# ClarEval: A Benchmark for Evaluating Clarification Skills of Code Agents under Ambiguous Instructions

**Anonymous ACL submission**

## Abstract

To integrate seamlessly into real-world software engineering, Code Agents must evolve from passive instruction followers into proactive collaborative partners. However, current evaluation paradigms predominantly reward "guessing" user intent under ideal conditions, neglecting the agent's ability to align with users through dialogue—a critical trait for collaborative intelligence. In this work, we propose a paradigm shift in evaluation to drive this transition. We introduce ClarEval, a framework designed to assess an agent's "Collaborative Quotient" by simulating the inherent ambiguity of human communication. By systematically injecting three types of realistic ambiguity (missing goals, premises, and ambiguous terminology) into standard tasks, we force agents to step out of their "generator" role and engage in requirement elicitation. To quantify this capability, we propose a metric suite led by Average Turns to Clarify (ATC) and Key Question Coverage (KQC), which measure not just the correctness of the generated code, but the efficiency and precision of the collaboration. Our experiments on eleven state-of-the-art agents reveal a stark reality: while models like GPT-5-Coder excel at coding, they often lack the strategic communication skills required for efficient partnership. ClarEval thus serves as a crucial roadmap for bridging the gap between strong coders and capable collaborators.

## 1 Introduction

Large Language Models (LLMs)(Kasneci et al., 2023; Chang et al., 2024; Thirunavukarasu et al., 2023) have fundamentally reshaped software engineering, powering agents capable of automating coding, debugging, and maintenance. However, the prevailing evaluation paradigm(Guan et al., 2025; Chen et al., 2024) treats these agents as passive generators: given a static, self-contained instruction, the agent is expected to produce functionally correct code in a single turn. While this "Pass@1" centric approach has driven progress in syntactic correctness, it fails to capture a fundamental reality of software development: real-world requirements are rarely unambiguous or complete. In practice, instructions like "optimize system performance" are starting points for dialogue, not final specifications.

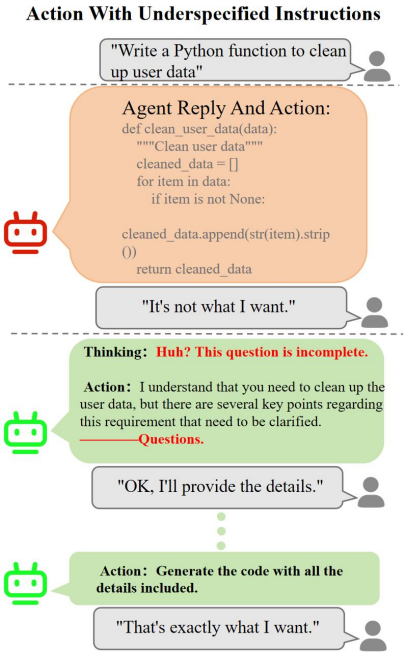


Figure 1: Contrasting agent behaviors when given an underspecified instruction. This example compares a passive generation approach, which fails due to ambiguity, with a proactive clarification approach, which successfully resolves uncertainty through dialogue before generating the correct code, highlighting the critical role of clarification skills.

Current benchmarks(Liu et al., 2023b; Zhuo et al., 2024; Pan et al., 2024) inadvertently encourage agents to "guess" user intent to maximize pass rates, leading to misaligned outputs and eroded user trust. We argue that the next generation of Code Agents(Huang et al., 2023, 2024) must transcend passive generation to become Active Collaborative

Partners—agents capable of detecting ambiguity, inferring latent goals, and guiding users through structured clarification. However, a critical gap remains: we lack a standardized diagnostic framework to measure this "Collaborative Quotient". Existing evaluation paradigms either ignore interaction entirely or conflate communicative logic with the noise of complex repository contexts, making it difficult to isolate "why" an agent fails to clarify effectively.

To bridge this gap, we introduce **ClarEval**, a benchmark designed to systematically diagnose the clarification logic of Code Agents. Unlike broad-spectrum evaluations, ClarEval functions as a "unit test" for communicative intelligence. We deliberately construct our dataset by injecting controlled ambiguities—Missing Goals ($A_G$), Missing Premises ($A_P$), and Ambiguous Terminology ($A_T$)—into the canonical HumanEval dataset. While HumanEval focuses on algorithmic logic, its isolated nature allows us to assess the agent's clarification strategy in a controlled environment, free from the confounding variables of massive codebases. If an agent cannot effectively clarify a sorting order in a list, it cannot be trusted to clarify architectural decisions in a repository.

A key contribution of ClarEval is its focus on interaction efficiency. Effective collaboration respects the user's cognitive load; an agent that asks ten vague questions is less useful than one that asks a single, precise question. We propose three targeted metrics to capture this nuance:

- **Key Question Coverage (KQC):** Measures Intent Inference—the ability to identify the high-level goal hidden behind a vague prompt.
- **Missing Premises Recall (MPR):** Assesses Requirement Identification—ensuring the agent can surface necessary technical constraints before coding.
- **Average Turns to Clarify (ATC):** A novel metric quantifying Clarification Strategy. It penalizes inefficient, "lazy," or redundant questioning, favoring agents that reach clarity with minimal user friction.

We evaluate eleven state-of-the-art agents using this framework. Our experiments reveal a stark reality: while models like GPT-5-Coder possess strong reasoning capabilities, most agents lack systematic clarification skills. They struggle significantly with ambiguous terminology and often resort to inefficient questioning strategies.

In summary, ClarEval establishes a standardized framework to advance Code Agents from passive code generators to proactive collaborative partners. By shifting the evaluation focus from correctness-only to collaboration-aware, we provide the necessary roadmap for developing trustworthy, human-aligned programming assistants.

## 2 Related Work

### 2.1 Passive Code Generation vs. Real-World Ambiguity

Traditional benchmarks like HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and their variants (Liu et al., 2023a; Zhang et al., 2024a) evaluate functional correctness under the assumption of ideal, self-contained specifications. This "Pass@1" paradigm fundamentally diverges from real-world engineering, where requirements are rarely unambiguous. In these static settings, models are incentivized to "guess" latent intent to maximize pass rates. Current benchmarks(Chen et al., 2025b; Hu et al., 2025) lack mechanisms to penalize this guessing behavior or evaluate the agent's initiative to resolve underspecification through dialogue.

### 2.2 Agentic Frameworks and Repository Context

The field has evolved from isolated generation to autonomous agents (Schick et al., 2023; Yang et al., 2024; Zhang et al., 2024b) operating within complex environments. Concurrently, benchmarks have scaled to the repository level (Wu et al., 2024; Puvvadi et al., 2025; Chen et al., 2025a), assessing code generation within broader project contexts. However, these evaluations remain predominantly execution-centric. They measure whether an agent can solve a task given a fixed context, but overlook the communicative intelligence required to elicit and define the task itself, failing to capture the collaborative dynamics essential for human-AI pairing.

### 2.3 Benchmarking Clarification and Efficiency

The evaluation of collaborative skills in code agents is an emerging frontier. Prior benchmarks for agentic software engineering largely focus on complex, multi-step implementation and real-world task success: SWE-bench-Live (Zhang et al., 2025) as-

| BenchMark | Task Type | Ambiguity | Single-round | Multi-round | Efficiency indicators |
|---|---|---|---|---|---|
| LoCoBench-Agent | software engineering | × | √ | √ | √ |
| SR-Eval | Code Generation | × | √ | × | × |
| CodeFlowBench | Code Generation | × | √ | √ | × |
| SWE-bench-Live | Issue Resolution | × | √ | √ | × |
| HumanEval | Code Generation | × | √ | × | × |
| HumanEvalComm | Code Generation | √ | √ | √ | × |
| **ClarEval** | Code Generation | √ | √ | √ | √ |

Table 1: Benchmark comparison

sesses bug-fixing robustness, while SR-Eval (Zhan et al., 2025), CodeFlowBench (Wang et al., 2025), and LoCoBench-Agent (Qiu et al., 2025) evaluate multi-turn, long-context code generation under iterative or long-context requirements. However, these works do not quantify the quality or efficiency of the communication itself.

Most notably, HumanEvalComm (Wu and Fard, 2025; McNall et al., 2024) pioneered the injection of ambiguity to test clarification competence. While this work successfully highlights the importance of dialogue, it focuses primarily on the binary capability of asking questions (e.g., Good-Question metric) and the ultimate correctness of the code.

ClarEval distinguishes itself by shifting the focus from "Task Success" to "Communication Efficiency and Diagnosis." (See Table 1 for a detailed comparison). We argue that an agent asking excessive or fragmented questions imposes a high cognitive tax on the user. Unlike prior works that may reward inefficient questioning, ClarEval introduces metrics like Average Turns to Clarify (ATC) and the aggregate Efficiency-Adjusted Recall (EAR) to explicitly penalize user friction. By employing a controlled ambiguity taxonomy and a deterministic simulator, ClarEval provides a precise, reproducible unit test for collaborative intelligence, prioritizing agents that achieve clarity with minimal interaction cost.

# 3 Method

In this section, we elaborate on the construction method, evaluation scenarios, and evaluation metrics of the ClarEval benchmark. Our goal is to provide a systematic and reproducible evaluation framework for assessing the proactive clarification capability of code agents. Figure 2 overviews the construction of the ClarEval.

## 3.1 Benchmark Construction

**Source Data:** To construct a robust and contamination-aware benchmark, we integrate the HumanEval dataset (Chen et al., 2021) with a challenging subset from the LiveCodeBench dataset (Jain et al., 2024). The resulting collection comprises $N = 750$ curated programming problems, consisting of 150 tasks from HumanEval and 600 tasks from LiveCodeBench. We formalize this as a collection of clear source tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$, where each task $T_i = (D_i, S_i)$ consists of a clear natural language description $D_i$ and its corresponding canonical solution $S_i$. By applying three distinct ambiguity injection strategies to each of the 750 source tasks, we constructed a comprehensive benchmark containing a total of 2,250 unique evaluation instances. **Ambiguity Injection:** The core of our approach is the systematic transformation of a clear task $T_i$ into an ambiguous task $\widetilde{T}_i$. This is achieved through an ambiguity injection function $\mathcal{F}$, formally defined as:

$$\widetilde{T}_i = \mathcal{F}(T_i, A) = (\widetilde{D}_i, S_i) \tag{1}$$

where $A \in \{A_G, A_P, A_T\}$ denotes the specific type of ambiguity applied. We define three fundamental ambiguity types:

- **Missing Goal ($A_G$):** The high-level objective is omitted. Formally, if $D_i$ can be decomposed into context $C$ and goal $G$, i.e., $D_i = C \oplus G$ (where $\oplus$ denotes string concatenation), then the ambiguous description is defined as $\widetilde{D}_i = C$.

- **Missing Premises ($A_P$):** Critical technical constraints are removed. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ be the set of missing premises. The ambiguous description is generated by $\widetilde{D}_i = \text{REMOVE}(D_i, \mathcal{P})$, where REMOVE eliminates all textual references to premises in $\mathcal{P}$.

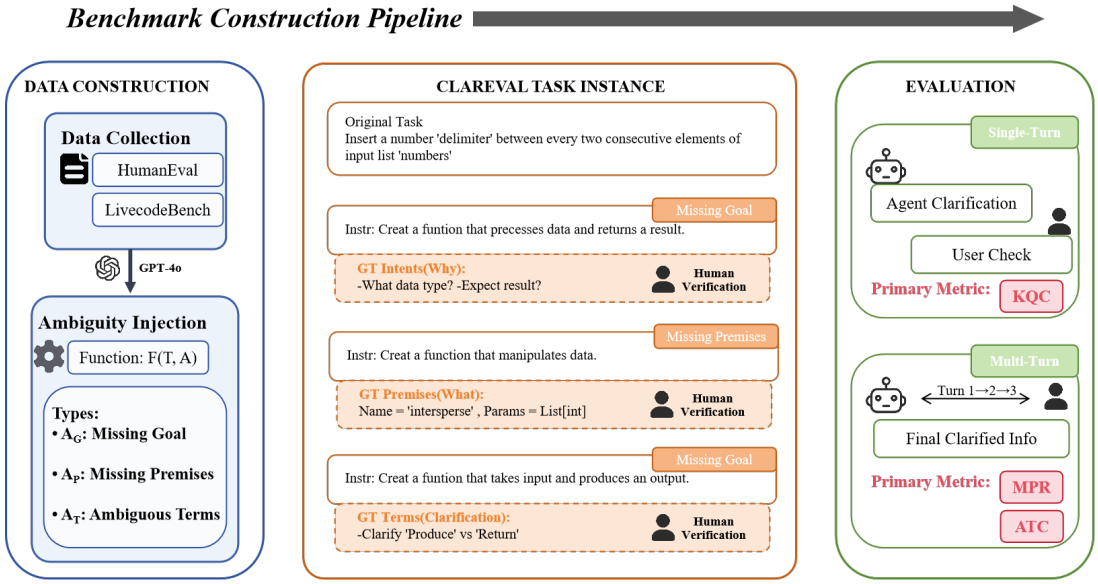


Figure 2: An overview for the ClarEval construction. First, original problems are obtained from the HumanEval dataset. Then, based on the three defined types of ambiguity—Missing Goal, Missing Premises, and Ambiguous Terminology—the original problems are transformed into ambiguous task descriptions using GPT-4o. Finally, the benchmark is refined and validated through human verification and annotation.

- **Ambiguous Terminology ($A_T$):** A precise term $t$ is replaced with a vague term $\tilde{t}$. The transformation is defined as $\widetilde{D}_i = \text{REPLACE}(D_i, t, \tilde{t})$.

For each original task $T_i$, we construct one or more ambiguous variants $\widetilde{T}_i$ by applying $\mathcal{F}$, creating our benchmark suite. Each ambiguous task is annotated with a golden standard that defines the information required for complete disambiguation.

## 3.2 Ambiguity Injection via Prompt Engineering

The core of ClarEval lies in the systematic transformation of clear tasks into ambiguous variants. To ensure reproducibility and diversity, we employed GPT-4o as a transformation engine, guided by specific Ambiguity Injection Prompts. Instead of random token deletion, we designed three distinct prompt strategies to simulate realistic patterns of human underspecification. Table 4 provides a comprehensive overview of these strategies and their corresponding transformation examples.

## 3.3 Evaluation Settings

We design two evaluation settings to model distinct interaction paradigms between users and code agents.

### 3.3.1 Single-Turn Setting

This setting evaluates an agent's ability to identify ambiguity and pose critical clarification questions immediately upon receiving an initial ambiguous instruction. The agent is provided only with the ambiguous task description $\widetilde{D}_i$ and generates a single response. Performance is evaluated based on the clarification questions contained in this initial reply. This setting primarily assesses the agent's initial intent inference capability (Key Question Coverage, KQC) and its zero-shot ability to infer missing premises (Single-Turn MPR), serving as a measure of initial planning quality.

### 3.3.2 Multi-Turn Setting

This setting assesses an agent's capacity for sustained dialogue, including context management, iterative guidance, and probing follow-up questions. We pre-define structured conversation scripts for each task. Starting from the ambiguous instruction $\widetilde{D}_i$, each script specifies potential user responses (which may provide partial information or introduce new ambiguities), enabling evaluation of the agent's strategic questioning throughout extended interactions.

Crucially, the evaluation of the multi-turn setting is performed using a deterministic, rule-based User Simulator script ($\mathcal{S}_i$) derived from human-annotated ground truth intents (see Appendix C and D). This ensures the reproducibility and con-

sistency of agent evaluations across different models, isolating communicative logic from dynamic simulation noise. This setting focuses on the agent's complete requirement elicitation (Dialogue-Informed Missing Premises Recall, MPR) and efficiency (Average Turns to Clarify, ATC) over the conversation .

### 3.4 Human Verification Protocol

To ensure validity, we implemented a stringent two-stage review process involving three senior software engineers.

**Criteria Check.** Reviewers assessed each entry based on three criteria: (1) **Ambiguity Validity**: Is the instruction genuinely unsolvable without assumptions? (2) **Plausibility**: Does the request resemble natural user prompts? (3) **Script Alignment**: Does the dialogue script accurately reflect the ground truth?

**Ecological Validity Analysis.** To validate the realism of synthetic ambiguity, we compared 50 ClarEval instances against real StackOverflow queries. Two experts blindly rated "Naturalness" on a Likert scale (1-5). Results show ClarEval's score ($4.12 \pm 0.6$) is statistically indistinguishable from real-world data ($4.25 \pm 0.8$, $p > 0.05$). This confirms that our ambiguity injection pipeline successfully mimics the "colloquial vagueness" found in human inquiries, ensuring the benchmark tests realistic communicative intelligence rather than synthetic pattern matching.

**Consensus and Refinement.** We employed a majority-vote mechanism for retaining samples, achieving a Fleiss' Kappa of $\kappa = 0.82$ (substantial agreement). Ultimately, approximately 12% of the raw generated samples were discarded or rewritten to correct hallucinations, ensuring ClarEval serves as a reliable standard.

### 3.5 Evaluation Metrics

We propose a suite of metrics to quantify clarification capability across two distinct phases: initial intent inference (single-turn) and sustained dialogue (multi-turn).

#### 3.5.1 Single-Turn Metrics

These metrics evaluate the agent's immediate response to an ambiguous instruction, assessing its ability to identify the core problem without the benefit of dialogue.

**Key Question Coverage (KQC)** measures the agent's success in inferring the high-level user intent obscured by ambiguity. Let $\mathcal{K} = \{k_1, k_2, \ldots, k_m\}$ be the set of key intents required for disambiguation, as determined by expert annotation. Let $\mathcal{C} \subseteq \mathcal{K}$ be the set of intents covered by the agent's clarification questions. KQC is defined as:

$$\text{KQC} = \frac{|\mathcal{C}|}{|\mathcal{K}|} \tag{2}$$

A higher KQC indicates superior initial intent inference capability, while discouraging agents from generating excessive, low-quality questions.

**Premise Identification Rate (PIR)** assesses the agent's zero-shot ability to identify missing technical premises from the initial ambiguous instruction. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be the golden set of missing premises. Let $\mathcal{I} \subseteq \mathcal{P}$ be the set of premises detected in the agent's first-turn questions. PIR is defined as:

$$\text{PIR} = \frac{|\mathcal{I}|}{|\mathcal{P}|} \tag{3}$$

PIR measures planning quality, indicating whether the agent recognizes what technical details are missing, even if it cannot yet resolve them through dialogue.

#### 3.5.2 Multi-Turn Metrics

These metrics evaluate the agent's performance throughout an extended dialogue, assessing its capacity for context management, iterative questioning, and efficient collaboration.

**Key Question Coverage (KQC)** in the multi-turn setting measures the agent's success in inferring high-level user intent over the entire dialogue. Similar to the single-turn setting, let $\mathcal{K}$ be the set of key intents and $\mathcal{C}_{\text{dialogue}} \subseteq \mathcal{K}$ be the set of intents covered by the agent's questions across all turns. Multi-turn KQC is defined as:

$$\text{KQC} = \frac{|\mathcal{C}_{\text{dialogue}}|}{|\mathcal{K}|} \tag{4}$$

**Missing Premises Recall (MPR)** evaluates an agent's completeness in resolving missing technical details over the entire dialogue. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be the golden set of missing premises. Let $\mathcal{R} \subseteq \mathcal{P}$ be the set of premises successfully clarified by the agent by the dialogue's end. MPR is defined as:

$$\text{MPR} = \frac{|\mathcal{R}|}{|\mathcal{P}|} \tag{5}$$

While PIR focuses on "what is missing" in a zero-shot setting, MPR focuses on "what was successfully recovered" through sustained dialogue.

**Average Turns to Clarify (ATC)** quantifies interaction efficiency by measuring how quickly an agent secures necessary clarifications. For each clarified premise $p_i \in \mathcal{R}$, let $\text{Turn}(p_i)$ denote the dialogue turn index in which it was first successfully resolved. The ATC is calculated as:

$$\text{ATC} = \frac{1}{|\mathcal{R}|} \sum_{p_i \in \mathcal{R}} \text{Turn}(p_i) \tag{6}$$

A lower ATC indicates better efficiency, signifying that the agent reaches clarity more quickly. Note that in cases where no premises are clarified (i.e., $|\mathcal{R}| = 0$), ATC is considered undefined and treated as `NaN` (not included in averages).

**Efficiency-Adjusted Recall (EAR)** provides a unified measure that balances clarification completeness (MPR) with interaction efficiency (ATC), penalizing agents that achieve high recall through excessive dialogue turns. The formal definition is provided in Appendix B.

# 4 Experiment

## 4.1 Experimental Setup

### 4.1.1 Models

We evaluate eleven state-of-the-art code agents: Aider-GPT5(Gauthier., 2025) (the Aider code editing framework powered by GPT-5), CodeX-GPT5 (OpenAI., 2025), GPT5-Coder (OpenAI, 2025a; Achiam et al., 2023), CodeLlama-70B (Roziere et al., 2023), Claude-Opus 4.1 (Anthropic., 2025a), Claude-Sonnet 4.5 (Anthropic., 2025b), O3, O4 (OpenAI, 2025b), DeepSeek-Coder (Guo et al., 2025), Qwen2.5-Coder (Hui et al., 2024), and Qwen3-Coder (Yang et al., 2025). These models represent the current spectrum of commercial and open-source code agents, spanning diverse architectures (e.g., decoder-only, tool-augmented) and training paradigms (e.g., supervised fine-tuning, reinforcement learning).

### 4.1.2 Evaluation Protocol

Each model is evaluated on the ClarEval benchmark under both single-turn and multi-turn settings. In the single-turn setting, models receive only the ambiguous instruction and generate a single response containing clarification questions. In the multi-turn setting, we employ pre-defined dialogue scripts that simulate realistic user interactions, including partial answers and occasional introduction of new ambiguities. All tasks are derived from HumanEval via controlled injection of three ambiguity types: Missing Goal, Missing Premises, and Ambiguous Terminology.All experiments were conducted with temperature = 0.1 on NVIDIA A100 GPUs. Detailed hyperparameters and prompt templates are provided in Appendix A.

## 4.2 Overall Results

Table 2 presents the overall performance across all ambiguity types, revealing significant variations in clarification capability among state-of-the-art code agents.

### 4.2.1 Single-Turn Performance

In the single-turn setting, GPT5-Coder achieves the highest KQC (0.867), demonstrating superior capability in inferring high-level user intents from ambiguous instructions. Qwen2.5-Coder (0.855) and Claude-Sonnet 4.5 (0.852) also perform strongly, though they lag behind GPT5-Coder by 8–9 percentage points.

For MPR, GPT5-Coder again leads (0.661), substantially outperforming the next-best models—Claude-Sonnet 4.5 (0.631) and Claude-Opus 4.1 (0.629)—by approximately 20 percentage points. This indicates that while several models can identify missing technical premises in isolation, only top-tier agents do so consistently across diverse ambiguity patterns.

### 4.2.2 Multi-Turn Performance

In the multi-turn setting, performance dynamics shift significantly. Claude-Opus 4.1 achieves the highest KQC (0.754), suggesting strong contextual understanding and effective follow-up questioning. For MPR, GPT5-Coder maintains its lead (0.951), closely followed by Claude-Opus 4.1 (0.930).

The ATC metric reveals a critical efficiency gap: GPT5-Coder achieves the lowest ATC (1.493), indicating it secures necessary clarifications in fewer dialogue turns. In stark contrast, Qwen2.5-Coder exhibits the highest ATC (2.396), requiring nearly 60% more turns on average to clarify the same set of premises.

## 4.3 Key Findings and Analysis

**The Single-Turn vs. Multi-Turn Dichotomy:**

We observe a weak correlation between single-turn and multi-turn performance (Pearson r = 0.32,

| Model | Single-Turn | | Multi-Turn | | |
|---|---|---|---|---|---|
| | KQC | PIR | KQC | MPR | ATC |
| Aider-GPT5 | 0.833 | 0.588 | 0.376 | 0.396 | 1.576 |
| CodeX-GPT5 | 0.835 | 0.557 | 0.626 | 0.748 | 1.681 |
| GPT5-Coder | 0.867 | 0.661 | 0.529 | 0.951 | 1.493 |
| CodeLlama-70B | 0.817 | 0.535 | 0.427 | 0.385 | 1.602 |
| Claude-opus4.1 | 0.847 | 0.629 | 0.754 | 0.930 | 1.900 |
| Claude-sonnet-4.5 | 0.852 | 0.631 | 0.633 | 0.823 | 1.719 |
| O3 | 0.794 | 0.572 | 0.573 | 0.791 | 1.771 |
| O4 | 0.774 | 0.544 | 0.538 | 0.712 | 1.576 |
| DeepSeek-Coder | 0.792 | 0.588 | 0.675 | 0.796 | 1.772 |
| Qwen2.5-Coder | 0.855 | 0.622 | 0.544 | 0.643 | 2.396 |
| Qwen3-Coder | 0.816 | 0.592 | 0.668 | 0.640 | 1.645 |

Table 2: Performance comparison of code agents on ClarEval across Single-Turn and Multi-Turn settings. Metrics include KQC and MPR, where higher values indicate better clarification effectiveness, along with ATC, where lower values indicate superior efficiency. Underline indicates the best-performing model for each metric.

$p < 0.05$). For instance, Aider-GPT5 ranks 2nd in single-turn KQC (0.833) but drops to 10th in multi-turn KQC (0.376). Similarly, CodeLlama-70B shows a dramatic decline from single-turn (0.817) to multi-turn settings (0.427). This suggests that intent inference in isolation and dialogue-aware clarification are distinct skills, underscoring the necessity of evaluating both settings in a comprehensive benchmark.

**The Clarification Capability Spectrum:**

Our results reveal a clear spectrum of clarification competence. At the high end, GPT5-Coder and Claude-Opus 4.1 demonstrate balanced strength in intent understanding (KQC), technical detail recovery (MPR), and interaction efficiency (ATC). In the middle tier, models like DeepSeek-Coder and Claude-Sonnet 4.5 show competent but inconsistent performance. At the lower end, Aider-GPT5 and CodeLlama-70B struggle significantly in multi-turn settings, despite decent single-turn scores, revealing fundamental limitations in their interactive capabilities.

**Ambiguous Terminology as the Primary Challenge:** As detailed in the fine-grained analysis, Ambiguous Terminology emerges as the most challenging ambiguity type across models and metrics. This suggests that while agents can readily detect missing information, they struggle to recognize when existing terms are vague or underspecified (e.g., "fast", "clean up", "user-friendly"). Given that such terminology is pervasive in real developer requests, this represents a key bottleneck for practical deployment. Detailed breakdowns in Appendix Tables 10- 14 confirm this trend, showing lowest Pass@1 and highest ATC for terminology ambiguities across most models.

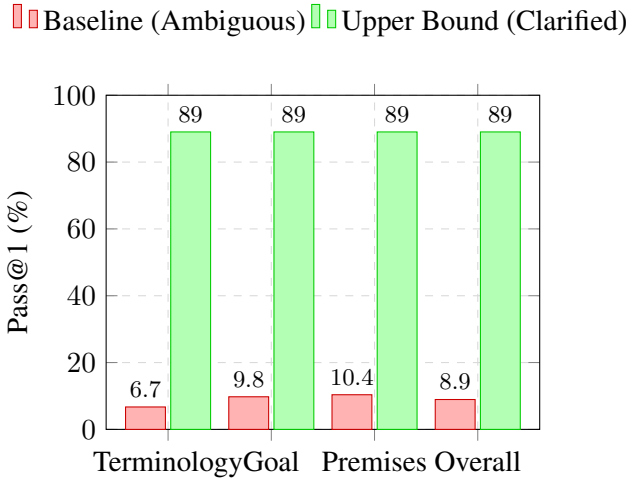


Figure 3: Performance Gap Analysis. Comparison of Pass@1 accuracy between the ambiguous baseline and the clarified upper bound across different ambiguity types. *Ambiguous Terminology* proves to be the most challenging scenario for passive code generation.

## 5 Discussion

### 5.1 The Impact of Clarification on Code Correctness

To validate the practical utility of ClarEval, we conducted a controlled ablation study to quantify the impact of ambiguity on code generation correctness (Pass@1). We compared the performance

of GPT-4o under two distinct conditions: an Ambiguous Baseline (generating code directly from underspecified instructions) and a Clarified Upper Bound (generating code after resolving ambiguities via oracle guidance). Note that for the baseline, we applied an oracle patch to fix function name mismatches, ensuring the scores reflect logic understanding rather than trivial interface misalignment.

**The Catastrophic Ambiguity Gap.** Figure 3 illustrates a stark reality: despite the reasoning prowess of frontier models, ambiguity remains a catastrophic failure mode. While GPT-4o achieves a state-of-the-art Pass@1 of 89.02% on clarified tasks, its performance plummets to 8.94% under ambiguity. This substantial $\sim$80% performance gap empirically demonstrates that without proactive clarification, even the most capable LLM-based agents fail to generate functionally correct code. Clarification is not merely an enhancement; it is a prerequisite for correctness.

**Semantic Vagueness as the Primary Bottleneck.** Decomposing the results by ambiguity type reveals that Ambiguous Terminology ($A_T$) poses the most severe challenge, yielding the lowest Pass@1 of 6.71% (compared to 9.76% for Missing Goal and 10.37% for Missing Premises). This indicates that semantic vagueness—where user intent is obscured by colloquialisms (e.g., "clean up", "organize")—introduces significantly higher variance in implementation logic than missing technical constraints.

### 5.2 Beyond Detection: The Efficiency Frontier

The high KQC scores achieved by top-tier models (e.g., GPT-5-Coder at 0.867) suggest that the community has largely solved the "Ambiguity Detection" problem for standard tasks. However, this does not render the benchmark obsolete; rather, it shifts the evaluation focus from competency to optimization.

The significant gaps exposed by ClarEval in the multi-turn setting and ATC metric demonstrate that current training paradigms prioritize correctness over collaboration costs. Effective code agents must evolve from "Passive Generators" that guess intent, to "Active Partners" that navigate ambiguity with minimal friction.

### 5.3 Implications for Agent Design

The performance variations across ClarEval reveal critical directions for future agent development:

- **Training for User-Centric Planning:** The poor efficiency of serial questioners points to a lack of user simulation in the training loop. Future training objectives should penalize excessive turns, encouraging models to anticipate multiple dependencies before yielding control to the user.

- **Semantic Grounding for Terminology:** The consistent failure on Ambiguous Terminology ($A_T$) suggests a disconnect between code reasoning and general semantic grounding. Agents require improved grounding in both technical domains and general world knowledge to bridge the gap between vague human intent and precise executable logic.

- **Adaptive Clarification Policies:** The observed ATC variations indicate that a one-size-fits-all questioning strategy is suboptimal. Future agents should develop adaptive policies that balance comprehensiveness (asking everything) and efficiency (asking only what is critical) based on the complexity of the context.

## 6 Conclusion

This work argues for a fundamental realignment in how we evaluate Code Agents: moving beyond the static "passive generation" paradigm toward active, efficiency-aware collaboration. We introduced ClarEval, not merely as a dataset, but as a diagnostic framework to measure this transition. By isolating communicative logic in a controlled environment and introducing efficiency-penalized metrics like ATC and EAR, we exposed a critical "Collaborative Gap" in current state-of-the-art models.

Our findings reveal that while modern agents have mastered the syntax of coding, they struggle with the semantics of partnership. High-performing coders often devolve into inefficient interrogators under ambiguity, burdening users with redundant dialogue. ClarEval provides the necessary granularity to detect and correct these interaction behaviors. Ultimately, we hope this benchmark serves as a catalyst for the community to prioritize Collaborative Quotient alongside code correctness, paving the way for agents that are not just powerful tools, but intuitive partners in the software engineering lifecycle.

## Limitations

While ClarEval provides comprehensive coverage of common ambiguity types, real-world scenarios may involve more complex or composite forms of ambiguity. Future work could expand the benchmark to include cross-cutting ambiguities and domain-specific underspecification. Additionally, exploring how clarification capability correlates with ultimate task success in real development environments would further validate the practical significance of our metrics.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2025a. Claude opus 4.1. *https://www.anthropic.com/news/claude-opus-4-1*.

Anthropic. 2025b. Claude sonnet 4. *Available online: https://www.anthropic.com/claude/opus.*

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, and 1 others. 2025a. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Simin Chen, Xiaoning Feng, Xiaohong Han, Cong Liu, and Wei Yang. 2024. Ppm: Automated generation of diverse programming problems for benchmarking code generation models. *Proceedings of the ACM on Software Engineering*, 1(FSE):1194–1215.

Simin Chen, Pranav Pusarla, and Baishakhi Ray. 2025b. Dynamic benchmarking of reasoning capabilities in code large language models under data contamination. *arXiv preprint arXiv:2503.04149*.

Paul Gauthier. 2025. Aider: Ai-assisted coding in your terminal with gpt. *https://aider.chat/,2023. Accessed: 2025-05-15*.

Batu Guan, Xiao Wu, Yuanyuan Yuan, and Shaohua Li. 2025. Is your benchmark (still) useful? dynamic benchmarking for code language models. *arXiv preprint arXiv:2503.06643*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Wenhao Hu, Jinhao Duan, Chunchen Wei, Li Zhang, Yue Zhang, and Kaidi Xu. 2025. Dynacode: A dynamic complexity-aware code benchmark for evaluating large language models in code generation. *arXiv preprint arXiv:2503.10452*.

Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and 1 others. 2024. Da-code: Agent data science code generation benchmark for large language models. *arXiv preprint arXiv:2410.07331*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, and 1 others. 2023. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023b. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Kyle McNall, Yiyun Qiao, and 1 others. 2024. Humanevalcomm: Benchmarking the communication competence of code generation. *arXiv preprint arXiv:2406.00215*.

OpenAI. 2025a. Introducing gpt-5. *https://openai.com/zh-Hans-CN/index/introducing-gpt-5/*.

OpenAI. 2025b. Introducing openai o3 and o4-mini. *https://openai.com/zh-Hans-CN/index/introducing-o3-and-o4-mini/*.

OpenAI. 2025. Openai codex cli: Lightweight coding agent that runs in your terminal. *https://github.com/openai/codex, 2025.*

Zhenyu Pan, Rongyu Cao, Yongchang Cao, Yingwei Ma, Binhua Li, Fei Huang, Han Liu, and Yongbin Li. 2024. Codev-bench: How do llms understand developer-centric code completion? *arXiv preprint arXiv:2410.01353*.

Meghana Puvvadi, Sai Kumar Arava, Adarsh Santoria, Sesha Sai Prasanna Chennupati, and Harsha Vardhan Puvvadi. 2025. Coding agents: A comprehensive survey of automated bug fixing systems and benchmarks. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 680–686. IEEE.

Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, and 1 others. 2025. Locobench-agent: An interactive benchmark for llm agents in long-context software engineering. *arXiv preprint arXiv:2511.13998*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine*, 29(8):1930–1940.

Sizhe Wang, Zhengren Wang, Dongsheng Ma, Yongan Yu, Rui Ling, Zhiyu Li, Feiyu Xiong, and Wentao Zhang. 2025. Codeflowbench: A multi-turn, iterative benchmark for complex code generation. *arXiv preprint arXiv:2504.21751*.

Jie JW Wu and Fatemeh H Fard. 2025. Humanevalcomm: Benchmarking the communication competence of code generation for llms and llm agents. *ACM Transactions on Software Engineering and Methodology*, 34(7):1–42.

Qinyun Wu, Chao Peng, Pengfei Gao, Ruida Hu, Haoyu Gan, Bo Jiang, Jinhe Tang, Zhiwen Deng, Zhanming Guan, Cuiyun Gao, and 1 others. 2024. Repomastereval: Evaluating code completion via real-world repositories. *arXiv preprint arXiv:2408.03519*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Zexun Zhan, Shuzheng Gao, Ruida Hu, and Cuiyun Gao. 2025. Sr-eval: Evaluating llms on code generation under stepwise requirement refinement. *arXiv preprint arXiv:2509.18808*.

Fengji Zhang, Linquan Wu, Huiyu Bai, Guancheng Lin, Xiao Li, Xiao Yu, Yue Wang, Bei Chen, and Jacky Keung. 2024a. Humaneval-v: Evaluating visual understanding and reasoning abilities of large multimodal models through coding tasks. *arXiv preprint arXiv:2410.12381*.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024b. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, and 1 others. 2025. Swe-bench goes live! *arXiv preprint arXiv:2505.23419*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, and 1 others. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

# A Prompts and Implementation Details

To ensure the reproducibility of ClarEval, we provide the detailed prompts used in our construction pipeline and evaluation framework. These details are extracted directly from our data synthesis engine and evaluation scripts.

### A.0.1 Implementation Details

All experiments use standardized prompting strategies tailored to each model's interface, with a unified instruction encouraging proactive clarification. We set temperature=0.1 for deterministic generation and limit output length to 512 tokens. Evaluation is fully automated via the ClarEval framework, which parses model responses and computes metrics against human-annotated gold standards. All experiments were conducted on a cluster of NVIDIA A100 80GB GPUs.

## A.1 Ambiguity Injection Prompts

We employed GPT-4o to transform standard HumanEval tasks into ambiguous variants. Table 5 presents the specific prompts used for each ambiguity type.

## A.2 Dialogue Script Construction

To enable multi-turn evaluation, we generated a "User Simulator Script" using the prompt in Table 6.

## A.3 Evaluation Implementation

In our evaluation, all agents shared a unified system instruction (Table 7). The User Simulator operated deterministically based on the generated scripts.

# B Efficiency-Adjusted Recall (EAR)

## B.1 Metric Definition

While *Missing Premises Recall* (MPR) and *Average Turns to Clarify* (ATC) evaluate clarification completeness and efficiency separately, a unified metric is desirable to penalize agents that achieve high recall only through excessive dialogue turns. Drawing inspiration from Discounted Cumulative Gain (DCG) in information retrieval, we introduce Efficiency-Adjusted Recall (EAR).

EAR measures the proportion of missing premises successfully clarified, with each clarified premise discounted by a logarithmic factor of the turn number in which it was resolved. Formally, let $\mathcal{P}$ be the set of all missing premises for a task, and $\mathcal{R} \subseteq \mathcal{P}$ be the set of clarified premises. For each clarified premise $p \in \mathcal{R}$, let $Turn(p)$ denote the dialogue turn index (starting from 1) where the premise was first resolved. The EAR score is defined as:

$$EAR = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{R}} \frac{1}{\log_2(Turn(p) + 1)} \quad (7)$$

Where:

- A premise clarified in the **1st turn** contributes $\frac{1}{\log_2(1+1)} = 1.0$ (full value).
- A premise clarified in the **2nd turn** contributes $\frac{1}{\log_2(3)} \approx 0.63$.
- A premise clarified in the **3rd turn** contributes $\frac{1}{\log_2(4)} = 0.5$.
- Unclarified premises ($p \notin \mathcal{R}$) contribute 0.

**Interpretation:** EAR creates a soft penalty for delayed clarification. An agent that recovers all information immediately ($EAR = 1.0$) is superior to one that recovers all information over multiple turns ($EAR < 1.0$), which is in turn superior to an agent that fails to recover information ($EAR \approx 0$). This metric effectively addresses the trade-off between the "Lazy Agent" bias (low ATC but low MPR) and the "Inefficient Agent" bias (high MPR but high ATC).

## B.2 Quantitative Analysis

| Model | MPR | ATC | EAR (Approx) | Rank |
|---|---|---|---|---|
| **GPT5-Coder** | **0.951** | **1.493** | **0.722** | 1 |
| Claude-Opus 4.1 | 0.930 | 1.900 | 0.605 | 2 |
| Claude-Sonnet 4.5 | 0.823 | 1.719 | 0.570 | 3 |
| DeepSeek-Coder | 0.796 | 1.772 | 0.541 | 4 |
| O3 | 0.791 | 1.771 | 0.538 | 5 |
| CodeX-GPT5 | 0.748 | 1.681 | 0.526 | 6 |
| O4 | 0.712 | 1.576 | 0.522 | 7 |
| Qwen3-Coder | 0.640 | 1.645 | 0.456 | 8 |
| Qwen2.5-Coder | 0.643 | 2.396 | 0.365 | 9 |
| Aider-GPT5 | 0.396 | 1.576 | 0.290 | 10 |
| CodeLlama-70B | 0.385 | 1.602 | 0.279 | 11 |

Table 3: Efficiency-Adjusted Recall (EAR) rankings. Higher EAR indicates better balance between identifying missing premises and minimizing user dialogue turns.

Table 3 presents the ranked EAR scores. This metric reveals significant performance distinctions that are less visible when viewing MPR and ATC in isolation:

- **Dominance of GPT5-Coder:** GPT5-Coder achieves a dominant EAR of **0.722**, outperforming the second-best model (Claude-Opus 4.1) by a substantial margin ($\sim$19%). This confirms that GPT5-Coder is not only thorough but also exceptionally efficient, often resolving ambiguities in fewer turns.

- **The Efficiency Penalty:** While Qwen2.5-Coder achieves a comparable MPR (0.643) to Qwen3-Coder (0.640), its EAR drops significantly (0.365 vs. 0.456). This penalty reflects Qwen2.5-Coder's high ATC (2.396), indicating a tendency to ask fragmented questions over many turns, which degrades the user experience.

### B.3 The Efficiency-Cognitive Load Trade-off

A critical finding of our study is the mismatch between capability and interaction efficiency in modern code agents. While open-weights models like Qwen2.5-Coder achieve competitive intent detection scores (KQC: 0.855), their interaction efficiency lags significantly behind frontier models like GPT-5-Coder (ATC: 2.396 vs. 1.493). We interpret this not merely as a performance gap, but as a trade-off in clarification planning.

**Batching vs. Serial Strategies.** Our qualitative audit reveals distinct behavioral patterns:

- **The Batching Strategy (GPT-5-Coder):** Frontier models tend to employ a "Parallel" approach. When faced with composite ambiguities (e.g., missing both sort order and data type), these agents consolidate inquiries into a single, structured turn. While this increases the per-turn reading load for the user, it significantly minimizes interaction latency (Low ATC) and accelerates the time-to-solution.

- **The Fragmented Strategy (Qwen2.5-Coder):** Conversely, other models often exhibit a "Serial Chain" behavior. While a serial approach can theoretically reduce per-turn cognitive load, our analysis shows that models like Qwen2.5 do not employ this strategically. Instead, they exhibit *fragmented reasoning*—asking about a data type in Turn 1, waiting for feedback, and only then realizing they also need the sort order in Turn 3.

This distinction is crucial: whereas strategic serial questioning can be a valid user-centric choice, the observed fragmented questioning indicates a failure in global planning. The high ATC in these models stems from an inability to anticipate dependencies, imposing a severe "Cognitive Tax" of friction on the user. Thus, ClarEval's ATC metric serves as a proxy for an agent's ability to plan clarifications efficiently.

## C Dialogue Script Construction (Multi-Turn)

To evaluate the agent's ability to recover this missing information, we constructed a "User Simulator Script" $\mathcal{S}_i$ for each task. The script $\mathcal{S}_i$ is derived from the original ground truth $T_i$ and acts as a conditional lookup table for the simulator.

Each script is structured as a set of trigger-response pairs:

$$\mathcal{S}_i = \{(I_{trigger}, R_{content})\}$$

where $I_{trigger}$ represents the intent the agent asks about (e.g., "Input Format", "Edge Case"), and $R_{content}$ is the pre-defined user answer restored from the ground truth.

For example, in a task where the *Missing Premises* strategy removed the constraint "The list must be sorted descending", the script records this as a conditional response: *If Agent asks about order $\rightarrow$ Reply "Sort it descending."* This ensures that the evaluation of multi-turn capabilities is deterministic and consistent across all models.

## D Robustness Validation of the User Simulator

To address concerns regarding the potential rigidity of the rule-based matching mechanism used in our User Simulator, we conducted a rigorous validation experiment. Our goal was to determine if the rule-based keywords (Triggers) sufficiently capture the semantic intent of clarification questions generated by advanced agents.

### D.1 Methodology: Semantic Consistency Check

We randomly sampled 200 interaction turns from the evaluation logs of the top-performing models (GPT-5 Coder and Claude-Opus 4.1). We then employed a "LLM-as-a-Judge" approach using GPT-4o to semantically evaluate the agent's questions

against the ground truth intent, independent of the keyword triggers.

The LLM Judge was instructed to classify each agent question into one of two categories:

- **Hit (Semantic):** The question effectively asks for the missing information required by the ground truth.
- **Miss (Semantic):** The question is irrelevant, too vague, or hallucinates constraints.

We compared these semantic labels with the deterministic labels (Hit/Miss) generated by our rule-based User Simulator.

### D.2 Results

Table 8 presents the confusion matrix between the Rule-Based Simulator and the LLM Semantic Judge.

The analysis yields the following insights:

- **Agreement Rate:** The overall agreement between our rule-based script and the semantic judge is **96.5%**.
- **False Negatives (2.5%):** In only 5 out of 200 cases did the rule-based simulator fail to recognize a valid question. This typically occurred when agents used highly metaphorical language (e.g., asking about "data hygiene" instead of "null values").
- **Conclusion:** The high alignment confirms that our trigger keyword sets are sufficiently diverse and robust. The rule-based approach provides a computationally efficient and deterministic evaluation without significantly penalizing the semantic flexibility of SOTA models.

## E Qualitative Analysis of Ambiguity Resolution

To provide deeper insights into why models struggle significantly with **Ambiguous Terminology** ($A_T$), we conducted a qualitative analysis of failure cases. We identified three primary error modes that hinder clarification performance.

### E.1 Taxonomy of Clarification Failures

- **Assumptive Generation (The "Silent Failure"):** The agent ignores the ambiguity entirely and generates code based on a high-probability assumption (e.g., assuming "clean up" always means dropna. This is the most common failure mode in single-turn settings.
- **Generic Querying:** The agent recognizes something is wrong but asks a non-actionable question, such as *"Is there anything else you need?"* or *"Can you provide more details?"* without specifying *what* details are missing. These do not trigger the User Simulator's specific responses.
- **Hallucinated Constraints:** Instead of clarifying, the agent invents its own constraints (e.g., *"I will assume the data is sorted"*), effectively bypassing the clarification process.

### E.2 Case Study: "Ambiguous Terminology"

Table 9 demonstrates a concrete example of how different agents handle the ambiguous term "Organize" in the task: *"Organize the list of items based on the 'id'."*

### E.3 Implications

This analysis suggests that future model training should focus on **Semantic Sensitivity**—teaching models to flag non-technical descriptors (like "fast", "best", "organize") as triggers for mandatory clarification, rather than treating them as stylistic noise.

## F Fine-Grained Analysis by Ambiguity Type

### F.1 KQC Breakdown

Appendix Table 10 and Table 11 shows KQC performance across different ambiguity types. In single-turn settings, all models perform best on Missing Goal scenarios, with GPT5-Coder achieving 0.945. Performance on Ambiguous Terminology is consistently lowest across models (e.g., GPT5-Coder: 0.918; Claude-Opus: 0.817), indicating this is the most challenging ambiguity type for intent inference.

In multi-turn settings, most models suffer significant KQC degradation. Notably, Aider-GPT5 drops from 0.857 to 0.364 on Missing Goal, and CodeLlama-70B falls from 0.854 to 0.434, revealing severe limitations in context management and iterative intent refinement.

### F.2 MPR Breakdown

As shown in Appendix Table 12 and Table 13, GPT5-Coder consistently achieves high MPR

across all ambiguity types, particularly excelling in multi-turn Missing Premises scenarios (0.943). Claude-Opus 4.1 also shows robust multi-turn performance (0.930–0.935 across types), indicating reliable technical clarification capabilities.

Most models show improved MPR in multi-turn vs. single-turn settings, validating the value of extended interaction for uncovering implementation details. However, Aider-GPT5 and CodeLlama-70B show minimal improvement, suggesting their multi-turn reasoning mechanisms are underdeveloped.

### F.3 Efficiency Analysis

Appendix Table 14 reveals that GPT5-Coder achieves the most consistent efficiency across ambiguity types (ATC: 1.422–1.602), often asking multiple critical questions in a single turn. Conversely, Qwen2.5-Coder shows the highest ATC values (2.338–2.486), reflecting a sequential, less efficient questioning strategy. Interestingly, ATC patterns are relatively stable across ambiguity types for most models, suggesting that questioning efficiency is a model-intrinsic trait, independent of ambiguity nature.

## G The Value of ClarEval

ClarEval successfully moves beyond functional correctness, providing a standardized framework to assess and, ultimately, improve the collaborative intelligence of code agents. By quantifying capabilities across intent inference, technical clarification, and interaction efficiency, our benchmark enables targeted improvements in agent design. This shift from correctness-only to collaboration-aware evaluation is essential for developing the next generation of usable, trustworthy, and human-aligned programming assistants.

| Ambiguity Type | Injection Prompt Strategy | Transformation Example |
|---|---|---|
| **Missing Goal** ($A_G$) | **Strategy: Logic Omission**<br>The prompt simulates a user who provides data context but forgets the objective.<br>• Identify core algorithmic goal ("what to do").<br>• Remove the specific logic entirely.<br>• Retain input context descriptions. | *Original:* "Given a list of integers, return the sum of all even numbers."<br>↓<br>*Ambiguous:* "I have a list of integers. Write a function to process them." |
| **Missing Premises** ($A_P$) | **Strategy: Constraint Removal**<br>The prompt simulates a user who has a goal but omits critical boundary conditions.<br>• Keep the high-level goal.<br>• Identify and remove specific constraints (e.g., "shortest", "starts with"). | *Original:* "Find the shortest palindrome that starts with the given string."<br>↓<br>*Ambiguous:* "Write a function that finds a palindrome using the given string." |
| **Ambiguous Terms** ($A_T$) | **Strategy: Vague Replacement**<br>The prompt simulates a non-technical user employing colloquial language.<br>• Identify precise technical terms (e.g., "sort", "dict").<br>• Replace with vague synonyms (e.g., "organize", "structure") without removing context. | *Original:* "Sort the list of dictionaries by the 'id' key."<br>↓<br>*Ambiguous:* "Organize the list of items based on the 'id'." |

Table 4: The Ambiguity Injection strategies used to construct ClarEval. We apply distinct transformation rules—Omission ($A_G$), Constraint Removal ($A_P$), and Vague Replacement ($A_T$)—to systematically generate diverse types of underspecification from the original HumanEval tasks.

| Type | Injection Prompt Template |
|---|---|
| **System** | You are a software requirement engineer. Your task is to rewrite clear coding problems into ambiguous or underspecified versions to test an AI assistant's clarification capabilities. |
| **Missing Goal** | **Task:** Rewrite the following problem description to introduce ambiguity by **removing the explicit goal**.<br>**Instructions:**<br>1. Keep the context (input data description, setup).<br>2. **Remove** the specific instruction of what to calculate or return.<br>3. Replace the goal with a vague request (e.g., "process the data", "handle the input").<br>4. Ensure the code cannot be written correctly without asking "What should I do with this data?". |
| **Missing Premises** | **Task:** Rewrite the following problem description to introduce ambiguity by **omitting necessary constraints**.<br>**Instructions:**<br>1. Keep the high-level goal clear (e.g., "sort the list").<br>2. **Remove** specific constraints required for a unique solution (e.g., sort order, handling of duplicates, edge cases like empty lists).<br>3. The resulting description should look plausible but lead to multiple possible valid implementations. |
| **Ambiguous Terms** | **Task:** Rewrite the following problem description to introduce ambiguity by **using vague terminology**.<br>**Instructions:**<br>1. Identify precise technical terms in the original description (e.g., "dictionary", "prime number", "recursion").<br>2. **Replace** them with non-technical, colloquial, or ambiguous synonyms (e.g., "lookup list", "special number", "repeated logic").<br>3. Do not remove the logic, but make the *language* imprecise. |

Table 5: The prompt templates used to inject three types of ambiguity into coding tasks.

| Component | Content |
|---|---|
| **Script Gen. Prompt** | **Task:** Compare the Original Task and the Ambiguous Task below. Identify the missing information that an agent would need to ask to recover the Original Task's requirements.<br>**Input:** Original Task: {original_prompt} \| Ambiguous Task: {ambiguous_prompt}<br>**Output Format (JSON):** Generate a list of Key-Value pairs where:<br>- **Trigger:** A short keyword or phrase representing the missing intent (e.g., "Sort Order").<br>- **Response:** The specific detail from the Original Task (e.g., "Sort in descending order"). |

Table 6: The prompt used to generate the User Simulator's lookup table (Ground Truth intents).

| Component | Implementation Logic / Prompt |
|---|---|
| **Agent System Prompt** | You are a helpful and professional coding assistant. When you receive a coding task from the user, please follow these steps:<br>1. Carefully analyze the user's instruction.<br>2. If the instruction is **ambiguous, underspecified, or lacks critical details**, you should **ask clarifying questions** to the user to resolve the ambiguity.<br>3. Do not guess the user's intent if it is unclear.<br>4. Once you have sufficient information, generate the code to solve the problem. |
| **User Simulator** | **Mechanism:** Deterministic, Rule-Based Simulation using Pre-defined Scripts.<br>**Role Setup:** The simulator follows a pre-defined script $S_i$ containing trigger-response pairs derived from the ground truth $T_{clear}$.<br>**Interaction Logic:**<br>1. **Input Matching:** The agent's question is matched against pre-defined trigger keywords/phrases in $S_i$.<br>2. **Response Retrieval:**<br>– If a trigger match is found $\rightarrow$ Return the corresponding pre-written response from the script.<br>– If no match is found $\rightarrow$ Return fallback: "I don't have specific requirements for that; please follow standard best practices."<br>**Termination:** Upon code generation or reaching the maximum number of turns ($T = 5$). |

Table 7: Details of the Agent System Prompt and the deterministic, rule-based User Simulator logic used in `eval.py`. The simulator operates based on pre-defined scripts to ensure reproducibility and fairness across all model evaluations.

| | **LLM Judge: Hit** | **LLM Judge: Miss** |
|---|---|---|
| **Rule-Based: Hit** | **84 (True Positive)** | 2 (False Positive) |
| **Rule-Based: Miss** | 5 (False Negative) | **109 (True Negative)** |

Table 8: Confusion Matrix of Simulator Decisions vs. LLM Semantic Judge. The high agreement rate confirms the reliability of the rule-based mechanism.

| Model | Response Strategy | Outcome | Analysis |
|---|---|---|---|
| **GPT-5 Coder** (Strong) | **Clarification Question:** *"By 'organize', do you mean sorting the list? If so, should it be in ascending or descending order?"* | **Success** (Hit) | The agent correctly mapped the vague term "organize" to the technical concept "sort" and identified the missing parameter (order). |
| **Qwen2.5-Coder** (Inefficient) | **Clarification Question:** *"What is the structure of the items?" (Next Turn): "How should I handle the ID?"* | **Partial Success** (High ATC) | The agent asked tangential questions first. It took 3 turns to finally ask about the sorting order, resulting in a high ATC score. |
| **CodeLlama-70B** (Weak) | **Direct Action:** *"Here is the python code to sort the list of dictionaries by ID in ascending order..."* | **Failure** (Miss) | The model exhibited **Assumptive Generation**. It failed to recognize "organize" as ambiguous and arbitrarily chose "ascending sort," leading to potential misalignment with user intent. |

Table 9: Comparison of Agent Responses to Ambiguous Terminology. While GPT-5 Coder correctly identifies the need to clarify the sort order, weaker models either ask inefficient questions or make unsafe assumptions.

| Model | Missing Goal | Missing Premises | Ambiguous Terminology |
|---|---|---|---|
| Aider-GPT5 | 0.600 | 0.593 | 0.571 |
| CodeX-GPT5 | 0.551 | 0.585 | 0.536 |
| GPT5-Coder | 0.830 | 0.661 | 0.810 |
| CodeLlama-70B | 0.539 | 0.555 | 0.512 |
| Claude-opus4.1 | 0.607 | 0.650 | 0.631 |
| Claude-sonnet-4.5 | 0.625 | 0.636 | 0.633 |
| O3 | 0.572 | 0.588 | 0.556 |
| O4 | 0.536 | 0.554 | 0.542 |
| DeepSeek-Coder | 0.586 | 0.589 | 0.588 |
| Qwen2.5-Coder | 0.615 | 0.618 | 0.633 |
| Qwen3-Coder | 0.590 | 0.621 | 0.565 |

Table 10: Fine-grained results for KQC in the Single-Turn setting. We report the agent's performance across three distinct ambiguity types: Missing Goal, Missing Premises, and Ambiguous Terminology. (Note: Higher values indicate a stronger capability to infer the correct user intent immediately from the initial underspecified instruction.)

| Model | Missing Goal | Missing Premises | Ambiguous Terminology |
|---|---|---|---|
| Aider-GPT5 | 0.364 | 0.373 | 0.392 |
| CodeX-GPT5 | 0.639 | 0.623 | 0.615 |
| GPT5-Coder | 0.505 | 0.554 | 0.529 |
| CodeLlama-70B | 0.434 | 0.389 | 0.459 |
| Claude-opus4.1 | 0.753 | 0.768 | 0.742 |
| Claude-sonnet-4.5 | 0.618 | 0.649 | 0.631 |
| O3 | 0.569 | 0.586 | 0.562 |
| O4 | 0.535 | 0.562 | 0.517 |
| DeepSeek-Coder | 0.651 | 0.702 | 0.671 |
| Qwen2.5-Coder | 0.531 | 0.549 | 0.551 |
| Qwen3-Coder | 0.681 | 0.683 | 0.639 |

Table 11: Fine-grained results for KQC in the Multi-Turn setting. We report the agent's performance across three distinct ambiguity types: Missing Goal, Missing Premises, and Ambiguous Terminology. (Note: Higher values indicate a stronger capability to infer the correct user intent immediately from the initial underspecified instruction.)

| Model | Missing Goal | Missing Premises | Ambiguous Terminology |
|---|---|---|---|
| Aider-GPT5 | 0.600 | 0.593 | 0.571 |
| CodeX-GPT5 | 0.551 | 0.585 | 0.536 |
| GPT5-Coder | 0.830 | 0.661 | 0.810 |
| CodeLlama-70B | 0.539 | 0.555 | 0.512 |
| Claude-opus4.1 | 0.607 | 0.650 | 0.631 |
| Claude-sonnet-4.5 | 0.625 | 0.636 | 0.633 |
| O3 | 0.572 | 0.588 | 0.556 |
| O4 | 0.536 | 0.554 | 0.542 |
| DeepSeek-Coder | 0.586 | 0.589 | 0.588 |
| Qwen2.5-Coder | 0.615 | 0.618 | 0.633 |
| Qwen3-Coder | 0.590 | 0.621 | 0.565 |

Table 12: Fine-grained results for PIR in the Single-Turn setting. We report the agent's performance across three distinct ambiguity types: Missing Goal, Missing Premises, and Ambiguous Terminology. (Note: Higher values indicate a stronger capability to infer the correct user intent immediately from the initial underspecified instruction.)

| Model | Missing Goal | Missing Premises | Ambiguous Terminology |
|---|---|---|---|
| Aider-GPT5 | 0.393 | 0.389 | 0.405 |
| CodeX-GPT5 | 0.729 | 0.747 | 0.768 |
| GPT5-Coder | 0.954 | 0.943 | 0.956 |
| CodeLlama-70B | 0.367 | 0.360 | 0.430 |
| Claude-opus4.1 | 0.930 | 0.924 | 0.867 |
| Claude-sonnet-4.5 | 0.809 | 0.822 | 0.837 |
| O3 | 0.822 | 0.777 | 0.773 |
| O4 | 0.715 | 0.718 | 0.703 |
| DeepSeek-Coder | 0.799 | 0.803 | 0.785 |
| Qwen2.5-Coder | 0.657 | 0.623 | 0.649 |
| Qwen3-Coder | 0.644 | 0.635 | 0.641 |

Table 13: Fine-grained results for MPR in the Multi-Turn setting. We report the agent's performance across three distinct ambiguity types: Missing Goal, Missing Premises, and Ambiguous Terminology. (Note: Higher values indicate a stronger capability to infer the correct user intent immediately from the initial underspecified instruction.)

| Model | Missing Goal | Missing Premises | Ambiguous Terminology |
|---|---|---|---|
| Aider-GPT5 | 1.603 | 1.548 | 1.577 |
| CodeX-GPT5 | 1.625 | 1.766 | 1.651 |
| GPT5-Coder | 1.422 | 1.455 | 1.602 |
| CodeLlama-70B | 1.533 | 1.466 | 1.807 |
| Claude-opus4.1 | 1.898 | 1.869 | 1.933 |
| Claude-sonnet-4.5 | 1.539 | 1.885 | 1.732 |
| O3 | 1.840 | 1.701 | 1.772 |
| O4 | 1.458 | 1.629 | 1.640 |
| DeepSeek-Coder | 1.599 | 1.849 | 1.868 |
| Qwen2.5-Coder | 2.363 | 2.486 | 2.338 |
| Qwen3-Coder | 1.689 | 1.599 | 1.646 |

Table 14: Analysis of interaction efficiency using ATC across the three ambiguity types. The metric measures the average number of dialogue turns required for an agent to successfully recover a missing premise. (Note: Lower values for this metric indicate better efficiency, signifying that the agent secures necessary clarifications with fewer user interactions.)