Interactive Evaluation of Large Language Models for Multi-Requirement Software Engineering Tasks

Dimitrios Rontogiannis,[1*] Maxime Peyrard,[2] Nicolas Baldwin,[3] Martin Josifoski,[3†] Robert West,[4] Dimitrios Gunopulos[5]

[1]Max Planck Institute for Software Systems
[2]Université Grenoble Alpes, CNRS, Grenoble INP, LIG
[3]FAIR at Meta
[4]EPFL
[5]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens
drontogi@mpi-sws.org, maxime.peyrard@univ-grenoble-alpes.fr, {nicolas.baldwin,robert.west}@epfl.ch,
martinjosifoski@meta.com, dg@di.uoa.gr

Abstract

Standard single-turn, static benchmarks fall short in evaluating the nuanced capabilities of Large Language Models (LLMs) on complex tasks such as software engineering. In this work, we propose a novel interactive evaluation framework that assesses LLMs on multi-requirement programming tasks through structured, feedback-driven dialogue. Each task is modeled as a requirement dependency graph, and an "interviewer" LLM, aware of the ground-truth solution, provides minimal, targeted hints to an "interviewee" model to help correct errors and fulfill target constraints. This dynamic protocol enables fine-grained diagnostic insights into model behavior, uncovering strengths and systematic weaknesses that static benchmarks fail to measure. We build on DevAI, a benchmark of 55 curated programming tasks, by adding ground-truth solutions and evaluating the relevance and utility of interviewer hints through expert annotation. Our results highlight the importance of dynamic evaluation in advancing the development of collaborative code-generating agents.

Code + Datasets — https://bit.ly/43Z6UfG
Extended version — https://arxiv.org/pdf/2508.18905

Introduction

The integration of Large Language Models (LLMs) into software development has transformed coding from a solitary, linear process into a dynamic, iterative collaboration. Modern tools like ChatGPT (Schulman et al. 2022), DeepSeek (DeepSeek-AI 2025), and AI-first IDEs such as Cursor exemplify this shift, where developers no longer simply request code but refine it through multiturn dialogues. Feedback, whether clarifications, corrections, or incremental constraints, has become the scaffold for progress, allowing models to adapt to ambiguities, edge cases, and evolving requirements. Yet, despite this reality, the prevailing benchmarks continue to evaluate LLMs as static single-turn code generators, ignoring the very interactions that define their practical utility.

[*]Work done while at EPFL and continued at NKUA
[†]Work done while at EPFL

The Gap in Current Evaluation Current evaluation paradigms for software engineering problems suffer from two critical misalignments with real-world software workflows. (i) First, they treat tasks as monolithic problems (Chen et al. 2021; Hendrycks et al. 2021), ignoring their compositional nature. For example, building a recommendation system requires strict dependencies: data loading → feature engineering → model training → API exposure. Yet static evaluations force models to "guess correctly" on the first attempt, conflating understanding of requirements with luck in initial output. This penalizes models for early errors (e.g., data loading) and obscures their ability to recover in downstream steps (e.g., model training), even though real development often involves debugging partial solutions. (ii) Second, while recent work explores interactive evaluation (Wang et al. 2023; Pan et al. 2025), these efforts rely on shallow feedback (e.g., binary correctness checks) or unstructured hints, failing to capture the directed repair behavior of human-AI collaboration. In practice, a model's value depends on its ability to adapt, say fixing a missing edge case after a developer's nudge, but benchmarks rarely measure this. The gap is systemic: without evaluating how LLMs leverage feedback to navigate dependencies or rectify cascading errors, we risk overestimating failures (where models could recover) or underestimating pitfalls (where models pass single-turn tests but reveal critical flaws when exposed to step-by-step refinement) - precisely the dynamics that define their practical utility.

Our framework: Interactive, Dependency-Grounded Assessment We propose a structured, feedback-driven evaluation framework (Figure 1) for software engineering tasks. Each task is decomposed into a directed acyclic graph (DAG) of requirements, capturing the hierarchical dependencies between subtasks. A model is evaluated not only on its initial output but also on its ability to improve iteratively through targeted feedback loops. These hints are automatically generated by an LLM-based interviewer with access to ground truth solutions and task requirements. If a model fails an early subtask, we guide it past the error to assess its performance on subsequent steps - mirroring how human de-

velopers work around intermediate bugs to evaluate deeper functionality.

A critical design feature is our lightweight integration protocol. The framework exposes simple interfaces that allow any LLM to participate as either interviewer or interviewee with minimal adaptation. Researchers can evaluate new models by implementing just these basic interaction primitives, while still benefiting from the full power of our dependency-aware assessment pipeline. This modular design ensures wide applicability without compromising the richness of evaluation, enabling both controlled benchmarking and real-world deployment testing.

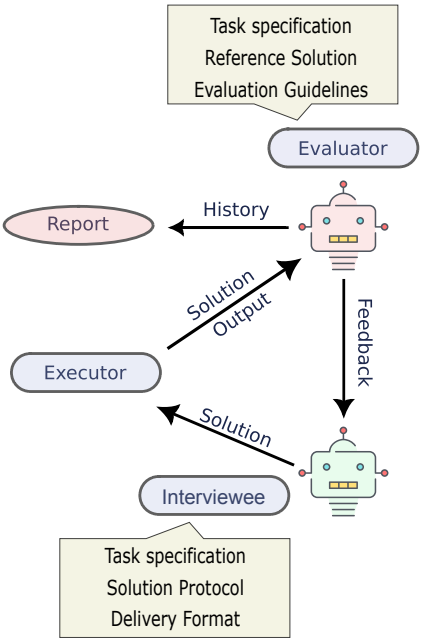

Figure 1: The Interactive Evaluation pipeline.

Contributions Our main contributions are:

1. A Dependency-Driven Interactive Evaluation Protocol: We introduce the first framework that jointly models software task decomposition and iterative feedback for LLM assessment. The framework's novel structure enables quantifying error propagation and recovery through guided feedback and easy integration via minimal interface requirements, allowing any LLM to participate as interviewer or interviewee with trivial adaptation.
2. An enhanced DevAI benchmark: We augment DevAI (Zhuge et al. 2024) with verified Ground-Truth solutions, using the original Agent-as-a-Judge methodology to ensure correctness. This extension enables guided, multi-stage evaluation through our framework's structured feedback mechanism, resulting in an improved benchmark that serves as both (i) an evaluation platform for our experiments and (ii) a reusable resource for future interactive assessment frameworks.
3. Our experiments reveal that failures in static evaluations become recoverable with targeted feedback, suggesting that single-turn benchmarks severely underestimate LLM capabilities. We also identify critical failure modes where models cannot effectively incorporate feedback, revealing limitations in their ability to refine solutions even with iterative guidance.

By bridging the gap between static benchmarks and real-world software workflows, our work advances practical LLM evaluation for software engineering problems.

Related Work

Traditional evaluations of LLMs rely on static benchmarks with fixed inputs and binary success criteria. While benchmarks such as HumanEval (Chen et al. 2021), APPS (Hendrycks et al. 2021), and MBPP (Austin et al. 2021) have driven rapid progress in code generation, they fail to capture the process-oriented, iterative nature of real-world problem solving. These benchmarks typically assess models based on functional correctness of output in a single shot setting, which assumes complete and unambiguous task specifications, an assumption that does not hold in most practical development scenarios. Extensions like CodeXGLUE (Lu et al. 2021) and SWE-bench (Jimenez et al. 2024) move towards more realistic evaluation tasks, such as bug fixing and issue resolution in real codebases. However, they still emphasize static correctness over dynamic reasoning, offering limited insight into how models handle ambiguity, adapt over time, or respond to developer intent.

To address these shortcomings, recent work has explored *interactive evaluation*, where models are assessed over multiple turns with access to feedback or clarification. Human-in-the-loop setups such as iEval (Svikhnushina, Filippova, and Pu 2022) and CheckMate (Collins et al. 2023) demonstrate that interactivity reveals model competencies overlooked by static scoring, particularly in complex domains like mathematics or natural language understanding. These studies show that model performance can vary substantially when they are allowed to ask questions, request hints, or revise outputs based on critique. More scalable frameworks simulate interaction using LLMs both as agents and evaluators, as in IQA-Eval (Li et al. 2024), KIEval (Yu et al. 2024), and medical roleplay systems (Liao et al. 2024), enabling broader experimentation without relying on human annotators. These frameworks highlight the potential of structured feedback to surface model behaviors that are otherwise invisible in single-turn evaluations.

Complementary to interactivity, *adaptive evaluation* dynamically adjusts testing based on model responses. DyVal (Zhu et al. 2024) and DyVal 2 extend this idea using reasoning graphs and skill-specific probes to isolate weaknesses and trace error propagation through multistep reasoning tasks. These tools allow for a more diagnostic view of model performance, showing not just whether a model fails, but how and why it fails across different cognitive skills. Similarly, AdaTest (Ribeiro and Lundberg 2022) and benchmark self-evolving frameworks (Wang et al. 2024) generate targeted adversarial examples to stress-test models under varying conditions. By continuously updating the test set in response to model behavior, these approaches create a mov-

ing target that reveals brittleness or blind spots that static benchmarks overlook.

In the software engineering domain, a few agent-based approaches have emerged to better reflect realistic development pipelines. Notably, Agent-as-a-Judge (Zhuge et al. 2024) evaluates LLMs on tasks involving interdependent components such as planning, execution, and evaluation. These methods begin to address compositionality and dependency tracking, yet often lack structured mechanisms for feedback-based refinement. They typically evaluate outputs at isolated checkpoints, without modeling how developer guidance might help correct or improve the model's trajectory through a task.

Our work builds on this trajectory by introducing an interactive evaluation framework tailored to software engineering tasks. Unlike prior efforts that isolate interactivity, adaptivity, or software-specific evaluation, our approach unifies these aspects through requirement decomposition, dependency-aware scoring, and guided iterative feedback. This allows for a more granular and realistic assessment of how models reason, adapt, and improve in complex engineering workflows, capturing the collaborative dynamics that characterize human-AI co-programming.

Problem Formulation

We define **Interactive Software Engineering Evaluation** as a multi-stage assessment framework designed to evaluate large language models through iterative refinement cycles guided by structured feedback. This approach specifically addresses complex, decomposable tasks characterized by three key properties: first, the presence of hierarchical dependencies among requirements; second, the potential need for incremental correction of partial solutions; and third, the necessity to evaluate both initial capability and adaptive improvement. The framework represents tasks as Directed Acyclic Graphs (DAGs) of requirements, where vertices correspond to verifiable subtasks, and edges encode functional dependencies.

Unlike traditional binary evaluations, which assess success or failure on a task as a whole, interactive evaluation captures both the model's initial performance and its ability to refine and repair its output in response to minimal guidance. This approach aligns closely with real-world software engineering, where developers iteratively build and correct solutions in response to feedback.

Structured Tasks with Hierarchical Dependencies We focus on problems that consist of multiple interdependent requirements, where progress on earlier subtasks enables progress on later ones. Formally, let a task T be defined by a set of requirements $R = \{r_1, r_2, \ldots, r_m\}$, with each r_j representing a subcomponent or constraint of the overall solution.

To capture the hierarchical and sequential structure of such tasks, we can model their dependencies as a DAG $G = (R, E)$, where an edge $(r_i, r_j) \in E$ indicates that requirement r_j can only be addressed after requirement r_i has been successfully completed. Let $P(r_j)$ denote the set of prerequisite requirements for r_j, and let $g\colon R \to \{0, 1\}$ be the initial evaluation function that checks whether each requirement $r \in R$ is satisfied (1) or not (0). Then, r_j is evaluable only if all its parents are satisfied: $\forall r_i \in P(r_j), g(r_i) = 1$.

Report Example

1. **Error Handling in Image Downloading**: The initial implementation of the `download_image` function did not adequately handle connection errors...
2. **URL Accessibility**: The model initially used a URL for the style image that resulted in a 404 error...
3. **Logging and Feedback**: The model's initial logging for download attempts was minimal...
4. **Code Organization**: While the code was well-structured...
5. **Adaptability**: The model demonstrated good adaptability...

Overall, the hints provided were instrumental...

Figure 2: Example evaluation report

The effective (dependency-aware) evaluation score for the task is then defined as:

$$S_G = \frac{1}{m} \sum_{j=1}^{m} g(r_j) \cdot \mathbb{I}\left[\forall r_i \in P(r_j),\ g(r_i) = 1\right] \quad (1)$$

This formulation allows us to credit partial progress while respecting the task's logical structure, avoiding overly coarse binary evaluations.

Guided Evaluation via Feedback Crucially, we are interested not just in how well a model performs on its first attempt, but in how effectively it improves when given feedback. In software engineering, a developer might suggest minimal edits ("rename this variable", "fix the off-by-one error"), guiding progress without solving the problem outright. We aim to replicate this process in evaluation.

Let $R_{\text{fail}} \subseteq R$ be the set of requirements the model initially fails, and let $H = \{h_1, h_2, \ldots, h_k\}$ be a minimal set of corrective hints provided by the evaluator. These hints serve as feedback for revision. The updated evaluation function $g'_H(r_j)$ checks if the revised response meets r_j given H.

We define the final interactive evaluation score as:

$$S'_G = \frac{1}{m} \sum_{j=1}^{m} g'_H(r_j) \cdot \mathbb{I}\left[\forall r_i \in P(r_j),\ g'_H(r_i) = 1\right] \quad (2)$$

By comparing S_G and S'_G, we gain insight into a model's capacity not just for initial accuracy, but for refinement—an essential skill in real-world applications. This interactive framework enables more efficient exploration of the solution space through feedback, aligning model evaluation with realistic software development workflows.

Post-Evaluation Report Following the interactive evaluation process, we generate a structured performance report to analyze the model's strengths, weaknesses, and adaptability. Rather than providing a single aggregate score, this report captures multiple dimensions of the model's behavior. It assesses problem-solving ability (e.g., whether the model can comprehend complex tasks and produce structured solutions), optimization awareness (e.g., consideration of time or space complexity), and, where applicable, code quality and organization. It also examines the model's ability to recognize and correct its own mistakes, its responsiveness to minimal feedback, and its handling of ambiguity or incomplete information.

Methodology

We introduce a structured methodology to evaluate LLMs on complex, structured software engineering tasks. The process consists of three stages: requirement extraction and initial evaluation, interactive refinement through feedback, and post-evaluation analysis.

Ground Truth Construction, Requirement Extraction and Initial Evaluation

Given a task T, we construct a ground-truth solution S^* and define a set of core requirements $R = \{r_1, r_2, \ldots, r_m\}$, representing essential aspects a correct solution must satisfy. These are structured as a DAG $G = (R, E)$, where an edge $(r_i, r_j) \in E$ indicates that r_j depends on the prior satisfaction of r_i. We demonstrate our framework using the DevAI benchmark, which provides structured requirements but lacks Ground-Truth solutions.

To evaluate a model-generated solution S, we segment it into chunks $C = \{c_1, c_2, \ldots, c_n\}$ and embed both requirements and chunks using a sentence encoder f_{enc}. For each requirement r_j, we retrieve the most similar chunk c_k^* via cosine similarity. The pair (r_j, c_k^*) is then passed to an LLM-based classifier, which predicts whether the requirement is satisfied, conditioned on the satisfaction of its parent requirements in the DAG.

This initial evaluation procedure follows the *Agent as a Judge* approach (Zhuge et al. 2024), and we adopt their judge implementation in our experiments.

Interactive Evaluation

To measure a model's ability to improve its solution with guidance, we introduce an iterative evaluation loop. At each iteration t, the model submits a revised solution $S^{(t)}$, which is executed in a sandboxed Python environment to produce outputs $O^{(t)}$ and errors $E^{(t)}$ if any.

A separate LLM-based component, the *interviewer* $\mathcal{I}$, analyzes the current output, execution errors, the evaluation graph G, and the ground-truth solution S^*. Based on this, it generates a minimal set of natural language hints $H^{(t)}$ intended to help the model correct its current deficiencies. These hints target specific failed requirements while preserving as much of the model's original reasoning as possible. A concrete example of such hint can be seen in Figure 3, demonstrating how they guide iterative improvement without overcorrecting. Additional examples showing hint variation across different failure modes are provided in Appendix C.

Interviewee: o3-mini, Problem: S26

Your solution currently does not explicitly load the Electronics subset of the Amazon Reviews 2023 dataset using the `datasets` *library as required. Instead, it reads from a local CSV or uses a dummy dataset fallback. To meet the requirement, please implement data loading in* `src/data_loader.py` *using the* `load_dataset` *function from the* `datasets` *library with the* `"McAuley-Lab/Amazon-Reviews-2023"` *dataset and* `"raw_review_Electronics"` *configuration, as shown in the reference solution. Also, please add explicit inline comments referencing the requirement for data loading and preprocessing steps.*

Figure 3: Hint provided by the interviewer

The evaluated model receives $H^{(t)}$ as input and produces an updated response $S^{(t+1)}$. This loop continues until either all requirements are satisfied according to $\mathcal{I}$, or a predefined maximum number of iterations is reached.

At the end of the process, we compute the final interactive score using Equation (2).

Post-Evaluation Reporting

Beyond correctness scores, we produce a qualitative report analyzing the model's behavior throughout the evaluation trajectory $\{S^{(t)}, H^{(t)}\}_{t=1}^{T}$. This report is generated by an LLM-based analyzer $\mathcal{R}$, which synthesizes insights about the model's reasoning process, adaptability, and robustness.

The analysis covers multiple dimensions, including problem-solving ability, sensitivity to feedback, optimization awareness (e.g., runtime or memory considerations), handling of ambiguity, and quality of code structure and organization. Rather than summarizing with a single metric, this report provides a structured breakdown of the model's strengths and failure patterns, offering deeper insight into its underlying capabilities. The example in Figure 2 shows a typical report. Additional reports showcasing varied response patterns across different model architectures and task categories are available in Appendix B.

An overview of this multi-phase evaluation process is illustrated in Figure 1, showing how model responses evolve through feedback and refinement.

Experimental Setup: Benchmark and Models

For the Interactive Evaluation experiments, we utilize problems sourced from the **DevAI** benchmark, running all computations on an Apple M2 Pro system (12-core CPU with 8 performance/4 efficiency cores, 19-core GPU with Metal 3 acceleration, and 16GB unified memory). These software engineering problems span several machine learning and data science domains, including classification, natural language processing, and recommender systems. Among its

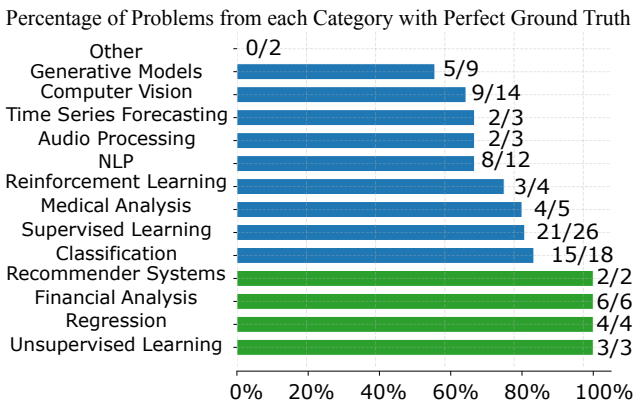

Figure 4: Percentage or problems with perfect ground truth accuracy by category.

several components, Figure 4 presents the categorical distribution of problems in DevAI. Each problem in Devai is not merely a question with a binary correct/incorrect outcome, but rather a structured task, decomposed into multiple requirements.

As illustrated in Figure 5, every problem is accompanied by: 1) The main question statement, which describes the problem to be solved. 2) A set of requirements, representing the individual steps, constraints, or conditions that a correct solution must satisfy. 3) A dependency graph, capturing the logical dependencies between requirements. Certain requirements can only be evaluated if prerequisite requirements have already been satisfied.

For the granular evaluation process, which assesses the quality of a solution produced by a candidate model, we employ OpenAI's `gpt-4o-mini`. For each requirement, the model provides a binary judgment (satisfied or unsatisfied), along with a natural language explanation justifying its decision.

Since DevAI does not provide official Ground-Truth solutions for its tasks, we constructed reference solutions. To ensure their reliability, we verified that each solution satisfied all predefined requirements using our granular evaluation framework prior to inclusion in experiments. As shown in Figure 6, most tasks achieve 100% requirement satisfaction. In a few cases, lower satisfaction scores occur due to two main factors: (1) some tasks rely on external datasets that are no longer publicly available, and (2) the LLM judge occasionally misclassifies correct outputs as unsatisfied due to limitations in understanding. Notably, this issue persists even with more capable judge models. However, even in cases with lower percentages, the absolute number of unsatisfied requirements is often small, usually a single missed requirement in tasks with few total requirements.

Our interactive evaluation experiments employ `gpt-4.1-mini` and `gpt-4o-mini` in separate evaluation runs, with each model serving independently as evaluator. The evaluator provides iterative feedback by analyzing the ground truth solution, the set of predefined requirements, the solution produced by the evaluated model

Example task: Tweets sentiment analysis on Sentiment140 dataset from HF

Query

Build a sentiment analysis system using the Sentiment140 dataset from Hugging Face. Load and clean the data (remove stop words, punctuation, special characters) in `src/data_loader.py`. *Use Word2Vec or GloVe for vectorization in the same file. Train an SVM classifier in* `src/model.py` *and save the accuracy in* `results/metrics/accuracy_score.txt`.

Requirements

- **R0** *Sentiment140 is loaded in* `src/data_loader.py`.
  Dependencies →{}
- **R1** *Dataset is cleaned (stop words, punctuation, special characters) in* `src/data_loader.py`.
  Dependencies →{**R0**}
- **R2** *Word2Vec or GloVe embeddings applied in* `src/data_loader.py`.
  Dependencies →{**R0, R1**}
- **R3** *SVM model trained in* `src/model.py`.
  Dependencies →{**R0, R1, R2**}
- **R4** *Accuracy written to* `results/metrics/accuracy_score.txt`.
  Dependencies →{**R1, R2, R3**}

Figure 5: A task example in DevAI.

(interviewee), as well as any errors encountered during execution in a Python interpreter.

For the interactive interviewer evaluator model, we set the temperature parameter to 0.3 to encourage responses that balance determinism and creativity, while ensuring a degree of consistency across repeated evaluations. The interviewee model uses the same temperature setting (0.3) for comparable behavior in solution generation. We configure the maximum token limit to 2000 tokens for the interviewer, allowing it to handle detailed feedback within each evaluation step, while permitting 5000 tokens for the interviewee to accommodate longer solutions to complex problems.

To ensure consistent behavior during interactive evaluation, we design a set of role-specific prompts for both the evaluator and the interviewee model. The evaluator is guided by a system prompt that defines its objectives and communication style, as well as an assistant prompt that specifies evaluation criteria, feedback strategies, and hinting procedures. The interviewee model receives a system prompt outlining its role, expected behavior, and response format, along with a detailed user prompt that directs its problem-solving approach and ensures adherence to task requirements. All prompts are provided in Appendix A.

Evaluation

We begin by rigorously evaluating the quality of our enhanced DevAI benchmark through multiple complementary analyses. First, we examine the requirement satisfaction rates of our curated Ground-Truth solutions. Figure 6 reveals that 92.6% of all requirements are satisfied on average across

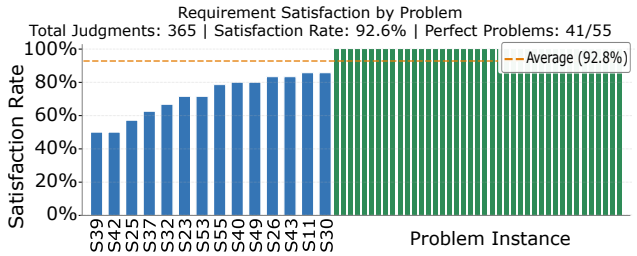

Figure 6: Requirement satisfaction rates of ground truths across all problems. Green bars indicate fully satisfied problems (100%). The orange dashed line shows the average requirement satisfaction (92.6%).

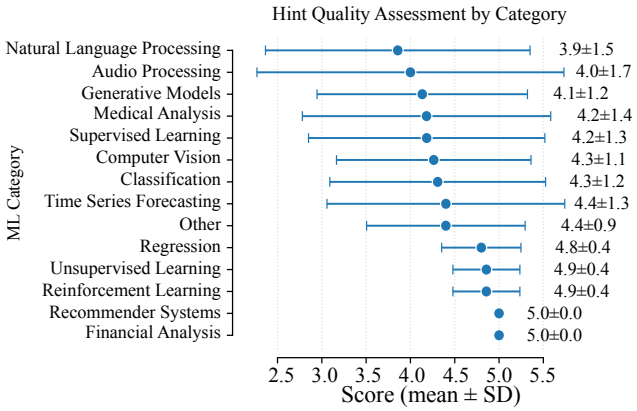

Figure 7: Hint quality scores across problem categories (mean = 4.32, σ = 1.18). These hints were produced by GPT-4.1-mini.

the benchmark, with a strong majority of problems achieving perfect 100% compliance (shown in green). This high overall quality ensures that the interviewer model generates hints based on fundamentally sound reference implementations, establishing a solid foundation for reliable interactive evaluation.

Deeper category-level analysis in Figure 4 exposes important variations in solution quality across different software engineering domains. While well-structured tasks maintain near-perfect ground truth rates, more complex domains exhibit noticeable gaps. Specifically, generative models, computer vision, and NLP tasks demonstrate lower compliance rates. We attribute these differences to three key factors: (1) inherent ambiguity in problem specifications for creative tasks, (2) dependency on external data sources that may become unavailable, and (3) greater implementation variability in cutting-edge domains where best practices are still evolving.

To assess how these benchmark characteristics translate to actual interactive evaluation quality, we conducted a comprehensive user study with 100 expert-annotated hints sampled from real evaluation sessions (20 hints $\times$ 5 interviewee models) using GPT-4.1-mini as the interviewer. The results in Figure 7 demonstrate that overall hint quality remains strong (μ=4.32/5, σ=1.18). We observe the predicted correlation between ground truth quality and hint effectiveness: where in categories with lower ground truth quality, we have slightly lower scores and hints showed greater variability. We replicated this study with GPT-4o-mini as the interviewer, with complete results and comparative analysis presented in Appendix C, revealing consistent patterns in hint effectiveness across both interviewer models.

To set baseline expectations, we begin by contextualizing model capabilities using OpenAI's published benchmark results (OpenAI 2025a,b,c). The GPT-4.1-mini outperforms the GPT-4o-mini in traditional coding and instruction evaluations, scoring 24% versus 9% on SWE-bench, 35% compared to 4% on Aider's Polyglot benchmark (Gauthier 2025), and 84% against 78% on IFEval (Zhou et al. 2023). Notably, the o3-mini surpasses both variants in standalone coding assessments with a 42.9% score on SWE-bench, while the o4-mini exhibits comprehensive superiority across all major benchmarks.

These static benchmark results present an intriguing paradox when contrasted with our interactive evaluation findings. In the context of complex, multi-requirement software engineering tasks requiring iterative feedback and refinement, in Figure 8, where GPT-4.1-mini acts as the interviwer, we observe that (interviewee) GPT-4.1-mini's performance degrades relative to GPT-4o-mini, with guided variants only matching GPT-4o-mini's baseline unguided performance. Furthermore, while o4-mini initially demonstrates suboptimal performance on certain problem categories, its capacity for instruction following becomes evident through the guidance process, ultimately surpassing all other model variants in final performance. This divergence suggests that GPT-4.1-mini exhibits limitations in processing and incorporating multi-turn feedback during refinement cycles, while o4-mini's robust instruction-following capabilities enable it to overcome initial implementation challenges (particularly those related to dataset and environment configuration, as detailed in Figure 10) and achieve superior final results.

Figure 9 shows that most models achieve only marginal gains when using hints from GPT-4o-mini, suggesting these hints provide limited value. Notably, GPT-4.1-mini exhibits a similar pattern – its performance deteriorates when relying on such hints. This decline stems from two compounding issues: the hints' inherent weaknesses and the model's inability to effectively utilize iterative refinements. Ultimately, this combination produces worse results than when the model generates solutions independently.

The observed behavior of GPT-3.5-turbo aligns with expectations given its architectural limitations. As a smaller model with constrained context window size, it struggles to: (1) retain and apply past refinements across iterations, (2) consistently produce complete solutions when task complexity exceeds its capacity, and (3) reliably follow instructions to regenerate full solutions – a weakness also documented in Aider's Polyglot benchmark. These limitations manifest consistently regardless of the interviewer model (GPT-4o-mini or GPT-4.1-mini), confirming fundamental

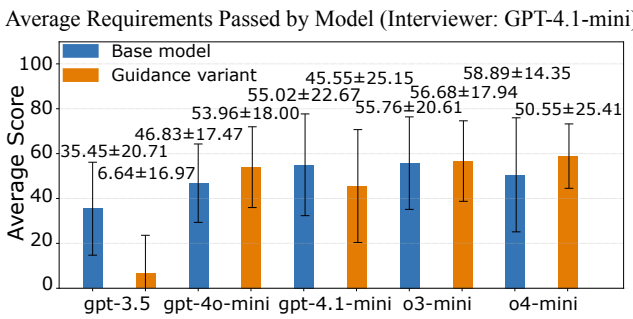

Figure 8: Average requirements passed by model variant using GPT-4.1.mini as interviewer. Blue bars represent base models, orange bars show guided variants.

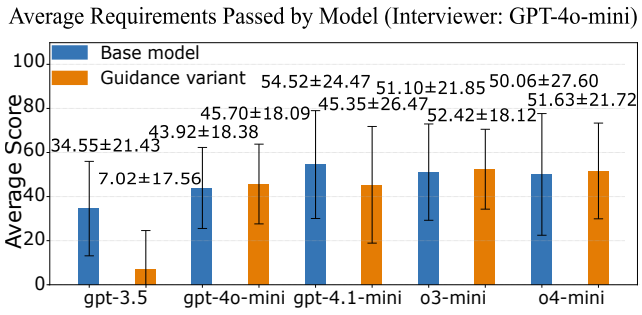

Figure 9: Average requirements passed by model variant using GPT-4o-mini as interviewer. Blue bars represent base models, orange bars show guided variants. Interviewer: GPT-4o-mini

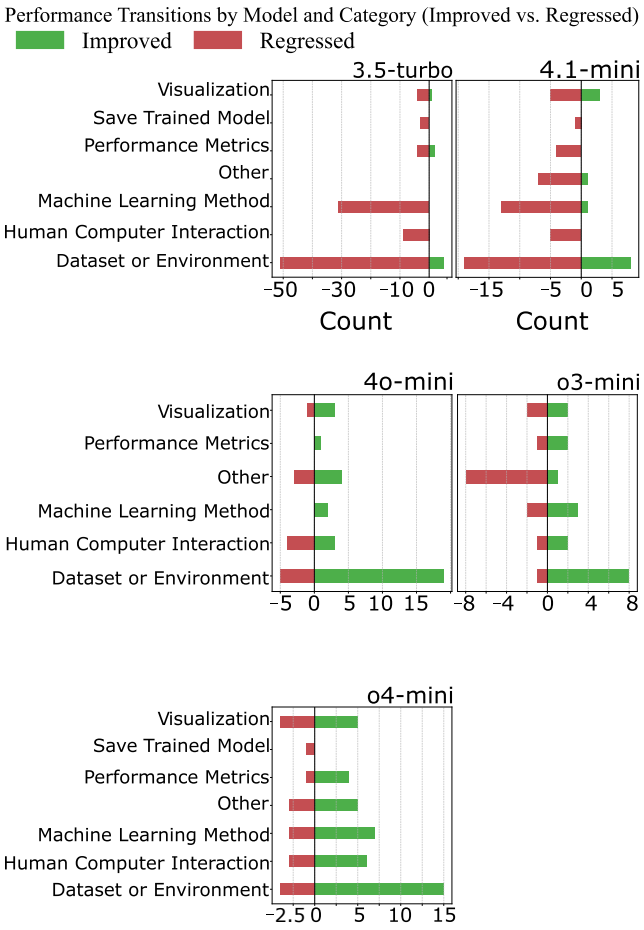

Figure 10: Breakdown of guidance impact across model variants per requirement category

capability constraints rather than interviewer-specific effects.

The transition plot (Figure 10) reveals distinct patterns in how models respond to hint interventions across task categories. All models exhibit their most pronounced performance improvements in the Dataset or Environment category, with consistent positive counts observed for every model variant. This trend likely stems from the inherent challenges posed by benchmark tasks requiring manipulation of recent or niche datasets, where models frequently lack sufficient pretraining exposure or precise location information. The availability of ground truth references in hints appears particularly effective for resolving such environment-specific ambiguities. Beyond this commonality, models demonstrate divergent response profiles to hinting. These differential responses suggest that hint efficacy depends both on the task domain and the specific model's capability profile, with no universal improvement pattern emerging across the evaluated categories.

Conclusion

Our work establishes a new paradigm for evaluating LLMs in software engineering tasks through three fundamental contributions. First, we demonstrate that dependency-aware interactive evaluation reveals capabilities and limitations obscured by static benchmarks - where models like GPT-4.1-mini show unexpected performance degradation when processing iterative feedback, while o4-mini leverages its superior instruction follow-up capacity to overcome initial implementation challenges. Second, we enhance with Ground-Truths and validate the DevAI benchmark, and we expose how error recovery in later stages (e.g., model training) often compensates for early failures (e.g., data loading) when guided by targeted hints. Third, our findings challenge the prevailing assumption that benchmark performance directly translates to interactive settings, as evidenced by the weak correlation between static scores and guided improvement rates across model variants.

Several limitations warrant attention. Requirement extraction can carry ambiguities from natural-language specifications, introducing noise into task decomposition. Automated feedback generation also raises fairness concerns: although our interviewer model provides consistent hints to all systems, their usefulness may differ across model architectures. Finally, the framework must balance guidance intensity: overly specific hints may reveal solutions, while vague ones may fail to prompt meaningful improvement.

Acknowledgments

Rontogiannis acknowledges support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101169607).

West's lab is partly supported by a grant from the Swiss National Science Foundation (TMSGI2_211379) and by generous gifts from Google and Microsoft. We also gratefully acknowledge compute support from the Microsoft "Accelerate Foundation Model Academic Research" program.

This work was supported by the European Union through the Horizon Europe AutoFair project (Grant No. 101070568) and the Horizon Europe CoDiet project (Grant No. 101084642). Additional support for CoDiet was provided by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee (Grant No. 101084642).

References

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; and Sutton, C. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Collins, K. M.; Jiang, A. Q.; Frieder, S.; Wong, L.; Zilka, M.; Bhatt, U.; Lukasiewicz, T.; Wu, Y.; Tenenbaum, J. B.; Hart, W.; Gowers, T.; Li, W.; Weller, A.; and Jamnik, M. 2023. Evaluating Language Models for Mathematics through Interactions. arXiv:2306.01694.

DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437.

Gauthier, P. 2025. Aider Chat Leaderboards Documentation. https://aider.chat/docs/leaderboards/. Accessed: 2025-08-1.

Hendrycks, D.; Basart, S.; Kadavath, S.; Mazeika, M.; Arora, A.; Guo, E.; Burns, C.; Puranik, S.; He, H.; Song, D.; and Steinhardt, J. 2021. Measuring Coding Challenge Competence With APPS. arXiv:2105.09938.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770.

Li, R.; Li, R.; Wang, B.; and Du, X. 2024. IQA-EVAL: Automatic Evaluation of Human-Model Interactive Question Answering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Liao, Y.; Meng, Y.; Wang, Y.; Liu, H.; Wang, Y.; and Wang, Y. 2024. Automatic Interactive Evaluation for Large Language Models with State Aware Patient Simulator. arXiv:2403.08495.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C.; Drain, D.; Jiang, D.; Tang, D.; Li, G.; Zhou, L.; Shou, L.; Zhou, L.; Tufano, M.; Gong, M.; Zhou, M.; Duan, N.; Sundaresan, N.; Deng, S. K.; Fu, S.; and Liu, S. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv:2102.04664.

OpenAI. 2025a. GPT-4.1 Preview. https://openai.com/index/gpt-4-1/. Accessed: 2025-08-1.

OpenAI. 2025b. Introducing O3 and O4 Mini. https://openai.com/index/introducing-o3-and-o4-mini/. Accessed: 2025-08-1.

OpenAI. 2025c. OpenAI O3 Mini. https://openai.com/index/openai-o3-mini/. Accessed: 2025-08-1.

Pan, J.; Shar, R.; Pfau, J.; Talwalkar, A.; He, H.; and Chen, V. 2025. When Benchmarks Talk: Re-Evaluating Code LLMs with Interactive Feedback. *arXiv preprint arXiv:2502.18413*.

Ribeiro, M. T.; and Lundberg, S. 2022. Adaptive Testing and Debugging of NLP Models. In Muresan, S.; Nakov, P.; and Villavicencio, A., eds., *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3253–3267. Dublin, Ireland: Association for Computational Linguistics.

Schulman, J.; Zoph, B.; Kim, C.; Hilton, J.; Menick, J.; Weng, J.; Uribe, J. F. C.; Fedus, L.; Metz, L.; Pokorny, M.; et al. 2022. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2(4).

Svikhnushina, E.; Filippova, A.; and Pu, P. 2022. iEval: Interactive Evaluation Framework for Open-Domain Empathetic Chatbots. In Lemon, O.; Hakkani-Tur, D.; Li, J. J.; Ashrafzadeh, A.; Garcia, D. H.; Alikhani, M.; Vandyke, D.; and Dušek, O., eds., *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 419–431. Edinburgh, UK: Association for Computational Linguistics.

Wang, S.; Long, Z.; Fan, Z.; Wei, Z.; and Huang, X. 2024. Benchmark Self-Evolving: A Multi-Agent Framework for Dynamic LLM Evaluation. arXiv:2402.11443.

Wang, X.; Wang, Z.; Liu, J.; Chen, Y.; Yuan, L.; Peng, H.; and Ji, H. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Yu, Z.; Gao, C.; Yao, W.; Wang, Y.; Ye, W.; Wang, J.; Xie, X.; Zhang, Y.; and Zhang, S. 2024. KIEval: A Knowledge-grounded Interactive Evaluation Framework for Large Language Models. arXiv:2402.15043.

Zhou, J.; Lu, T.; Mishra, S.; Brahma, S.; Basu, S.; Luan, Y.; Zhou, D.; and Hou, L. 2023. Instruction-Following Evaluation for Large Language Models. arXiv:2311.07911.

Zhu, K.; Chen, J.; Wang, J.; Gong, N. Z.; Yang, D.; and Xie, X. 2024. DyVal: Dynamic Evaluation of Large Language Models for Reasoning Tasks. arXiv:2309.17167.

Zhuge, M.; Zhao, C.; Ashley, D.; Wang, W.; Khizbullin, D.; Xiong, Y.; Liu, Z.; Chang, E.; Krishnamoorthi, R.; Tian, Y.; Shi, Y.; Chandra, V.; and Schmidhuber, J. 2024. Agent-as-a-Judge: Evaluate Agents with Agents. arXiv:2410.10934.