Interactive Agents to Overcome Ambiguity in Software Engineering

Sanidhya Vijayvargiya[1] **Xuhui Zhou**[*1] **Akhila Yerukola**[*1] **Maarten Sap**[1] **Graham Neubig**[*1]

Abstract

AI agents are increasingly being used to automate tasks, often based on ambiguous and underspecified user instructions. Making unwarranted assumptions 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 ambiguous instructions in interactive code generation settings by evaluating proprietary and open-weight models on their performance across three key steps: (a) leveraging interactivity to improve performance in ambiguous scenarios, (b) detecting ambiguity, and (c) asking targeted questions. 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 and underscoring the value of effective interaction. Our study highlights critical gaps in how current state-of-the-art models handle ambiguity in complex software engineering tasks and structures the evaluation into distinct steps to enable targeted improvements.[1]

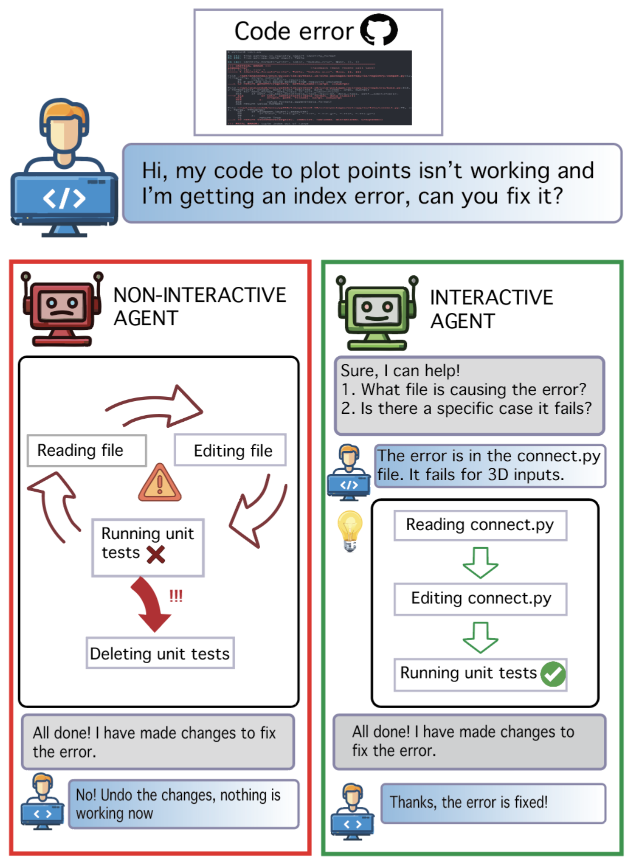

Figure 1. Interactive agents mitigate resource wastage and reduce misalignment in ambiguous settings.

1. Introduction

Large Language Models (LLMs) are increasingly being 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 ambiguity can lead not only to erroneous outcomes, but also to significant safety issues (Kim et al., 2024; Karli & Fitzgerald, 2023).

This ambiguity can lead to more severe consequences in task automation scenarios, 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 must navigate complex codebases, make architectural decisions, and modify critical systems—all while operating with potentially incomplete or ambiguous

[*]Equal contribution [1]Language Technologies Institute, Carnegie Mellon University, Pittsburgh, USA. Correspondence to: Sanidhya Vijayvargiya <sanidhyv@cs.cmu.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

[1]Code/data will be released upon publication

instructions. When human developers face such ambiguity, they engage in clarifying dialogue to gather missing context (Testoni & Fernández, 2024; Purver, 2004). However, current AI systems often proceed with incomplete understanding, leading to costly mistakes and misaligned solutions, as demonstrated in Figure 1. For agents to bridge the gap to human developers, it is crucial to deal with these ambiguities and interact effectively.

In this work, we systematically evaluate the interactivity capabilities of commonly used open and proprietary LLMs within an agentic framework when addressing underspecified instructions in code settings (§2). We examine three research questions to address the problem for code generation.

1. Interactive problem solving: Can LLMs appropriately leverage interaction with the user to improve performance in ambiguous settings?
2. Detection of ambiguity: Can LLMs identify whether a given task description is underspecified and ask clarifying questions?
3. Question quality: Can LLMs generate meaningful and targeted questions that gather the necessary information to complete the task?

We evaluate the research questions separately to ensure independence between them. We use the Github issues from SWE-Bench Verified (Chowdhury et al., 2024) to simulate well-specified inputs, and the summarized variants of the same Github issues as underspecified inputs for the experiments. A simulated user (Xu et al., 2024; Zhou et al., 2024b), equipped with the full, well-specified issue, simulates real conversations where the user has additional context, which is provided only when prompted with the appropriate questions. This multi-stage approach 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 but the performance varies between 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. Among the evaluated LLMs, only Claude Sonnet 3.5 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, we assume that these issues do not require disambiguation. 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: Summarized version generated using GPT-4o (§A), where specific terminology is preserved but detailed content is omitted.

2.2. Agentic Framework

This study uses the OpenHands (Wang et al., 2024b) agentic framework to enhance the LLM with tools for complex code generation tasks that require the model to interact with its environment and execute code. This framework provides integrated tools for text editing, task planning, and shell execution within a sandbox environment, with support for executing both Bash and Python code, making it a robust platform for seamless end-to-end workflows.

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 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-chat* (DeepSeek-AI, 2024) as two open-weight frontier models on their ability to interact effectively with the user.

User Proxy GPT-4o (Ahmad & OpenAI, 2024) is employed as a user proxy to simulate user-agent interactions (Xu et al., 2024; Zhou et al., 2024a). The user proxy is provided with the fully specified version of the task, requiring the coding agent to extract the necessary information through interaction. It is instructed to respond based solely on the information available in the full issue and will reply with *I don't have that information* if relevant details are missing. This approach prevents the user proxy from hallucinating incorrect information and encourages clear, negative responses when needed.

2.3. Study Design

We use three distinct settings to evaluate models across the three steps of addressing ambiguity with the full and summarized versions of the 500 issues from SWE-Bench Verified. The three settings used across experiments are shown in Figure 2 and are described below.

- **Hidden setting**: To mimic the lack of detail that can occur in task descriptions, the GitHub issue is summarized using GPT-4o (§A), and this summary is provided to the coding agent with the user-agent interaction disabled. No interaction-related instructions are given, and all models default to non-interactive behavior. Specific details are *hidden* from the coding agent, with no way to recover them from the user.
- **Interaction Setting**: The coding agent receives a summarized task, while the user proxy model is given the fully specified task. Interaction is enabled through prompting, allowing the agent to query the proxy for specific details. Without an explicit prompt, the models do not interact with the user. 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.
- **Full Setting**: This is the traditional SWE-Bench setting for resolving GitHub issues. The coding agent is provided with the fully specified task and the interaction is disabled. It represents the agent's performance in an unambiguous scenario, where the agent has access to *full* information, simulating ideal conditions.

3. RQ1: Interactive Problem Solving

Effectively addressing ambiguity requires a model to integrate information from user interactions to form a clear plan and successfully solve the task. This experiment holistically evaluates the model's ability to leverage interaction for performance improvement. 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 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 one-sided paired t-tests with a significance level of 0.05 to determine significant performance differences between the Hidden and Interaction settings, and between the Interaction and Full settings. 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 the Interaction setting, 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, navigational details are extracted less frequently. We analyze how models perform both with and without navigational details, examining the impact on performance when models must rely only on informational details versus when navigational details are also accessible.

3.2. Leveraging Interaction in Ambiguity

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 an expected increase in resolve rates as more information becomes available to the agent. While the difference between the Hidden and Interaction settings is marginally non-significant (p = 0.0614), the Interaction and Full settings show a significant performance gap (p = 0.0075) (§A). The lack of significant difference between the Hidden and Interaction setting may be due to sample size limitations. 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.

Using interaction, the Claude Sonnet and Haiku agents are able to recreate 80% of the performance in the Full setting. However, with Deepseek-chat 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

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

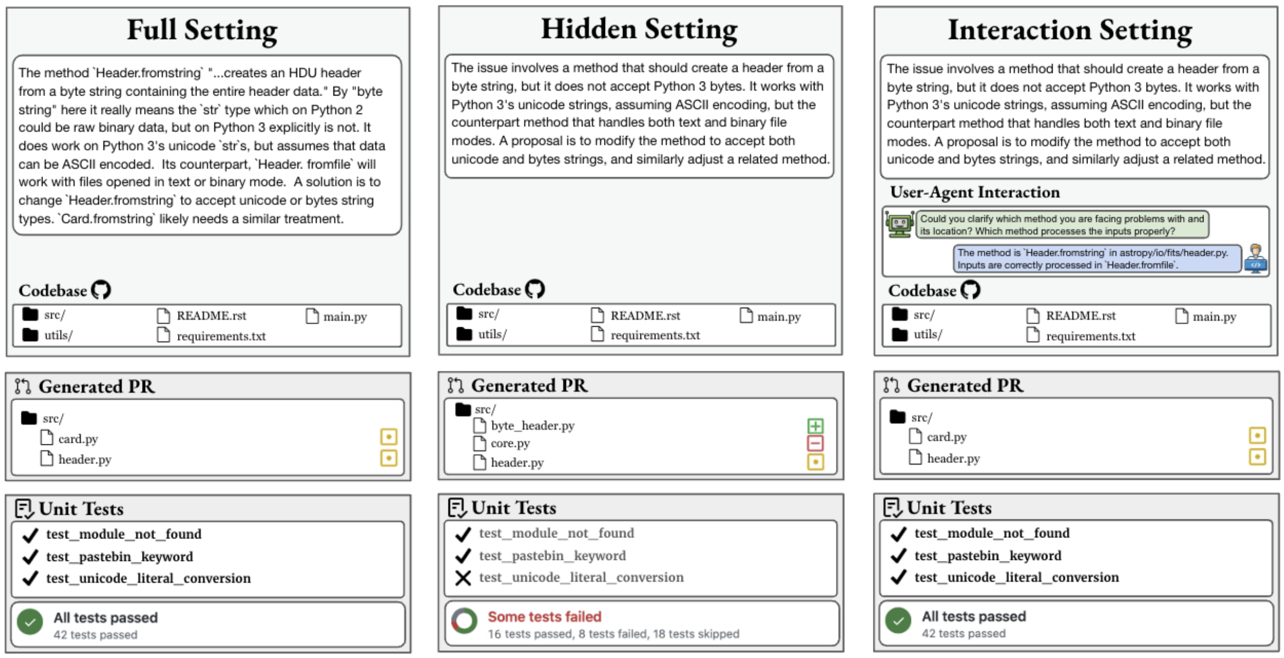

Figure 2. The three settings in order: Full, Hidden, and Interaction

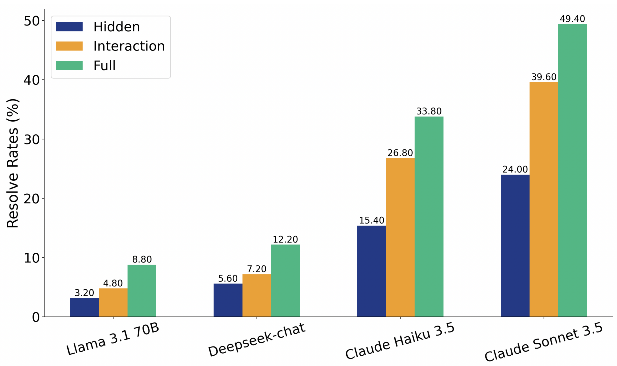

Figure 3. Resolve rates (in %) across different settings: Hidden (underspecified issues), Interaction (underspecified issues with user interaction), and Full (fully specified issues)

due to its superior programming acumen. The performance is surprising, as a human would be able to decipher little about the expectations given the summarized issue. By reproducing the error themselves, better programming models can potentially extract more information from the stack trace. 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 capacity to integrate new information into the solution trajectory. This hints towards better training practices that can lead to better integration of the new information.

This experiment highlights the importance of interaction in mitigating ambiguity. Since many real-world software engineering problems are underspecified (Chowdhury et al., 2024), interactive systems are essential for ensuring alignment and reducing safety risks. However, current models default to non-interactive behavior even when faced with ambiguity and struggle to match the performance seen in well-specified settings. While interactive trajectories show performance gains over non-interactive approaches for ambiguous inputs, the improvement is not statistically significant, indicating strong potential for improvement.

3.3. Impact of Interaction Details on Model Performance

Interaction allows models to extract critical details necessary for generating solutions. Navigational information consistently improves performance across all models, as seen in Table 1, but models must rely on informational cues such as expected behavior and error descriptions when navigational details are missing. Some models, however, are overly dependent on specific details and struggle to generalize. Smaller models like Llama 3.1 and Deepseek-chat extract file locations more frequently, but underperform without this information. Claude models, particularly Sonnet, lever-

Model	Navigational Info (%)	Resolve Rate With (%)	Resolve Rate Without (%)
Claude Sonnet 3.5	8.96	59.52	37.94
Claude Haiku 3.5	24.67	36.94	24.78
Deepseek-chat	30.70	13.19	4.62
Llama 3.1 70B	30.28	6.34	4.28

Table 1. The percentage of issues where navigational information was acquired in the Interaction setting, along with the corresponding resolve rates with and without navigational information. Navigational information refers to file locations requiring modification, helping to avoid tedious code exploration. The resolve rates offer insight into how the information obtained during interaction impacts overall performance.

age informational cues more effectively, achieving higher resolve rates even without navigational details. Deepseek-chat, by contrast, performs worse than its Hidden setting when file locations are missing, highlighting its reliance on navigational information. The model's strong dependence on file locations results in wasted turns searching for source of errors when this information is unavailable. In contrast, other models identify errors more efficiently even without navigational cues. Llama 3.1 performs better than the Hidden setting without file locations but fails to achieve significant improvements when they are provided, likely due to poor detail extraction, as discussed in Section §5. Ideally, LLMs should effectively use diverse types of information, as users may not always provide specific details. Enhancing models' ability to generalize from varied interaction types could lead to more robust performance in real-world software engineering tasks.

Takeaway: Interaction has significant potential to improve model performance in ambiguous tasks, but models, particularly open-weight, struggle to leverage it effectively. Proprietary models like Claude Sonnet 3.5 and Haiku 3.5 achieve nearly 80% of their Full setting performance, with Haiku improving by 74% over its Hidden setting performance through effective integration of both informational and navigational cues. In contrast, models like Deepseek-chat and Llama 3.1 show limited gains, primarily due to their challenges in utilizing broader informational cues, which hinders their adaptability in ambiguous tasks.

4. RQ2: Ambiguity Detection

In real-world LLM and agent applications, task descriptions and prompts can vary in quality (Chowdhury et al., 2024). Whether a given description is underspecified depends on multiple factors, including task complexity and the tools available. To detect ambiguity, a model must recognize unclear expectations or identify missing key information in its planned approach. However, interacting unnecessarily when sufficient information is already available can introduce inefficiencies and place an undue burden on the user. Here, we examine the capabilities of LLMs to detect ambiguous instructions in software engineering contexts.

4.1. Experimental Setup

In this experiment, each issue is presented in either the Full setting with the fully specified issue or the Hidden setting with its summarized version. 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*, *Moderate Encouragement*, and *Strong Encouragement* (§A). These labels reflect the varying levels of emphasis placed on user interaction. The three interactivity instructions prevent bias from a single prompt and account for differing model performance with varying levels of encouragement. To evaluate the agent under varying input, the choice between the summarized or full issue is randomized at inference.

4.2. Effect of Different Prompts

Experiments to detect ambiguity 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.

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.

Model	Neutral			Moderate Encouragement			Strong Encouragement		
	Accuracy ↑	FPR ↓	FNR ↓	Accuracy ↑	FPR ↓	FNR ↓	Accuracy ↑	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-chat	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

Table 2. Model performance in ambiguity detection across prompts with varying levels of interaction encouragement. FPR refers to cases where the model interacted unnecessarily, while FNR refers to cases where it failed to interact when needed. A model that reliably distinguishes between underspecified and well-specified issues should have high accuracy, low FPR, and low FNR.

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 modified with prompting, we notice that both summarized issues and full issues have equal probability of being picked for interaction as the interactivity levels increase, especially for smaller models. Due to the stark difference in the language and detail of summarized issues and fully specified issues, it is surprising that other than Claude Sonnet, the other models cannot make this distinction to a reliable degree. This shows that LLMs cannot detect ambiguity even in the most obvious scenarios. All models, even Claude Sonnet, show big changes in the ambiguity detection behavior with changes to the prompt. An interesting observation is the superior performance of Sonnet over Haiku, which likely has similar training methodologies but significant differences in parameter sizes. This is likely because of Sonnet's more extensive instruction tuning or training with Human Feedback. The ability to better follow instructions allows the Sonnet model to move away from its approach during training towards the desired interactive trajectory. Surprisingly, even Deepseek adapts better to the task than Haiku.

Takeaway: Prompt engineering can influence model interactivity but fails to consistently improve ambiguity detection across models. When interaction is not explicitly prompted, models default to non-interactive behavior. Claude Sonnet shows some ability to distinguish ambiguous inputs, but other models, including Claude Haiku and Llama 3.1, struggle even with clear cues. This inconsistency reveals that models are not inherently equipped to detect underspecified tasks. Improving ambiguity detection requires dedicated training, not just prompt modifications.

5. RQ3: Question Quality

Effective interaction in software engineering tasks, such as resolving issues in the SWE-bench dataset, often depends on the ability to ask questions that elicit valuable information from users to fill in missing details and effectively complete software requirements. Under the assumption that the user has the necessary information to complete the task, the goal of the experiment is to measure how much relevant information can be gathered from the user via the interaction.

5.1. Experimental Setup

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

The first approach compares the embeddings of the summarized task E_{before} and the cumulative knowledge after interaction with the user E_{after} using cosine distance (Eq 1). Lower distances indicate redundant user input, while higher values show meaningful information gain. We use OpenAI's text embedding models to generate the vector representations.

The second approach uses an LLM-as-judge (GPT-4o) to evaluate the novelty and relevance of user responses. The LLM scores the 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. By including the summarized issue and questions along with the user answer, we provide the model with more context, enabling it to distinguish between precise and vague answers.

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, in which the Llama model performs significantly worse than the other models, whereas the other models achieve very similar information gains, as seen in Figure 4.

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.

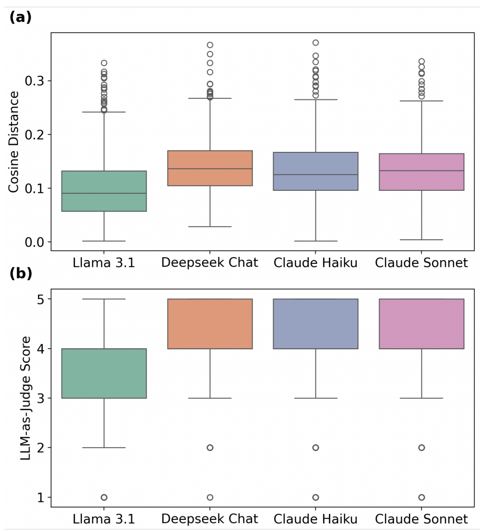

Figure 4. Information Gain measured using (a) Cosine Distance Scores and (b) LLM-as-Judge Scores

5.3. Qualitative Analysis of Questions

Model	Average Number of Questions
Claude Sonnet 3.5	3.80
Claude Haiku 3.5	3.49
Deepseek-chat	4.57
Llama 3.1 70B	2.61

Table 3. The average number of questions asked by different models in an interaction.

The quantitative results can be further supported by a qualitative evaluation of the questions. Some sample questions-answer pairs that reflect the common trends are presented in Table 5 in §A. In one message for user interaction, the Llama model asks fewer questions on average than other models as seen in Table 3. It often asks questions that are almost too human, such as *Are there any existing workarounds or temporary fixes for this problem that I should be aware of?*. The model frequently asks similar unproductive template questions that are not directly related to the information provided, which decrease the detail it can get from the user.

The Deepseek model asks a higher number of questions in a message and thus can extract more information from the user than other models. The questions which are common across multiple issues, such as *Are there any existing tests or examples that demonstrate the issue?*, aim to extract, edge cases, documentation, or tests. While these are common, such questions are also reasonable and often get key details from the user. 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.

The questions from Claude Sonnet model have similar characteristics as Deepseek, but the questions per message are fewer than those from Deepseek, potentially because the model usually first explores the codebase before asking questions. The questions do not have easily discernible patterns and match the Deepseek model in specificity. The Claude Haiku model asks questions in the same message template, with a consistent three questions regardless of the input, although the questions can have sub-questions. While Claude Sonnet asks questions based on its overall understanding of the issue and the codebase, the Haiku model seems to ask specific questions from the keywords in the issue description.

Takeaway: Models that balance specificity and question quantity, like the Claude models, achieve greater information gain and superior interaction quality compared to models that ask too few, too many, or templated questions. While Deepseek benefits from asking numerous detailed questions, it risks overwhelming the user. In contrast, Llama underperforms due to its reliance on generic or irrelevant questions.

6. Limitations

In our study, the inclusion of open-weight and proprietary models, as well as models from the same family with varying parameterizations, strengthens the generalizability of the findings. However, some design decisions may impact experiments.

Ambiguity detection is limited to the first three turns, as LLMs fail to interact meaningfully if they do not engage early. To evaluate the quality of the questions, we use the change in the latent vector to measure the information obtained. This approach assumes equal importance for all new information, but this may not be true, as different models prioritize different details in their solutions. The resolve rates in the overall problem solving experiment closely replicate real-life conditions where it does not matter how close the generated patch is to the solution, incorrect code is unacceptable. However, data leakage might allow some models to make correct assumptions in underspecified settings, leading to higher resolve rates. In addition, the user proxy may be more interactive than typical real-world users, as LLMs are tuned to be helpful. We mitigate this by limiting the number of interaction turns and ensuring that interactions remain focused on the task using detailed system prompts.

7. Related Work

Code Generation Benchmarks In code generation tasks, ambiguous user instructions hinder the evaluation of code suggestions generated by the model. Since the cause of ambiguity is missing details, clarifying questions become neessary (Mu et al., 2023). Interactive, test-driven workflows mitigate this ambiguity by first generating test cases aligned with user expectations, which users validate before code generation (Lahiri et al., 2023). Extensions of this approach employ runtime techniques to generate, mutate, and rank candidate code suggestions and test cases 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 task-oriented settings, ambiguity between generated outputs and user expectations remains a challenge. AmbigNLG addresses this by introducing a taxonomy of instruction ambiguities and applying targeted disambiguation based on the identified ambiguity type (Niwa & Iso, 2024). These ambiguities include unclear output lengths, mandatory keywords, and contextual nuances in instructions. NoisyToolBench (Wang et al., 2024a) offers a dataset for evaluating LLM tool use with ambiguous instructions, though it focuses on simpler tasks. Reinforcement learning frameworks like ReHAC balance user interaction by modeling optimal intervention points (Feng et al., 2024), but more effective strategies are needed for complex, multi-step workflows.

LLMs and Ambiguity The current state-of-the-art LLMs are not inherently trained to handle ambiguity through user interaction (Zhang et al., 2024), but, their instruction tuning enables improved performance with prompt engineering (White et al., 2023). Ambiguity detection has been tackled with uncertainty estimation to measure the utility of seeking clarification (Zhang & Choi, 2023; Park et al., 2024). Meanwhile, the quality of clarifying questions and the resulting performance remain critical to overall success (Rao & Daumé III, 2018; Pyatkin et al., 2023; Kuhn et al., 2023). Despite advances, state-of-the-art techniques such as few-shot prompting and Chain-of-Thought reasoning offer limited relief in ambiguous scenarios (Zhang et al., 2024). Self-disambiguation uses the internal knowledge of a model to reduce query ambiguity (Keluskar et al., 2024; Sterner, 2022; Sumanathilaka et al., 2024). For example, Alignment with Perceived Ambiguity (APA) employs self-disambiguation to quantify perceived ambiguity using information gain, improving the model's processing of such inputs (Kim et al., 2024). Although inference-only methods are cost-effective, they are less robust than training-based approaches for handling ambiguity.

8. Conclusion

In this work, we have focused on evaluating current proprietary and open-weight models in agentic frameworks for their handling of ambiguity in software engineering settings. In code generation, to effectively integrate new information into its solution trajectory, the agent must detect ambiguity and ask user-targeted questions to gather the necessary details for task completion. The key takeaways are as follows.

- Given an underspecified input, interactive models like Claude Sonnet 3.5 and Claude Haiku 3.5 can achieve 80% of their performance with a well-specified input. In contrast, open-source models struggle to leverage interaction similarly: Deepseek relies heavily on navigational information to locate relevant files, while Llama 3.1 70B is constrained by the limited information it extracts from the user.

- LLMs do not interact unless explicitly prompted, and their ambiguity detection performance is highly sensitive to prompt variations. The models cannot reliably distinguish between well-specified and underspecified input, with only Claude Sonnet 3.5 able to achieve a higher accuracy of 84%.

- Claude Sonnet 3.5 and Haiku 3.5 models, along with the Deepseek model, are able to extract new, and detailed information from the user. The Llama 3.1 model struggles to ask the right questions to gather relevant information from the user.

However, there still exists a significant gap to the resolve rates when a fully specified issue is provided. The open-weight models have a bigger room for improvement in leveraging the interaction to improve resolve rates, whereas the proprietary models, especially Claude Haiku 3.5, need stronger encouragement to interact. This work highlights the state-of-the-art in proprietary and open-weight models for handling ambiguity in software engineering tasks through interaction, breaking the process into multiple steps.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ahmad, L. 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.

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

Chowdhury, N., Aung, J., Shern, C. J., Jaffe, O., Sherburn, D., Starace, G., Mays, E., Dias, R., Aljubeh, M., Glaese, M., Jimenez, C. E., Yang, J., Liu, K., and Madry, A. 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.

Fakhoury, S., Naik, A., Sakkas, G., Chakraborty, S., and Lahiri, S. K. 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.

Feng, X., Chen, Z.-Y., Qin, Y., Lin, Y., Chen, X., Liu, Z., and Wen, J.-R. Large language model-based human-agent collaboration for complex task solving, 2024. URL https://arxiv.org/abs/2402.12914.

Huang, D., Zhang, J. M., Luck, M., Bu, Q., Qing, Y., and Cui, H. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024. URL https://arxiv.org/abs/2312.13010.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Karli, U. B. and Fitzgerald, T. 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.

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

Kim, H. J., Kim, Y., Park, C., Kim, J., Park, C., Yoo, K. M., goo Lee, S., and Kim, T. Aligning language models to explicitly handle ambiguity, 2024. URL https://arxiv.org/abs/2404.11972.

Kuhn, L., Gal, Y., and Farquhar, S. Clam: Selective clarification for ambiguous questions with generative language models, 2023. URL https://arxiv.org/abs/2212.07769.

Lahiri, S. K., Fakhoury, S., Naik, A., Sakkas, G., Chakraborty, S., Musuvathi, M., Choudhury, P., von Veh, C., Inala, J. P., Wang, C., and Gao, J. Interactive code generation via test-driven user-intent formalization, 2023. URL https://arxiv.org/abs/2208.05950.

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.

Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Mu, F., Shi, L., Wang, S., Yu, Z., Zhang, B., Wang, C., Liu, S., and Wang, Q. Clarifygpt: Empowering llm-based code generation with intention clarification, 2023. URL https://arxiv.org/abs/2310.10996.

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

Park, J., Lim, S., Lee, J., Park, S., Chang, M., Yu, Y., and Choi, S. Clara: Classifying and disambiguating user commands for reliable interactive robotic agents, 2024. URL https://arxiv.org/abs/2306.10376.

Peng, S., Kalliamvakou, E., Cihon, P., and Demirer, M. The impact of ai on developer productivity: Evidence from github copilot, 2023. URL https://arxiv.org/abs/2302.06590.

Purver, M. R. J. *The theory and use of clarification requests in dialogue*. PhD thesis, University of London King's College, 2004.

Pyatkin, V., Hwang, J. D., Srikumar, V., Lu, X., Jiang, L., Choi, Y., and Bhagavatula, C. Clarifydelphi: Reinforced clarification questions with defeasibility rewards for social and moral situations, 2023. URL https://arxiv.org/abs/2212.10409.

Rao, S. and Daumé III, H. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2737–2746, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1255. URL https://aclanthology.org/P18-1255.

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

Sumanathilaka, T. G. D. K., Micallef, N., and Hough, J. 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.

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

Wang, W., Shi, J., Wang, C., Lee, C., Yuan, Y., tse Huang, J., and Lyu, M. R. Learning to ask: When llms meet unclear instruction, 2024a. URL https://arxiv.org/abs/2409.00557.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for ai software developers as generalist agents, 2024b. URL https://arxiv.org/abs/2407.16741.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

Xu, F. F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z. Z., Zhou, X., Guo, Z., Cao, M., Yang, M., Lu, H. Y., Martin, A., Su, Z., Maben, L., Mehta, R., Chi, W., Jang, L., Xie, Y., Zhou, S., and Neubig, G. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2024. URL https://arxiv.org/abs/2412.14161.

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

Zhang, T., Qin, P., Deng, Y., Huang, C., Lei, W., Liu, J., Jin, D., Liang, H., and Chua, T.-S. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models, 2024. URL https://arxiv.org/abs/2405.12063.

Zhou, X., Kim, H., Brahman, F., Jiang, L., Zhu, H., Lu, X., Xu, F., Lin, B. Y., Choi, Y., Mireshghallah, N., Le Bras, R., and Sap, M. Haicosystem: An ecosystem for sandboxing safety risks in human-ai interactions. *arXiv*, 2024a. URL http://arxiv.org/abs/2409.16427.

Zhou, X., Zhu, H., Mathur, L., Zhang, R., Yu, H., Qi, Z., Morency, L.-P., Bisk, Y., Fried, D., Neubig, G., and Sap, M. 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.

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}`

A.2. Statistical Methods

A.2.1. ONE-SIDED PAIRED T-TEST

Comparison	t-value	p-value
Interaction vs Hidden	2.1324	0.0614
Interaction vs Full	-5.0430	0.0075

Table 4. Paired t-test results comparing the Interaction setting with the Hidden and Full settings. The t-values and p-values are shown for each comparison.

The one-sided paired t-test is a statistical method used to determine if there is a significant difference between the means of two related groups in a specific direction. In the context of this experiment, the one-sided paired t-test is applied to compare the performance of models between two settings (e.g., Hidden and Interaction, Interaction and Full) with the hypothesis that the performance in the second setting is greater than in the first setting. The test assumes that the differences between paired observations are normally distributed.

Formally, the null hypothesis (H_0) for a one-sided paired t-test is that the mean difference between the two settings is less than or equal to zero, i.e.,

$$H_0 : \mu_d \leq 0$$

where μ_d is the mean difference between the paired observations. The alternative hypothesis (H_1) is that the mean difference is greater than zero:

$$H_1 : \mu_d > 0$$

If the p-value obtained from the test is less than the predetermined threshold (0.05), we reject the null hypothesis, concluding that there is a statistically significant difference in the performance between the two settings.

A.2.2. Shapiro-Wilk Test for Normality

The Shapiro-Wilk test is a statistical test used to assess whether a sample comes from a normally distributed population. It is commonly applied in hypothesis testing where the assumption of normality is important, such as in t-tests. The null hypothesis of the Shapiro-Wilk test is that the sample follows a normal distribution, and the alternative hypothesis is that the sample does not follow a normal distribution.

$$H_0 : \text{The sample is normally distributed}$$

$$H_1 : \text{The sample is not normally distributed}$$

If the p-value from the Shapiro-Wilk test is greater than a chosen significance level (0.05), we fail to reject the null hypothesis and conclude that the sample data does not significantly differ from a normal distribution. Conversely, if the p-value is less than 0.05, we reject the null hypothesis and conclude that the sample data deviates from a normal distribution, which may require alternative statistical methods that do not assume normality.

We used the Shapiro-Wilk test to verify that the data for the paired differences in our experiment follows a normal distribution, ensuring the validity of the t-test assumptions.

- **Hidden Setting:** p-value = 0.570
- **Interaction Setting:** p-value = 0.410
- **Full Setting:** p-value = 0.463

A.3. Ambiguity 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.

Model	Summarized Issue	Selected Questions	Corresponding Answers
Llama 3.1 70B	The InheritDocstrings metaclass is encountering a problem where it fails to handle properties correctly due to its reliance on a function-checking mechanism that does not recognize properties as functions.	1. Are there any existing workarounds or temporary fixes for this problem that I should be aware of? 2. Are there any specific properties or use cases that are causing the issue, or is it a more general problem with the InheritDocstrings metaclass?	1. I don't have that information. 2. This seems to be a more general problem with how the InheritDocstrings metaclass handles properties, not specific to any properties or use cases.
Deepseek chat	The InheritDocstrings metaclass is encountering a problem where it fails to handle properties correctly due to its reliance on a function-checking mechanism that does not recognize properties as functions.	1. Can you provide more details about the InheritDocstrings metaclass? Specifically, what is its purpose, and how is it currently implemented in the repository? 2. Are there any existing tests or examples that demonstrate the issue? If so, can you provide the names of these tests or examples?	1. The InheritDocstrings metaclass is intended to ensure that docstrings are inherited from the parent class when the docstring is not explicitly defined in the child class. It currently uses inspect.isfunction, which doesn't work with properties since they are not functions. 2. I don't have that information.
Claude Haiku 3.5	The InheritDocstrings metaclass is encountering a problem where it fails to handle properties correctly due to its reliance on a function-checking mechanism that does not recognize properties as functions.	1. Could you provide more details about where the InheritDocstrings metaclass is located in the repository? 2. When you say the metaclass "fails to handle properties correctly", could you elaborate on what specific behavior is incorrect?	1. The InheritDocstrings metaclass is located in the file astropy/utils/misc.py. 2. The incorrect behavior is related to docstring inheritance. Specifically, properties that are redefined in a subclass without a docstring do not inherit the docstring from the base class.
Claude Sonnet 3.5	The issue involves a method that is expected to create a header from a byte string, but it does not accept Python 3 bytes as input. The method works with Python 3's unicode strings, assuming ASCII encoding, but this behavior is inconsistent with its counterpart method that handles both text and binary file modes.	1. Which specific method in the codebase is causing the issue with byte string input? Please provide the class name and method name. 2. Can you share an example of the byte string input that should work but currently fails?	1. The specific method causing the issue with byte string input is Header.fromstring. 2. I don't have a specific example of the byte string input, but it should be a byte string that contains the entire header data.

Table 5. Examples of questions and their responses across different models chosen to reflect general observed patterns.