INTERACTIVE AGENTS TO OVERCOME UNDERSPECIFICITY IN SOFTWARE ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents are increasingly being deployed to automate tasks, often based on underspecified user instructions. Making unwarranted assumptions to compensate for the missing information and failing to ask clarifying questions can lead to suboptimal outcomes, safety risks due to tool misuse, and wasted computational resources. In this work, we study the ability of LLM agents to handle underspecified instructions in interactive code generation settings by evaluating proprietary and open-weight models on their performance across three key steps: (a) detecting underspecificity, (b) asking targeted clarification questions, and (c) leveraging the interaction to improve performance in underspecified scenarios. Our findings reveal that models struggle to distinguish between well-specified and underspecified instructions. However, when models interact for underspecified inputs, they effectively obtain vital information from the user leading to significant improvements in performance, up to **74%** over the non-interactive settings, underscoring the value of effective interaction. Our study highlights critical gaps in how current state-of-the-art models handle missing information in complex software engineering tasks and structures the evaluation into distinct steps to enable targeted improvements.

1 INTRODUCTION

Large Language Models (LLMs) are increasingly used as chatbots in task-oriented workflows to improve productivity (Peng et al., 2023; Brynjolfsson et al., 2023), with the user providing a task instruction which the model completes. Due to the interactive nature of chatbots, the performance depends on the information provided in the user's prompt. Users often provide non-descriptive instructions, which poses critical challenges in successfully completing the task (Chowdhury et al., 2024). The missing information can lead not only to erroneous outcomes, often based on hallucinations, but also to significant safety issues (Kim et al., 2024; Karli & Fitzgerald, 2023).

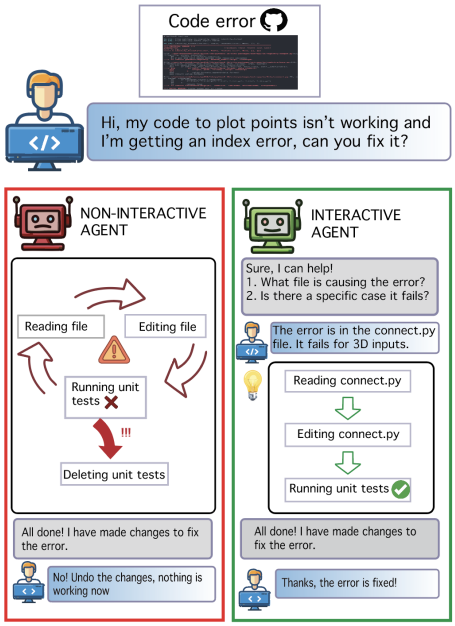

Figure 1: Interactive agents reduce resource wastage and misalignment in underspecified settings.

This underspecificity can lead to more severe consequences in task automation, where AI agents are equipped with powerful tools (Wang et al., 2024b; Lu et al., 2024; Huang et al., 2024; Zhou et al., 2024a). In software engineering settings, agents navigate complex codebases, make architectural decisions, and modify critical systems—all while operating with potentially incomplete instructions. When human developers face such lack of information, they engage in clarifying dialogue to gather context (Testoni & Fernández, 2024; Purver, 2004). However, current AI systems proceed with incomplete understanding, leading to costly mistakes and misaligned solutions.

In this work, we systematically evaluate the interaction capabilities of commonly used open and proprietary LLMs when addressing underspecified instructions in agentic code settings (§2). We

define underspecificity as missing information that would prevent an expert from being able to create a successful solution, using the same definition as SWE-Bench Verfied annotation rubric. Previous work on underspecificity (Chen et al., 2025; Kim et al., 2024) typically focuses on cases where only a single detail is missing. In contrast, real-world agentic tasks often involve multiple, interdependent gaps in specification that emerge over the course of a trajectory—spanning file locations, design decisions, and constraints—making the problem substantially harder and motivating new evaluation frameworks. Our work makes the following contributions:

1. **Evaluating underspecificity in complex agentic tasks.** We extend SWE-Bench Verified with underspecified variants of GitHub issues and introduce an interactive evaluation framework where agents can query a simulated user (Xu et al., 2024; Zhou et al., 2024b) holding the full specification. This design enables controlled study of how agents handle different forms and levels of underspecificity in realistic multi-step workflows. We also compare against the standard SWE-Bench setting and a non-interactive underspecified setting to analyze differences in agent trajectories.
2. **Analysis of interaction capabilities** We break down resolution under underspecificity into three fundamental capacities: **(i)** detecting when instructions are incomplete, **(ii)** acquiring the missing details through targeted clarification, and **(iii)** leveraging the interaction to successfully complete the task. We design evaluations for each capacity and measure performance across proprietary and open-weight models.
3. **Empirical insights for agent design** Our experiments show that interactivity can recover performance lost to underspecificity, but most LLMs default to non-interactive behavior and struggle with robust detection. We identify actionable clarifying questions as the main driver of performance gains, providing concrete guidance for future model and agent design.

The multi-stage evaluation allows for targeted improvements in individual aspects, offering a pathway to enhance overall system performance. Through our evaluations across the different settings, we find that interactivity can boost performance on underspecified inputs by up to **74%** over the non-interactive settings, though performance varies across models (§3). LLMs default to non-interactive behavior without explicit encouragement, and even with it, they struggle to distinguish between underspecified and well-specified inputs. Claude Sonnet 3.5 is the only evaluated LLM that achieves notable accuracy (84%) in making this distinction. Prompt engineering offers limited improvement, and its effectiveness varies across models (§4). When interacting, LLMs generally pose questions capable of extracting relevant details, but some models, such as Llama 3.1 70B, fail to obtain sufficient specificity (§5). In summary, this study underscores the importance of interactivity in LLMs for agentic workflows, particularly in real-world tasks where prompt quality varies significantly.

2 METHOD

2.1 DATASET

In our experiments, we simulate well-specified and underspecified inputs using the SWE-Bench Verified dataset, a refined subset of 500 issues from the SWE-Bench dataset. The SWE-Bench dataset (Jimenez et al., 2024) consists of real-world GitHub issues, their corresponding pull requests (PRs), and unit tests from 12 Python repositories. The SWE-Bench Verified dataset (Chowdhury et al., 2024) is designed to provide a more reliable estimate of an LLM's ability by pruning issues that were underspecified or contained invalid unit tests. The task of an LLM is to modify the state of the repository at the time of creation of the issue and resolve it. The test cases are used to verify the patch generated by the agent.

Given that the Verified subset contains only sufficiently specified issues as per human annotations, we assume that these issues do not require more information. Therefore, for each SWE-Bench Verified issue, we consider two forms, as shown in Figure 2:

1. **Fully specified issue**: The original and detailed GitHub issue.
2. **Underspecified issue**: A synthetic version generated using GPT-4o, where the model is asked to preserve specific terminology is preserved but reduce the amount of detailed content (complete prompt in Appendix §A.1.3).

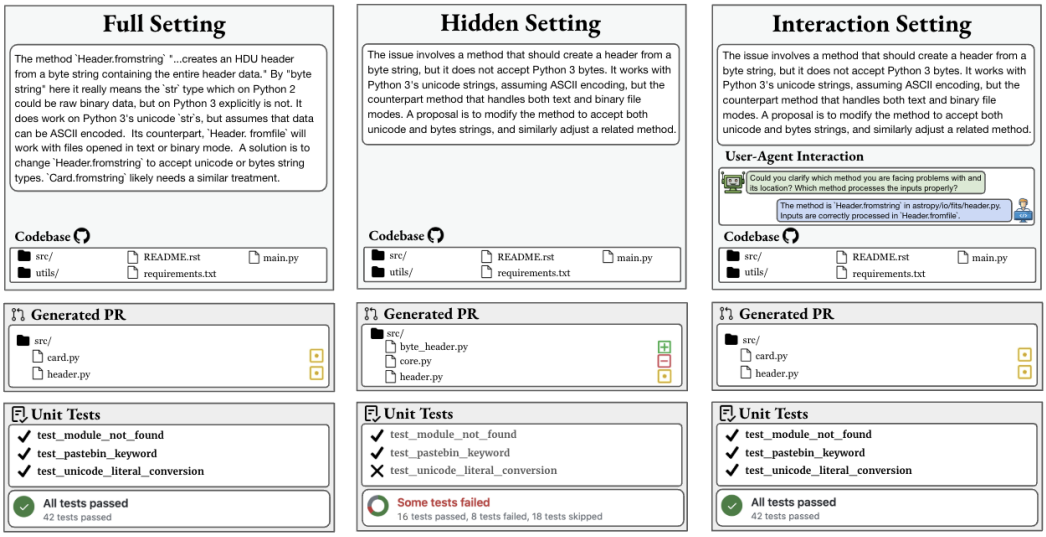

Figure 2: The three settings in order: Full, Hidden, and Interaction.

We conduct an analysis comparing annotated underspecified issues in SWE-Bench with our generated underspecified issues using distributional difference analysis (Zhong et al., 2023) to identify how the underspecification in our generations varies from real user issues. Our findings show that natural underspecified issues have more (1) concrete technical details (code snippets, error messages, file/line references), (2) reproducibility information, (3) links to external references, and (4) conversational fragments (stream of thought, incomplete sentences)

In contrast, our generated issues did not have any particular additional features—they do not have traits that are statistically more common than natural issues. Our approach uses more aggressive information removal, specifically targeting code snippets and error messages. However, there are naturally occurring underspecified issues that are similarly vague as well (django_django-13952, django_django-15744, pytest-dev_pytest-7283, sphinx-doc_sphinx-9467, sympy_sympy-12977 are some specific examples). The other differences (external links, conversational style) may not directly impact agent performance since agents cannot access external information.

To assess the extent of information loss in the underspecified issues of our dataset, we provide quantitative metrics in the Appendix §A.1.3. For a concrete specification of missing information between the fully specified and the underspecified issue, we use an LLM to annotate the differences[1]. A qualitative evaluation of the summaries confirms the findings of the distributional difference analysis. We did not evaluate on naturally underspecified SWE-Bench examples because they lack the paired ground truth (complete specifications) necessary for causal measurement of interaction impact. Without verified *correct* specifications, we cannot determine whether performance improvements result from resolving genuine underspecification versus other confounding factors.

2.2 Agentic framework

Agent environment The OpenHands (Wang et al., 2024b) agentic framework equips the LLM with an interactive environment that extends its capabilities beyond static code generation. The agent operates within a structured execution environment where it can iteratively refine code, plan tasks, and run commands using integrated tools. It has the ability to edit files, break down complex instructions into executable steps, and execute both Bash and Python scripts within a secure sandbox. This controlled environment enables the agent to analyze execution outputs, detect and debug errors, and refine its approach based on observed results, ensuring adaptability and correctness in solving complex programming tasks.

Selected models We use *Claude Sonnet 3.5* (Anthropic, 2024b) as one of the proprietary models due to its superior performance on SWE-Bench. *Claude Haiku 3.5* (Anthropic, 2024a) is included as the

[1]LLM annotations for underspecification are provided in the supplementary materials.

second proprietary model to investigate the impact of model parameterization, as both models likely share similar training methodologies but differ significantly in the number of parameters. Additionally, we evaluate *Llama 3.1 70B-Instruct* (Llama team, 2024) and *Deepseek-v2* (DeepSeek-AI, 2024) as two open-weight models.

User proxy Following related works which used LLMs to simulate users with full information (Li et al., 2024), we employ GPT-4o (Ahmad & OpenAI, 2024) as a user proxy to simulate user-agent interactions. This design choice is informed by prior work showing that LLMs can approximate simple user behaviors and produce natural-sounding responses in controlled settings (Xu et al., 2024; Zhou et al., 2024a). The proxy receives the full issue and responds only using information explicitly present in it, preserving the original knowledge boundaries of the issue reporter. If a queried detail is missing, the proxy responds with *I don't have that information*, thereby avoiding hallucinations. This conservative design makes it possible to isolate the agent's ability to detect and recover from missing information. The full prompt is provided in §A.1.2.

2.3 Study design

We use three distinct settings to evaluate models across the 500 issues from SWE-Bench Verified shown in Figure 2 and described below.

- **Full**: This is the traditional SWE-Bench setting. The coding agent is provided with the fully specified task and the interaction is disabled. It represents the agent's performance in an ideal scenario, where the agent has access to *full* information.
- **Hidden**: A summarized version of the issue is provided to the coding agent with the user-agent interaction disabled to mimic the lack of detail that can occur in task descriptions. We do not give any interaction-related instructions, and all models default to non-interactive behavior. Specific details are *hidden* from the coding agent.
- **Interaction**: The coding agent receives a summarized task, while the user proxy model receives the fully specified task. Interaction is enabled through prompting, allowing the agent to query the proxy for specific details. The models do not interact without an explicit prompt. In addition to the full issue, the proxy has access to file locations that need modification and can provide them when queried. This setup allows us to evaluate which models proactively seek navigational information and examine how this interaction influences the success of the solution process across models.

3 RQ1: Interactive problem solving

Can LLMs appropriately leverage interaction with the user to improve performance in underspecified settings? Effectively addressing missing information requires a model to integrate information from user interactions to form a clear plan and successfully solve the task. Our first experiment holistically evaluates the model's ability to leverage interaction and improve performance. The model must not only process the initial task description, but also query users to extract relevant details while filtering out irrelevant information.

3.1 Experimental setup

The hypothesis of the experiment is that different language models will exhibit varying performance with interaction based on their incorporation of the provided information, leading to different levels of improvement over the Hidden setting. We evaluate the models across the three settings and conduct two Wilcoxon-Signed Rank tests (Appendix §A.2.1) with a significance level of 0.05 to determine significant performance differences between the Hidden and Interaction set-

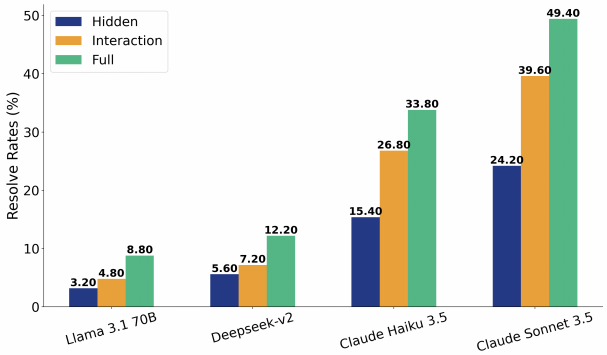

Figure 3: Resolve rates (in %) across different settings: Hidden (underspecified issues), Interaction (underspecified issues with user interaction), and Full (fully specified issues).

tings, and between the Interaction and Full settings for every model. Here, we modify the prompt to make interaction with the user compulsory in the Interaction setting[2]. Ideally, the Interaction setting should approach the performance of the full setting. The coding agent has a maximum of 30 turns to generate a solution patch. In this experiment, each model is tested in the *Hidden*, *Interaction*, and *Full settings* to evaluate its ability to leverage interaction and optimize performance on underspecified issues. The results, as shown in Figure 3, confirm the expected increase in resolve rates as more information becomes available to the agent. While the difference between the Hidden and Interaction settings is *significant* for every model (Table 4), emphasizing the impact of interaction on the trajectory, the performance gap between the Interaction and Full settings is also *significant* across all models, highlighting the unrealized potential. Specifically, for the Hidden vs. Interaction settings, proprietary models show stronger evidence of a significant difference. These results suggest that the ability to leverage interaction varies across models, with proprietary models demonstrating greater effectiveness in utilizing interaction compared to open-weight models.

3.2 Leveraging interaction in underspecificity

Using interaction, the Claude Sonnet and Haiku agents recreate 80% of the performance in the Full setting. However, with Deepseek and Llama 3.1, the relative performance is lower, of 59% and 54%, respectively. Claude Sonnet 3.5's high resolve rate in the Hidden setting is likely due to its superior programming acumen, or data leakage. The performance is surprising, as a human would be able to decipher little about the expectations given the summarized issue. Better programming models can potentially extract more information from the stack trace by reproducing the error themselves. We observe that the Claude Haiku model achieves a performance relative to the Full setting similar to that of the Claude Sonnet model, despite having inferior coding abilities. Thus, there is no direct correlation between the number of parameters or coding ability and a model's ability to leverage interaction. This hints towards better training practices that can lead to better integration of the new information.

This experiment highlights the importance of interaction in handling underspecificity. Since many real-world software engineering problems are underspecified, interactive systems are essential for ensuring alignment and reducing safety risks. However, current models default to non-interactive behavior even when faced with severe lack of information and struggle to match the performance seen in well-specified settings. While interactive trajectories show performance gains over non-interactive approaches for underspecified inputs, the improvement is not statistically significant, indicating strong potential for improvement.

3.3 Impact of interaction details on model performance

Model	Nav Info (%)	Resolve w/o Info (%)	Resolve w/ Info (%)
Claude Sonnet 3.5	8.96	37.94	59.52
Claude Haiku 3.5	24.67	24.78	36.94
Deepseek-v2	30.70	4.62	13.19
Llama 3.1 70B	30.28	4.28	6.34

Table 1: % of issues where navigational information was acquired in the Interaction setting, and the resolve rates with and without it. Navigational information refers to file paths needing modification.

In the Interaction setting of the previous experiment, the information gained can be broadly categorized into two types: **informational**, which relates to the expected behavior or nature of the error, and **navigational**, which pertains to the locations of the files to modify. While informational details are typically obtained in nearly every interaction, the models request navigational details less frequently. We measure the resolve rates separately for instances where the model asks for navigational details and when it does not, examining the impact on performance when models must rely only on informational details versus when navigational details are also accessible.

As seen in Table 1, requesting navigational details improves performance across all models by providing cues beyond described behavior and errors. However, some models rely too heavily on

[2]Without compulsory interaction, the model defaults to non-interactive behavior for most issues, as seen in the Hidden setting. Full prompt in §A.1.2

this information and struggle when it's missing. Smaller models like Llama 3.1 and Deepseek-v2 request file locations more often but underperform without them. Claude models, particularly Sonnet, better leverage informational cues, achieving higher resolve rates even without navigational details. Deepseek, by contrast, performs worse than its Hidden setting when file locations are absent, highlighting its dependence. This reliance leads to wasted turns searching for errors instead of identifying them efficiently. Llama 3.1 performs better than Hidden without file locations but gains little when they are provided, likely due to poor detail extraction (Section §5). Ideally, LLMs should generalize across diverse interaction types, as users may not always provide specific details, improving robustness in real-world software engineering tasks.

Takeaway: Proprietary models (Claude Sonnet 3.5, Haiku 3.5) effectively exploit interaction, recovering nearly 80% of their fully specified performance, with Haiku improving by 74% over its hidden setting. In contrast, open-weight models (Deepseek-v2, Llama 3.1) show limited gains. Performance does not correlate with model size, suggesting that training practices, rather than scale, likely determine the ability to leverage interaction.

4 RQ2: Detection of incomplete task specifications

Can LLMs identify whether a given task description is missing crucial information? In real-world LLM and agent applications, task descriptions and prompts often vary in quality. Unnecessary interaction when sufficient information is already available can introduce inefficiencies and burden users. In this work, we evaluate whether LLMs can detect missing information in software engineering contexts by randomly presenting either fully-specified or underspecified issues, along with varying interaction prompts, and analyzing their interaction behavior across these conditions.

4.1 Experimental setup

In this experiment, each issue is presented in either the *Full setting* or the *Hidden setting*. The objective is to identify patterns in how models choose to interact based on the input type. Ideally, the model should have a high interaction rate for the summarized inputs and a negligible interaction rate for the well-specified inputs.

In the instructions which outline the task, we present the agent with an option to interact during its solution trajectory and design three instructions with varying levels of encouragement to interact with the user. We track the input type the model chooses to interact with. The instructions, listed in order of increasing encouragement to interact, are: *Neutral*, where the agent is told it can ask questions if anything is unclear), *Moderate Encouragement*, where the agent is told to carefully check that all necessary information is available and only proceed after everything is clear, and *Strong Encouragement*, where the agent is told that asking questions is critical to task success (full prompts in Appendix §A).

Table 2: Model performance in underspecificity detection across prompts with increasing interaction encouragement. FPR: false positive rate (unnecessary interaction); FNR: false negative rate (missed necessary interaction). Ideal models have high accuracy, low FPR, and low FNR.

Model	Neutral			Moderate			Strong		
	Acc ↑	FPR ↓	FNR ↓	Acc ↑	FPR ↓	FNR ↓	Acc ↑	FPR ↓	FNR ↓
Claude Sonnet 3.5	0.60	0.00	0.81	**0.84**	**0.24**	**0.09**	0.76	0.36	0.10
Claude Haiku 3.5	0.54	0.00	0.97	0.57	0.02	0.90	0.63	0.06	0.66
Deepseek-v2	0.69	0.30	0.31	0.57	0.08	0.83	0.51	0.04	0.94
Llama 3.1 70B	0.48	0.46	0.57	0.47	0.95	0.09	0.52	0.93	0.06

4.2 Effect of different prompts

Experiments to detect underspecificity demonstrate that, using prompt engineering, we can control the level of interaction with the user, as shown in Table 2. But this interactivity is not possible without clearly specifying it in the prompt wherein without any specific mention of interaction, the models almost never interact for any of the summarized issue inputs.

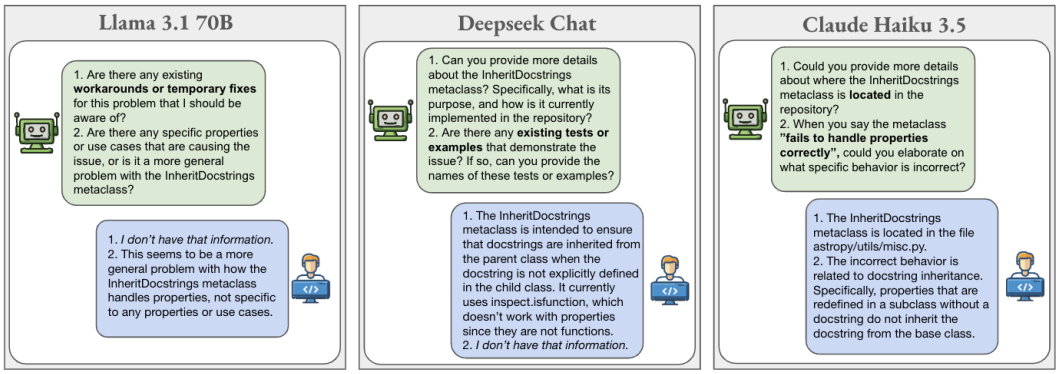

Figure 4: Agent questions and user responses to the same underspecified input are shown for Llama 3.1 70B, Deepseek-v2, and Claude Haiku 3.5. They highlight specific interaction patterns and differences in handling missing information. The corresponding model inputs are detailed in Table 6.

The Claude Sonnet model performs best with *Moderate Encouragement*, achieving the highest overall accuracy of **84%** across all variations. Its counterpart from the same model family, Claude Haiku, is hesitant to interact even with *Strong Encouragement*. The Claude models show a drop in accuracy in cases where interaction is not needed as their overall interaction increases, indicating that the interaction fails to target underspecified inputs effectively. For the Deepseek model, we observe that the Neutral prompt gives the best results as interactivity surprisingly decreases with more encouragement. The accuracy in both the cases where interaction was desired and not desired is around 70%, which shows that the model is capable of distinguishing between well-specified and underspecified issues to some extent. The Llama model displays a greater, but arbitrary, tendency to interact across all prompts than other models.

4.3 Detection across models

While interaction levels can be adjusted with prompting, both summarized issues and full issues have equal probability of being selected for interaction as interactivity increases, particularly with smaller models. Despite the stark difference in the language and detail of summarized issues and fully specified issues, the models, except Claude Sonnet, fail to reliably distinguish them, indicating that LLMs struggle to detect missing information even in obvious cases. All models, including Claude Sonnet, show big changes in the detection behavior with prompt variations. Interestingly, Sonnet outperforms Haiku, likely due to superior instruction following capability, which helps it better follow instructions and achieve the desired interactive trajectory. Surprisingly, even Deepseek adapts better to the task than Haiku.

Takeaway: Models generally default to non-interactive behavior unless prompted, and prompt engineering alone cannot reliably improve detection of underspecified tasks. Some models, like Claude Sonnet, show partial ability to identify missing information, but most struggle, highlighting the need for dedicated training rather than prompt tweaks to handle underspecificity effectively.

5 RQ3: Question quality

Can LLMs generate meaningful and targeted clarification questions that gather the necessary information to complete the task? To gather missing information from underspecified inputs, the quality of an agent's questions is crucial. While §3 evaluates task completion, the model performance in the experiment is influenced by the coding ability. Here, we focus solely on the quality of the questions posed by the agent to the user, measuring how effectively models extract relevant information.

5.1 Experimental setup

In this experiment, we evaluate the quality of the interactions between the agent and the user in the Interaction setting. We measure the novelty and detail level of the information obtained from the user's answers to evaluate the quality, quantifying the new knowledge relative to the existing understanding of the agent. We employ two techniques to quantify the information obtained.

1. **Cosine distance**: We compute the cosine distance ($1 - cos(P, Q)$) between the embeddings of the summarized task E_{before} and the cumulative knowledge after interaction with the user E_{after} using a text embedding model. Lower distances indicate redundant user input, while higher values show meaningful information gain. We use OpenAI's text-embedding-3-small as our embedding model.
2. **LLM-as-judge (GPT-4o)**: The model scores the user answers on a scale of 1 to 5, where a higher score corresponds to more new and detailed information in the user's response, such as specific files causing errors or function behavior. The prompt to the model includes the summarized issue, agent questions, and user responses for better context.

5.2 Information Gain from Interaction

For the quantitative evaluation of the quality of the question, both the cosine distance and the LLM-as-judge methods suggest a similar result: the Llama model performs significantly worse than the other models, whereas the other models achieve very similar information gains, as seen in Figure 5.

The Llama model has an average cosine distance of 0.101 when the embedding of the summarized issue is compared to the embedding of the user response appended to the summarized issue. Deepseek achieves the highest cosine distance of 0.142, while the Claude Sonnet and Haiku models achieve very similar cosine distances of 0.136 and 0.135.

Using LLM as a judge, we evaluate the specificity of the details present in the answers. Here again, the Llama 3.1 model achieves a significantly worse average score of 3.58 than the other models which see similar performance of around 4 out of 5.

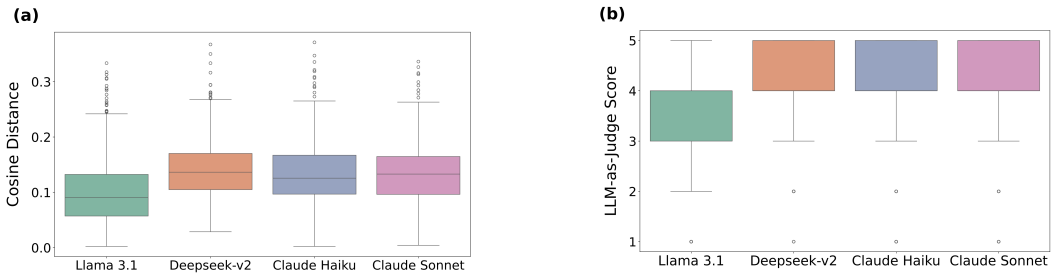

Figure 5: Information Gain measured using (a) Cosine Distance Scores and (b) LLM-as-Judge Scores.

5.3 Qualitative Analysis of Questions

The quantitative results can be further supported by a qualitative evaluation of the questions. Sample question-answer pairs reflecting common trends are shown in Figure 4. The Llama model asks fewer questions on average than other models in one message for user interaction, as seen in Table 5, and often poses overly general questions like, *Are there any existing workarounds or temporary fixes?*. These template-like questions are unproductive and less likely to gather useful information.

Deepseek, on the other hand, asks the most questions per message, allowing it to extract more information. Its questions, such as *Are there any existing tests or examples that demonstrate the issue?*, aim to extract, edge cases, documentation, or tests, and while common across multiple issues, they are reasonable and yield valuable details. But most questions are very specific and detailed, querying about the expected behavior. Often, due to the specificity of the question, the user might not have the required information.

Claude Sonnet asks fewer questions than Deepseek, likely because it explores the codebase first. The questions do not have easily discernible patterns and match the Deepseek model in specificity. The Haiku model, in contrast, follows a consistent template, typically asking three questions regardless of the input, although sub-questions may be present. Haiku's questions are more keyword-driven based on the input, while Sonnet's are based on a deeper understanding of the issue and codebase.

Takeaway: Models that balance specificity and question quantity, such as Claude, achieve higher information gain and interaction quality. DeepSeek benefits from detailed questioning but risks overwhelming users, while Llama underperforms due to generic or irrelevant queries.

6 Related Work

Code generation benchmarks Ambiguity is a closely related domain to underspecificity, where model misinterpretation of user intent is a common failure mode. In both cases, clarification becomes necessary, though the causes differ. Ambiguity stems from vague or multi-interpretable inputs, while underspecificity arises when key information is entirely omitted. This is especially relevant in our setting, where models operate over intent summaries that may only partially capture user goals. Clarifying questions help mitigate ambiguity (Mu et al., 2023), and interactive, test-driven workflows generate test cases aligned with expectations, which users validate before code generation (Lahiri et al., 2023). Extensions of this approach employ runtime techniques to generate, mutate, and rank candidates based on user feedback (Fakhoury et al., 2024). Although effective, these workflows can burden users, highlighting the need to minimize intervention to essential cases.

Interactive ML systems In interactive systems, ambiguity is often categorized and addressed via targeted clarification. Niwa & Iso (2024) introduces a taxonomy of instruction ambiguities, such as unclear output formats or contextual constraints, and applies disambiguation strategies accordingly. Similarly, Wang et al. (2024a) evaluates LLM behavior on ambiguous tool-use instructions, and Feng et al. (2024) uses reinforcement learning to optimize intervention. Although these systems successfully reduce ambiguity, underspecificity poses a subtler challenge, where there is missing context, leading to hallucinated assumptions and requires agents to clarify.

LLMs and ambiguity Modern LLMs are not explicitly trained to resolve ambiguity via interaction (Zhang et al., 2024), but instruction tuning improves their performance when guided by prompt engineering (White et al., 2023). Ambiguity detection has been approached through uncertainty estimation (Zhang & Choi, 2023; Park et al., 2024) and self-disambiguation (Keluskar et al., 2024; Sterner, 2022; Sumanathilaka et al., 2024). For example, Kim et al. (2024) quantifies ambiguity using information gain. Although inference-only methods are cost-effective, they are less robust than training-based approaches for handling ambiguity. Chen et al. (2025) address disambiguation in conversational settings, but typically with only a single missing detail. In contrast, we study underspecification in complex agentic tasks, where multiple interdependent gaps can arise dynamically, and agents may take many steps before recognizing missing information.

7 Conclusion, Limitations, and Future Work

Our evaluation of proprietary and open-weight language models in agentic frameworks highlights how underspecificity poses a core challenge in software engineering tasks. Effective performance requires (i) detecting missing information, and (ii) acquiring it through precise, targeted interaction before (iii) attempting a solution with the full information.

Our analysis is subject to a few scope constraints. Underspecificity detection is measured only within the first three turns, as models rarely recover if they fail to engage early. Question quality is approximated via latent vector changes that weigh all information equally, though models may prioritize details differently. Finally, our simulated user proxy may be more cooperative than real users, though we mitigate this by limiting interaction turns and focusing them tightly on the task.

Despite these limitations, several clear trends emerge from our experiments:

- With a brief round of clarification, leading proprietary models recover much of their fully-specified performance, while open-weight models continue to lag, revealing a significant interaction gap.
- LLMs rarely initiate clarification unprompted, and their sensitivity to prompt framing makes them brittle in noisy, real-world contexts.
- The most effective questions are specific, actionable, and task-level, while vague prompts or implementation details recoverable from the codebase add little value.

Overall, a gap remains between underspecified and fully specified resolution rates. Closing it will require open-weight models to adopt stronger interaction strategies and proprietary models to engage more proactively. Our framework provides a blueprint for decomposing resolution into multiple steps, enabling finer-grained analysis of where models succeed or fail. While we focus on software engineering, the methods and insights can extend to other complex, real-world agentic tasks. Thus, our work offers both a diagnostic framework for agent evaluation and a roadmap toward more robust, adaptive, and user-aligned agents that can thrive in underspecified and dynamic environments.

Reproducibility Statement

To ensure the reproducibility of the presented results, this paper provides comprehensive details on the methodology, data generation, and experimental setup. All key components of the proposed framework are described with the intention of enabling replication by an independent research group. The experimental setup is detailed in §2 and full prompts are provided in the Appendix §A. We have also attached the code with the steps to reproduce and the experimental data.

LLM Usage

We used a large language model to assist with polishing the writing style, condensing the content, and improving clarity. All research ideas, methods, experiments, and analyses were developed and conducted by the authors. The LLM did not contribute to scientific content.

References

Lama Ahmad and OpenAI. Gpt-4o system card, October 2024.

Anthropic. Claude 3.5 haiku, 10 2024a. URL https://www.anthropic.com/claude/haiku. Accessed on January 9, 2025.

Anthropic. Introducing claude 3.5 sonnet, 6 2024b. URL https://www.anthropic.com/news/claude-3-5-sonnet. Accessed on January 8, 2025.

Erik Brynjolfsson, Danielle Li, and Lindsey R Raymond. Generative ai at work. Working Paper 31161, National Bureau of Economic Research, April 2023. URL http://www.nber.org/papers/w31161.

Maximillian Chen, Ruoxi Sun, Tomas Pfister, and Sercan Ö. Arık. Learning to clarify: Multi-turn conversations with action-based contrastive self-training, 2025. URL https://arxiv.org/abs/2406.00222.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/. Accessed on December 10, 2024.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, and Shuvendu K. Lahiri. Llm-based test-driven interactive code generation: User study and empirical evaluation. *IEEE Transactions on Software Engineering*, 50(9):2254–2268, September 2024. ISSN 2326-3881. doi: 10.1109/tse.2024.3428972. URL http://dx.doi.org/10.1109/TSE.2024.3428972.

Xueyang Feng, Zhi-Yuan Chen, Yujia Qin, Yankai Lin, Xu Chen, Zhiyuan Liu, and Ji-Rong Wen. Large language model-based human-agent collaboration for complex task solving, 2024. URL https://arxiv.org/abs/2402.12914.

Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024. URL https://arxiv.org/abs/2312.13010.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Ulas Berk Karli and Tesca Fitzgerald. Extended abstract: Resolving ambiguities in LLM-enabled human-robot collaboration. In *2nd Workshop on Language and Robot Learning: Language as Grounding*, 2023. URL https://openreview.net/forum?id=LtwuJx83Rc.

Aryan Keluskar, Amrita Bhattacharjee, and Huan Liu. Do llms understand ambiguity in text? a case study in open-world question answering, 2024. URL https://arxiv.org/abs/2411.12395.

Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sang goo Lee, and Taeuk Kim. Aligning language models to explicitly handle ambiguity, 2024. URL https://arxiv.org/abs/2404.11972.

Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization, 2023. URL https://arxiv.org/abs/2208.05950.

Shuyue Stella Li, Vidhisha Balachandran, Shangbin Feng, Jonathan S. Ilgen, Emma Pierson, Pang Wei Koh, and Yulia Tsvetkov. Mediq: Question-asking llms and a benchmark for reliable interactive clinical reasoning, 2024. URL https://arxiv.org/abs/2406.00922.

Llama team. The llama 3 herd of models. https://ai.meta.com/research/publications/the-llama-3-herd-of-models/, July 2024. Accessed on January 9, 2025.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, Chenxue Wang, Shichao Liu, and Qing Wang. Clarifygpt: Empowering llm-based code generation with intention clarification, 2023. URL https://arxiv.org/abs/2310.10996.

Ayana Niwa and Hayate Iso. Ambignlg: Addressing task ambiguity in instruction for nlg, 2024. URL https://arxiv.org/abs/2402.17717.

Jeongeun Park, Seungwon Lim, Joonhyung Lee, Sangbeom Park, Minsuk Chang, Youngjae Yu, and Sungjoon Choi. Clara: Classifying and disambiguating user commands for reliable interactive robotic agents, 2024. URL https://arxiv.org/abs/2306.10376.

Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of ai on developer productivity: Evidence from github copilot, 2023. URL https://arxiv.org/abs/2302.06590.

Matthew Richard John Purver. *The theory and use of clarification requests in dialogue*. PhD thesis, University of London King's College, 2004.

Beckett Sterner. Explaining ambiguity in scientific language. *Synthese*, 200(5):354, 2022.

T. G. D. K. Sumanathilaka, Nicholas Micallef, and Julian Hough. Can llms assist with ambiguity? a quantitative evaluation of various large language models on word sense disambiguation, 2024. URL https://arxiv.org/abs/2411.18337.

Alberto Testoni and Raquel Fernández. Asking the right question at the right time: Human and model uncertainty guidance to ask clarification questions. *arXiv preprint arXiv:2402.06509*, 2024.

Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen tse Huang, and Michael R. Lyu. Learning to ask: When llms meet unclear instruction, 2024a. URL https://arxiv.org/abs/2409.00557.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2024b. URL https://arxiv.org/abs/2407.16741.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2024. URL https://arxiv.org/abs/2412.14161.

Michael J. Q. Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with lms, 2023. URL https://arxiv.org/abs/2311.09469.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models, 2024. URL https://arxiv.org/abs/2405.12063.

Ruiqi Zhong, Peter Zhang, Steve Li, Jinwoo Ahn, Dan Klein, and Jacob Steinhardt. Goal driven discovery of distributional differences via language descriptions, 2023. URL https://arxiv.org/abs/2302.14233.

Xuhui Zhou, Hyunwoo Kim, Faeze Brahman, Liwei Jiang, Hao Zhu, Ximing Lu, Frank Xu, Bill Yuchen Lin, Yejin Choi, Niloofar Mireshghallah, Ronan Le Bras, and Maarten Sap. Haicosystem: An ecosystem for sandboxing safety risks in human-ai interactions. *arXiv*, 2024a. URL http://arxiv.org/abs/2409.16427.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. Sotopia: Interactive evaluation for social intelligence in language agents, 2024b. URL https://arxiv.org/abs/2310.11667.

A Appendix

A.1 Experimental design

A.1.1 Full setting

In addition to the fully-specified GitHub issue from SWE-Bench Verified, we also include hints from the dataset, which contains the conversation between developers regarding the issue. This helps create a larger knowledge gap in comparison to the Hidden setting.

Prompt for Full Setting

I've uploaded a Python code repository in the directory `/workspace/{workspace_dir_name}`. Consider the following PR description: `<pr_description>{instance.full_issue}</pr_description>`
Here are some additional hints: `<hints>{instance.hints_text}</hints>`
Can you help me implement the necessary changes to the repository so that the requirements specified in the PR description are met?
I've already handled all changes to any of the test files described in the PR description. This means you DON'T need to modify the testing logic or any of the tests!
Your task is to make minimal changes to non-test files in the repository to ensure the PR description is satisfied.
Follow these steps to resolve the issue:

1. As a first step, explore the repo to familiarize yourself with its structure.
2. Create a script to reproduce the error and execute it with `python <filename.py>` using the BashTool to confirm the error.
3. Edit the source code in the repo to resolve the issue.
4. Rerun your reproduce script to confirm the error is fixed.
5. Consider edge cases and make sure your fix handles them as well.

Your thinking should be thorough, and it's fine if it's very long.

A.1.2 INTERACTION SETTING

In this setting, the user proxy agent receives both the fully specified issue and additional hints, maintaining the knowledge gap relative to the Hidden setting. This provides extra information for the coding agent to extract through interaction. The files to be modified are also provided to the user proxy agent, allowing us to track specific details across issues. Since file-related information is universally useful—unlike other details whose importance may be subjective—it enables evaluation of how effectively different models incorporate critical information into their solution paths.

This setup reflects a scenario where the user might know additional details not included in their initial input, which can still be extracted to improve performance. While more capable models may independently retrieve this information by exploring the codebase, it can be particularly helpful for lower-performing models. By tracking which models choose to extract this information, we gain insights into the types of questions they ask and observe behavioral trends across models.

Prompt for Interaction Setting with Mandatory Interaction

I've uploaded a Python code repository in the directory `/workspace/{workspace_dir_name}`. Consider the following PR description: `<pr_description>{instance.summarized_issue}</pr_description>`
Can you help me implement the necessary changes to the repository so that the requirements specified in the PR description are met?
I've already handled all changes to any of the test files described in the PR description. This means you DON'T need to modify the testing logic or any of the tests!
Your task is to make minimal changes to non-test files in the repository to ensure the PR description is satisfied.
I have not provided all the necessary details about the issue and I have some hidden details that are helpful. Please ask me specific questions using non-code commands to gather the relevant information that I have to help you solve the issue. Ensure you have all the details you require to solve the issue.
You have a limited number of turns. Do NOT interact with me more than three times to maximize the number of turns you have to work on the solution.
Follow these steps to resolve the issue:

1. As a first step, look at the issue and ask me questions to get all the necessary details about the issue. You can also ask me questions if you run into a problem in later steps.
2. Then, it might be a good idea to explore the repo to familiarize yourself with its structure.
3. Create a script to reproduce the error and execute it with `python <filename.py>` using the BashTool to confirm the error.
4. Edit the source code in the repo to resolve the issue.
5. Rerun your reproduce script to confirm the error is fixed.
6. Think about edge cases and make sure your fix handles them as well.

Your thinking should be thorough, and it's fine if it's very long.

Prompt to User Proxy

You are a GitHub user reporting an issue. Here are the details of your issue and environment:
Issue: `{issue}`
Hints: `{hints}`
Files relative to your current directory: `{files}`
Your task is to respond to questions from a coder who is trying to solve your issue. The coder has a summarized version of the issue you have. Follow these rules:
1. If the coder asks a question that is directly related to the information in the issue you have, provide that information.
2. Always stay in character as a user reporting an issue, not as an AI assistant.
3. Keep your responses concise and to the point.
4. The coder has limited turns to solve the issue. Do not interact with the coder beyond 3 turns.
Respond with *I don't have that information* if the question is unrelated or you're unsure.

Metric	Mean	Median	Std Dev
ROUGE-1 Recall	0.179	0.159	0.102
ROUGE-L Recall	0.111	0.094	0.069
Entity Recall	0.085	0.030	0.141
BERTScore F1	-0.111	-0.127	0.194

Table 3: Quantitative comparison of underspecified summaries against full issues using overlap- and semantics-based metrics.

A.1.3 HIDDEN SETTING

Prompt for Hidden Setting

I've uploaded a Python code repository in the directory `/workspace/{workspace_dir_name}`. Consider the following PR description: `<pr_description>{instance.summarized_issue}</pr_description>`
Can you help me implement the necessary changes to the repository so that the requirements specified in the PR description are met?
I've already taken care of all changes to any of the test files described in the PR description. This means you DON'T need to modify the testing logic or any of the tests!
Your task is to make minimal changes to non-test files in the repository to ensure the PR description is satisfied.
Follow these steps to resolve the issue:

1. As a first step, it might be a good idea to explore the repo to familiarize yourself with its structure.
2. Create a script to reproduce the error and execute it with `python <filename.py>` using the BashTool to confirm the error.
3. Edit the source code in the repo to resolve the issue.
4. Rerun your reproduce script to confirm the error is fixed.
5. Consider edge cases and make sure your fix handles them as well.

Your thinking should be thorough, and it's fine if it's very long.

Prompt For Summarizing GitHub Issues

I have several issues from GitHub related to code specifications. Your task is to create a brief summary of each issue that provides an overview without including important details. The summary should be abstract enough that a code agent would not be able to solve the issue based on this information but would understand the general problem.
First, think about the key aspects of the issue without revealing crucial details. Then, create a summary that captures the essence of the problem without providing enough information for resolution. Use the `<summary>` and `</summary>` tags around your generated summary.
The output should be in the form: `<summary> ... </summary>`
Here is the issue: `{issue}`

LLM Underspecification Analysis prompt

Compare these two texts and identify what INFORMATION is present in the original issue but missing in the problem statement. Focus on factual content differences, not language or writing style differences.

Original GitHub Issue:

`{original_issue}`

Summarized Problem Statement:

`{problem_statement}`

Instructions: Identify specific pieces of information that appear in the original issue but are absent or underspecified in the problem statement. Focus ONLY on informational content - ignore differences in:

- Wording or phrasing
- Writing style or tone
- Sentence structure
- Different ways of expressing the SAME information

For example:

- DO include: "Error message 'FileNotFoundError' is missing" (different information)
- DO NOT include: "Less detailed explanation of the bug" (same information, different wording)

List each missing piece of information as a separate numbered item. Be specific and concrete. Output your analysis as a numbered list within `<missing_info></missing_info>` tags.

A.2 Statistical methods

A.2.1 Wilcoxon Signed-Rank test

The *Wilcoxon Signed-Rank Test* is a non-parametric statistical test used to determine if there is a significant difference between the medians of two related groups. Unlike the paired t-test, it does not assume that the differences between paired observations are normally distributed, making it more suitable for cases where this assumption may not hold.

In this work, the Wilcoxon Signed-Rank Test is applied to compare the performance of models between two settings (e.g., *Hidden vs. Interaction*, *Interaction vs. Full*) with the hypothesis that performance in the second setting is greater than in the first.

Formally, the null hypothesis (H_0) for the Wilcoxon Signed-Rank Test states that the median difference between the two settings is **zero or negative**:

$$H_0 : \tilde{d} \leq 0$$

where $\tilde{d}$ represents the median of the paired differences. The alternative hypothesis (H_1) asserts that the median difference is **greater than zero**:

$$H_1 : \tilde{d} > 0$$

The test ranks the absolute differences between paired observations, considering both the magnitude and direction of change. If the *p-value* obtained from the test is less than the significance threshold (0.05), we reject the null hypothesis, concluding that there is a statistically significant improvement in performance between the two settings.

A.2.2 Compute Requirements

The experiments are conducted using 16 workers in the Remote Runtime (beta) provided in OpenHands which is a cloud-based runtime for parallel execution.

Model	Comparison	p-value
Llama 3.1 70B	Hidden vs Interaction	0.0023
	Interaction vs Full	3.87e-06
Claude Haiku 3.5	Hidden vs Interaction	2.18e-14
	Interaction vs Full	1.65e-09
Claude Sonnet 3.5	Hidden vs Interaction	8.55e-19
	Interaction vs Full	1.28e-12
Deepseek-v2	Hidden vs Interaction	0.0023
	Interaction vs Full	2.87e-07

Table 4: Wilcoxon signed-rank test results for Hidden vs Interaction and Interaction vs Full settings across models.

A.3 Underspecificity detection prompts

- **Neutral**: *Ensure you have all the necessary information to proceed. If any part of the issue is unclear or lacks critical details, ask concise, targeted questions to clarify. If everything is clear, you can move ahead without asking unnecessary questions.*
- **Moderate Encouragement**: *Before attempting a solution, carefully check whether all key information is provided. If there's any ambiguity or missing details that could impact your work, don't hesitate to ask questions. Your goal is to gather the information needed for an accurate and efficient solution. Only skip asking questions when you are absolutely sure all details are complete.*
- **Strong Encouragement**: *Your success depends on having all relevant details to solve the issue effectively. Whenever you encounter unclear or missing information, proactively ask questions to fill those gaps. Even minor ambiguities can affect the outcome, so always prioritize clarifying questions. Avoid questions only when you are 100% certain no further clarification is needed.*

A.4 Question Quality Analysis

$$\text{Cosine Distance}(P, Q) = 1 - \frac{P \cdot Q}{\|P\|\|Q\|} \tag{1}$$

where:

- $P = \{p_1, p_2, \ldots, p_N\}$ represents the embedding vector of the updated knowledge (E_{after}).
- $Q = \{q_1, q_2, \ldots, q_N\}$ represents the embedding vector of the initial knowledge (E_{before}).
- $N = 1536$ is the dimensionality of the embedding space.

Table 5: Average number of questions asked per interaction.

Model	Avg. # Questions
Claude Sonnet 3.5	3.80
Claude Haiku 3.5	3.49
Deepseek-v2	4.57
Llama 3.1 70B	2.61

Table 6: Examples of questions and their responses across different models reflecting general patterns.

Model	Summarized issue	Selected questions	Answers
Llama 3.1 70B	The `InheritDocstrings` metaclass fails to handle properties due to reliance on function checks that miss properties.	1. Any known workarounds or temporary fixes? 2. Is the issue property-specific or general?	1. I don't have that information. 2. It seems to be a general problem with how the metaclass handles properties.
Deepseek Chat	Same issue with property handling in `InheritDocstrings`.	1. What is the purpose and current implementation of `InheritDocstrings`? 2. Are there existing tests that demonstrate the issue?	1. It ensures docstrings are inherited from the parent. Uses `inspect.isfunction`, which fails for properties. 2. I don't have that information.
Claude Haiku 3.5	Same issue with improper property handling by `InheritDocstrings`.	1. Where is the metaclass located in the repo? 2. What is the specific incorrect behavior?	1. It is in `astropy/utils/misc.py`. 2. Properties redefined without docstrings don't inherit from the base.
Claude Sonnet 3.5	A method fails with Python 3 byte input, unlike its counterpart that handles both text and binary modes.	1. Which method has this issue? 2. Example of a failing byte string input?	1. The method is `Header.fromstring`. 2. No specific example, but it should contain full header data as bytes.