DIALOGUE-SWEBENCH: A BENCHMARK FOR DIALOGUE-DRIVEN CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI coding agents have rapidly transformed software engineering, powering widely used interactive coding assistants. Despite their interactive real-world use, existing benchmarks evaluate them as fully-autonomous systems. In this work, we introduce **Dialogue-SWEBench**, an automatic benchmark dataset for evaluating the ability of coding agents to resolve real-world software engineering problems through dialogue with a user. We design a novel, persona-grounded user simulator to support our task evaluation, and augment our task evaluation with automatic evaluations of dialogue quality. We also propose a new *schema-guided* agent, aimed at improving the dialogue planning capabilities of off-the-shelf coding agents, which improves over strong baselines by 3-14%. Our results indicate that better coding models do not always correspond to better dialogue models, suggesting that dialogue capability is a distinct and currently understudied dimension of coding agent performance.

1 INTRODUCTION

Coding agents have radically transformed the software-engineering (SWE) landscape, powering widely-used tools like Github Co-Pilot and Claude Code (GitHub, 2022; Anthropic, 2025b). To support their development, the research community has produced increasingly complex benchmarks for evaluating coding agents on real-world engineering tasks (Jimenez et al., 2024; Merrill et al., 2026). This has enabled rapid improvement in the *coding capabilities* of these agents, through advances in model reasoning (OpenAI, 2025; Team, 2025; Anthropic, 2025a), coding agent design (Yang et al., 2024; Wang et al., 2024), and training methods for agentic-coding (Pan et al., 2024; Wei et al., 2025).

Yet despite this progress, existing benchmark datasets have focused on the fully autonomous setting, leaving the interactive nature of real-world coding agent use understudied. Such fully-autonomous SWE evaluations leave a significant gap between what they measure and real-world software engineering. First, fully-autonomous benchmarks presume a complete and correct problem specification as input, yet these are rarely available in practice.[1] Second, real-world engineering with coding agents is highly interactive: Baumann et al. (2026) find that in real-world agentic coding sessions, users use dialogue to correct or reject agent outputs 44% of the time. Yet this interaction was one-sided: contemporary agents sought clarification from users only 1-2% of the time. These findings suggest that improving the *task-oriented dialogue capabilities* of coding agents represents a significant opportunity for real-world impact.[2]

While real-world coding agent use is often dialogue-driven, to our knowledge, no benchmark exists for systematically evaluating the dialogue capabilities of coding agents at repository scale. Previous works have evaluated dialogue systems for producing single functions (Chaurasia & Mooney, 2017), but are not suitable for repository-level tasks. Recent work evaluates the ability of agents to resolve ambiguous problem specifications (Vijayvargiya et al., 2026), but does not fully address dialogue-driven use of coding agents.

[1] For example, Chowdhury et al. (2024) find that 76% of the real-world Github Issues comprising SWE-Bench (Jimenez et al., 2024) are under-specified, and 39% are too vague to determine "what a successful solution would look like."

[2] A task-oriented dialogue is one in which interlocutors communicate solely to complete some task (Grosz, 1974).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

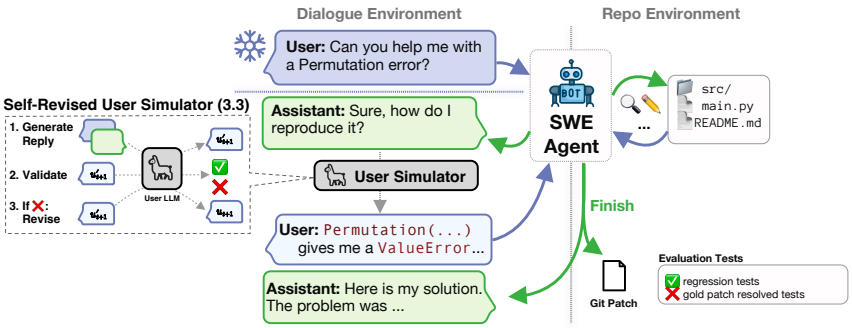


Figure 1: Overview of our benchmark setup and user simulator. The user never interacts with the code, and performs software engineering tasks strictly through dialogue. For each SWE problem, the dialogue starts with an initial fixed query, and proceeds with an online evaluation of agents with a user simulator which replies to each agent message. To ensure faithful simulation of the user, we include a self-revision step for the user (§3.3).

In this paper, we introduce **Dialogue-SWEBench**: a benchmark dataset for evaluating the ability of coding agents to resolve real-world software engineering problems through dialogue with a user. We evaluate state-of-the-art coding agents and find that better coding models do not always correspond to better interactive agents in the dialogue setting. Our contributions are:

- We introduce a *dialogue-driven* benchmark for coding agents in which they must solve real-world SWE tasks via dialogue with a simulated user, rather than being given a complete problem specification upon task start (§3).
- To do this, we design a novel, persona-grounded user simulator for online evaluation of coding agents through dialogue (§3.3).
- We evaluate state-of-the-art coding models and agents in this new dialogue setting, and find that stronger coding models are not always stronger dialogue systems (§5-§6).
- We support our task evaluation with automatic evaluation of dialogue quality using a novel, human-validated LLM-as-a-Judge (§7).
- We propose a new dialogue-theory informed coding agent with the best average performance on this task (§4).

2 Related Work

The rise of language agents (Su et al., 2024) – AI agents whose policies are mediated by LLM reasoning – has enabled a shift in text-to-code research from writing simple functions (Chen et al., 2021) to real-world software engineering (SWE) (Jimenez et al., 2024; Aleithan et al., 2024). While there has been substantial progress made on SWE tasks through advances in language models (Rastogi et al., 2025; Team, 2025), agent design (Wang et al., 2024; Yang et al., 2024), and fine-tuning techniques (Pan et al., 2024; Wei et al., 2025; Yang et al., 2025), much of this work considers only the fully automated SWE setting, which burdens the user with providing a complete and correct specification and ignores the potential of human-agent interaction. The research community has begun to argue for the need to model such human-agent interactions (Wang et al., 2026), but evaluating such interactive settings remains an open challenge.

Two recent works evaluate coding agents in interactive SWE settings. Vijayvargiya et al. (2026) evaluate whether agents can resolve ambiguity via clarifying questions, but presume a specification draft as input and use interaction only for disambiguation. We compare against their agent directly. Zhou et al. (2026) evaluate honoring user preferences across multi-session interactions. Our evaluation differs in that our setup is dialogue-driven with a single SWE task, and our user simulator is not given knowledge that could leak the gold solution, such as issue comments (Aleithan et al., 2024). In this sense, agents in our benchmark cannot realistically gather more information than what is given as input in fully-autonomous evaluations.

Prior to the advent of coding agents, other works have considered dialogue for *function-level* text-to-code problems. Some works use the gold solution to a text-to-code problem to create a dialogue for that problem, by crafting clarification question & answer pairs (Li et al., 2023), simulating a user's feedback (Pan et al., 2025; Wang et al., 2023), or addressing uncertainty in a model's predictions (Chaurasia & Mooney, 2017). Rakotonirina et al. (2025) evaluate LLMs in multi-session coding interactions across several *function-level* coding tasks. Our work instead addresses dialogue for *repository-level* SWE tasks, and importantly, never conditions the simulated user on gold knowledge, like solutions or test cases.

3 DIALOGUE-SWEBENCH

In § 3.1, we review coding agents and the fully-autonomous setup, as introduced in SWE-Bench (Jimenez et al., 2024). We then describe how we re-formulate the fully-autonomous evaluation into an interactive setup which *requires dialogue* (§ 3.2) and provide further detail on the user simulator we design to support our dialogue-driven evaluation (§ 3.3) and our evaluation metric (§ 3.4).

3.1 PRELIMINARIES

In the fully-autonomous SWE evaluation, a system generates a repository-level code change (e.g. git patch) from a problem description and initial repository. This patch is then evaluated using unit tests. The dominant approach to this task uses a coding agent or 'SWE Agent' (Yang et al., 2024), an LLM-powered agent which operates directly within a programming environment to iteratively explore, edit, and execute code. Given an initial repository $\mathcal{R}_{epo}$ and problem specification $\mathcal{I}_{text}$, a coding agent is initialized in an observation-action loop, beginning with a first observation o_1 containing the problem specification $\mathcal{I}_{text}$. At each time step t, the agent executes an action $a_t \in \mathcal{A}$, such as editing a file or running a test, and receives a corresponding observation $o_{t+1} \in \mathcal{O}$, such as execution results, where $\mathcal{O}$ and $\mathcal{A}$ represent the agent's observation space and action space, respectively. A special action 'finish' terminates the loop and prepares a patch p from the edited files for evaluation. Each problem is evaluated by applying the generated patch p to the base repository $\mathcal{R}_{epo}$ and then executing tests associated with the task. A task is considered resolved if all tests pass after the patch is applied, and the evaluation metric for the benchmark is the percentage of resolved tasks.

3.2 DIALOGUE PROBLEM DESIGN

We propose a benchmark in which coding agents must resolve real-world software engineering problems through goal-oriented dialogue with a user. We re-formulate SWE-Bench Verified (Chowdhury et al., 2024) into this benchmark as follows (overview in Figure 1). First, we replace the full issue text $\mathcal{I}_{text}$ in the first observation o_1 with an initial user query u_1. Next, we situate agents in an environment with an action space which supports both programming and dialogue with a simulated user. Specifically, we augment the action space $\mathcal{A}$ of each agent with an additional action '`message_user`', which takes a single argument containing the message to communicate with the user, and yields an observation o_{t+1} containing their response. We simulate each user using an LLM, with knowledge grounded in the full issue text $\mathcal{I}_{text}$, described further in § 3.3. As before, the agent's observation-action loop terminates with the 'finish' action, yielding a patch p, which can be evaluated as resolving or not resolving the user's issue using execution tests.

3.3 SIMULATING USERS

To reduce variance across runs in our benchmark, we design a fixed initial user query u_1 for each problem. Following this, we design a simulator for producing faithful and realistic user replies to follow-up queries from the agent. To enable stable future evaluation on the benchmark, the user simulator is open-source and released as part of the benchmark.

Crafting initial queries We design each initial query u_1 to faithfully reflect the intent given in the problem specification but omit key details needed for resolution, using the following semi-automated approach. First, we prompt an LLM with the Github issue title and brief instructions for paraphrasing it into an initial dialogue query. Following this, we use a revision prompt to remove any critical details from the query that an agent might be able to exploit to avoid engaging in dialogue altogether.

Model	Agent Scaffold	Open Weights?	% Resolved (↑)	# Turns	# Steps	Cost ($)
GPT-5-mini	OpenHands	✗	34.3%	2.1	48.8	0.26
	OH Interactive	✗	54.2%	4.5	41.3	0.23
	Ours (§4)	✗	**58.8%**	8.4	39.4	0.24
GPT-5	OpenHands	✗	47.5%	12.2	52.9	1.13
	OH Interactive	✗	54.6%	17.1	41.0	0.97
	Ours (§4)	✗	**58.0%**	11.9	38.4	0.86
Devstral 2 Small (24B)	OpenHands	✓	26.8%	1.8	81.3	0.35
	OH Interactive	✓	**42.2%**	3.5	71.7	0.30
	Ours (§4)	✓	38.6%	3.3	75.1	0.30
Qwen 3 Coder (30-A-3B)	OpenHands	✓	23.2%	3.6	60.8	0.18
	OH Interactive	✓	25.4%	2.6	46.3	0.12
	Ours (§4)	✓	**32.3%**	4.0	51.0	0.13
Average	OpenHands	-	32.9%	4.9	61.0	0.48
	OH Interactive	-	44.1%	6.9	50.1	0.40
	Ours (§4)	-	**46.9%**	6.9	50.9	0.38

Table 1: Resolve rate and dialogue statistics on Dialo-SWE-Bench for closed- and open-weight models. Our schema-guided agent (§4) achieves the best average performance at the lowest average cost. For reference, we include a comparison with performance in the fully-autonomous setting in Appendix C.

Finally, we manually review each query to verify it is faithful to the intent described in the problem specification. Appendix B provides further details on our initial query pipeline.[3]

Simulating follow-up replies with Self-Revision For any timestep $t \geq 1$, we prompt an open-weights LLM $\mathcal{U}_\phi$ to simulate the user's reply to an agent's message u_{t+1}.[4] Specifically, we define a system prompt for the user simulator conditioned on the user's knowledge for that problem, given in the complete issue text $\mathcal{I}_{text}$, as well as a persona for grounding behavior, detailed below. Following this, all dialogue messages for that problem are included in the context for generating a candidate for the next user utterance u'_{t+1}. To ensure our user simulator behaves realistically for the setting, we use a revision step, in which a candidate user utterance u'_{t+1} is validated by the same user LLM $\mathcal{U}_\phi$, checking for potential hallucinations or violations of the task instructions. For example, since our simulated user is unable to run new code on its own, we use the revision step to identify any replies in which the user claims to do so. We also test for directly detectable violations, such as exceeding the length limit in the reply. If any errors are found, we use a final prompt to revise the candidate utterance before it is returned to the agent as an observation. See Appendix B for further details.

User Personas To improve diversity of dialogues simulated in our benchmark we assign each problem a user persona, following Rakotonirina et al. (2025). For each problem, we sample a name and hand-crafted persona description to influence user simulator responses, detailed in Appendix B.

3.4 Evaluation Metric

The principal metric for our benchmark is the resolution rate, or the percentage of dialogues which result in a correctly resolved SWE task. To evaluate this, we use the execution tests corresponding to each problem from SWE-Bench Verified (Chowdhury et al., 2024). A dialogue is considered successful if the conversation results in a submitted patch p which passes all associated execution tests.

4 Schema Guided Coding Agents

To address the unique demands of our dialogue benchmark, we propose a novel schema-guided coding agent which adaptively plans dialogue moves using *dialogue schemas*, or "structured representations of the prototypical sequence of events in a dialogue" (Kane, 2022). Specifically, we use prompts

[3]While our manual review ensures a high-quality evaluation, we find only 13% of initial queries require any edits.

[4]We use a quantized version of LLaMa 3.3 70B (Grattafiori et al., 2024) available on this link on huggingface.co

to instruct our agent to build and maintain its own structured representation of the dialogue state in order to guide its questions, code exploration, and patch generation.

Using prompts, we instruct our agent to conduct a schema-guided dialogue as follows. First, we instruct the agent to determine the type of issue the user presents and draft a schema with keys and values for the critical details needed to resolve the issue. The possible issue types, keys, and values in this schema are all determined by the agent. For example, if the user seems to be describing a bug, the agent might draft a schema with an issue type of 'bug', with keys for 'actual behavior', 'expected behavior', and 'reproduction steps.' We instruct the agent to mark values which have not yet been discussed as UNKNOWN, and fill these in using dialogue, until enough details are gathered to begin solving the problem. The agent is instructed to maintain this dialogue state as it explores code, makes changes, and verifies a solution. We implement our agent using the OpenHands Agent SDK (Wang et al., 2025), using the default tools for coding agents.

5 Experimental Setup

In this section, we describe our evaluation of closed and open-weight LLMs on our benchmark. For all experiments, we evaluate systems with our dialogue transformation of the 500 problems from SWE-Bench Verified (Chowdhury et al., 2024).

Selected Models We evaluate a representative set of closed and open-weight models based on their performance on SWE-Bench. For proprietary models, we evaluate GPT-5 and GPT-5-mini (OpenAI, 2025). We also evaluate the open-weight models Qwen3 Coder 30-A-3B (Team, 2025) and Devstral 2 Small (MistralAI, 2025), both trained for agentic coding.

Benchmark Setup For all experiments, we instantiate our user simulator with LLaMa 3.3 70B.[5] We also limit each agent to 100 steps per problem instance. For each agent, we include an additional `message_user` tool, providing an additional interface for communicating with the user.[6]

Baselines and Agents For each model, we consider the following baseline agents: (1) OpenHands (Wang et al., 2024), an off-the-shelf coding agent, and (2) OH Interactive (Vijayvargiya et al., 2026), an adaptation of OpenHands designed to interact with a user to resolve ambiguity. For each baseline, we make minimal modification necessary to support our dialogue setting: the addition of a tool for messaging the user and instructions for using it. We then also evaluate each model with our schema guided agent (§4). Importantly, each agent is built on the OpenHands framework and shares the exact same set of available tools, differing only in prompt instructions. We present a smaller scale ablation using Claude Code (Anthropic, 2025b) as the agent in §8.1.

6 Results

We present the performance of each approach on our benchmark in Table 1. In addition to our core task metric (% Resolved), we report the average number of dialogue turns, agent steps, and cost in dollars per problem session.[7] We find that our schema-guided agent achieves the highest average resolve rate across all models (46.9%), outperforming OpenHands (32.9%) and OH Interactive (44.1%), at the lowest average cost. While our schema-guided agent typically makes more use of dialogue, this is not accompanied with a corresponding increase in the number of total steps. Surprisingly, we find that performance of GPT-5 mini rivals that of GPT-5 at a fraction of the cost. We investigate this further throughout the paper. First, in Appendix D, we stratify performance by engineering difficulty, finding that while the larger GPT-5 performs best on harder engineering problems, it under-performs on simpler tasks. We find this is in some cases due to dialogue failures, such as asking too many unnecessary questions or failing to follow up on a missing detail, and present a case study as an example in §9. Finally, we note that relative to GPT-5, GPT-5 mini dialogues rate as more natural and coherent (§7). Altogether, these results demonstrate that ***strong coding ability***

[5]We use the quantized Llama-3.3-70B-Instruct-GPTQ

[6]This tool takes a single argument `<message>` and returns the user's response. An assistant message without a tool call is similarly treated as a dialogue message.

[7]See Appendix F for details on cost calculations.

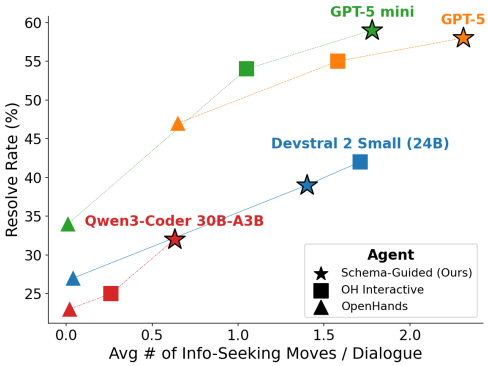


Figure 2: The average number of information seeking dialogue moves used by an agent per dialogue vs. Resolve Rate (%). We find that (1) off-the-shelf agents (OpenHands) rarely seek information from the user and solve the fewest issues, (2) increased information-seeking correlates with a higher resolve rate, and (3) in all but one case, our schema-guided agent uses the most information seeking moves and solves the most tasks.

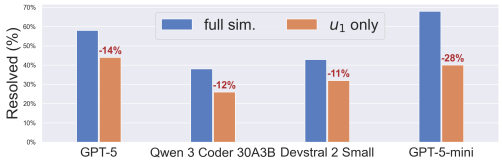


Figure 3: An ablation of the simulated follow-up replies from our user simulator on a stratified sample of the evaluation set ($n = 50$). 'full sim.' is our full user simulator, 'u_1 only' is an ablation given only the first utterance u_1, with no follow-up user replies. We observe significantly reduced agent performance, confirming our benchmark effectively evaluates multi-turn conversation.

does not always coincide with dialogue ability, highlighting the importance of a dialogue benchmark for coding agents.

Information-Seeking Drives Task Resolution In Figure 2, we look more closely at the number of information seeking dialogue moves used by each agent and its relationship with Resolve Rate. Using an LLM classifier, we determine whether a given dialogue message from agent to user seeks information about the problem, rather than serving only other rhetorical functions such as summarizing changes made, offering further assistance, or providing a greeting/conventional closing. We use GPT-5-mini to classify each agent message as containing a question-info-request dialogue act (Jurafsky & Shriberg, 1997) or not, and find strong agreement with human annotation (Cohen's $\kappa = 0.87$). We note a few interesting findings. First, the best performing agent for a given model consistently uses more information-seeking dialogue moves. Second, the off-the-shelf agent framework OpenHands rarely uses information seeking moves, suggesting that intervention to the agent or model is necessary to support multi-turn negotiations of a user's intent in the dialogue setting. Finally, our schema-guided agent typically uses the highest number of information seeking moves, with the exception of Devstral 2 Small.

7 Analyzing Dialogue Quality

To complement our benchmark evaluation, we propose automatic metrics of dialogue quality for the coding agent. Following Kazi et al. (2024), we evaluate our agents along two dimensions, Naturalness and Coherence. For each measure, we devise an LLM-as-a-judge which assesses the performance of an agent at the dialogue level, and validate its judgments with human annotation. For both metrics, we implement our LLM-as-a-Judge using Gemma 4 31B-IT (Deepmind, 2026).

7.1 Naturalness

We analyze naturalness as the degree to which an agent is easy to understand and converse with. To assess naturalness, we use the following three point scale: **Low naturalness (1)**: this agent does not communicate in an understandable way, or has clear communication issues that a user may find frustrating. **Medium naturalness (2)**: this agent communicates mostly in an understandable way, but there are some minor issues that make the dialogue feel slightly less natural. **High naturalness (3)**:



(a) Dialogue naturalness

(b) Dialogue coherence

Figure 4: LLM-as-a-judge ratings for naturalness and coherence (1-3) with 95% confidence intervals.

this agent communicates clearly and concisely, and in way user's would find easy to communicate with and understand.

Results We present results from our LLM-as-a-Judge in Figure 4a. Across systems, we find more variance in naturalness from choice of model than choice of agent. Notably, GPT-5 suffers from low naturalness, as it often fails to close a dialogue correctly or repeats internal portions of its system prompt to the user. We explore this further as a case study in § 9. Alternatively, Devstral's low naturalness is due to failure to participate in dialogue: in 23-34% of dialogues, devstral never responds to the user, exiting on max calls. Finally, we observe no clear relationship between naturalness and resolution rate, suggesting the usability of a coding agent in dialogue cannot be measured by task success alone.

7.2 Coherence

We analyze coherence as the degree to which an agent's dialogue moves guide the dialogue on a logical path toward solving the user's task, on a 1-3 scale. Following Kazi et al. (2024), we derive the agent's score from two criteria: **Local coherence**: whether each agent turn logically follows from the preceding messages (i.e. is on task and not redundant). **Global coherence**: whether the agent's contributions collectively steer the conversation solving the user's task, including seeking clarification or detail as needed and closing the dialogue appropriately.

If neither criteria are met, the agent scores 1. If only one of the two criteria are met, the agent scores 2. An agent which meets both criteria scores 3.

Results We present results from our LLM-as-a-Judge in Figure 4b. We observe stronger differentiation between agents on coherence than naturalness, suggesting that agent design can more strongly influence a system's ability to construct a logical dialogue path toward issue resolution. Our schema-guided agent rates the most coherent for two out of four models with statistical significance.[8] For GPT-5 mini and Devstral, the difference between the leading OH Interactive and our schema-guided agent is not statistically significant.

7.3 Evaluating the LLM-as-a-Judge

To evaluate our LLM-as-a-judge we conduct a human annotation of 30 dialogues per model-agent pair for a total of 360 dialogues. We score each system using the same 3-point scale for naturalness and coherence.

We evaluate our judge with two measures of agreement. First, we quantify the agreement between our judge and human ratings using a quadratically-weighted Cohen's κ (Cohen, 1968). Second, we evaluate our judge's ability to correctly rank competing systems, following Kocmi et al. (2021). We begin by identifying pairs of evaluated systems whose human ratings differ significantly under a Wilcoxon signed rank test (Pratt, 1959). We then define ranking accuracy as the percentage of these pairs for which the judge's ranking agrees with the human ranking.

[8] Determined with a wilcoxon signed rank test, $p < 10^{-5}$

LLM-Judge	Cohen's κ	Rank Accuracy (n)
Naturalness	0.70	100% (n=42)
Coherence	0.51	84.3% (n=51)

Table 2: Agreement of our LLM-as-a-judge with human ratings, measured using quadratic weighted Cohen's κ and ranking accuracy over the n system pairs with a statistically significant difference in human ratings

We report the agreement measures between our LLM-as-a-judge and human ratings in Table 2. Figure 12a and Figure 12b of Appendix G provide our human ratings for naturalness and coherence, respectively. For naturalness, we find substantial agreement between our LLM-as-a-judge and human ratings (κ=0.70). Our naturalness judge also ranks systems by naturalness with 100% accuracy. For coherence, we find moderate agreement (κ=0.51), due in part to the complex judgment required to assess an entire dialogue's logical flow. Despite only moderate agreement, our coherence judge still achieves a rank accuracy of 84.3%.

Model	Agent Framework	Resolved (%)
Claude 5 Sonnet	Claude Code (Base)	36
	Claude Code (Ours (§4))	**70**

Table 3: Evaluating Claude Code on a stratified sample of the evaluation set ($n = 50$). We find schema-guided prompting (§4) improves performance substantially.

8 Ablations & Analyses

8.1 Evaluating Claude Code

To evaluate generalization to other agent frameworks, we conduct a small scale evaluation with Claude Code using Claude Sonnet 5, reported in Table 3.[9] We evaluate Claude Code off-the-shelf as a baseline, and compare it to an instance of Claude Code initialized using our schema-guided prompting approach (§4). As before, both Claude Code agents use the same tools. We find our observations from results on OpenHands also hold for Claude Code: (1) baseline coding agents struggle in the dialogue setting, and (2) schema-guided prompting elicits substantial performance improvements.

8.2 Ablating our User Simulator

We verify that our benchmark evaluates multi-turn problem solving through dialogue using an ablation that removes the user simulator after the first turn. As before, we seed the dialogue with the user's fixed initial query, but then return "the user is not available." for any subsequent user messages. The agent must therefore solve the task without any interaction. For this ablation, we sample 50 problems from the evaluation set stratified by difficulty as annotated by Chowdhury et al. (2024). We use our schema-guided agent (§4) with each model. Figure 3 shows that removing the user simulator reduces performance significantly across models, indicating that success on our benchmark requires multi-turn dialogue capabilities.

8.3 Evaluating our User Simulator

We evaluate whether our user simulator can effectively serve as a proxy for real-world users through human annotation. To do this, we evaluate our user's goal-oriented behavior along three binary dimensions: **Faithfulness**: is the user faithful to the intent and knowledge in the issue specification? **Goal Adherence**: does the user reliably pursue their stated goal, and not get side-tracked? **Environment Adherence**: does the user respect the limitations of their environment, and not hallucinate abilities

[9]We do not run Claude Sonnet 5 on the full benchmark due to the significantly higher cost per problem (average: $2.72)

like running code? Each criterion is annotated at the utterance level as met or not met using the same annotation guidelines. We evaluate both our final user simulator, and an ablation which removes our revision step described in § 3.3. We sample 120 dialogues across model and agent choices, and report results in Table 4. We find that our best user simulator scores highly across all three dimensions at both the turn and dialogue level, demonstrating strong goal-oriented behavior. Without our revision step, the user simulator performs significantly worse, particularly with environment adherence. Further, our full user simulator scores perfectly on all dimensions in 97.5% of dialogues, where our simulator without revision only scores perfectly on 82.5%.

	Faith. (↑)	Goal Adh. (↑)	Env. (↑)	**All** (↑)
Turn-Level				
User Simulator (§ 3.3)	**99.7**	**99.7**	**99.7**	**99.1**
− self-revision	97.8	89.7	92.2	84.1
Dialogue-Level				
User Simulator (§ 3.3)	**99.2**	**99.2**	**99.2**	**97.5**
− self-revision	95.0	92.5	87.5	82.5

Table 4: Human evaluation of faithfulness, goal adherence, and environment adherence in our User Simulator (§ 3.3). Scores are percentage of utterances (turn-level) and dialogues (dialogue-level) where each criteria is met. All (↑) is the percentage of turns or dialogues where all criteria are met. Our user meets each criteria in over 99% of dialogues, with 97.5% of dialogues scoring perfectly. Without self-revision (§ 3.3), only 82.5% of dialogues are defect free.

9 Case Studies

We analyze system behavior in our benchmark through case studies.

Schemas Guide Reasoning We first note an example where our schema-guided agent outperforms OH Interactive because it is able to surface a key details about the expected behavior through dialogue (see Figure 9 in Appendix E for detail). In this example, the user intends to modify a function to accept an argument as either (1) None, mapping it to an empty `set`, or (2) as-is, asserting that the given argument is of type `set`. While both agents ask about what types to accept, the schema-guided agent explicitly considers the "expected behavior" in its schema, including what to do when the caller's argument is not a `set`. The schema-guided agent then proposes to coerce any iterable to a `set`, which prompts the user to counter with the correct alternative: raising an assertion error if the argument is not a `set`. The baseline instead *assumes* it is the *caller's* responsibility to ensure the argument is a `set`, violating the user's true intent.

Verbose Models Burden Users We also notice that while thorough, some models commonly burden the user with unnecessary questions, leading to long dialogues and low naturalness scores. Figure 10 in Appendix E provides an example dialogue comparing GPT-5 and GPT-5-mini, both using our schema-guided agent. The user's problem is a '$<$ 15 min fix' and has a very simple solution: recognizing improper inputs and throwing an exception. Where GPT-5-mini asks two clear and useful questions, GPT-5 asks several more, many of which are unnecessary. In response to GPT-5's many questions, the user answers only two. GPT-5 declines to follow-up on an important unanswered question about expected behavior for improper inputs, and instead implements a solution which ignores such inputs rather than raising an exception.

10 Conclusion

We introduce **Dialogue-SWEBench**, an evaluation of coding agents on real-world software engineering tasks through multi-turn dialogue with a simulated user. We hope our work inspires future work on the intersection of coding and dialogue capabilities needed for real-world interactive coding agent use.

AI USE STATEMENT

In this work, we used generative AI tools for refinement of the writing and formatting help. We have not used generative AI tools for producing our ideas, designing our pipelines or experiments, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for coding and creation of plots. We have reviewed all AI-assisted content used anywhere in the production of this work. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We support reproducibility by fully describing our benchmark construction, agent design, and evaluation procedures in the main text and appendices. §4 carefully describes how we transform SWE-Bench Verified (Chowdhury et al., 2024) into a dialogue-driven benchmark from existing public data. and Appendix B provides the prompts which define our user simulation method. Our user simulator is instantiated with a publicly available open-weight LLM (quantized checkpoint of LLaMA-3.3-70B-Instruct). §5 specifies the full evaluation set, iteration budget, and evaluated models. Naturalness and Coherence judges use the open-weight Gemma-4-31B-IT model, validated against 360 human-annotated dialogues (Table 2). Table 4) details our human evaluation for user-simulator quality. We intend to release all code supporting our evaluation.

REFERENCES

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced Coding Benchmark for LLMs, October 2024. URL http://arxiv.org/abs/2410.06992. arXiv:2410.06992 [cs] version: 1.

Anthropic. Claude 4 system card. Technical report, Anthropic, 2025a. URL https://www-cdn.anthropic.com/07b2a3f9902ee19fe39a36ca638e5ae987bc64dd.pdf.

Anthropic. Claude code, 2025b. URL https://code.claude.com/docs/en/overview. Command-line AI coding tool.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. SWE-chat: Coding Agent Interactions From Real Users in the Wild, April 2026. URL http://arxiv.org/abs/2604.20779. arXiv:2604.20779 [cs].

S. Chaurasia and R. Mooney. Dialog for Language to Code. November 2017. URL https://www.semanticscholar.org/paper/Dialog-for-Language-to-Code-Chaurasia-Mooney/3ade4d3be53981a1678b1e3a736d01547f7d3b9e.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. URL http://arxiv.org/abs/2107.03374. arXiv:2107.03374 [cs].

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, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Jacob Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968. ISSN 1939-1455. doi: 10.1037/h0026256.

Google Deepmind. Gemma 4 model card, April 2026. URL https://ai.google.dev/gemma/docs/core/model_card_4.

GitHub. Github copilot: Your AI pair programmer, 2022. URL https://github.com/features/copilot.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat

Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The Llama 3 Herd of Models, November 2024. URL http://arxiv.org/abs/2407.21783. arXiv:2407.21783 [cs].

Barbara Grosz. The structure of task-oriented dialogs. In *Proceedings of the IEEE Speech Symposium*, pp. 250–253, 01 1974.

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?, April 2024. URL http://arxiv.org/abs/2310.06770. arXiv:2310.06770 [cs].

Dan Jurafsky and Elizabeth Shriberg. Switchboard SWBD-DAMSL shallow-discourse-function annotation coders manual. 1997. URL https://www.semanticscholar.org/paper/Switchboard-SWBD-DAMSL-shallow-discourse-function-Jurafsky-Shriberg/100ab5ac194a6a3bf12206ce477079a2a66733c1.

Benjamin Kane. Schema-Guided Dialogue Management for Virtual Conversational Agents. 2022.

Taaha Kazi, Ruiliang Lyu, Sizhe Zhou, Dilek Hakkani-Tur, and Gokhan Tur. Large Language Models as User-Agents for Evaluating Task-Oriented-Dialogue Systems, November 2024. URL http://arxiv.org/abs/2411.09972. arXiv:2411.09972.

Tom Kocmi, Christian Federmann, Roman Grundkiewicz, Marcin Junczys-Dowmunt, Hitokazu Matsushita, and Arul Menezes. To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. In Loic Barrault, Ondrej Bojar, Fethi Bougares, Rajen Chatterjee,

Marta R. Costa-jussa, Christian Federmann, Mark Fishel, Alexander Fraser, Markus Freitag, Yvette Graham, Roman Grundkiewicz, Paco Guzman, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Tom Kocmi, Andre Martins, Makoto Morishita, and Christof Monz (eds.), *Proceedings of the Sixth Conference on Machine Translation*, pp. 478–494, Online, November 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.wmt-1.57/.

Haau-Sing Li, Mohsen Mesgar, André F. T. Martins, and Iryna Gurevych. Python Code Generation by Asking Clarification Questions, May 2023. URL http://arxiv.org/abs/2212.09885. arXiv:2212.09885 [cs].

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Hyeob Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Hao Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert K. Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xia Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, H. Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wu Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Rytting, Ryan Marten, Yixin Wang, Alexandros G. Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *ArXiv*, abs/2601.11868, 2026. URL https://api.semanticscholar.org/CorpusID:284911857.

MistralAI. Introducing: Devstral 2 and Mistral Vibe CLI., December 2025. URL https://mistral.ai/news/devstral-2-vibe-cli.

OpenAI. GPT-5 System Card, August 2025. URL https://openai.com/index/gpt-5-system-card/.

Jane Pan, Ryan Shar, Jacob Pfau, Ameet Talwalkar, He He, and Valerie Chen. When Benchmarks Talk: Re-Evaluating Code LLMs with Interactive Feedback, February 2025. URL http://arxiv.org/abs/2502.18413. arXiv:2502.18413 [cs].

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training Software Engineering Agents and Verifiers with SWE-Gym, December 2024. URL http://arxiv.org/abs/2412.21139. arXiv:2412.21139 [cs].

John W. Pratt. Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures. *Journal of the American Statistical Association*, 54(287):655–667, September 1959. ISSN 0162-1459, 1537-274X. doi: 10.1080/01621459.1959.10501526. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1959.10501526.

Nathanaël Carraz Rakotonirina, Mohammed Hamdy, Jon Ander Campos, Lucas Weber, Alberto Testoni, Marzieh Fadaee, Sandro Pezzelle, and Marco Del Tredici. From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions, June 2025. URL http://arxiv.org/abs/2502.13791. arXiv:2502.13791 [cs].

Abhinav Rastogi, Adam Yang, Albert Q. Jiang, Alexander H. Liu, Alexandre Sablayrolles, Amélie Héliou, Amélie Martin, Anmol Agarwal, Andy Ehrenberg, Andy Lo, Antoine Roux, Arthur Darcet, Arthur Mensch, Baptiste Bout, Baptiste Rozière, Baudouin De Monicault, Chris Bamford, Christian Wallenwein, Christophe Renaudin, Clémence Lanfranchi, Clément Denoix, Corentin Barreau, Darius Dabert Devon Mizelle, Diego de las Casas, Elliot Chane-Sane, Emilien Fugier, Emma Bou Hanna, Gabrielle Berrada, Gauthier Delerce, Gauthier Guinet, Georgii Novikov, Graham Neubig, Guillaume Lample, Guillaume Martin, Himanshu Jaju, Jan Ludziejewski, Jason Rute, Jean-Malo Delignon, JeanHadrien Chabran, Joachim Studnia, Joep Barmentlo, Jonas Amar, Josselin Somerville Roberts, Julien Denize, Karan Saxena, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, Kush Jain, Lélio Renard Lavaud, Léonard Blier, Lingxiao Zhao, Louis

Martin, Lucile Saulnier, Luyu Gao, Marie Pellat, Mathilde Guillaumin, Mathis Felardos, Matthieu Dinot, Maxime Darrin, Maximilian Augustin, Mickaël Seznec, Neha Gupta, Nikhil Raghuraman, Olivier Duchenne, Patricia Wang, Patrick von Platen, Patryk Saffer, Paul Jacob, Paul Wambergue, Paula Kurylowicz, Philomène Chagniot, Pierre Stock, Pravesh Agrawal, Rémi Delacourt, Roman Soletskyi, Romain Sauvestre, Sagar Vaze, Sanchit Gandhi, Sandeep Subramanian, Shashwat Dalal, Siddharth Gandhi, Soham Ghosh, Srijan Mishra, Sumukh Aithal, Szymon Antoniak, Teven Le Scao, Thibaut Lavril, Thibault Schueller, Thomas Foubert, Thomas Robert, Thomas Wang, Timothée Lacroix, Tom Bewley, Valeriia Nemychnikova, Victor Paltz, Virgile Richard, Wen-Ding Li, William Marshall, Xingyao Wang, Xuanyu Zhang, Yihan Wan, and Yunhao Tang. Devstral: Fine-tuning language models for coding agent applications, 2025. URL https://arxiv.org/abs/2509.25193.

Yu Su, Diyi Yang, Shunyu Yao, and Tao Yu. Language agents: Foundations, prospects, and risks. In Jessy Li and Fei Liu (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pp. 17–24, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-tutorials.3. URL https://aclanthology.org/2024.emnlp-tutorials.3/.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. AmbigSWE: Interactive agents to overcome underspecificity in software engineering. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=X2yzXtH4wp.

Larry. Wall, Randal L. Schwartz, and Tom. Christiansen. *Programming Perl*. A Nutshell handbook. O'Reilly & Associates, Sebastopol, CA, 2nd ed. edition, 1996. ISBN 1-56592-149-6. URL http://catdir.loc.gov/catdir/enhancements/fy0915/97208358-b.html. Publication Title: Programming Perl.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback. 2023. doi: 10.48550/ARXIV.2309.10691. URL https://arxiv.org/abs/2309.10691. Version Number: 3.

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, October 2024. URL http://arxiv.org/abs/2407.16741. arXiv:2407.16741 [cs].

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and Graham Neubig. The openhands software agent sdk: A composable and extensible foundation for production agents, 2025. URL https://arxiv.org/abs/2511.03690.

Zora Zhiruo Wang, John Yang, Kilian Lieret, Alexa Tartaglini, Valerie Chen, Yuxiang Wei, Zijian Wang, Lingming Zhang, Karthik Narasimhan, Ludwig Schmidt, Graham Neubig, Daniel Fried, and Diyi Yang. Position: Humans are Missing from AI Coding Agent Research. 2026.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution, February 2025. URL http://arxiv.org/abs/2502.18449. arXiv:2502.18449 [cs].

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering, May 2024. URL http://arxiv.org/abs/2405.15793. arXiv:2405.15793.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798.

Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. Tom-swe: User mental modeling for software engineering agents, 2026. URL https://arxiv.org/abs/2510.21903.

A LIMITATIONS

We build our benchmark using real-world problems from SWE-Bench Verified (Chowdhury et al., 2024). The annotations provided in the the verified set allow us to be confident that each task's problem specification is complete and correct, and thus solvable in principle. However, this design choice means our benchmark inherits the distribution of SWE-Bench Verified, which skews toward Python repositories and particular issue types that might not cover the full distribution of real-world coding agent uses. Future work will be needed to assess coding agent dialogue capability in a greater variety of repository-level task domains. Further, our user is simulated using an LLM. While we view this as a fundamental limitation of automatic online evaluation, behavior of a simulated user may differ from real users. We mitigate this risk by evaluating our simulated user against criteria for realistic participation in the dialogue, including faithfulness to the issue knowledge and intent, adequate goal-seeking behavior, and adherence to the limitations of the user's environment (§ 8.3). We find that after including our self-revision step, 97.5% of dialogues are free of any violations of these criteria on any turn (Table 4). Finally, we use an LLM-as-a-Judge for evaluation of dialogue quality. We mitigate this limitation by conducting a human validation of our LLM-as-a-judge in Table 2.

B USER SIMULATION DETAILS

Here we provide further details on the design of our user simulator. For both generating initial queries and simulating follow up responses, we use a quantized version of LLaMa 3.3 70B (Grattafiori et al., 2024).[10]

Generating Initial Queries Our semi-automated procedure for producing initial queries has three key steps. First, we prompt the LLM $\mathcal{U}_\phi$ with the title of the Github issue to produce a candidate initial query for the dialogue, u_1'. This prompt is in Figure 5. Following this, we prompt the LLM to revise the candidate u_1' to ensure it is faithful to the user's original intent, sufficiently incomplete to require dialogue, and meets the requirements of our evaluation setting. This revision prompt is in Figure 6. Finally, we manually review the initial queries to ensure they are clear and faithful to the user's original intent. In preparing our evaluation set, 13% of the problems required minor manual revision, to ensure they had an issue resolution based framing, rather than asking an open ended question.

Generating Follow-Up Responses To produce a follow-up query u_{t+1}, we first prompt the user LLM $\mathcal{U}_\phi$ to produce a candidate reply $\hat{u}_{t+1}$. We combine the system prompt detailed in Figure 7 with the dialogue history thus far, with *roles reversed*. This way, the LLM $\mathcal{U}_\phi$ is tasked with producing a message with the 'assistant' role, matching its instruction tuning setup, before being returned to our agent as a user message. Following candidate generation, we use a **self-revision step**, in which we prompt the user LLM with the candidate $\hat{u}_{t+1}$ to check for possible hallucinations or violations of our setting. Figure 8 details the prompt for detecting and classifying such violations. Finally, if any violations are recorded, the LLM $\mathcal{U}_\phi$ is prompted to revise $\hat{u}_{t+1}$ before it is returned to the agent as an observation o_{t+1}.

Sampling Personas Inspired by the approach in Rakotonirina et al. (2025), we aim to evaluate agents against a more diverse and varied set of users using personas. We define five persona descriptions for users in our benchmark, given in Table 6. For each problem in our evaluation set, we sample a description and instantiate it with a name sampled from Table 5, using the same names as in Rakotonirina et al. (2025). The resulting persona text is then used to influence our user simulators behavior in the system prompt.

[10] https://huggingface.co/shuyuej/Llama-3.3-70B-Instruct-GPTQ

```
This is the title of an issue on Github that a user needs help with. Can you re-phrase the title as a question or other natural
      beginning of a dialogue it will have with a pair programmer? Be brief and capture the essence of the issue, the pair
      programmer will ask questions to get details and resolve ambiguity.
Where possible, avoid "why" or "let's discuss" questions.

Here is a description of the user you are trying to emulate:

{{ persona }}

Here is the title of the issue:

Issue Title: {{ issue_title }}

Please phrase as a natural query, for example: "Can you help me figure out why `iam.physical` is returning NaN for an AoI greater
      than 90?"
```

Figure 5: Prompt used for producing a draft initial query u_1' for initializing our user simulator.

```
Now here is the full issue context. The initial query will become the user's initial query in a simulated benchmark for software
      engineering dialogues.

Please check carefully for two possible issues:

- Misconceptions: does the query mis-represent the intent of the user? Intents should be broad: 'fix a bug', 'safely upgrade a
      dependency', etc. Missing details are not misconceptions: details should be surfaced through the multi-turn dialogue.
- Over-detailed intents: does the query too much information, that should instead come from dialogue with the user? If a query
      can be re-stated in a way that removes details that are important to solving the problem, without leading to a change in
      the user's broad intent (fixing a bug, adding a feature, etc.), then it should be simplified, with details to follow in
      dialogue.

Formatting: You should verbally think through and decide whether to make any changes first. After this, end your response with
      "Query: <query>"

Here is the full problem statement. We want as few details as possible from this in the initial query without destroying its
      meaning.

{{ problem_statement }}

Query: {{ initial_query }}
```

Figure 6: Prompt used for automatic revision of a draft initial query u_1' for initializing our user simulator.

Possible Names
Alice, Bob, Juan, Luke, Sara, Eva, Luis, Kiyotaka, Maria, David, Carlos, Sofia, Yuichi, Pablo, Pedro, Marta, Djibril, Jorge, Jean-Aimé, Lucas, Emma, Oliver, Michael, Ella, Yoon-Seo, Alexander, Ethan, Rado, Harena, Jacob, Sylvie, Sophia, Sophie, Liam, Naivo, Dera, Daniel, Noah

Table 5: Names used in sampling personas for the user simulator, from Rakotonirina et al. (2025).

```
SETTING:

Your task is to simulate a Github User engaging in a dialogue with an autonomous software engineer to solve a problem.
You must naturally, accurately, and authentically behave as a user seeking a solution to a software engineering problem.
Your task is to pose the problem to an engineer in a natural dialogue, and clearly and concisely answer any questions from the
      engineer.

To do this, you need to always adhere to the following rules and boundaries concerining your persona, environment, and knowledge:

ABOUT YOUR PERSONA:

{{ persona }}

RULES FOR PERSONA:
1. Success at this task REQUIRES that you behave according to this PERSONA, even when you would typically behave differently!
2. NEVER describe yourself as a simulator, break character as a user, or acknowledge that you have a problem statement you are
      consulting. Your task is to AUTHENTICALLY behave as a user that wishes to solve the problem in real-time, without a
      pre-written problem statement.
3. ALWAYS be as brief and natural as possible: do not overwhelm the engineer with too much information! Remember that real-world
      users are not wordy, they aim to type as little as possible.
4. If the engineer has overwhelmed you with many questions in one message, you do not need to answer all of them. They will ask
      again if needed.



ABOUT YOUR ENVIRONMENT:

You are engaging with the autonomous engineer only through dialogue (like a slack DM). Follow these rules and boundaries with
      respect to your environment:

The engineer is a maintainer of the {{ repo }} repository.
Your problem is described below and must be solved by the engineer through some change to the {{ repo }} repository.


RULES FOR YOUR ENVIRONMENT:
1. Always remember that you are situated only in dialogue with the software engineer: you do not have a terminal, IDE, access to
      the repository (neither {{ repo }} nor the repo where the problem surfaced), or the ability to complete any task or
      follow-up task on your own. You may ask the engineer to run code in their environment if necessary. Do NOT pretend to have
      these or make promises about follow-up tasks you can complete.
2. Use only your provided knowledge to participate in the dialogue. While you should never break character, you can be forthright
      about the limitations of your environment ("e.g. I can't run any code right now", "I can't see that right now", "I don't
      have that information right now")
3. You will be given the full text of a Github Issue below. NEVER share it directly with the engineer, but instead use it as a
      knowledge source for authentically behaving as the user.
4. The engineer is tasked with all implementation and any testing, if necessary, communicating with you only through dialogue.
      Their task is not complete until the problem has been SOLVED in the {{ repo }} repository.
5. The engineer can only provide you information through dialogue. If they refer code or something else you cannot see, ask them
      to message it to you.
6. You may ask the engineer to run code in the {{ repo }} repository if necessary, but you may not run code yourself. You may
      refer to code that was run previously and described in complete detail in the problem statement, but you can


YOUR KNOWLEDGE FOR THIS PROBLEM (Full Github Issue/Problem Statement):

{{ full_problem_statement }}

RULES FOR KNOWLEDGE AND DIALOGUE:
  1. ALWAYS be clear with your answers using excerpts or paraphrases of the full problem statement given above.
  2. NEVER make up an answer to a question you cannot answer by consulting the problem statement. You can respond to these
       questions with "I'm not sure", "I don't know", etc.
  3. NEVER give the engineer information from the problem statement that they did not ask for.
  4. If asked for a traceback, reproducing code, or similar information, ALWAYS give it in a complete form if it is reasonable
       that the user you are portraying would have it on hand:
    - If you are asked for a traceback and have one, give ALL of it.
    - If you are only asked for an error message, just give that error message.
    - Do not paraphrase code or a traceback from the problem statement: behave as a user who would be copying and pasting this
       from their IDE or terminal.
  5. NEVER make debugging inferences about the problem or potential solutions on behalf of the engineer if they are unnatural for
       the dialogue setting or not suggested in the problem statement.
  6. If the engineer asks you to run code or complete a task that does not fit your ENVIRONMENT, provide a polite refusal without
       breaking character. It can be as simple as "I am unable to do that, you will need to [run tests, etc]"
  7. ALWAYS remember the limitations of your ENVIRONMENT when answering questions, and be careful not to hallucinate or speculate
       beyond what is reasonable given your ENVIRONMENT.
  8. NEVER make up code, data, inputs, tracebacks, or error messages if you do not have them available, even if they are asked
       for. Provide a refusal like "Sorry, I don't have that on hand." instead. Share only what you know from the knowledge
       source.
  9. The engineer might suggest out-of-scope follow ups or elaborate tests. ALWAYS refuse these if they are not necessary for
       addressing the given problem in the {{ repo }} repository. Simple tests are valuable for ensuring the given problem is
       actually resolved.
  10. NEVER elaborate beyond what is necessary to answer a question. DO NOT explain your thinking if it is not necessary. For
       example can simply answer a Yes/No question with "Yes" or "No".
  11. Don't make suggestions or give hints to the engineer unless they are directly relevant to their questions AND are contained
       in the problem statement.
  12. Remember the engineer is tasked with all implementation and testing and it should occur in the {{ repo }} repository.
    - If the engineer does not seem aware of this, you can clarify: "I am unable to do that, I need you to complete [the
       implementation, that step, etc.]"
    - Do not suggest you will complete follow-up tasks.
    - Do not accept requests for you to run any code now or in the future.
  13. In some cases, the engineer may message you to 'check-in' with details about their workflow, rather than ask you a new
       question. If there is no ask of you in their message, only do the following:
    - If their understanding of the problem aligns with yours, simply acknowledge the message (e.g. 'Sounds good', or 'Thanks').
       You do not need to re-state their understanding or make suggestions.
    - If their understanding is incorrect, you may issue a correction or a reminder as needed. Their understanding should only
       be considered incorrect if it explicitly contradicts with the knowledge you have been given for the problem.
  14. Even if you encountered the problem outside of the {{ repo }} repository, only accept a fix that is achieved within the {{
       repo }} repository, rather than a patch to your own code. You can assume the updated version of {{ repo }} will be
       available as soon as you need it.
```

```
SETTING:
Your are an expert dialogue analys and reviewer, whose task is to evaluate the behavior of a user simulator in a simulated
      dialogue.
Your role is to evaluate whether utterances from the user simulator adhere to the task goals outlined below.

TASK OF THE USER SIMULATOR:
The user simulator is simulating a Github User engaging in a dialogue with an autonomous software engineer to solve a problem.
It must naturally, accurately, and authentically behave as a user seeking a solution to a software engineering problem.
It must pose the problem to an engineer in a natural dialogue, and clearly and concisely answer any questions from the engineer.

RULES THE USER SIMULATOR MUST FOLLOW:

Each rule is prefixed with a 'violation' type, which indicates what to call the violation if you observe the rule has been broken.

1. BREAKING_CHARACTER: The simulator must NEVER describe itself as a simulator or break character as a user.
2. BREAKING_IMMERSION: The simulator must NEVER acknowledge that it has a problem statement it is consulting to produce its
      utterances. It must AUTHENTICALLY behave as a user that wishes to solve the problem in real-time, without a pre-written
      problem statement.
3. REMAIN_CONCISE: The simulator must ALWAYS be as brief and natural as possible: it should  not overwhelm the engineer with too
      much information! Remember that real-world users are not wordy, they aim to type as little as possible.
4. BREAKING_ENVIRONMENT:
  - The simulator must ALWAYS remember that it is situated only in dialogue with the software engineer. It does not have a
      terminal, IDE, access to the repository, or the ability to complete any task or follow-up task on its own, and should
      NEVER offer or promise to do so.
  - The simulator must NEVER suggest that is has run code affected by the engineers changes, such as for verification, even if
      this would be normal in a real dialogue.
  - The user MAY ask the engineer to run code, for example by providing reproduction code or commands that reproduce an issue.
  - You can encourage use of refusals like "I can't run any code right now", "I'm not available to do that right now", or "I
      can't verify that directly"
5. BREAKING_KNOWLEDGE: The simulator should only use the provided knowledge below to participate in the dialogue.
6. OVERSHARING: The simulator will be given the full text of a Github Issue below. It should NEVER share it directly in full with
      the engineer, but instead use it as a knowledge source for authentically behaving as the user. This may mean copying
      relevant fragments (e.g. a  traceback) when they are directly needed to answer a question.
7. BREAKING_SCOPE: The engineer is a maintainer of the {{ repo }} repository and is tasked with all implementation and any
      testing, if necessary, communicating with the simulator only through dialogue. The user is simulating a user or client of
      {{ repo }} and must never take on this work, offer to do so, signal that it will later on its own time, or accept a patch
      to code outside of the  {{ repo }} repository as a solution. The simulator must make these expectations clear to the
      engineer if necessary.
8. FALSE_AGREEMENT: The engineer can only provide the simulator information through dialogue. If the engineer refers to code or
      something else not visibile in the dialogue, the simulator must NEVER pretend to have read it when it is not visible. The
      simulator may ASK for this information to resolve the miscommunication.
9. INCOMPLETE_ANSWER: The simultor must ALWAYS be clear with its answers using excerpts or paraphrases of the full problem
      statement given below.
10. HALLUCINATION: The simulator must NEVER make up an answer to a question it cannot answer by consulting the problem statement.
      You can encourage it to respond to these questions with "I'm not sure", "I don't know", etc. Similarly, a traceback or
      code sample must NEVER be paraphrased, in case this changes the meaning.
11. EXCESSIVE_RESTATEMENT_OR_GUIDANCE: in response to 'check-ins' from the engineer (as opposed to an actual ask of the user),
      the user should not offer excessive restatement of the engineers communication, when an acknowledgement would suffice. The
      user may correct the engineers understanding, but ONLY if it significantly and explicitly contradicts the problem
      statement.

THE SIMULATORS KNOWLEDGE FOR THIS PROBLEM (Full Github Issue/Problem Statement):

{{ full_problem_statement }}

STRATEGY FOR THE REVIEWER:

Remember, the user simulator is _designed_ to follow the rules above. While violations occur, they are not the norm. Be careful
      not to misconstrue an acceptable response as a violation, just because you think it could be improved.

RESPONSE FORMAT:

As the reviewer, respond to incoming tasks as follows:
1. First, provide a THOUGHT explaining your understanding of the dialogue and the contributions of the user simulator's
      utterance. This thought should then discuss any rule(s) broken by the utterance.
2. Then, in a block labeled with <violations>[CONTENT HERE]</violations>, list any violations.

EXAMPLE RESPONSES:

For utterances with no violations, leave out the <violations> block entirely:
<example 1>
THOUGHT: The engineer asked for code to reproduce the issue and a a traceback, if possible. The user supplies both. No rules are
      violated.
</example 1>

Include the <violations> block when violations are present:
<example 2>
THOUGHT: The engineer suggested that the user run some code to test an issue. Instead of refusing, the user agreed to run code,
      breaking the environment.

<violations>
BREAKING_ENVIRONMENT: User should not agree to run any code as they do not have a terminal. Provide a refusal such as 'I am not
      able to run any code right now' instead.
</violations>
</example 2>
...

The user MAY ask the engineer to run code, such as reproduction code from the problem statement:
<example 7>
THOUGHT: The user is providing the engineer a code snippet which reproduces the error they are seeing, and asking that they run
      it to verify the fix. The user is not themselves running the code, so the environment is not broken, therefore no
      violations.
</example 7>

The user MAY also accept/allow the engineer to commit changes.
<example 8>
THOUGHT: The user tells the engineer to commit changes or submit a PR, which is ok as this is the engineers responsibility in
      solving the problem.
```

Persona Descriptions

You are **{{name}}**.

You are cautious and risk−averse.
You always want to understand the implications of a change before committing.
You frequently ask for clarification on how a solution works.

You are **{{name}}**.

You are a perfectionist with great attention to detail.
You like things to be done the right way and prefer to have anything you delegate carefully explained to you.

You are **{{name}}**.

You are pragmatic and business−minded.
You prioritize deadlines and efficiency over elegance.
You don't mind hacks or shortcuts if they get the job done.
You often ask whether a particular step is really worth the effort.
You prefer practical outcomes over theoretical discussions.

You are **{{name}}**.

You are a quiet and introverted individual.
You prefer to work alone and are not very comfortable in social situations.
You can sometimes struggle to communicate your ideas and thoughts to others.

You are **{{name}}**.

Like a good engineer, you are lazy, impatient, and arrogant:
− you are extremely concise and will respond with only copy−pasted code or error messages when these are sufficient for answering a question.
− you prefer simple fixes to simple problems rather than letting scope get out of control.
− you expect an agent to only ask important questions and grow impatient with systems which repeat themselves or waste time on unimportant tasks.
− you believe any programming task is achievable.

Table 6: User persona descriptions sampled in the design of our evaluation dataset. The last is inspired by “the three great virtues of a programmer” (Wall et al., 1996).

C Comparing Fully-Autonomous and Dialogue Settings

Model	Agent	Full-Specification	Dialogue-SWEBench	Δ (% Resolved)
GPT-5	OpenHands	68.8	47.5	-21.3
GPT-5-mini	OpenHands	62.6	34.3	-28.3
Qwen 3 Coder 30-A-3B	OpenHands	51.6	23.2	-28.4
Devstral 2 Small (24B)	OpenHands	54.8	26.8	-28.0

Table 7: For each model, we compare performance of coding agents in the fully-autonomous (Full Specification) setting of SWE-Bench Verified Chowdhury et al. (2024) with performance in our dialogue-driven setting (Dialogue-SWEBench) using OpenHands Wang et al. (2024). Our evaluation is designed to be strictly harder than SWE-Bench Verified: our user simulator is only given knowledge which is directly available to the agent in the fully-autonomous setting. Correspondingly, we observe substantial performance drops across coding agents.

In Table 7, we provide a performance comparison between the standard, fully-autonomous SWE setting as exemplified by SWE-Bench Verified (Chowdhury et al., 2024), and our dialogue-driven transformation. Each model uses the OpenHands agent (Wang et al., 2024). By design, our benchmark does not seed the user simulator with any knowledge that is not already given directly to an agent in the fully-autonomous evaluation setting. Correspondingly, we observe significant performance drop due to the added dialogue challenge in our benchmark.

D Analyzing Results by SWE Difficulty

Model	Agent	<15m (n=261)		15m–1h (n=194)		1h+ (n=45)
GPT-5 mini	OH Interactive	**66.5**		52.5		11.1
GPT-5		(-4.6) 61.9		(+2.7) **55.2**		(+8.9) **20.0**
GPT-5 mini	Schema-guided (Ours)	**70.6**		56.7		20.0
GPT-5		(-3.6) 67.0		(+0.4) **57.1**		(+4.4) **24.4**

Table 8: GPT-5-mini vs GPT-5 results stratified by SWE difficulty, as annotated by Chowdhury et al. (2024). While gpt-5-mini is competitive with GPT-5 overall, GPT-5 still outperforms on more challenging SWE problems. However, GPT-5 under-performs on easier SWE problems.

In Table 8, we stratify our results for GPT-5 and GPT-5-mini from Table 1 by engineering difficulty, as annotated by Chowdhury et al. (2024). We first note that our schema-guided agent improves performance for models across difficulty settings. Second, we can see that while GPT-5-mini matches performance of the larger GPT-5 on the overall benchmark, the larger model still out-performed the smaller on the more difficult engineering problems, while performing worse on the easier ones. In § 9, we explore one of these surprising failures for GPT-5 on easy problems further in a case study.

E Case Study Details

In this section, we present additional case studies. Figure 9 and Figure 10 detail the case studies introduced in §9.

Solvable Single-Turn Dialogues Occasionally, a user's initial query is sufficiently clear for the agent to resolve the user's problem without seeking clarification, when considering repository context. Figure 11 provides an example. In this example, the user wants to add support for the DISTINCT keyword for two aggregate functions, Avg and Sum. Rather than seek clarification from the user, the agent is able to infer the correct solution from the implementation for a related aggregate function (Count).



Figure 9: A case study between OH Interactive (left, ✗) and our Schema-Guided agent (right, ✓). The baseline OH Interactive agent fails to discuss the **expected behavior** when the caller's argument is not of type `set`, choosing to assume it is without checking. Our schema-guided agent first reasons about the expected behavior for this case explicitly, then proposes a solution to the user: coercing iterables to sets. This elicits the precise expected behavior when the user proposes an alternative: asserting that the caller's argument is of type `set`.

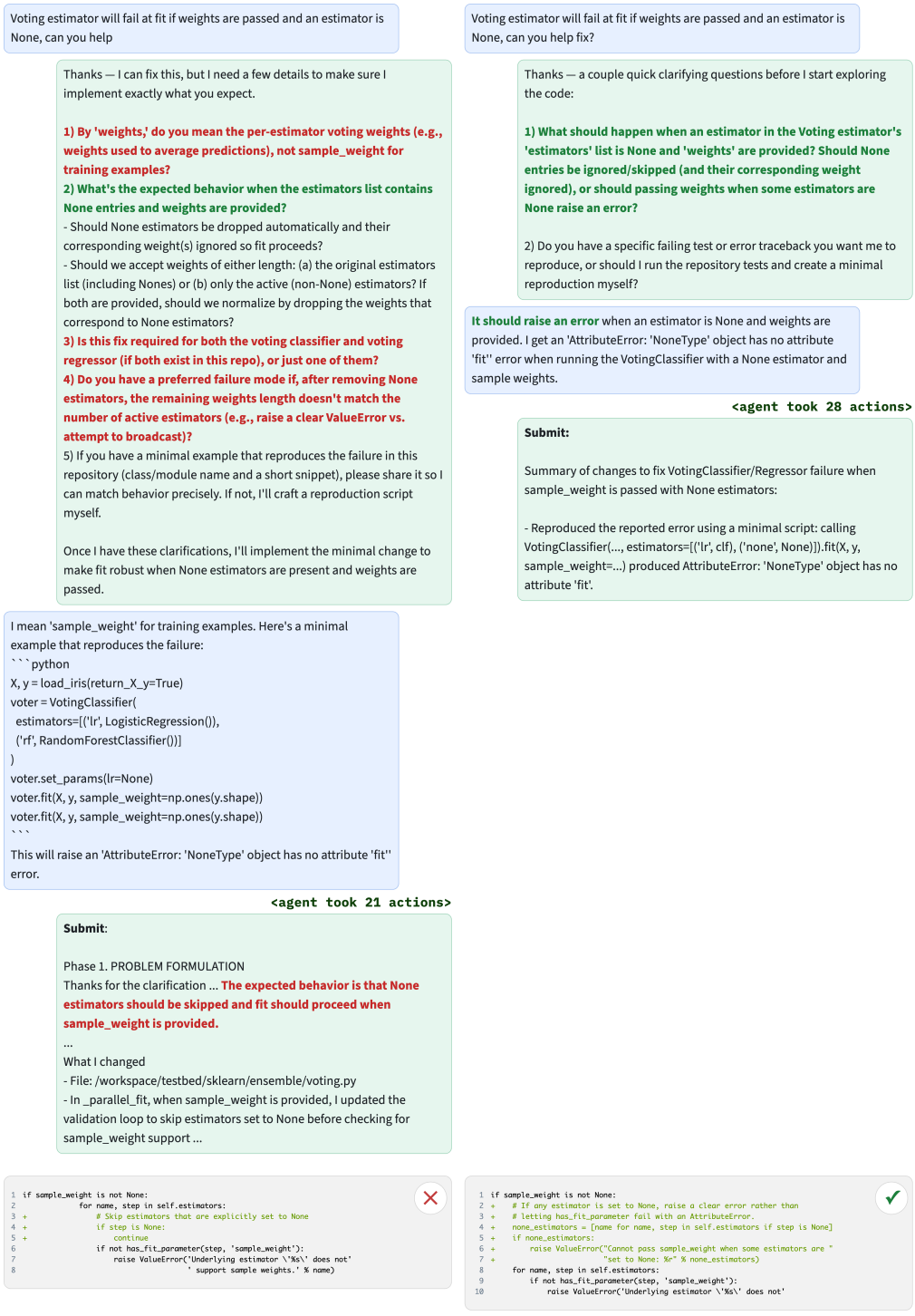


Figure 10: A case study between GPT-5 (left, ✗) and GPT-5-mini (right, ✓), both using our schema-guided agent. The solution to the user's problem is a minor change: **raise an error** if any estimators are None (right). GPT-5 mini asks only two questions, eliciting the critical detail: correct expected behavior for None estimators. GPT-5 asks *many* questions, some of which are unnecessary. The user gives an answer to questions 1 and 5, but the agent does not follow up on those the user chose not to address in their response. As a result, GPT-5 makes **incorrect assumptions about expected behavior** and applies the wrong fix.

Can we add support for DISTINCT in Avg and Sum aggregates?

<agent took 37 actions>

I have successfully implemented support for DISTINCT in Avg and Sum aggregates as requested.

Summary of Changes

Problem: The `Avg` and `Sum` aggregate functions did not support the `distinct=True` parameter, unlike `Count` which already had this functionality.

Root Cause: In `/workspace/testbed/django/db/models/aggregates.py`, the base `Aggregate` class had `allow_distinct = False`, and only the `Count` class explicitly set `allow_distinct = True`. The `Avg` and `Sum` classes inherited this default `False` value.

Solution: Made minimal changes to add `allow_distinct = True` to both `Avg` and `Sum` classes:

1. **Line 102**: Added `allow_distinct = True` to the `Avg` class
2. **Line 146**: Added `allow_distinct = True` to the `Sum` class

Verification

✅ **Functionality**: `Avg('field', distinct=True)` and `Sum('field', distinct=True)` now work correctly
✅ **Backward Compatibility**: All existing usage patterns continue to work unchanged
✅ **Consistency**: All three aggregate types (`Avg`, `Sum`, `Count`) now support `distinct` parameter uniformly
✅ **Tests**: All existing aggregation tests (59 tests) and regression tests (63 tests) continue to pass
✅ **No Regressions**: No existing functionality was broken

The implementation follows the exact same pattern as the existing `Count` class and maintains full backward compatibility while adding the requested DISTINCT support for Avg and Sum aggregates.

```
diff --git a/django/db/models/aggregates.py b/django/db/models/aggregates.py
index da4ff928aa..8b10829eb8 100644
--- a/django/db/models/aggregates.py
+++ b/django/db/models/aggregates.py
@@ -99,6 +99,7 @@ class Aggregate(Func):
 class Avg(FixDurationInputMixin, NumericOutputFieldMixin, Aggregate):
     function = 'AVG'
     name = 'Avg'
+    allow_distinct = True


 class Count(Aggregate):
@@ -142,6 +143,7 @@ class StdDev(NumericOutputFieldMixin, Aggregate):
 class Sum(FixDurationInputMixin, Aggregate):
     function = 'SUM'
     name = 'Sum'
+    allow_distinct = True


 class Variance(NumericOutputFieldMixin, Aggregate):
```

Figure 11: An example dialogue in which the agent (Schema-Guided Qwen 3 Coder) is able to resolve the user's initial query without any clarification questions, using only one dialogue turn to report its findings and solution. In this case, the user's initial intent can be clearly defined in only a few words, with the remaining details inferrable from the repository context.

Model	Agent	Prompt Tokens	Comp. Tokens	Prompt Price ($ / 1M)	Comp. Price ($ / 1M)	Cost ($)
GPT-5 mini	OpenHands	2.59M	57.4K	0.25	2.00	0.26
GPT-5	OpenHands	2.78M	64.4K	1.25	10.00	1.13
Devstral 2 Small (24B)	OpenHands	3.48M	17.7K	0.10	0.30	0.35
Qwen3-Coder 30B-A3B	OpenHands	2.02M	14.3K	0.09	0.32	0.18
GPT-5	Schema-Guided (Ours)	1.49M	52.5K	1.25	10.00	0.86
GPT-5 mini	Schema-Guided (Ours)	1.81M	55.7K	0.25	2.00	0.24
Devstral 2 Small (24B)	Schema-Guided (Ours)	2.91M	16.3K	0.10	0.30	0.30
Qwen3-Coder 30B-A3B	Schema-Guided (Ours)	1.51M	11.8K	0.09	0.32	0.13
GPT-5 mini	OH Interactive	1.87M	45.3K	0.25	2.00	0.23
GPT-5	OH Interactive	1.77M	58.8K	1.25	10.00	0.97
Devstral 2 Small (24B)	OH Interactive	2.93M	16.2K	0.10	0.30	0.30
Qwen3-Coder 30B-A3B	OH Interactive	1.31M	11.7K	0.09	0.32	0.12

Table 9: Details on average costs, prompt and completion token usage per problem. For GPT-5/GPT-5 mini, costs reflect average measured per-run, which may reflect savings due to prompt caching. For open-weight models, estimates are based on pricing from available API providers and measured token usage.

F Run Cost Estimation

For models run from API providers, we report an average of the actual cost of each run as calculated through API usage with `litellm` . For open-weight models, we run original experiments on our own hardware, and report cost estimates based on token usage and pricing quoted from an appropriate provider. Table 9 lists the token usage across runs, API pricing, estimated or reported average cost, and the API provider. The cost of user simulator inference is excluded from all runs.

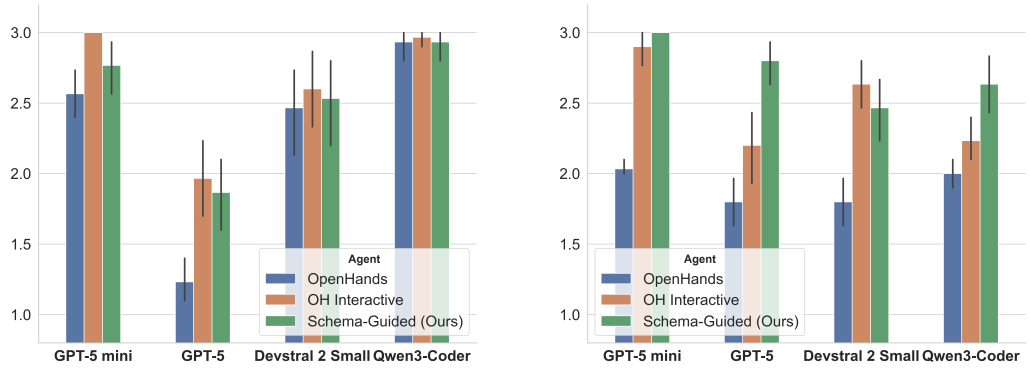
G Naturalness & Coherence Human Ratings

Figure 12a and Figure 12b respectively present the results of our human annotation for the naturalness and coherence on 360 dialogues, as described in §7. Figure 12c and Figure 12d present the LLM-as-a-judge results for the same *human-annotated subset*. While human annotators and our LLM-as-a-judge rank system naturalness identically, there are a few differences in ranking for system coherence. Notably, human coherence ratings rate schema-guided agent with GPT-5-mini highest, where the LLM-as-a-judge ranks it equal to or below other agents.

H Artifacts Used

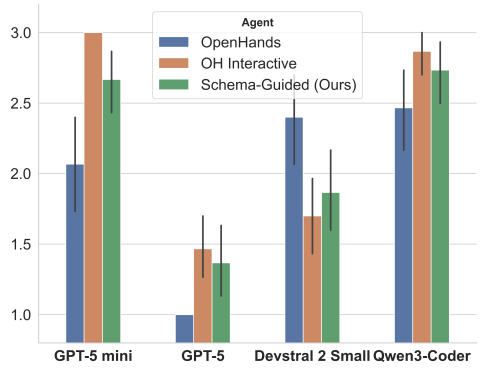
Our work utilizes the following artifacts:

1. SWE-Bench (Jimenez et al., 2024): dataset which distributed on Github with the MIT License.
2. Qwen 3 Coder (Team, 2025): distributed with the Apache 2.0 license, available on Huggingface.
3. Devstral 2 Small (MistralAI, 2025): distributed with the Apache 2.0 license, available on Huggingface.
4. GPT 5 & GPT-5 Mini (OpenAI, 2025): proprietary and hosted via API by OpenAI. Our usage respects the OpenAI terms of service.

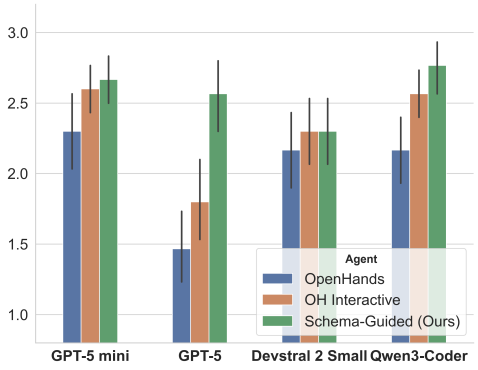


(a) Dialogue naturalness (human annotations)

(b) Dialogue coherence (human annotations)



(c) Dialogue naturalness (LLM annotations)



(d) Dialogue coherence (LLM annotations)

Figure 12: Human and LLM ratings for naturalness and coherence (1–3) with 95% confidence intervals for each system, evaluated on the human-annotated subset of the data (30 dialogues per agent-model pair)