain. In the RE task, the model has to explain why a requirement is labeled as either FR or NFR. Because of this, the explanation easily develops around the distinction between the two categories. In the code task, the model explains program behavior itself, so there is no equally strong contrast built into the task. As a result, non-contrastive explanations are more common there.

El-Assady et al. point out that explanations can differ not only in what they say, but also in how they are developed, meaning explanations can take inductive, deductive, or contrastive forms [31]. Our results show that, in SE tasks, these patterns are not evenly distributed, but vary across task types. Regarding contrastive explanations, Jacovi et al. argue that non-contrastive explanations often include more causal factors than necessary. The focus shifts for contrastive explanations since they focus on important factors that help to distinguish the fact from the foil, reducing cognitive load [35]. Wang et al. also suggest that causal explanations are often more informative when they are contrastive, because they explain an observed outcome in relation to a plausible alternative [32]. This is consistent with the strong contrastive pattern we observe in the RE task. Overall, explanation patterns are shaped not only by the model itself, but also by the task and the input form.

However, these results also need to be viewed together with the methodological choices for the study. Pattern analysis looks at the structural features of explanations in the output, not at the model's actual internal reasoning process. For this reason, the results show more directly how the model expresses an explanation, but they do not by themselves demonstrate that the model reasons internally in the same way. The distribution of patterns may also be affected by the prompt design. The detailed RE prompt includes an example whose explanation is itself framed contrastively, describing a requirement as "an environmental constraint...rather than a system function". Since the RE task is already mostly contrastive under the high-level prompt (8–9 of 10 explanations per model), the clear jump to 100% may be a result of the model picking up the "rather than" framing. This should remain as one possible interpretation rather than a confirmed cause since the effect cannot be clearly separated from the definitions and role description present in the prompt. A similar issue appears in the inductive vs. deductive dimension. All explanations are labeled as deductive, but this may also be related to the wording of the prompt, especially expressions such as "explain why...". This kind of framing tends to guide the model to give a conclusion first and then provide reasons.

The reasoning pattern findings have practical implications for how LLM explanations are used and evaluated in SE. For practitioners, the strong presence of contrastive patterns in the RE task suggests that LLM-generated explanations in classification settings often organize the reasoning around category differences. This makes them easier to follow when the decision involves a binary choice. At the same time, the appearance of generic explanations in the RE task, even at a relatively low rate, shows that LLM-generated justifications do not always stay close to the specific input. Practitioners who rely on these explanations should therefore check

whether the reasoning is actually grounded in the requirement being discussed. For researchers, these findings show that explanation pattern analysis can complement semantic similarity analysis, because it shows not only whether explanations have similar meanings, but also how models organize their justifications. These patterns also show that the way explanations are structured depends strongly on the task. Therefore, codebook-based analysis may be affected by how the task is designed, and findings from one task type should not be assumed to generalize to other settings without further investigation.

Perceived quality and prompt effects. Across the evaluated pairs, the results on high-level and detailed prompting show that perceived similarity, explanation quality, overall preference, helpfulness, and length compliance vary across task types, model pairs, and prompt conditions. For quality judgments, the variation also depends on the specific criterion being evaluated. Across these comparisons, prompt condition affects the evaluations, but it does not produce a single consistently preferred model across all comparisons. In particular, participants do not treat explanation quality as a single property for the evaluated pairs. Instead, they distinguish between concise explanations, complete explanations, and explanations that are clearly structured or grounded in the task context.

The human perceived similarity scores show a task-dependent pattern. In the code task, the mean perceived similarity is identical under both prompt conditions (3.37 for both), suggesting that similarity for participants is not bound to prompt detail. For the RE task, perceived similarity is slightly higher under the detailed prompt compared to the high-level prompt (3.67 vs. 3.53). This makes an interesting comparison with the semantic similarity scores under the RE task for the detailed prompt, where the cosine similarity was more unstable. In other words, RE explanations were less similar in their embeddings but perceived as more similar by human readers. One possible answer to this is that the detailed prompt is more anchored in shared framing, such as FR/NFR definitions and an example, which influence the models to produce similar explanations on a surface level. This may make the explanations feel more alike to readers even when their semantic content varies. Overall, this may suggest that semantic similarity and perceived similarity capture related but not identical aspects of cross-model agreement.

One possible explanation is that detailed prompts affect the form of the explanations, but this effect is not always the same across tasks. In the code task, the detailed prompt gives clearer guidance on what the explanation should cover, such as the purpose, inputs, return value, main logic, and edge cases. This may make the explanations easier to judge in terms of completeness and coherence. However, in the RE task, the detailed prompt gives the model more conceptual material, including definitions and examples. This may increase the number of possible justification paths, since a requirement can be explained through functional scope, system behavior, quality attributes, or user impact. As a result, the detailed prompt does not necessarily make the perceived quality pattern more uniform. This is consistent with the earlier semantic similarity results, where prompt detail does not have the same effect in the code and RE tasks.

Length compliance shows a similar task-dependent pattern. In the code task, the detailed prompt leads to much lower compliance with the target word range across all three models. This suggests that when the prompt asks the models to cover several specific elements, such as the purpose, inputs, return value, main logic, and edge cases, the content requirements may override the word limit. In contrast, length compliance in the RE task remains relatively stable across the two prompt conditions, but the models show different tendencies. This indicates that word-count compliance is affected not only by prompt condition, but also by task type and model-specific output tendencies. Therefore, in tasks such as code explanation, where several concrete content points need to be covered, a detailed prompt may help make the explanation more structured, but it can also make length control more difficult.

The results also show that human judgments and LLM-as-a-judge judgments partly overlap, but they are not interchangeable. Agreement between the survey majority and the LLM-as-a-judge is higher in the RE task than in the code task. This may indicate that the LLM-as-a-judge finds it easier to evaluate RE explanations using the given rubric, possibly because these explanations often follow more explicit classification-based reasoning. In contrast, the code task requires the evaluator to judge whether the explanation captures the behavior of a concrete code snippet. This may make some quality dimensions more sensitive to human interpretation. The clearest example is clarity in the code task, where the LLM-as-a-judge consistently treats the paired explanations as "About the same", while the survey majority favors one explanation in every pair. This suggests that human participants may notice differences in readability or explanatory usefulness that the LLM-as-a-judge does not separate as clearly.

The LLM-as-a-judge rates perceived similarity differently from human participants. The LLM-as-a-judge gives only scores of 4 or 5 across all 12 pairs (means of 4.00 in the code task, and 4.33 in the RE task), while human means range from 3.37 to 3.67 with a much wider spread. This suggests that LLM-as-a-judge treats the evaluated explanation pairs as more similar than humans do, possibly because it focuses more on the overall reasoning and content alignment in the explanations rather than smaller differences in wording, structure, or framing that human readers might notice. Therefore, these findings suggest that human and LLM-as-a-judge similarity judgments are not consistent and should not be treated as interchangeable.

The comparison between survey scores and LLM-as-a-judge scores further suggests that the LLM-as-a-judge gives more consistent and generally higher ratings than the human participants. For preference judgments, the LLM-as-a-judge and the survey often agree, but differences remain, especially in the RE high-level condition. This implies that LLM-as-a-judge evaluation can capture broad preference patterns in some cases, but may miss more detailed differences that human participants consider relevant. The answers to the open-ended question from both sources help to interpret these patterns. Both sources, in most cases, pointed to the same factors when explaining their judgments, where both groups mentioned structure and readability, preferring explanations with clear sections or bullet points over explanations with

only a wall of text. They also agreed that many comparisons were close calls, where the explanations were very similar in content, and the decision was based on smaller differences in wording or organization, suggesting that humans and the LLM-as-a-judge rely on broadly similar criteria when evaluating explanations. The main differences are how those criteria translate into ratings. Participants in the survey converted small perceived differences into criterion-level preferences, while LLM-as-a-judge more often rated these cases as "About the same".

A more specific pattern, highlighted in Section 5.3.2, is the use of only three out of the five scale options when judging the quality of the explanations. The judge never uses the two "clearly better" options across any of the 12 pairs and six rubric dimensions. The judge only leans toward the "slightly better" and "About the same" options. Furthermore, helpfulness is rated 4 for every pair, and similarity is limited to 4 or 5. In contrast, the survey responses spread across the full scale on all four judgment types. This may be an indication that the LLM-as-a-judge is more conservative when differentiating between explanations than human participants, even when given the same rubric and scale. The reflections from the open-ended question from the LLM-as-a-judge offer one possible interpretation of this, where it clearly describes the explanations as very similar in content and reasoning, phrasing, and parsimony, and notes that most RE instances are relatively clear-cut, reducing the chance for stronger quality differences to appear in the explanations. Looking at it from this perspective, the narrower ratings from the judge may reflect how it perceives the differences rather than simply being unwilling to use the full scale. This should be seen as one possible explanation, since the behavior could also reflect more general tendencies of the model itself.

The observed differences between human and LLM-based judgments are also consistent with prior work showing that LLM-as-a-judge can be affected by position bias and self-preference, and that their agreement with human judgment depends on the task and evaluation setting [52, 53].

The findings also relate to earlier work on prompt engineering in SE tasks. Ekin argues that clearer instructions, explicit constraints, and contextual examples can improve the structure and consistency of LLM outputs [8]. Binkhonain and Alfayez, and Ronanki et al. also show that prompt design can affect model behavior in requirements engineering tasks [22, 23]. Our findings are consistent with their view, since the perceived quality patterns differ between the high-level and detailed prompt conditions. However, the results also show that adding more detail to the prompt does not always lead to better or more consistent explanations across all tasks and models. In the code task, the detailed prompt appears to support more complete and coherent explanations in some comparisons. In the RE task, the effects are more mixed and depend more on the specific criterion. This suggests that prompt detail affects perceived explanation quality, but its effect depends on the task type, the task context, and the quality criterion being evaluated.

These results also connect to earlier work on human and LLM-based evaluation. Van der Lee et al. and Howcroft et al. argue that generated text should be evaluated

through clearly defined dimensions rather than only through a single overall quality score [49, 50]. Our findings support this idea, since model preference changes depending on whether participants judge clarity, correctness, completeness, coherence, parsimony, or groundedness. The findings also relate to work on LLM-as-a-judge evaluation. Zheng et al. and Hashemi et al. show that LLM-as-a-judge can provide useful and scalable evaluations, especially when guided by explicit criteria [51, 40]. Therefore, these evaluations are better treated as complementary to human evaluation rather than as replacements.

These findings should be interpreted with the study's methodological constraints in mind. In particular, the pairwise comparisons, which are drawn from lower-similarity explanation pairs rather than a random sample, meaning the results examine divergent cases rather than providing a full ranking of the three models. These and other limitations, including the sample size and the use of a single judge model, are discussed further in Section 6.2.

For practitioners, the findings suggest that prompt design and model choice should be considered together with the intended quality goal. If the goal is a concise explanation, the model that performs best on parsimony may be more suitable. If the goal is a more complete or clearly structured explanation, another model or a more detailed prompt may be more useful. The findings also suggest that LLM-as-a-judge evaluation can be useful as an initial assessment, especially when many explanations need to be compared. However, it should not be used as the sole source of evaluation when clarity, user understanding, or domain-specific correctness is important. For researchers, these findings highlight the value of combining human evaluation with LLM-based evaluation. Human judgments can reveal differences in perceived quality that the LLM-as-a-judge may not capture on its own. At the same time, LLM-based evaluation can still be used to compare multiple explanations when the results are interpreted carefully.

6.1 Ethical Considerations

For this study, there are two main ethical considerations: the protection of survey participants and the use of generative AI throughout the research process. Participants in the survey were fully anonymous, and no personal information, such as names or email addresses, was collected. Participation was voluntary, and participants could stop at any time before submitting their responses. Generative AI was used as part of the methodology; we used Claude Opus 4.7 as the LLM-as-a-judge for RQ3, and AI was also used to select the 120 explanations for RQ2, an approach confirmed with our supervisors. The use of LLMs also carries an environmental cost, as these models require substantial computational resources. While this was unavoidable given the study's design, it is worth acknowledging as part of the broader ethical context of LLM-based research.

6.2 Threats to Validity

In the following section, the main threats to validity are discussed using the four types: internal validity, statistical conclusion validity, construct validity, and external validity, all introduced by Shadish, Cook, and Campbell [65].

We begin with internal validity, which looks at whether the results were actually caused by what we are studying, and not by other potential factors [65].

- **Semantic similarity score influences:** Several factors may have influenced the semantic similarity scores: the word range constraint in the prompts, the short RE instances (around 1-2 sentences, giving models less content to anchor to), and structural features such as headings or the number of paragraphs used, which might affect cosine similarity independently of meaning. Directly testing whether the word limit affected similarity was not feasible, since removing it produced explanations exceeding 700 words, which was beyond the processing capacity of the embedding model. A more systematic RE instance selection using requirements with richer content could reduce this threat in future work. The structural factor would be hard to mitigate since it is more dependent on the model itself rather than separate design choices.

- **Prompt framing shaping reasoning patterns:** The wording of the prompts may have driven the observed reasoning patterns rather than reflecting how the models reason. Asking a model to "explain why" a classification holds naturally creates a conclusion-first structure, which likely explains why all explanations were deductive. Similarly, one of the detailed RE prompts' examples is itself phrased as contrastive, which may have pushed the already dominant contrastive pattern within the RE high-level prompt to 100% under the detailed prompt. Neither effect could be isolated within the current design, since the framing is inseparable from the explanation task. Future work could use a task not framed as an explanation, and a detailed prompt with a non-contrastive phrased example, to test whether inductive and non-contrastive patterns emerge.

- **LLM-as-a-judge limitations:** Three related threats affect the LLM-as-a-judge evaluation. First, it was run only once, and because the chat interface does not allow setting the temperature to 0.0, repeated runs could produce slightly different judgments. Running the evaluation 3–5 times and aggregating the results would give a more stable estimate. Second, the evaluation used the browser interface rather than the API, which may introduce smaller differences compared to a more controlled API-based setup. This threat was mitigated by disabling memory and access to previous conversations, though the effect cannot be fully ruled out. Lastly, the evaluation used only one judge model, so it is unclear whether the judgments would hold across different models or model families. Future work could use multiple LLMs as judges to assess whether their judgments converge or differ.

- **Survey participants and fatigue:** The participant sample skewed toward students with limited professional experience, with only three industry professionals per task version, meaning judgments may not represent a broader and more diverse audience. Survey fatigue may also have affected response quality toward later pairs, as suggested by the low number of open-ended answers. This was partially mitigated by splitting the original 12-pair survey into two task-specific versions, reducing completion time from roughly 45 to 20–25 minutes. Future work can focus on a larger, more diverse participant pool, which would strengthen generalizability.

Next, we consider statistical conclusion validity, which looks at whether the patterns we observed are stable and well-estimated enough to support the conclusions we draw from them [65].

- **Survey subjectivity:** The pairwise judgment format used in the survey requires participants to make fine-grained decisions, such as choosing whether "A is better" or "A is slightly better", which are decisions that could shift slightly if the same participant completed the survey on a different occasion. Answers may even differ entirely simply because the participant reads the explanations slightly differently on the second occasion, shifting their preference from A to B. This introduces some instability into the survey measurements that is difficult to fully reduce without changing the evaluation format entirely.

We then turn to construct validity, which looks at whether what we measured really captures what we were trying to measure [65].

- **Model selection and prompt calibration:** The three models were chosen based on cost and recency within each family rather than a systematic evaluation of their suitability for explanation tasks, so the study may not be comparing models that are truly equivalent in explanation capability. Related to this, the prompts were designed and validated using newer models (ChatGPT-5.2, Claude Sonnet 4.6, Gemini 3) while the experiment used older ones (GPT-4.1, Claude Sonnet 4, Gemini 2.5 Flash), so the prompts may be better suited for the newer models than for those actually used. The newer models were used for prompt testing because the older ones were not available in the chat interfaces where that testing was done. Using the same models for prompt design and the final experiment would have been the cleanest mitigation, but the API cost of the newer models was too high and outside the study's scope.

Finally, we discuss external validity, which looks at how well the findings apply beyond the specific models, tasks, and prompts used in this study [65].

- **Model generalizability:** The findings are specific to the three model versions used in the study. Newer model versions would likely produce somewhat similar results, with a slight shift in similarity scores and quality patterns, while older models would probably show more divergence. Testing newer models

was outside the study's scope due to cost.

- **Task generalizability:** The findings are specific to the two SE tasks used in the study. Other SE tasks, such as bug fixing or code review, would likely produce different patterns, since models are trained to handle and explain different tasks differently; however, framing them as explanation tasks could reduce that difference. The two tasks were deliberately chosen to be far apart within SE while both involving explanation, allowing comparison between two different domains.

- **Instance generalizability:** The code instances we chose were reasonably representative, since most were well-known intermediate Python exercises that are broadly recognized within the programming community. For the RE task however, the instances were less representative since requirements vary a lot in practice, and the 20 selected instances cover a small portion of the wide variety of requirement statements that exist.

- **Prompt generalizability:** The prompts were specifically constructed for this study and its tasks, meaning the findings about prompt effects are unlikely to generalize to other prompting strategies or formulations. Other techniques, such as chain-of-thought, would most likely produce entirely different reasoning patterns and quality outcomes. However, the prompts can serve as a starting point for other researchers working with similar tasks, requiring only minor modifications to fit different task contexts.

6.2.1 Disclosure of AI Use

Beyond its methodological role, AI was used as a supportive research tool to search for related papers, clarify technical terms, and brainstorm ideas on specific problems. However, every cited paper was personally read and assessed for credibility, and all proposed solutions were carefully evaluated and adapted by us. Additionally, AI tools such as ChatGPT, Claude, and Gemini were used to refine grammar, improve flow, and adjust tense in text that we had already drafted, and to provide alternative phrasings for sections where we sought additional input. AI was never used to write first drafts or entire sections from scratch.

7

Conclusion

This study uses a controlled experiment to compare the semantic similarity, reasoning patterns, and perceived similarity and quality of explanations generated by three different LLMs for the same SE tasks. It compares GPT-4.1, Claude Sonnet 4, and Gemini 2.5 Flash on Python code comprehension and requirements classification tasks.

For semantic similarity, explanations from different models are often close in meaning, but the degree of similarity varies across model pairs, task types, and prompt conditions. This suggests that different LLMs can explain the same task in broadly similar ways, but their explanations should not be treated as fully interchangeable.

For reasoning patterns, the three models produce structurally very similar explanations when the task is held constant. Variation appears across task types and prompt conditions, but not across models. This suggests that the explanation task shapes reasoning structure more than the choice of LLM.

For perceived explanation evaluation, different prompting strategies (high-level and detailed) are associated with small, task-dependent differences in perceived similarity and quality across the evaluated pairs. The clearest prompt effect appears in length compliance, measured directly against the target range. No model consistently leads across all quality dimensions, and perceived similarity does not always align with semantic similarity. Human evaluators and the LLM-as-a-judge agree more often on some tasks and criteria than others. For instance, the RE task receives notably higher agreement than the code task. Because this agreement is uneven, automated evaluation should be interpreted alongside human judgment rather than in isolation.

Overall, these findings show that LLM explanations in SE can be semantically close yet differ in reasoning patterns and perceived similarity and quality. Our findings also show that no single evaluation method captures the full picture.

This study makes two main contributions. For research, it provides empirical evidence on how similar explanations from different LLMs are in SE, an area that has received less direct attention than task-level LLM performance. It also shows that explanation evaluation benefits from combining several perspectives, because

embedding-based similarity, reasoning-pattern analysis, human judgments, and LLM-as-a-judge results each show different parts of the picture. For practice, the findings suggest that developers and organizations should be careful about treating explanations from different LLMs as equivalent, even when the task and prompt are the same. The results also show that LLM-as-a-judge evaluation can be useful as a supporting method, but should not replace human evaluation when trustworthiness and explanation quality are important.

Future work can explore several directions. Removing the word-length constraints from prompts is one direction worth studying, since these constraints may influence how models express their explanations. Unconstrained outputs could reveal different patterns of cross-model variation. Future studies can also include more models, more SE tasks, larger participant samples, and additional prompting strategies to see whether these findings hold in broader settings.

Bibliography

[1] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering*, 30(2):50, 2025.

[2] Jasmin Jahić and Ashkan Sami. State of Practice: LLMs in Software Engineering and Software Architecture. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, pages 311–318, Hyderabad, India, 2024. IEEE.

[3] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. Comparing code explanations created by students and large language models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pages 124–130, 2023.

[4] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024.

[5] Matteo Colombo. Experimental Philosophy of Explanation Rising: The Case for a Plurality of Concepts of Explanation. *Cognitive Science*, 41(2):503–517, 2017.

[6] Roberto Confalonieri, Tarek R. Besold, Tillman Weyde, Kathleen Creel, Tania Lombrozo, Shane T. Mueller, and Patrick Shafto. What makes a good explanation? cognitive dimensions of explaining intelligent machines. In Ashok K. Goel, Colleen M. Seifert, and Christian Freksa, editors, *Proceedings of the 41st Annual Meeting of the Cognitive Science Society*, pages 25–26, Montreal, Canada, 2019. Cognitive Science Society.

[7] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for Large Language Models: A Survey. *ACM Transactions on Intelligent Systems and Technology*, 15(2), April 2024.

[8] Sabit Ekin. Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practices. *TechRxiv*, May 2023.

[9] June Sallou, Thomas Durieux, and Annibale Panichella. Breaking the Silence: The Threats of Using LLMs in Software Engineering. In *Proceedings of the 2024 ACM/IEEE 46th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '24, pages 102–106, Lisbon, Portugal, 2024. ACM.

[10] Jiasheng Zheng, Boxi Cao, Zhengzhao Ma, Ruotong Pan, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. Beyond Correctness: Benchmarking Multi-dimensional Code Generation for Large Language Models, September 2024. ICLR 2025 Conference withdrawn submission.

[11] Huashan Chen, Zhenyu Qi, Haotang Li, Hong Chen, Jinfu Chen, Kebin Peng, In Kee Kim, Kyu Hyung Lee, and Sen He. Beyond Single LLMs: Enhanced Code Generation via Multi-Stage Performance-Guided LLM Orchestration. *arXiv preprint arXiv:2510.01379*, October 2025.

[12] Dipin Khati, Yijin Liu, David N. Palacio, Yixuan Zhang, and Denys Poshyvanyk. Mapping the Trust Terrain: LLMs in Software Engineering – Insights and Perspectives. *ACM Transactions on Software Engineering and Methodology*, October 2025.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[15] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 3:111–132, 2022.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, 55(12), March 2023.

[17] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[18] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[19] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.*, 15(3), March 2024.

[20] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[22] Manal Binkhonain and Reem Alfayez. Are prompts all you need? Evaluating prompt-based Large Language Models (LLM) s for software requirements classification. *Requirements Engineering*, 30(4):423–443, 2025.

[23] Krishna Ronanki, Beatriz Cabrero-Daniel, Jennifer Horkoff, and Christian Berger. Requirements engineering using generative ai: Prompts and prompting patterns. In *Generative AI for Effective Software Development*, pages 109–127. Springer, 2024.

[24] EG Santana Jr, Gabriel Benjamin, Melissa Araujo, Harrison Santos, David Freitas, Eduardo Almeida, Paulo Anselmo da Neto, Jiawei Li, Jina Chun, and Iftekhar Ahmed. Which Prompting Technique Should I Use? An Empirical Investigation of Prompting Techniques for Software Engineering Tasks. *arXiv preprint arXiv:2506.05614*, 2025.

[25] Paul Denny, David H Smith IV, Max Fowler, James Prather, Brett A Becker, and Juho Leinonen. Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages

283–289. 2024.

[26] Ranim Khojah, Francisco Gomes de Oliveira Neto, Mazen Mohamad, and Philipp Leitner. The impact of prompt programming on function-level code generation. *IEEE Transactions on Software Engineering*, 2025.

[27] Sabinakhon Akbarova, Felix Dobslaw, and Robert Feldt. Understanding on the Edge: LLM-generated Boundary Test Explanations. *arXiv preprint arXiv:2601.22791*, 2026.

[28] Dhairya Dalal, Marco Valentino, Andre Freitas, and Paul Buitelaar. Inference to the best explanation in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 217–235, 2024.

[29] Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. LLM Reasoners: New Evaluation, Library, and Analysis of Step-by-Step Reasoning with Large Language Models. *arXiv preprint arXiv:2404.05221*, April 2024.

[30] Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering*, 37(4):1620–1634, 2025.

[31] Mennatallah El-Assady, Wolfgang Jentner, Rebecca Kehlbeck, Udo Schlegel, Rita Sevastjanova, Fabian Sperrle, Thilo Spinner, and Daniel Keim. Towards XAI: Structuring the Processes of Explanations. In *Proc. HCML Workshop, CHI'19*, Glasgow, UK, May 2019.

[32] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y. Lim. Designing Theory-Driven User-Centric Explainable AI. In *Proc. 2019 CHI Conf. Human Factors in Computing Systems (CHI '19)*, pages 1–15, Glasgow, UK, 2019.

[33] Zichen Chen, Jianda Chen, Ambuj Singh, and Misha Sra. XplainLLM: A knowledge-augmented dataset for reliable grounded explanations in LLMs. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 7578–7596, 2024.

[34] Brian Y Lim, Anind K Dey, and Daniel Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2119–2128, 2009.

[35] Alon Jacovi, Swabha Swayamdipta, Shauli Ravfogel, Yanai Elazar, Yejin Choi, and Yoav Goldberg. Contrastive explanations for model interpretability. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language*

Processing, pages 1597–1611, 2021.

[36] Dhivya Chandrasekaran and Vijay Mago. Evolution of semantic similarity—a survey. *Acm Computing Surveys (Csur)*, 54(2):1–37, 2021.

[37] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[38] Shaik Arsh Hussain, Ravishta Kohli, Saniya Zahoor, and Shabir A. Sofi. Transforming the gui landscape: Harnessing the power of mpnet base v2 sentence transformers. *Procedia Computer Science*, 259:1809–1816, 2025. Sixth International Conference on Futuristic Trends in Networks and Computing Technologies (FTNCT06), held in Uttarakhand, India.

[39] Alfirna Rizqi Lahitani, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International conference on cyber and IT service management*, pages 1–6. IEEE, 2016.

[40] Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13806–13834, 2024.

[41] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46, 2000.

[42] Sadia Khalid, Uzair Rasheed, Uzair Khaleeq uz Zaman, Majed Alfayad, Mohammed Assiri, Wasi Haider Butt, Mamoona Humayun, and Mahmood Niazi. Ensuring quality in software requirement engineering process: A comparative study. *Egyptian Informatics Journal*, 31:100754, 2025.

[43] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, and Dan Port. NFR. Zenodo [Data set]. https://doi.org/10.5281/zenodo.268542, March 2007.

[44] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2017.

[45] Marvin Wyrich, Justus Bogner, and Stefan Wagner. 40 years of designing code comprehension experiments: A systematic mapping study. *ACM computing surveys*, 56(4):1–42, 2023.

[46] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an LLM to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[47] Yangtian Zi, Luisa Li, Arjun Guha, Carolyn Anderson, and Molly Q Feldman. "I Would Have Written My Code Differently': Beginners Struggle to Understand LLM-Generated Code. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pages 1479–1488, 2025.

[48] Brandon Smith, Mohamed Reda Bouadjenek, Tahsin Alamgir Kheya, Phillip Dawson, and Sunil Aryal. A comprehensive analysis of large language model outputs: Similarity, diversity, and bias. *arXiv preprint arXiv:2505.09056*, 2025.

[49] Chris Van der Lee, Albert Gatt, Emiel Van Miltenburg, and Emiel Krahmer. Human evaluation of automatically generated text: Current trends and best practice guidelines. *Computer Speech & Language*, 67:101151, 2021.

[50] David M Howcroft, Anja Belz, Miruna Clinciu, Dimitra Gkatzia, Sadid A Hasan, Saad Mahamood, Simon Mille, Emiel Van Miltenburg, Sashank Santhanam, and Verena Rieser. Twenty years of confusion in human evaluation: NLG needs evaluation sheets and standardised definitions. In *Proceedings of the 13th international conference on natural language generation*, pages 169–182, 2020.

[51] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[52] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, et al. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9440–9450, 2024.

[53] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM Evaluators Recognize and Favor Their Own Generations. https://arxiv.org/abs/2404.13076, 2024.

[54] Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1105–1116, 2021.

[55] Sebastian Lubos, Alexander Felfernig, Thi Ngoc Trang Tran, Damian Garber, Merfat El Mansi, Seda Polat Erdeniz, and Viet-Man Le. Leveraging llms for the quality assurance of software requirements. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pages 389–397. IEEE, 2024.

[56] Klaas-Jan Stol and Brian Fitzgerald. The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.*, 27(3), September 2018.

[57] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

[58] Vishal. Intermediate Python Exercises: 65 Coding Problems with Solutions. https://pynative.com/intermediate-python-exercises/, 2026. PYnative. Accessed: 2026-02-10.

[59] Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. Guidelines for conducting surveys in software engineering. 2015.

[60] Ron Garland. The mid-point on a rating scale: Is it desirable. *Marketing bulletin*, 2(1):66–70, 1991.

[61] Carolyn C Preston and Andrew M Colman. Optimal number of response categories in rating scales: reliability, validity, discriminating power, and respondent preferences. *Acta psychologica*, 104(1):1–15, 2000.

[62] Anthropic. Introducing Claude Opus 4.7. https://www.anthropic.com/news/claude-opus-4-7, April 2026. Accessed: 2026-05-10.

[63] Thomas W. MacFarland and Jan M. Yates. *Introduction to Nonparametric Statistics for the Biological Sciences Using R*. Springer International Publishing, Cham, Switzerland, 2016.

[64] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg, 2nd edition, 2024.

[65] Kylie Anglin, Qing Liu, and Vivian C. Wong. A primer on the validity typology and threats to validity in education research. *Asia Pacific Education Review*, 25:557–574, 2024.

Appendices

A.1 Prompt Templates and Task Instances

A.1.1 RE Task: High-level Prompt

You will receive a software requirement and its pre-assigned classification (either FR or NFR). Provide an explanation for why the given classification is suitable. Do not reconsider or change the classification. Aim for around 100–120 words in your explanation.

Requirement: [REQUIREMENT]
Classification: [CLASSIFICATION]

A.1.2 RE Task: Detailed Prompt

You are an experienced requirements analyst who explains software requirement classifications clearly and concisely.

Definitions:

Functional Requirement (FR): Specifies what the system should do, its behaviors, functions, or capabilities.

Non-functional Requirement (NFR): Specifies how the system should perform, including qualities like performance, security, usability, or constraints.

Examples:

Requirement: "The system shall display a launcher to the News-app version 3.1.4 on the main menu."
Classification: FR
Explanation: This describes a tangible system capability: displaying a launcher. It details what the system must do, which is the essence of a functional requirement.

Requirement: “The app shall run on a smartphone with Android OS version 2.3.”
Classification: NFR
Explanation: This defines an environmental constraint, compatibility with a platform, rather than a system function, making it a non-functional (portability-related) requirement.

Task:

You will receive a software requirement and its pre-assigned classification (either FR or NFR).

Provide an explanation for why the given classification is suitable. Do not reconsider or change the classification.

Aim for around 100–120 words in your explanation.

Choose the explanation structure and output pattern that you believe is clearest and best for the task; do not adhere to or mimic any specific format.

Requirement: [REQUIREMENT]
Classification: [CLASSIFICATION]

A.1.3 Code Task: High-level Prompt

Given the Python code below, please explain what it does and what it returns (in general, based on its inputs). Aim for around 135–150 words in your explanation. No printing is required.

Python code:
[CODE]

A.1.4 Code Task: Detailed Prompt

Given the Python code below, explain what it does without running it. No printing is required.

Requirements:

Be true to the code: do not add behavior that the code does not imply, and do not rewrite or change the code.

Use simple language suitable for beginner-intermediate Python learners. Aim for around 135–150 words in your explanation.

In your explanation, make sure to cover: (1) purpose, (2) inputs and

return value, (3) the main steps/logic, and (4) any important edge cases/assumptions.

Choose the explanation structure and output pattern that you believe is clearest and best for the task; do not adhere to or mimic any specific format.

Python code:
[CODE]

A.1.5 Task Instances

The complete set of task instances for both the RE and code tasks is available at this **link**.

A.1.5.1 Task Instance Examples

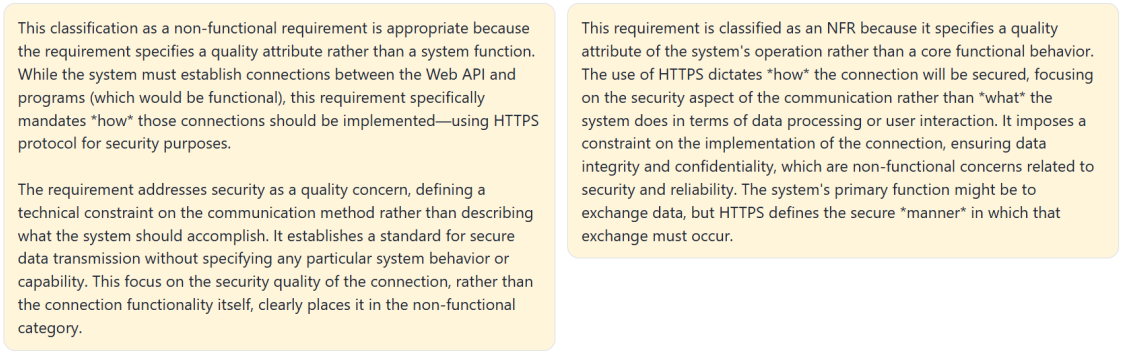

Figure A.1: Lowest-scoring similarity pair (0.609); Claude (left) vs. Gemini (right). Instance: ex014; Task: RE; Prompt variant: Detailed.

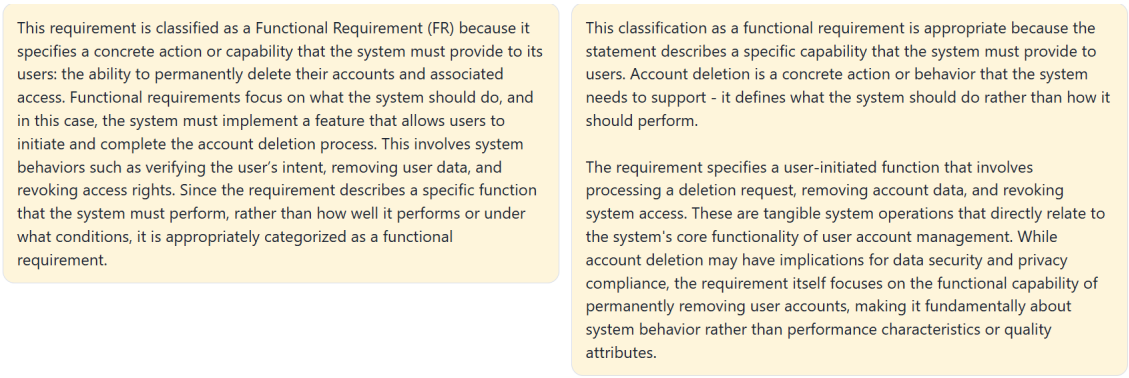

Figure A.2: One of the highest-scoring similarity pairs (0.948); GPT (left) vs. Claude (right). Instance: ex002; Task: RE; Prompt variant: Detailed.

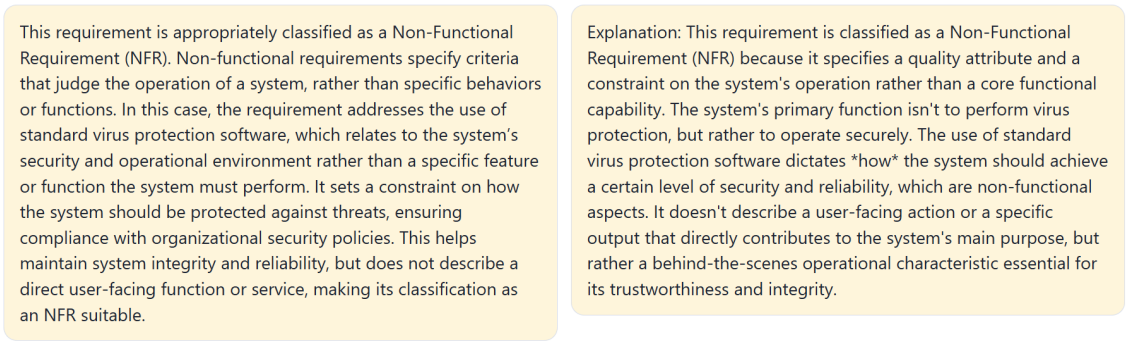

Figure A.3: Output pattern: Grounded vs. Generic; GPT Grounded (left) vs. Gemini Generic (right). Instance: ex013 for both outputs; Task: RE; Prompt variant: High-level.

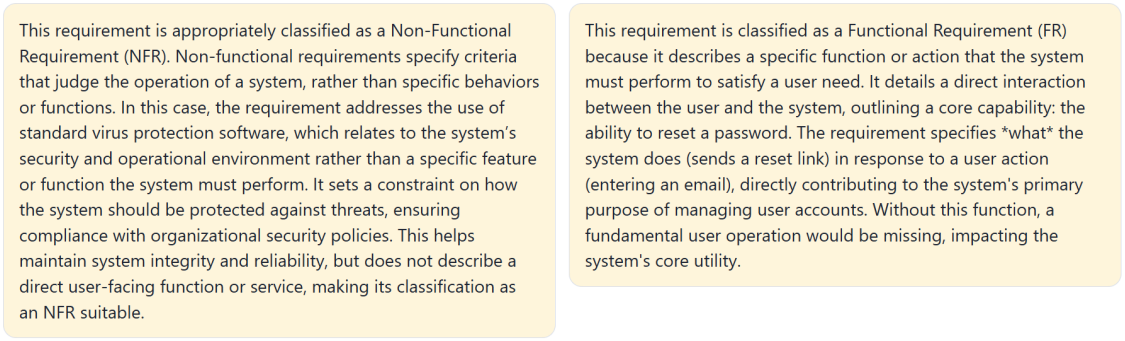

Figure A.4: Output pattern: Contrastive vs. Non-contrastive; GPT Contrastive (left) vs. Gemini Non-contrastive (right). Instance: ex013 (left), ex004 (right); Task: RE; Prompt variant: High-level.

A.2 Reasoning Pattern Codebook

A.2.1 Inductive (Bottom-Up)

Definition: An explanation where the model presents intermediate reasoning steps before the conclusion, building step by step from the input to the answer. The answer emerges at or near the end.

Decision rule: Label as inductive when the explanation works toward the conclusion through multiple linked reasoning steps, rather than just stating the answer and then justifying it.

Example 1 (RE): "The requirement says the system shall log in users with email and password. This describes a system behavior. Since it specifies what the system must do, it is a functional requirement."

Why is it inductive: Moves through steps before concluding.

Example 2 (Code): "The function first loops through the list, then checks whether each value is even, and finally adds matching values to the result. Because it performs filtering based on a condition, the code returns only even numbers."

Why is it inductive: Reasoning unfolds step by step.

Example 3 (RE): "The statement focuses on response time under load. Response time is a performance quality, not a system feature. Therefore, the requirement is non-functional."

Why is it inductive: It builds toward the classification through intermediate reasoning.

A.2.2 Deductive (Top-Down)

Definition: An explanation in which the model states the final answer or conclusion first, then provides the reasoning that elaborates or justifies it.

Decision rule: Label as deductive when the conclusion appears at the beginning, and the rest of the explanation mainly serves as justification to support it.

Example 1 (RE): "This is a functional requirement because it describes a feature the system must provide, namely, generating a monthly report."

Why is it deductive: The conclusion comes first: "This is a functional requirement."

Example 2 (Code): "This code removes duplicates from the list. It does this by checking whether each item has already been seen before adding it to the result."

Why is it deductive: The answer appears first, then the explanation.

Example 3 (RE): "This is a non-functional requirement because it specifies that the system must respond within two seconds, which is a performance constraint."

Why is it deductive: The classification is stated immediately.

A.2.3 Grounded

Definition: An explanation that supports its reasoning with concrete evidence from the specific instance, such as exact requirement phrases, code lines, conditions, variables, or behaviors.

Decision rule: Label as grounded when the explanation refers to specific details in the actual input, not just general principles.

Example 1 (RE): "The phrase 'within two seconds' shows that the requirement is about performance, which makes it non-functional."

Why is it grounded: Uses an exact phrase from the requirement.

Example 2 (Code): "The condition `if x % 2 == 0` shows that the code checks whether a number is even before adding it to the output."

Why is it grounded: Refers to a specific code condition.

Example 3 (Code): "The line `result.append(name.upper())` shows that each name is converted to uppercase before being stored."

Why is it grounded: Points to a concrete code statement.

A.2.4 Generic

Definition: An explanation that stays general or vague and could fit many different cases, without pointing to concrete evidence from the specific instance.

Decision rule: Label as generic when the explanation could mostly be reused for another similar example with little or no change.

Example 1 (RE): "This is functional because it describes what the system should do."

Why is it generic: No specific phrase or feature from the actual requirement is cited.

Example 2 (Code): "The code processes the data and returns the correct result."

Why is it generic: Too vague; could apply to many snippets.

Example 3 (RE): "This is non-functional because it is about system quality."

Why is it generic: General category-level wording, but no evidence from the instance.

A.2.5 Contrastive

Definition: An explanation that explains why the chosen answer was made instead of an alternative, for example, "why X rather than Y."

Decision rule: Label as contrastive when the explanation explicitly or clearly compares the chosen outcome to another possible outcome or class.

Example 1 (RE): "This is non-functional rather than functional because it describes how fast the system should respond, not a feature the system performs."

Why is it contrastive: Explicitly compares the chosen class to the alternative.

Example 2 (Code): "The code is filtering values rather than sorting them, because it uses a condition to include certain items but never changes their order."

Why is it contrastive: Explains the output against another possible interpretation.

Example 3 (RE): "This should be classified as functional, not non-functional, because 'send an email confirmation' describes a concrete system action rather than a quality attribute."

Why is it contrastive: Uses a direct "X not Y" comparison.

A.2.6 Non-contrastive

Definition: An explanation that explains only why the chosen answer was selected, without comparing it to any alternative.

Decision rule: Label as non-contrastive when there is no meaningful comparison to another possible answer or outcome.

Example 1 (RE): "This is a functional requirement because it describes a feature the system must provide."

Why is it non-contrastive: Explains only the chosen label, with no comparison.

Example 2 (Code): "The function returns the maximum value in the list by comparing each number during iteration."

Why is it non-contrastive: Only explains what the code does.

Example 3 (RE): "This is non-functional because it specifies a security constraint for user authentication."

Why is it non-contrastive: No "rather than" or alternative class is discussed.

A.3 Complete Survey Design and Participant Answers

A.3.1 RE Survey

The complete survey used for the RE task is available at this **link**.

The answers from the respondents for the RE survey are available at this **link**.

A.3.2 Code Survey

The complete survey used for the code task is available at this **link**.

The answers from the respondents for the code survey are available at this **link**.

A.4 LLM-as-a-judge Prompt Templates and Answers

A.4.1 RE Template and Answers

The complete template used for the LLM-as-a-judge evaluation for the RE task is available at this **link**.

The answers from the LLM-as-a-judge for the RE task are available at this **link**.

A.4.2 Code Template and Answers

The complete template used for the LLM-as-a-judge evaluation for the code task is available at this **link**.

The answers from the LLM-as-a-judge for the code task are available at this **link**.

A.5 Supplementary Statistical Results

A.5.1 Normality Test Results

Table A.1: Shapiro–Wilk normality test results for the similarity score distributions. Tests are reported for each model pair within a condition ($n = 20$) and for each condition with all model-pair scores combined ($n = 60$). p-values are evaluated against $\alpha = 0.05$; values below this threshold indicate a statistically significant difference from normality (Non-normal*). HL = high-level prompt; Det. = detailed prompt; GPT = GPT-4.1; C = Claude Sonnet 4; G = Gemini 2.5 Flash.

Condition	Distribution	W	p-value	Non-normal*
Code HL	GPT vs. C	0.929	0.150	NO
	GPT vs. G	0.863	0.009	YES
	C vs. G	0.948	0.334	NO
Code Det.	GPT vs. C	0.756	<0.001	YES
	GPT vs. G	0.888	0.024	YES
	C vs. G	0.787	0.001	YES
RE HL	GPT vs. C	0.960	0.544	NO
	GPT vs. G	0.804	0.001	YES
	C vs. G	0.862	0.009	YES
RE Det.	GPT vs. C	0.958	0.502	NO
	GPT vs. G	0.938	0.216	NO
	C vs. G	0.870	0.012	YES
Code HL	Combined ($n = 60$)	0.927	0.001	YES
Code Det.	Combined ($n = 60$)	0.793	<0.001	YES
RE HL	Combined ($n = 60$)	0.797	<0.001	YES
RE Det.	Combined ($n = 60$)	0.923	0.001	YES