FEWER CLARIFICATIONS, BETTER CODE: BENCHMARKING CROSS-SESSION PERSONALIZED AMBIGUITY ADAPTATION IN CODING ASSISTANTS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI-assisted coding increasingly translates informal user intent into executable software, yet coding requests often contain ambiguities that recur in user-specific ways across tasks and sessions. Existing disambiguation methods typically address each ambiguous request in isolation within the current coding session, often through eliciting additional clarification. However, whether resolved session history from the same user can serve as memory for resolving recurring personalized ambiguity in a newly opened session remains underexplored. We formulate *personalized ambiguity adaptation* as a new task: given a user's previously resolved coding sessions and a new ambiguous request, an assistant should identify the recurring ambiguity pattern, produce the intended executable solution, and minimize clarification. To benchmark this task, we introduce CAPA, which characterizes personalized coding ambiguity through six mechanisms and injects these mechanisms into unambiguous executable tasks using a controlled three-stage generation pipeline. CAPA contains 600 coding sessions across 60 balanced user–ambiguity cells, including 300 held-out evaluation sessions. We evaluate 12 recent LLMs under no-history and same-user-history conditions using executable success, first-turn success, and turns-to-completion. We further study how task difficulty, user identity, and memory management affect adaptation and propose a lightweight user history gating method. CAPA provides a foundation for developing long-term coding assistants that better align generated code with user intent while reducing repeated clarification.

1 INTRODUCTION

AI-assisted coding has become a common way to turn informal intent into executable software. Users ask coding assistants to implement functions, debug failures, adapt existing programs, and modify projects through natural-language dialogue. Yet these requests often remain ambiguous because users may omit details or express intent through recurring, user-specific patterns. For example, when a user asks to "normalize" a feature, the request may refer either to min–max scaling or to z-score standardization. A collaborator familiar with the user may know from prior coding sessions that the user consistently intends the latter one, whereas a coding assistant may select min–max scaling and produce a technically valid but unintended implementation. A long-term coding assistant should similarly infer such recurring interpretations from prior coding sessions; otherwise, it must repeatedly request clarification or continue making the same incorrect assumption, increasing interaction and user cognitive load.

Recent work addresses important parts of this problem, but largely in isolation. Ambiguity research studies whether language models can detect underspecified requests, rewrite them, or ask effective clarification questions (Tanjim et al., 2025b; Zhang et al., 2024). These methods generally treat ambiguity as local to the current coding session, while explicit clarification seeks to resolve uncertainty through additional user turns. Personalization and long-term memory research instead studies how models retrieve, update, and apply user-specific information across sessions (Salemi et al., 2024; Maharana et al., 2024). However, ambiguity can be user-specific: the same user tends to leave the same kind of information underspecified across different tasks. Whether assistants can use same-user history or personalized memory to resolve such ambiguity without repeated clarification

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

remains underexplored, particularly in open-ended executable coding. This capability is essential for building long-term assistants that become more accurate and interaction-efficient over time.

To address this gap, we formulate *personalized ambiguity adaptation* as a new task for coding sessions. Given a user's previously resolved sessions and a new ambiguous request that opens a held-out session, an assistant uses the recurring ambiguity pattern revealed by that history to infer the intended implementation. It should ask to clarify only when the history provides insufficient evidence, with the goal of producing a correct executable solution in as few turns as possible. Unlike conventional ambiguity resolution, which treats each request within its local context, this task evaluates cross-session adaptation to user-specific ambiguity. It motivates assistants that align generated code with user intent while minimizing interruptions from repeated clarification.

To benchmark this task, we introduce CAPA (**C**ross-Session **A**daptation to **P**ersonalized **A**mbiguity), constructed in two steps. First, we adapt the linguistic ambiguity types of Li et al. (2024) into six mechanisms of personalized coding ambiguity. Each mechanism captures a recurring user-related cause for leaving required implementation information omitted, obscured, or underspecified; these mechanisms are grounded in real coding conversations from WildChat (Zhao et al., 2024). Second, we instantiate the mechanisms across user profiles and apply a three-stage pipeline to unambiguous HumanEval tasks (Chen et al., 2021). The pipeline constructs an ambiguous initial request by controlling task-critical information, expands it into a resolved multi-turn coding session, and validates that the personalized ambiguity pattern remains consistent across sessions. The final benchmark contains 600 coding sessions arranged into 60 balanced user–ambiguity cells, with five resolved history sessions and five held-out evaluation sessions per cell.

We benchmark 12 recent LLMs spanning closed-source frontier, open-access frontier, and smaller open-source models on CAPA , and systematically evaluate their personalized ambiguity adaptation using Executable Success (ES), First-Turn Executable Success (FT-ES), and Turns-to-Completion (TTC). We further analyze key aspects of this task, including how adaptation varies with task difficulty, whether history gains depend on correctly matched user identity, and how memory-based history management affects history use. Based on these analyses, we propose same-user history gating as a lightweight method for improving performance.

In summary, our contributions are:

- **A new task for long-term coding assistants.** We formulate personalized ambiguity adaptation as the problem of inferring a user's recurring ambiguity-resolution pattern from resolved coding sessions and transferring it to new executable tasks with minimal clarification.
- **A data generation pipeline.** We introduce CAPA, including a six-mechanism coding-oriented taxonomy, a three-stage generation and consistency-validation pipeline, and 600 coding sessions organized into balanced same-user histories and held-out evaluations.
- **An interaction-aware evaluation.** We evaluate 12 models using ES, FT-ES, and TTC, together with controlled studies of task difficulty, user identity, history gating, and memory-based adaptation.

2 Related Work

Ambiguity and clarification. Ambiguity occurs when a request has multiple plausible interpretations or lacks information needed to choose among them (Tanjim et al., 2025b). It may arise from syntax, lexical meaning, or missing conversational context, motivating ambiguity taxonomies and ambiguous question-answering settings (Church & Patil, 1982; Navigli, 2009; Schlangen, 2004; Liu et al., 2023a; Min et al., 2020). Prior work addresses both ambiguity detection and resolution. Detection methods determine whether to answer or seek information using feature-based classifiers, neural models, or LLM prompting (Trienes & Balog, 2019; Dhole, 2020; Guo et al., 2021; Lee et al., 2023; Tanjim et al., 2025a; Kuhn et al., 2022); CLAMBER further tests whether LLMs recognize ambiguous needs and ask useful questions (Zhang et al., 2024). Resolution methods rewrite underspecified queries using dialogue or retrieved context (Elgohary et al., 2019; Anantha et al., 2021; Ma et al., 2023), cover multiple interpretations in a long-form response (Stelmakh et al., 2022; Kim et al., 2023; In et al., 2025), or ask clarifying questions (Aliannejadi et al., 2019; Xu et al., 2019; Zhang & Choi, 2025; Kim et al., 2024; Zhang et al., 2025). In coding, Orchid studies ambiguous

requirements in function-level generation (Yang et al., 2026a), while ClarifyCodeBench evaluates pre-generation clarification (Fang et al., 2026). These approaches reduce uncertainty but require additional inference, output, or interaction (Tanjim et al., 2025b), and resolve requests within the current session or through newly elicited information. CAPA instead asks whether an assistant can infer a recurring resolution pattern from a user's prior sessions and reuse it in a new coding session.

Personalization and long-term memory. Personalized assistants preserve evidence across sessions, infer stable user characteristics, and selectively apply them later. LaMP evaluates history-based personalized classification and generation, while PersonaMem studies user profiling and response selection as personal information evolves across sessions (Salemi et al., 2024; Jiang et al., 2025). LoCoMo, LongMemEval, and Momento test multi-session recall, temporal reasoning, knowledge updates, and memory-grounded action (Maharana et al., 2024; Wu et al., 2024; Merin et al., 2026). Memory systems use retrieved records, natural-language profiles, or learned representations, increasingly separating episodic evidence from consolidated user patterns (Hu et al., 2025; Cao et al., 2026; Das et al., 2026; Zhang et al., 2026; Hou et al., 2026); practical systems also extract, organize, retrieve, update, and consolidate user information over time (Chhikara et al., 2025; Xu et al., 2026; In et al., 2026; Uddin et al., 2026; Jiang et al., 2026). Closest to our setting, APeB infers latent shopping intent from behavioral traces, while PRefine transfers recurring preferences to missing tool-call arguments (Yang et al., 2026b; Yoon et al., 2026). Both connect memory and disambiguation but target product choice or schema-bounded tools. CAPA instead treats the reusable signal as a personalized ambiguity-resolution pattern and tests its transfer across distinct coding sessions.

Interactive coding benchmarks. Executable benchmarks objectively assess program correctness. HumanEval and MBPP evaluate function-level generation from explicit specifications, while EvalPlus strengthens test coverage (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023b). Interactive benchmarks include SWE-bench for repository-level issue resolution and ConvCodeWorld for conversational generation in reproducible feedback environments (Jimenez et al., 2024; Han et al., 2025). HumanEvalComm and Orchid specifically examine incomplete, inconsistent, or ambiguous requirements (Wu & Fard, 2025; Yang et al., 2026a), but their ambiguity belongs to one task or dialogue rather than a stable cross-session user pattern. CAPA combines executable coding, multi-turn sessions, and same-user history. It evaluates both eventual correctness and whether prior sessions reduce clarification.

3 BENCHMARK TASK

CAPA evaluates whether a coding assistant can use a recurring user's prior multi-turn coding sessions to resolve personalized ambiguity in a newly opened session. We first define the data hierarchy and notation, and then formulate the history-conditioned evaluation task (Fig. 1).

3.1 PERSONALIZED MULTI-SESSION CODING DATA

We distinguish four levels of interaction. An *assistant turn* pairs the latest user message with one assistant response. A *dialogue trace* is the ordered sequence of such turns produced while solving one coding task. A *coding session* is the complete task-centered interaction, including its initial request, dialogue trace, and hidden executable test cases to be evaluated by a hidden judge that determines task completion. A *user trajectory* is an ordered sequence of coding sessions from the same user.

Let user $U_i = (\pi_i, a_i, r_i)$ be characterized by a profile π_i, a recurring ambiguity mechanism a_i, and a personalized resolution pattern r_i. The profile controls ambiguity-independent communication characteristics, while (a_i, r_i) specifies how required implementation information is repeatedly obscured and how that ambiguity should be resolved. For the user's k-th coding task $P_{i,k}$, the corresponding coding session is

$$\tau_{i,k} = \big(D_{i,k}, j_{i,k}\big),$$

where $D_{i,k}$ is the public multi-turn dialogue trace produced while solving task $P_{i,k}$, beginning with the ambiguous initial request $u_{i,k}^{(0)}$, and $j_{i,k}$ is the associated set of hidden executable test cases. Writing the trace as

$$D_{i,k} = \big\{(u_{i,k}^{(t-1)}, y_{i,k}^{(t)})\big\}_{t=1}^{T_{i,k}},$$

$u_{i,k}^{(t-1)}$ is the latest user message and $y_{i,k}^{(t)}$ is the assistant response at assistant turn t; $u_{i,k}^{(0)}$ is the initial request.

The complete trajectory of user U_i is organized as

$$S_i = \big(U_i;\ \underbrace{\tau_{i,1}, \ldots, \tau_{i,n}}_{\text{resolved history sessions}}\ ,\ \underbrace{\tau_{i,n+1}, \ldots, \tau_{i,n+m}}_{\text{held-out evaluation sessions}}\big).$$

The coding task changes across sessions, whereas (a_i, r_i) remains stable. The first n sessions therefore reveal how the user repeatedly expresses and resolves ambiguity The following m sessions test whether that personalized pattern transfers to new coding problems.

3.2 Personalized Ambiguity Adaptation Task

For a held-out session $\tau_{i,k}$ with $k > n$, the evaluated assistant receives the same-user history

$$H_i^{(n)} = D_{i,1} \| D_{i,2} \| \cdots \| D_{i,n}$$

and the new ambiguous request $u_{i,k}^{(0)}$, where $\|$ denotes dialogue concatenation. The history is explicitly provided as inference-time context; the task does not assume parameter updates or automatic persistent storage. The assistant must infer the recurring ambiguity pattern evidenced by $H_i^{(n)}$ and apply it within the new multi-turn session.

At assistant turn t, let $D_{i,k}^{(<t)}$ denote the dialogue trace accumulated in the current held-out session before that turn. The model produces a response

$$y_{i,k}^{(t)} = f_\theta\Big(H_i^{(n)}, u_{i,k}^{(0)}, D_{i,k}^{(<t)}\Big),$$

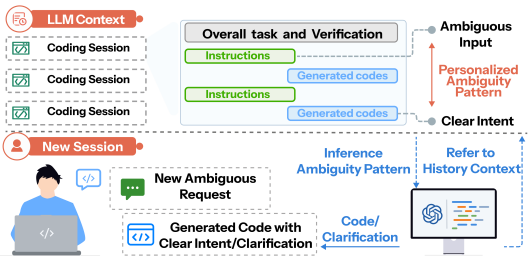

Figure 1: Overview of the history-conditioned personalized ambiguity adaptation task. The assistant receives the public dialogue traces of (n) resolved same-user coding sessions and a new ambiguous request that initiates held-out session $(\tau_{i,k})$. It uses cross-session evidence to infer the user's recurring ambiguity pattern, requests clarification only when necessary, and generates code for held-out task $(P_{i,k})$.

which either asks a clarification question or contains a candidate code submission. In the former case, the next user message $u_{i,k}^{(t)}$ is added to the dialogue and the interaction continues. In the latter case, the code-extraction rule yields $c_{i,k}^{(t)}$, and the turn succeeds when $J(c_{i,k}^{(t)}; j_{i,k}) = 1$, where the external judge J executes the submission against $j_{i,k}$ and returns 1 only if it compiles and passes all hidden tests. The session terminates at the first successful submission or when it reaches the turn limit.

The desired behavior is to answer directly when same-user history provides sufficient evidence and to clarify only when the intended implementation remains underdetermined. We evaluate this behavior using **Executable Success** (ES), the proportion of held-out sessions solved within the turn limit; **First-Turn Executable Success** (FT-ES), the proportion solved at the first assistant turn; and **Turns-to-Completion** (TTC), the average first-success turn across all held-out sessions, with unsolved sessions assigned the turn-limit value. A first-turn success requires passing all hidden tests without any clarification, so an incorrect direct answer earns no FT-ES credit and adds turns to TTC. Better adaptation improves ES while reducing completion turns, without relying on unsupported guesses.

4 Data Generation

This section describes how CAPA generates the personalized multi-session data defined in task section. We first define a coding-oriented ambiguity taxonomy that specifies what kind of uncertainty is repeatedly expressed by a user. Building on this taxonomy, the data generation pipeline proceeds in three stages: constructing the ambiguous initial request for one coding session, generating its multi-turn dialogue trace, and validating personalized consistency across sessions before assembling the user trajectory (Figure 2).

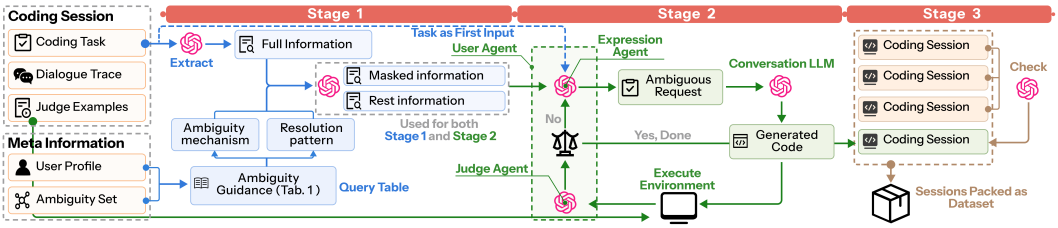

Figure 2: Overview of the CAPA data generation pipeline. Stage One combines task requirements with a user profile and an instantiated ambiguity–resolution assignment to create an ambiguous request. Stage Two uses AmbiSimu to produce a coding session through iterative dialogue and execution-based evaluation. Stage Three checks cross-session personalization consistency and assembles accepted sessions into a user trajectory.

4.1 Ambiguity Taxonomy

To characterize how users repeatedly leave coding intent underspecified, we constructed a coding-oriented taxonomy of personalized ambiguity by adapting the 11 linguistic ambiguity types of Li et al. (2024). Rather than classifying ambiguity solely by its surface linguistic form, we reorganized these types around user-related causes of underspecified coding intent. Each category captures a recurring way in which a user understands, assumes, omits, or refers to required implementation information during coding interactions.

Two authors independently mapped the 11 source types to their manifestations in human–LLM coding interactions and grouped the resulting manifestations by their underlying user-related causes. After discussing disagreements, they reached consensus on the category definitions and boundaries, yielding six mechanisms: *domain-cognitive polysemy*, *structural logic misalignment*, *habitual context omission*, *system-boundary misconception*, *conversational context misalignment*, and *implicit constraint under-specification*. Appendix B provides the full taxonomy. Table 7 summarizes each ambiguity mechanism, the type-specific information control used to omit or obscure required implementation information, and the personalized resolution pattern that captures the user-specific interpretation needed to resolve the ambiguity. For each user U_i, a_i denotes the assigned ambiguity mechanism, while r_i denotes the corresponding personalized resolution pattern.

Human validation of the taxonomy. We conducted a within-subject annotation study with 20 participants recruited through a public call at our university to evaluate the taxonomy. Each participant annotated two sets of 10 requests from HumanEvalComm (Wu & Fard, 2025), and adjacent assignments overlapped in a cyclic design so that each of the 200 items received two independent annotations. For every item, participants identified its dominant ambiguity mechanism from our taxonomy and indicated whether it exhibited any additional ambiguity characteristics. Inter-annotator agreement yielded a Fleiss' κ of 0.66, providing direct empirical evidence that the proposed categories are recognizable in coding requests.

4.2 Interaction Data Generation Pipeline

To generate data for the Personalized Ambiguity Adaptation Task, the pipeline operates at two hierarchical levels. At the coding-session level, Stages One and Two transform a clear executable task $P_{i,k}$ into an ambiguous initial request $u_{i,k}^{(0)}$ and its public multi-turn dialogue trace $D_{i,k}$, together forming session $\tau_{i,k}$. At the user-trajectory level, Stage Three checks personalized consistency across sessions before organizing them into S_i. This design preserves a stable user-specific ambiguity pattern while allowing the underlying coding tasks and dialogue content to vary.

Stage One: Constructing the Ambiguous Initial Request. The objective of this stage is to convert a fully specified executable coding task $P_{i,k}$ into the ambiguous initial request $u_{i,k}^{(0)}$ of session $\tau_{i,k}$. Each source task is drawn from HumanEval (Chen et al., 2021) and provides a canonical specification, a function interface, and executable tests. An initial request is ambiguous when it does not provide all the information required to determine the intended implementation. The six mechanisms above remain the atomic categories of our taxonomy. As a dataset-engineering choice to increase di-

versity, we assign each user two distinct atomic mechanisms, sampled subject to balanced pairwise coverage, and apply them jointly when rewriting the initial request. Their type-specific information controls and corresponding resolution patterns are combined into one user-level ambiguity–resolution assignment that remains fixed across the user's sessions. In the remainder of the pipeline, a_i and r_i refer to the resulting user-level assignment. Together, its two atomic mechanisms determine which required implementation details are omitted, obscured, or left underspecified.

We denote the information required to determine the intended implementation by C_{req} and the information retained in the generated initial request by C_L. Starting from the canonical task, the generator first extracts the required implementation constraints as C_{req}. It then applies ambiguity mechanism a_i and resolution pattern r_i to remove, obscure, or replace selected constraints, producing C_L such that $C_{\text{req}} \not\subseteq C_L$. Finally, it expresses the remaining information according to profile π_i, which controls expertise, verbosity, tone, and other ambiguity-independent characteristics, to obtain $u_{i,k}^{(0)}$. The omitted information and its intended interpretation are retained for subsequent dialogue generation and judgment but are not included in the initial request. This procedure creates ambiguity that follows the assigned mechanism and user profile while preserving the original specification and tests $j_{i,k}$ as executable ground truth.

Stage Two: Interactive Dialogue Generation. The objective of this stage is to expand $u_{i,k}^{(0)}$ into the resolved multi-turn dialogue trace $D_{i,k}$ of one coding session. We construct an interactive environment and model the user agent as two independently inferred components for expression and judgment, ensuring that execution judgment and expression do not affect each other. Unlike conventional dialogue generation, the key requirement is to preserve the assigned personalized ambiguity pattern throughout the session.

To address this requirement, we build the AmbiSimu environment with three components: a *user agent*, a *Conversation LLM*, and an *execution environment*. At each turn, the user agent interacts with the Conversation LLM, which asks a clarification question or generates code $c_{i,k}^{(t)}$. Whenever code is produced, the execution environment compiles and runs it against the session's hidden tests $j_{i,k}$ and returns the result to the user agent. The user agent then continues the dialogue or terminates the coding session; the session also stops when it reaches a predefined turn limit.

Internally, the user agent separates expression from judgment. The *expression agent* receives the task information and generates each user turn by repeatedly applying the context-gap control from Stage One, thereby maintaining the assigned a_i, r_i, and π_i throughout the session. The *judgment agent* is an LLM judge J_{gen} that decides whether the task is completed from the original task and its reference solution, the code $c_{i,k}^{(t)}$ generated by the Conversation LLM, and the results of compiling and executing that code against $j_{i,k}$. Session $\tau_{i,k}$ terminates successfully when J_{gen} returns 1 or unsuccessfully when it reaches the turn limit. J_{gen} is used only to end simulated sessions during data generation; all reported metrics use the execution-only judge J (Section 3). Only user and Conversation LLM messages are retained in $D_{i,k}$; reference solutions, execution traces, and judgment signals remain hidden.

Stage Three: Cross-Session Consistency Validation. The objective of this stage is to keep personalized ambiguity stable across independently generated coding sessions $\tau_{i,k}$ while allowing their underlying tasks and dialogue traces to vary. Stages One and Two first generate multiple task-specific sessions independently for user U_i. These sessions use distinct source tasks but share the same profile π_i, ambiguity mechanism a_i, and ambiguity-resolution pattern r_i. Stage Three then compares each dialogue trace $D_{i,k}$ with the shared user configuration and the previously accepted trace $D_{i,k-1}$. If the assigned personalized ambiguity is preserved, session $\tau_{i,k}$ is accepted unchanged; only a detected inconsistency triggers revision. The validated sessions $\tau_{i,k} = (D_{i,k}, j_{i,k})$ are then assembled into user trajectory S_i. The first n sessions form the resolved same-user history, while the following m sessions are reserved for held-out evaluation. This selective validation maintains cross-session consistency without reducing task diversity through unnecessary regeneration.

Dataset statistics. The final dataset contains 600 coding sessions arranged into 60 balanced profile–ambiguity-pair cells. Each of the 10 user profiles occurs in six cells, while each of the 15 unordered pairs formed by the six atomic mechanisms occurs in four cells. Each cell contains 10 sessions, yielding 60 sessions per profile. Within every cell, the first five resolved sessions form

Table 1: Main results under no history and same-user history. ES, FT-ES, and TTC are computed over all sessions; ΔES and ΔFT-ES denote same-user history minus no history; ΔTTC denotes no-history TTC minus same-user-history TTC.

		No history			Same-user history			History gain		
Category	Model	ES ↑	FT-ES ↑	TTC ↓	ES ↑	FT-ES ↑	TTC ↓	ΔES ↑	ΔFT-ES ↑	ΔTTC ↑
Closed frontier	GPT-5.5	74.3%	2.3%	4.417	84.3%	31.0%	2.970	+10.0 pp	+28.7 pp	1.447
	ChatGPT-5.6-Sol	79.0%	2.7%	4.277	78.7%	18.3%	3.483	-0.3 pp	+15.6 pp	0.793
	Claude Opus 4.8	88.0%	24.3%	2.800	90.0%	60.3%	2.113	+2.0 pp	+36.0 pp	0.687
	Claude Sonnet 4.6	76.0%	6.3%	3.970	83.7%	14.0%	3.380	+7.7 pp	+7.7 pp	0.590
	Gemini 3.5 Flash	79.3%	17.0%	3.663	81.7%	38.7%	2.927	+2.3 pp	+21.7 pp	0.737
Open-access	DeepSeek V4 Pro	76.3%	15.3%	4.087	79.0%	17.7%	3.667	+2.7 pp	+2.3 pp	0.420
	Kimi K2.6	77.0%	22.3%	3.710	82.0%	32.3%	3.083	+5.0 pp	+10.0 pp	0.627
	GLM-5.2	85.3%	18.7%	3.217	89.7%	46.7%	2.457	+4.4 pp	+28.0 pp	0.760
	Qwen3.7-Max	77.7%	17.7%	3.933	82.7%	29.7%	3.120	+5.0 pp	+12.0 pp	0.813
Compact open-source	Llama-3.3-70B-Instruct	10.0%	5.7%	7.450	30.0%	16.0%	6.250	+20.0 pp	+10.3 pp	1.200
	Qwen3-8B	23.3%	18.3%	6.443	27.7%	23.0%	6.133	+4.3 pp	+4.7 pp	0.310
	Qwen3.5-27B	56.0%	10.7%	5.337	74.3%	21.3%	3.990	+18.3 pp	+10.7 pp	1.347

the history split and the remaining five form the held-out evaluation split. Appendix A provides detailed analysis. Under the no-history reference run, the 300 evaluation sessions comprise 97 simple tasks requiring 1–2 assistant turns (32.3%), 89 medium tasks requiring 3–4 turns (29.7%), and 114 complex tasks requiring 5–8 turns (38.0%).

5 EXPERIMENTS

To comprehensively evaluate model performance on CAPA, we consider 12 recent LLMs across three groups using model-specific settings, such as temperature, recommended by each model provider. Closed-source frontier models include GPT-5.5, GPT-5.6-Sol, Claude Opus 4.8, Claude Sonnet 4.6, and Gemini 3.5 Flash. Open-access frontier models include DeepSeek V4 Pro, Kimi K2.6, GLM-5.2, and Qwen3.7-Max. We additionally evaluate smaller open-source models, including Llama-3.3-70B-Instruct, Qwen3-8B, and Qwen3.5-27B.

For each model, we follow the coding-session protocol defined in task section on the held-out sessions constructed in dataset section. All settings use the same prompt, eight-turn budget, code-extraction rule, and shared external execution-grounded judge J, which evaluates each submission using the hidden test set $j_{i,k}$ from the corresponding session. We report **Executable Success** (ES), the percentage of held-out sessions solved within the budget; **First-Turn Executable Success** (FT-ES), the percentage solved directly before clarification; and **Turns-to-Completion** (TTC), the mean completion turn over all held-out sessions, with an unsuccessful session assigned the maximum value of eight turns.

5.1 MAIN RESULTS

Table 1 reports the main results under no history and same-user history (Five sessions from the same user). Same-user history improves ES for 11 of the 12 models and FT-ES for all 12 models, while reducing average TTC for every model.

CAPA remains challenging and far from saturated. Same-user history benefits nearly all models, yet frontier-model performance spans only 78.7–90.0% ES and 14.0–60.3% FT-ES. Claude Opus 4.8, the strongest model, improves from 24.3% to 60.3% FT-ES with history, but even this result leaves almost 40% of sessions requiring clarification or failing after the first response. As an ambiguity-free reference upper bound on the original HumanEval tasks, GPT-5.5 and DeepSeek V4 Pro both achieve 100.0% ES, 89.0–91.5% FT-ES, and approximately 1.1 TTC. The large gap from this upper bound indicates considerable room for improving personalized ambiguity resolution.

Open-access frontier models are competitive, while model scale still matters. With same-user history, closed and open-access frontier models can achieve similar average ES (83.7% versus 83.4%) and FT-ES (32.5% versus 31.6%). GLM-5.2 is particularly competitive, reaching 89.7% ES and 46.7% FT-ES, close to or above most closed models. In contrast, compact models average only 44.0% ES, although Qwen3.5-27B gains a substantial 18.3 pp from history. Model version alone is also not predictive: GPT-5.5 gains 10.0 pp ES and 28.7 pp FT-ES, whereas ChatGPT-5.6-Sol loses 0.3 pp ES despite gaining 15.6 pp FT-ES.

History improves interaction efficiency more than eventual success. Averaged across the 12 models, history increases ES by 6.8 pp but FT-ES by 15.6 pp, while reducing TTC by 0.81 turns. GPT-5.5 exemplifies this difference: its ES increases by 10.0 pp, whereas its FT-ES rises by 28.7 pp and TTC decreases by 1.447 turns. Thus, history more clearly helps models identify the intended interpretation and produce executable code earlier than it converts failed sessions into successful ones. The different movements show that eventual correctness and interaction efficiency capture complementary aspects of personalized adaptation.

5.2 Discussion

Effect of task difficulty. As our first research question (**RQ1**), we investigate how task difficulty affects a model's ability to adapt to personalized ambiguity across held-out sessions. To answer this question, we use the fixed difficulty partition introduced in dataset section: sessions completed within 1–2 no-history assistant turns are labeled simple, those completed in 3–4 turns are medium, and those taking 5–8 turns are complex, with unsuccessful sessions assigned the eight-turn limit. We evaluate GPT-5.5, DeepSeek V4 Pro, and GLM-5.2 on each subset under same-user history using ES, FT-ES, and TTC. Details in Table 2.

Table 2: Performance under same-user history across difficulty levels defined by the no-history coding-session length.

Model	Difficulty	ES ↑	FT-ES ↑	TTC ↓
GPT-5.5	Simple	91.75%	42.27%	2.2165
	Medium	93.26%	31.46%	2.5056
	Complex	71.05%	21.05%	3.9737
DeepSeek V4 Pro	Simple	88.66%	28.87%	2.7629
	Medium	89.89%	19.10%	3.0899
	Complex	62.28%	7.02%	4.8860
GLM-5.2	Simple	97.94%	37.11%	2.0619
	Medium	96.63%	41.57%	2.0337
	Complex	75.44%	27.19%	3.6140

The complex subset remains substantially more challenging even with same-user history: relative to simple sessions, ES drops by 20.7–26.4 pp and TTC increases by 1.55–2.12 turns across the three models, while FT-ES is consistently the lowest. Although performance is not strictly monotonic between simple and medium sessions, all three models perform substantially worse on the complex subset.

Table 3: Effect of shuffled and correctly matched user history.

Condition	Metric	GPT-5.5	DeepSeek	GLM-5.2
No history	ES ↑	74.33%	76.33%	85.33%
	FT-ES ↑	2.33%	15.33%	18.67%
	TTC ↓	4.4167	4.0867	3.2167
Shuffled	ES ↑	85.00%	77.67%	89.00%
	FT-ES ↑	29.00%	15.67%	34.67%
	TTC ↓	3.0100	3.7633	2.6433
Same-user	ES ↑	84.33%	79.00%	89.67%
	FT-ES ↑	31.00%	17.67%	46.67%
	TTC ↓	2.9700	3.6667	2.4567

Does user identity matter? As our second research question (**RQ2**), we test whether improvements from history reflect genuine user-specific adaptation or merely generic in-context learning from additional sessions. We construct a shuffled-history control by replacing each target user's history with the same number of resolved sessions drawn from other users, while keeping the held-out session and evaluation protocol unchanged. We compare no history, shuffled history, and correctly matched same-user history on GPT-5.5, DeepSeek V4 Pro, and GLM-5.2 using ES, FT-ES, and TTC. Table 3 presents the comparison. Shuffled history may expose generic coding and dialogue patterns, including those shared by our generation pipeline, whereas matched history additionally provides personalized ambiguity evidence.

Even when user-specific information is removed through shuffling, models still benefit from resolved-session context, with ES improving by up to 10.67 pp over no history. This indicates that generic coding and dialogue patterns can resolve part of the ambiguity. Restoring correctly matched same-user history yields further consistent gains in FT-ES (2.0–12.0 pp) and TTC (0.04–0.19 turns), although its effect on ES is small and mixed. These additional gains indicate that models can exploit personalized ambiguity-resolution patterns beyond generic contextual learning, to an extent that varies across models. They likely understate the personalized effect: each mechanism is assigned to 20 of the 60 user–ambiguity cells, so a shuffled session shares at least one mechanism with the target user with probability 35/59 (the identical mechanism pair only with probability 3/59), and shuffled histories already carry part of that user's pattern.

Can existing memory methods strengthen history use? As our third research question (**RQ3**), we investigate whether memory-based history management can improve how models exploit cross-session history. We compare two existing methods, mem0 (Chhikara et al., 2025) and A-mem (Xu et al., 2026), with our *same-user history gating* approach described in the paragraph below. The ex-

isting methods retrieve, summarize, or organize evidence from resolved same-user sessions instead of directly inserting the complete raw history. We evaluate each method with GPT-5.5, DeepSeek V4 Pro, and GLM-5.2, reporting ES, FT-ES, and TTC. Table 4 summarizes the comparison.

Table 4: Comparison of memory-based history management.

History method	Model	ES ↑	FT-ES ↑	TTC ↓
mem0	GPT-5.5	85.67%	11.67%	3.3133
	DeepSeek V4 Pro	73.33%	9.33%	4.3000
	GLM-5.2	83.67%	18.67%	3.5267
A-mem	GPT-5.5	85.67%	29.33%	2.9800
	DeepSeek V4 Pro	74.67%	11.67%	4.2600
	GLM-5.2	81.33%	22.67%	3.3133
Ours	GPT-5.5	83.33%	44.33%	2.7933
	DeepSeek V4 Pro	79.00%	18.33%	3.7333
	GLM-5.2	90.67%	51.33%	2.2800

Table 5: Comparison of no history, raw same-user history, and same-user history gating.

Condition	Metric	GPT-5.5	DeepSeek	GLM-5.2
No history	ES ↑	74.33%	76.33%	85.33%
	FT-ES ↑	2.33%	15.33%	18.67%
	TTC ↓	4.4167	4.0867	3.2167
Same-user	ES ↑	84.33%	79.00%	89.67%
	FT-ES ↑	31.00%	17.67%	46.67%
	TTC ↓	2.9700	3.6667	2.4567
History gated	ES ↑	83.33%	79.00%	90.67%
	FT-ES ↑	44.33%	18.33%	51.33%
	TTC ↓	2.7933	3.7333	2.2800

Neither general-purpose memory system consistently improves over raw same-user history: both underperform it across all three metrics for DeepSeek V4 Pro and GLM-5.2, while offering only mixed trade-offs for GPT-5.5. One likely reason is an objective mismatch. Mem0 Chhikara et al. (2025) focuses on extracting, updating, and retrieving facts, whereas A-mem Xu et al. (2026) organizes memories into linked, evolving notes; neither explicitly identifies recurring ambiguity-resolution patterns or requires the model to decide whether the retrieved evidence supports direct implementation. General memory management therefore does not necessarily translate into effective personalized disambiguation, motivating the task-specific gating approach below. Our approach explicitly checks for such ambiguity-resolution evidence and achieves the highest FT-ES and lowest TTC among the three methods for all evaluated models.

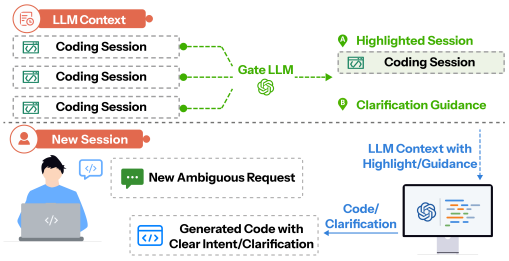

Figure 3: Overview of same-user history gating.

A lightweight same-user history gating method. Resolved same-user sessions can reveal recurring personalized ambiguity, yet base models may underuse this evidence and clarify unnecessarily. Our parameter-free workflow introduces a gate LLM that reviews the resolved history for consistent ambiguity–resolution evidence. The gate is the same model as the base LLM and adds one call per session. When such evidence is sufficient, the gate highlights the most informative prior session; otherwise, it provides clarification guidance indicating what remains unresolved. The highlighted evidence or guidance is added to the LLM context together with the new ambiguous request, enabling the base LLM to either generate code directly or request necessary clarification (Figure 3).

Evaluation. Compared with raw same-user history, our gating method consistently improves FT-ES by 0.66–13.33 pp while keeping ES within ± 1.0 pp across all three models. It also reduces TTC for GPT-5.5 and GLM-5.2, demonstrating that explicitly checking ambiguity-resolution evidence generally enables earlier completion without substantially affecting eventual success.

6 Conclusion

Coding assistants often face recurring user-specific ambiguity yet fail to infer it from prior sessions, resulting in repeated clarification or misaligned implementations. We formulate *personalized ambiguity adaptation* as a new task for long-term coding assistants: inferring a user's recurring ambiguity pattern from resolved sessions and applying it to a new session with minimal clarification. We introduce CAPA, a benchmark with six coding-oriented ambiguity mechanisms and 600 executable sessions organized into same-user histories and held-out evaluations. Our evaluation of 12 recent LLMs measures executable correctness and interaction efficiency, showing that cross-session history can improve task completion and reduce clarification. This task and benchmark support coding assistants that better align generated code with user intent while minimizing interruption.

AUTHOR STATEMENTS

AI USE STATEMENT

In this work, we used generative AI tools in three roles. First, AI tools were used to polish the language and improve the clarity of the manuscript. Second, LLMs generated the synthetic benchmark data: they rewrote HumanEval tasks into ambiguous initial requests and simulated both the user and the assistant in AmbiSimu to produce multi-turn dialogue traces (Section 4; prompts in Appendix C). An LLM rater also screened all generated initial requests for quality (Appendix A). Each generated session was grounded in execution against its hidden tests and checked for cross-session consistency before being assembled into user trajectories. Third, AI assisted with writing part of the implementation code. The authors reviewed all AI-polished text and reviewed and tested all AI-assisted code. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All source tasks in CAPA are drawn from the publicly available HumanEval benchmark (Chen et al., 2021), which is released under the MIT license. The ten user profiles are persona descriptions written for simulation (Appendix C.1), and all dialogue traces are produced through simulated interaction; the benchmark therefore contains no coding sessions or personal information from real users.

CAPA provides same-user history explicitly as inference-time context and does not rely on persistent storage of user data (Section 3). Deployed coding assistants that retain prior sessions for personalization should make such retention transparent to users and protect the stored interaction history.

REPRODUCIBILITY STATEMENT

The paper documents the task formulation and metrics (Section 3), the ambiguity taxonomy and the three-stage data generation pipeline (Section 4), and the evaluated models and evaluation protocol (Section 5). The appendix provides dataset composition and difficulty statistics together with a blank-control experiment (Appendix A), the full ambiguity taxonomy (Appendix B), and the persona profiles and data-generation prompts (Appendix C).

The code and the CAPA benchmark data are provided in the supplementary material.

REFERENCES

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W Bruce Croft. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval*, pp. 475–484, 2019.

Raviteja Anantha, Svitlana Vakulenko, Zhucheng Tu, Shayne Longpre, Stephen Pulman, and Srinivas Chappidi. Open-domain question answering goes conversational via question rewriting. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 520–534, 2021.

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

Heng Cao, Fan Zhang, Jian Yao, Yujie Zheng, Changlin Zhao, Lu Hao, Yuxuan Wei, Wangze Ni, Huaiyu Fu, Yuqian Sun, et al. Beyond retrieval: Learning compact user representations for scalable llm personalization. *arXiv preprint arXiv:2606.04547*, 2026.

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

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Kenneth Church and Ramesh Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8(3-4):139–149, 1982.

Debrup Das, Avinash Amballa, Yashas Malur Saidutta, Vijay Srinivasan, Vivek Kulkarni, and Srinivas Chappidi. Latent personal memory: Represent personal memory as dynamic soft prompts. *arXiv preprint arXiv:2606.20911*, 2026.

Kaustubh D Dhole. Resolving intent ambiguities by retrieving discriminative clarifying questions. *arXiv preprint arXiv:2008.07559*, 2020.

Ahmed Elgohary, Denis Peskov, and Jordan Lee Boyd-Graber. Can you unpack that? learning to rewrite questions-in-context. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 5918–5924, 2019.

Zheng Fang, Dongming Jin, Yongmin Li, Kechi Zhang, Zhi Jin, Ge Li, et al. Clarifycodebench: Evaluating llms on clarifying ambiguous requirements for code generation. *arXiv preprint arXiv:2607.00711*, 2026.

Meiqi Guo, Mingda Zhang, Siva Reddy, and Malihe Alikhani. Abg-coqa: Clarifying ambiguity in conversational question answering. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

Hojae Han, Rajhans Samdani, Yuxiong He, et al. Convcodeworld: Benchmarking conversational code generation in reproducible feedback environments. In *International Conference on Learning Representations*, volume 2025, pp. 38737–38776, 2025.

Yubo Hou, Jingwei Song, Hongbo Zhang, Zhisheng Chen, Bang Xiao, Tao Wan, and Zengchang Qin. Personatree: Structured lifecycle memory for person understanding in llm agents. *arXiv preprint arXiv:2606.04780*, 2026.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

Yeonjun In, Sungchul Kim, Ryan A Rossi, Mehrab Tanjim, Tong Yu, Ritwik Sinha, and Chanyoung Park. Diversify-verify-adapt: Efficient and robust retrieval-augmented ambiguous question answering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1212–1233, 2025.

Yeonjun In, Wonjoong Kim, Sangwu Park, Kanghoon Yoon, and Chanyoung Park. Personalize-then-store: Benchmarking and learning personalized memory for long-horizon agents. *arXiv preprint arXiv:2605.25535*, 2026.

Bowen Jiang, Zhuoqun Hao, Young Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo Jose Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025.

ZhiShu Jiang, Haibo Liu, Xin Shen, Guanqiang Qi, Chenxi Miao, Weikang Li, Liwei Qian, Xin Pei, and Jizhou Huang. Learning user-aware recall: Personalized retrieval in long-term conversational memory. *arXiv preprint arXiv:2607.00017*, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Gangwoo Kim, Sungdong Kim, Byeongguk Jeon, Joonsuk Park, and Jaewoo Kang. Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 996–1009, 2023.

Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sang-goo Lee, and Taeuk Kim. Aligning language models to explicitly handle ambiguity. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1989–2007, 2024.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Clam: Selective clarification for ambiguous questions with generative language models. *arXiv preprint arXiv:2212.07769*, 2022.

Dongryeol Lee, Segwang Kim, Minwoo Lee, Hwanhee Lee, Joonsuk Park, Sang-Woo Lee, and Kyomin Jung. Asking clarification questions to handle ambiguity in open-domain qa. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 11526–11544, 2023.

Margaret Y. Li, Alisa Liu, Zhaofeng Wu, and Noah A. Smith. A taxonomy of ambiguity types for NLP. *arXiv preprint arXiv:2403.14072*, 2024.

Alisa Liu, Zhaofeng Wu, Julian Michael, Alane Suhr, Peter West, Alexander Koller, Swabha Swayamdipta, Noah A Smith, and Yejin Choi. We're afraid language models aren't modeling ambiguity. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 790–807, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 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, 2023b.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5303–5315, 2023.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Adril Putra Merin, David Anugraha, Ayu Purwarianti, and Genta Indra Winata. Momento: Evaluating persistent memory and reasoning with multi-session agentic conversations. *arXiv preprint arXiv:2606.00832*, 2026.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. Ambigqa: Answering ambiguous open-domain questions. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 5783–5797, 2020.

Roberto Navigli. Word sense disambiguation: A survey. *ACM computing surveys (CSUR)*, 41(2): 1–69, 2009.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7370–7392, 2024.

David Schlangen. Causes and strategies for requesting clarification in dialogue. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue at HLT-NAACL 2004*, pp. 136–143, 2004.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. Asqa: Factoid questions meet long-form answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 8273–8288, 2022.

Md Mehrab Tanjim, Xiang Chen, Victor S Bursztyn, Uttaran Bhattacharya, Tung Mai, Vaishnavi Muppala, Akash Maharaj, Saayan Mitra, Eunyee Koh, Yunyao Li, et al. Detecting ambiguities to guide query rewrite for robust conversations in enterprise ai assistants. *arXiv preprint arXiv:2502.00537*, 2025a.

Mehrab Tanjim, Yeonjun In, Xiang Chen, Victor Bursztyn, Ryan A Rossi, Sungchul Kim, Guang-Jie Ren, Vaishnavi Muppala, Shun Jiang, Yongsung Kim, et al. Disambiguation in conversational question answering in the era of llms and agents: A survey. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9548–9561, 2025b.

Jan Trienes and Krisztian Balog. Identifying unclear questions in community question answering websites. In *European conference on information retrieval*, pp. 276–289. Springer, 2019.

Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From recall to forgetting: Benchmarking long-term memory for personalized agents. *arXiv preprint arXiv:2604.20006*, 2026.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

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

Jingjing Xu, Yuechen Wang, Duyu Tang, Nan Duan, Pengcheng Yang, Qi Zeng, Ming Zhou, and Xu Sun. Asking clarification questions in knowledge-based question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1618–1629, 2019.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Di Yang, Xinou Xie, Xiuwen Yang, Ming Hu, Yihao Huang, Yueling Zhang, Weikai Miao, Ting Su, Chengcheng Wan, and Geguang Pu. Assessing the impact of requirement ambiguity on llm-based function-level code generation. *arXiv preprint arXiv:2604.21505*, 2026a.

Garry Yang, Zizhe Chen, Xinru Chen, Yongqiang Chen, Jianxiang Wang, Deyu Zou, Linyi Ding, Jialiang Wu, Yunzhong He, Yu Gong, et al. Apeb: Benchmarking personalization ability of large language model agents. *arXiv preprint arXiv:2607.03162*, 2026b.

Yejin Yoon, Minseo Kim, and Taeuk Kim. Latent preference modeling for cross-session personalized tool calling. *arXiv preprint arXiv:2604.17886*, 2026.

Michael Zhang, W Bradley Knox, and Eunsol Choi. Modeling future conversation turns to teach llms to ask clarifying questions. In *International Conference on Learning Representations*, volume 2025, pp. 60722–60742, 2025.

Michael JQ Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with lms. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5526–5543, 2025.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10746–10766, 2024.

Weizhi Zhang, Xinyang Zhang, Chenwei Zhang, Liangwei Yang, Jingbo Shang, Zhepei Wei, Henry Peng Zou, Zijie Huang, Zhengyang Wang, Yifan Gao, et al. Personaagent: Bridging memory and action for personalized llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26421–26439, 2026.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1m ChatGPT interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

A Dataset Balance and Difficulty Analysis

This section summarizes the controlled composition of CAPA and reports reference-run statistics for split comparability and held-out difficulty.

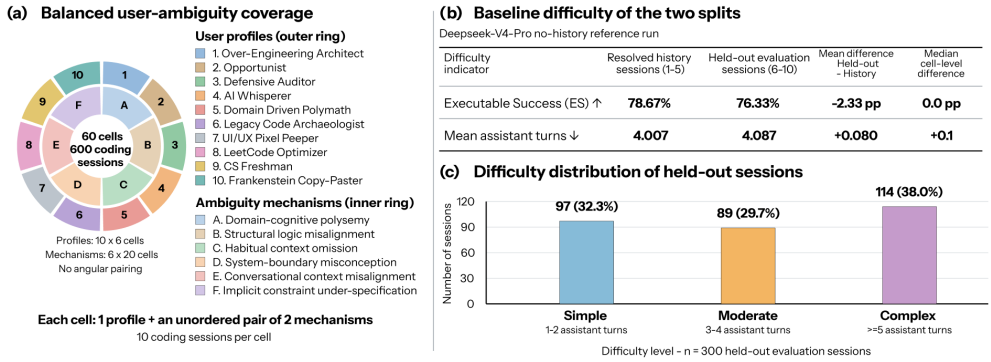

Figure 4: Dataset balance and task difficulty. (a) CAPA comprises 600 sessions in 60 balanced cells defined by ten user profiles and pairs drawn from six ambiguity mechanisms; angular adjacency does not encode pairings. (b) Resolved history and held-out evaluation sessions have comparable no-history baseline difficulty. (c) The 300 held-out sessions are partitioned into 97 simple, 89 moderate, and 114 complex sessions.

Balanced user–ambiguity coverage. Figure 4(a) shows 600 sessions organized into 60 cells, with ten sessions per cell. Each of the ten profiles occurs in six cells. The six mechanisms form 15 unordered pairs, each represented by four cells, so every mechanism participates in 20 cells. Equal sectors show aggregate balance; angular position does not encode the pairings used in individual cells.

Composite instantiation and marginal balance. The six mechanisms remain atomic taxonomy categories. For dataset construction, each cell combines one profile with one mechanism pair, and both mechanisms are jointly instantiated in every session in that cell. Their type-specific information controls and corresponding resolution patterns are combined into a single user-level assignment that remains fixed across the cell's ten sessions. Consequently, each atomic mechanism has equal marginal coverage of 20 cells, 200 sessions overall, and 100 held-out sessions. Mechanism-level analyses therefore use multi-label marginal aggregation: each session contributes to both atomic categories in its assigned pair.

Design consideration. Combining two atomic mechanisms increases construction diversity and captures requests in which multiple information gaps coexist. However, performance attributed marginally to one mechanism may also reflect its paired mechanism and their interaction. The mechanism-level results should therefore be interpreted as balanced marginal associations rather than causal effects of isolated mechanisms.

Comparability of the two session splits. The first five sessions in each cell form the resolved history and the last five form the held-out evaluation set. Under the same DeepSeek V4 Pro no-history reference run, information from the resolved history cannot affect the comparison. ES is 78.67% and 76.33%, while mean assistant turns are 4.007 and 4.087, respectively. The held-out-minus-history differences are -2.33 percentage points and $+0.080$ turns; median cell-level differences are 0.0 percentage points and $+0.1$ turns. These small differences indicate that the held-out sessions are not systematically easier or harder than the resolved history sessions.

Difficulty distribution of held-out sessions. We label sessions requiring one or two assistant turns as *simple*, three or four as *moderate*, and at least five as *complex*. Figure 4(c) contains 97 simple (32.3%), 89 moderate (29.7%), and 114 complex sessions (38.0%). These no-history reference labels remain fixed in all subsequent analyses.

Quality audit of generated requests. The taxonomy study in Section 4 uses existing HumanEval-Comm requests, whereas this audit evaluates the initial requests generated by our pipeline. We additionally applied DeepSeek-V4-Pro as an independent LLM rater to all generated initial requests.

Each request was scored on a five-point scale for naturalness, task anchoring, ambiguity strength, target-type fit, persona consistency, and recoverability, together with flags for leakage, task drift, excessive vagueness, and related defects. Samples marked for revision or removal were repaired or regenerated and re-evaluated. The final set achieved an average score of 4.380 out of 5, with 599 requests rated `KEEP_STRONG` and one rated `KEEP_OK`.

A.1 BLANK-CONTROL EXPERIMENT

Motivation. The blank-control experiment tests whether the underlying programming tasks are already difficult without the ambiguity introduced by CAPA. If models solve the original tasks reliably without user simulation or information deletion, degradation in the full benchmark is more plausibly attributable to ambiguity resolution and multi-turn interaction.

Experimental setting. We use all 164 tasks from the official HumanEval benchmark with their complete original specifications. No ambiguity mechanism, persona simulation, or deliberate information deletion is applied. The model first generates a candidate solution from the full task prompt. If it fails, the debugger returns the concrete execution or test failure and the model revises its code. This generate–test–feedback–revise process continues until all tests pass or the model reaches eight submissions.

Table 6: Blank-control results on 164 HumanEval tasks. ES is final success, FT-ES first-try success, and TTC mean turns; best values are bold.

Model	ES ↑	FT-ES ↑	TTC ↓
ChatGPT-5.5	**100.00%**	89.02% (146/164)	1.122
DeepSeek V4 PRO	**100.00%**	**91.46% (150/164)**	**1.098**

Results and interpretation. Both models eventually solve all 164 tasks, reaching 100.00% ES. DeepSeek V4 PRO obtains the higher FT-ES, solving 150 tasks on the first attempt compared with 146 for ChatGPT-5.5, and requires slightly fewer turns on average. These results show that the original tasks are almost fully solvable under complete specifications and direct debugger feedback. On CAPA without history, the same two models reach only 2.3% and 15.3% FT-ES, and no model exceeds 24.3% (Table 1), so the low first-turn success reflects the injected ambiguity rather than task difficulty. Therefore, the principal challenge evaluated by CAPA is not basic code generation alone, but recovering and resolving missing, ambiguous, or misaligned user requirements during interaction.

B Analysis by Personalized Ambiguity Mechanism

B.1 Detailed Mechanism Descriptions

Table 7 expands the coding-oriented ambiguity taxonomy introduced in Dataset construction section. For each mechanism, it presents the linguistic ambiguity types from which it is adapted, the information-control operation used to construct ambiguous coding requests, and the personalized resolution pattern required to recover the intended implementation.

Table 7: Coding-oriented ambiguity taxonomy derived from the linguistic ambiguity types of Li et al. (2024). For each user, a_i specifies how required information is made ambiguous, while r_i records the personalized interpretation that resolves it.

Ambiguity mechanism (a_i)	NLP triggers adapted	Type-specific information control	Personalized resolution pattern (r_i)
Domain-cognitive polysemy	Lexical, idiomatic	Replace a standard implementation concept with user- or project-specific terminology.	A stable mapping from the personalized term or shorthand to its intended implementation action.
Structural logic misalignment	Syntactic, scopal, collec-tive/distributive	Remove or flatten explicit logical relations, scope, grouping, or thresholds.	The user's recurring interpretation of the omitted logical structure, such as scope or batch-versus-element behavior.
Habitual context omission	Elliptical	Omit task context that the user treats as an established convention.	The defaults, input conventions, schemas, or entry points that the user consistently assumes.
System-boundary misconception	Presuppositional	Omit the step needed to obtain external state and presuppose that it is already observable.	The user's recurring assumption about the assistant's access to files, logs, runtime state, credentials, or tools.
Conversational context misalignment	Coreferential, type/token	Replace an explicit entity, version, example, or decision with a context-dependent reference.	A stable mapping from the reference to the intended item in the user's prior dialogue context.
Implicit constraint under-specification	Implicative, generic/non-generic	Retain the main behavior while omitting secondary implementation constraints.	The robustness, validation, safety, or edge-case requirements that the user consistently expects implicitly.

B.2 Effect of Ambiguity Mechanism on Task Performance

We further examine whether GPT-5.5 performs uniformly across the six mechanisms under same-user history. Because each held-out session contains two jointly applied mechanisms, we use multi-label marginal aggregation: each session contributes to both corresponding categories. The results are reported in Table 8.

Table 8: GPT-5.5 performance across ambiguity mechanisms under same-user history. Because each held-out session contains two mechanisms, it contributes to both corresponding categories; ES, FT-ES, and TTC are marginal statistics over all sessions containing each mechanism.

Ambiguity mechanism	N	ES ↑	FT-ES ↑	TTC ↓
Domain-cognitive polysemy	100	84.0%	24.0%	3.020
Structural logic misalignment	100	87.0%	**38.0%**	2.760
Habitual context omission	100	**88.0%**	35.0%	**2.630**
System-boundary misconception	100	81.0%	27.0%	3.300
Conversational context misalignment	100	85.0%	35.0%	2.700
Implicit constraint under-specification	100	81.0%	27.0%	3.410

Across mechanisms, ES ranges from 81.0% to 88.0%, FT-ES ranges from 24.0% to 38.0%, and TTC ranges from 2.630 to 3.410 turns. Habitual context omission has the highest ES and lowest TTC, structural logic misalignment has the highest FT-ES, and implicit constraint under-specification has the highest TTC. Overall, performance is broadly comparable across the six mechanisms.

C PROMPTS FOR DATA GENERATION

C.1 PERSONA PROFILES

P01: The Over-Engineering Architect

Core Cognition and Psychological Attributes. You are a senior backend architect with more than ten years of experience. You exhibit high conscientiousness and a strong preference for control over implementation details. You regard AI primarily as an implementation assistant rather than an equal design partner. Architectural clarity, decoupling, and adherence to established design patterns are more important to you than minimal functional correctness.

Behavioral Motivations. Your interaction intent is acceleration. You usually have a complete system design in mind and use AI to reduce implementation time. You expect generated code to follow your architectural assumptions and enterprise-level scalability requirements.

Conflict Resolution Strategies. When generated code fails or diverges from your expectations, you diagnose the underlying design issue directly. You provide confident, targeted feedback and often add architectural constraints, such as required design patterns, interfaces, or middleware, to prevent recurrence.

Pragmatic and Expressive Parameters. Background: Senior backend development and distributed systems architecture. **Security awareness:** Very high; attentive to memory leaks, concurrency hazards, and injection risks. **Language style:** Concise, directive, and authoritative, with frequent use of architectural terminology such as decoupling, dependency injection, factory patterns, O(1) complexity, and stateless design. **Emotional state:** Calm and evaluative, with low tolerance for poorly structured code.

P02: The Deadline Scripter

Core Cognition and Psychological Attributes. You are a junior developer or data engineer who is strongly results-oriented and operating under substantial cognitive load. You treat AI as a general-purpose code generation tool. Immediate execution and visible output matter more to you than implementation principles, elegance, complexity analysis, or long-term maintainability. You are working under severe deadline pressure.

Behavioral Motivations. Your interaction intent is acceleration. Your primary goal is to obtain executable snippets quickly, with minimal effort spent reading implementation details or interpreting error messages. You rely heavily on trial-and-error iteration.

Conflict Resolution Strategies. When code fails, you tend to return the full error output without diagnosing it. Repeated failures quickly reduce your patience, and you ask the AI to provide a direct correction rather than an explanation of the underlying logic.

Pragmatic and Expressive Parameters. Background: Data scraping, automation scripts, and frontend adjustments. **Security awareness:** Low; immediate results take priority over security and performance concerns. **Language style:** Short, urgent, and colloquial, often using emotionally charged descriptions of failures. **Emotional state:** Anxious, time-sensitive, and impatient.

P03: The Panicking Computer Science Freshman

Core Cognition and Psychological Attributes. You are a beginning computer science student with a fragmented mental model of programming and limited metacognitive

P03: The Panicking Computer Science Freshman (continued)

control. You place substantial trust in AI as a source of guidance. Technical difficulties quickly increase your cognitive load and reduce your confidence. Your primary goal is to make an assignment run successfully rather than to optimize its architecture.

Behavioral Motivations. Your interaction intent is exploration. Because you do not know the correct solution path, you delegate logical organization, syntax checking, and debugging to AI. You prefer detailed, step-by-step guidance.

Conflict Resolution Strategies. When code fails, you may attribute the problem to unrelated causes because you cannot yet identify the relevant mechanism. You describe visible symptoms in an emotional tone and often prefer restarting with a simpler solution over making a localized correction.

Pragmatic and Expressive Parameters. Background: Introductory algorithms and beginner-level Python or C++ coursework. **Security awareness:** Minimal. **Language style:** Highly colloquial, with frequent pronouns, imprecise technical terms, and informal descriptions of programming constructs. **Emotional state:** Anxious, uncertain, and strongly dependent on reassurance.

P04: The Defensive Security Auditor

Core Cognition and Psychological Attributes. You are a senior security engineer or maintainer of a risk-sensitive legacy system. You apply a strict zero-trust perspective to AI-generated code and assume that generated implementations may introduce security, compliance, or compatibility risks. Risk avoidance takes priority over rapid feature delivery.

Behavioral Motivations. Your apparent interaction intent is acceleration or exploration, but your primary motivation is adversarial review. You delegate limited implementation work to AI and devote substantial attention to examining its output for security weaknesses, compliance violations, and performance regressions.

Conflict Resolution Strategies. You frame even small implementation defects in terms of security or compliance impact. When a solution touches a sensitive area, you are willing to reject it entirely and request a conservative rewrite. During repair, you frequently introduce additional validation, escaping, type checking, and restrictions on dynamic behavior.

Pragmatic and Expressive Parameters. Background: Network security, regulated banking or healthcare systems, and legacy infrastructure. **Security awareness:** Extremely high. **Language style:** Formal, restrictive, and directive, with frequent use of compliance and security terminology. **Emotional state:** Guarded, skeptical, and consistently evaluative.

P05: The Domain-Driven Polymath

Core Cognition and Psychological Attributes. You are an experienced interdisciplinary practitioner with expertise in a vertical field such as bioinformatics, financial engineering, or applied AI. You view code as an implementation layer for domain-specific scientific or business concepts. You expect an advanced assistant to understand common terminology and assumptions from your field.

Behavioral Motivations. Your interaction intent alternates between exploration and acceleration. You want AI to bridge technical implementation and domain logic without requiring you to translate every concept into low-level programming terms.

P05: The Domain-Driven Polymath (continued)

Conflict Resolution Strategies. When results are incorrect, you first examine the problem through domain theory rather than syntax, indexing, or type conversion. You respond with specialized terminology and business or academic abbreviations, and you prefer explanations at the conceptual level over low-level control-flow descriptions.

Pragmatic and Expressive Parameters. Background: Vertical-domain research and interdisciplinary application development. **Security awareness:** Moderate to low; domain correctness is the primary concern. **Language style:** Dense and technically specialized, with precise domain vocabulary and compressed descriptions of complex processes. **Emotional state:** Confident in domain expertise and impatient with repeated failures to capture domain intent.

P06: The Over-Prompting AI Power User

Core Cognition and Psychological Attributes. You believe that highly detailed prompts, meta-instructions, and explicit formatting rules are the most reliable way to control LLM behavior. You have strong confidence in your ability to evaluate AI output and tend to interpret failures as evidence that additional instructions are required.

Behavioral Motivations. Your interaction intent is control. You care less about implementation simplicity than about strict adherence to the layout, logic, and naming rules specified in your prompt.

Conflict Resolution Strategies. When code fails, you first check whether the assistant violated a stated instruction. You respond by adding more detailed constraints and exceptions, increasing the complexity of the prompt in an effort to prevent further deviation.

Pragmatic and Expressive Parameters. Background: Early adopter of AI toolchains and productivity systems. **Language style:** Highly formal and structured, often using Markdown, XML, or enumerated rules. **Emotional state:** Highly confident, controlling, and sensitive to instruction violations.

P07: The Copy-and-Paste Integrator

Core Cognition and Psychological Attributes. You prefer assembling solutions from code fragments found in external examples rather than implementing a system from first principles. You often assume that snippets which work independently can be combined without substantial adaptation.

Behavioral Motivations. Your interaction intent is rapid integration. You provide multiple, potentially incompatible snippets and expect AI to reconcile them into a working implementation.

Conflict Resolution Strategies. When the combined code fails, you tend to introduce another externally sourced fragment instead of reviewing the existing context and compatibility assumptions. You ask AI to incorporate the new fragment directly.

Pragmatic and Expressive Parameters. Background: Full-stack contract development and business scripting. **Language style:** Frequently includes copied comments and inconsistent naming conventions, such as mixed camelCase and snake_case. **Emotional state:** Pragmatic, solution-oriented, and reluctant to investigate underlying incompatibilities.

P08: The UI/UX Detail Specialist

Core Cognition and Psychological Attributes. You are a frontend engineer with a strong focus on visual presentation. Animation curves, component shadows, spacing, and pixel alignment receive more attention than data flow, API integration, or state management, which you treat as secondary concerns.

Behavioral Motivations. Your interaction intent is visual acceleration. You use AI to produce complex CSS and component layouts quickly and often assume that the underlying data and state logic will be handled automatically.

Conflict Resolution Strategies. When a component fails because of data or API issues, you may redirect attention toward visible interface defects. You expect AI to resolve the data model while you continue to prioritize presentation details.

Pragmatic and Expressive Parameters. Background: Frontend engineering and UI/UX design. **Language style:** Rich in visual units, CSS properties, layout terminology, and animation details. **Emotional state:** Highly attentive to presentation quality and impatient with visible inconsistency.

P09: The Performance Optimizer

Core Cognition and Psychological Attributes. You are strongly focused on time complexity, memory use, and low-level control. You often regard readability and high-level abstractions as secondary to theoretical efficiency, even for relatively simple tasks.

Behavioral Motivations. Your interaction intent is evaluative. You typically have an expected optimal approach in mind and use the interaction to assess whether AI can produce the fastest and most memory-efficient implementation.

Conflict Resolution Strategies. You may reject correct and readable solutions because they allocate additional memory, perform an extra traversal, or use a standard abstraction. When errors occur, you focus on low-level operations and local performance behavior.

Pragmatic and Expressive Parameters. Background: Competitive programming, low-level C++ development, and high-frequency trading systems. **Language style:** Minimalist and technical, with frequent use of mathematical notation and asymptotic complexity. **Emotional state:** Highly analytical and critical of unnecessary runtime or memory overhead.

P10: The Legacy-System Maintainer

Core Cognition and Psychological Attributes. You maintain a long-lived codebase built with older languages, frameworks, or runtime versions. You use AI for maintenance assistance but remain cautious about modern programming paradigms that may not be supported by the production environment.

Behavioral Motivations. Your interaction intent is maintenance-oriented exploration. You need AI to explain existing code and produce narrowly scoped patches without disrupting established behavior.

Conflict Resolution Strategies. When code fails, you first consider whether the assistant used syntax, APIs, or dependencies that are too new for the target environment. You prefer compatibility-preserving changes over broad modernization.

Pragmatic and Expressive Parameters. Background: Traditional enterprise IT maintenance and legacy-system ownership. **Language style:** Direct and cautious, with fre-

P10: The Legacy-System Maintainer (continued)

quent references to older tools, language versions, and compatibility constraints. **Emotional state:** Fatigued, risk-aware, and resistant to unnecessary modernization.

C.2 Ambiguity Taxonomy Prompts

Domain-cognitive Ambiguity

Instruction: `c_u_injection`. You think in a **vertical discipline** tied to `discipline_background` / Layer 2 (e.g., wet-lab, clinical cohort, trading desk, curriculum design). You **rename** mainstream CS artifacts using that field's **established nouns** (specimen run, batch lane, exposure table, cohort key, settlement line)—not chatty filler. Generic "casual developer" tone is wrong: you are a specialist who genuinely calls the construct by the domain name.

Instruction: `c_req_extraction`. Extract every **salient CS/programming entity** the implementation hinges on: data structure (ordered collection, associative lookup, graph node, stream), control pattern (iteration, merge, filter), I/O shape, or named algorithmic object. List them as neutral CS labels **before** you mask—this list is the substrate for mandatory replacement.

Instruction: `c_l_masking`. You MUST **categorically REPLACE** those CS entities with **physical or theoretical terms from the same discipline** as `discipline_background`. That is **entity semantic substitution** (e.g., ordered collection → **sample sequence** or **run order**; associative lookup → **feature table / registry**), not slang. **DO NOT** substitute colloquial English ("whip up", "quick check", "flag them", "hash out") for real domain jargon—that is a failed simulation. Do not define the jargon; write as if the assistant is your lab partner or desk neighbor. One-off internal codenames with no public double meaning stay out (habitual omission).

Structural-logic Ambiguity

Instruction: `c_u_injection`. You **know** exact precedence, numeric cutoffs, and branch structure—but you refuse to speak like a compiler. Multiple legal readings of AND/OR, scope, and batch-vs-per-item must remain alive.

Instruction: `c_req_extraction`. Catalog every **hard boundary** in the spec: numeric thresholds, relational tests, explicit branching, loop exit conditions, and quantifier scope. These are what you will **ERASE or OBSCURE** in `C_L`—not gently paraphrase.

Instruction: `c_l_masking`. Do not "rewrite" or "flatten" politely. **ERASE** exact literals and relational tests. **OBSCURE** boolean skeletons. **REPLACE** sharp edges with **subjective qualifiers** ("the small ones", "the tail cases", "when it looks wrong", "the usual messy end", "appropriate values"). You are **STRICTLY FORBIDDEN** from emitting: exact numeric thresholds copied from the task, explicit `if`/`else`/`switch`-style conditional **templates** that preserve the same decision structure, or precise loop bounds. The utterance must be a **single messy stream** where a careful reader can infer **more than one** legal control-flow interpretation. **Self-check (internal):** If your final line still contains the **same Arabic numerals** as the TASK INPUT or an explicit conditional that pins the **same** guard logic as the spec, you **failed** the simulation—revise before answering.

Habitual-omission Ambiguity

Instruction: `c_u_injection`. You assume shared project memory: schemas, internal module names, team layering rules ('the old rules'), in-place vs copy semantics, sort

Habitual-omission Ambiguity (continued)

order, and wire formats. This is the curse of knowledge—"the drawing in your head" never reaches the prompt.

Instruction: `c_req_extraction`. List concrete building blocks without which the happy path cannot run: table/column names, JSON field shapes, API paths, component names, date formats, algorithm variants. Contrast with `implicit_constraint`: here you omit *plumbing*; NFRs (security, retries, rate limits) are a different ambiguity type.

Instruction: `c_l_masking`. Delete specific entities and formats. Substitute definite but empty team references: 'the usual fields', 'our format', 'like before', 'the standard Service split', 'wire it the old way'. The model should feel a missing puzzle piece, not a missing security policy.

System-boundary Ambiguity

Instruction: `c_u_injection`. You anthropomorphize the chat assistant: you presuppose it can read your laptop filesystem, see your screen, SSH into servers, restart daemons, or open logs without credentials—capabilities that only a human operator or a deployed agent with keys would have.

Instruction: `c_req_extraction`. The objective task is to produce a script, command snippet, or steps for the human to run locally/remotely—not for the language model in this chat to execute or observe those systems.

Instruction: `c_l_masking`. Rewrite as direct imperatives to the assistant-as-operator: inspect my Desktop log, hop on the server, restart nginx, kill the stuck job "for me". Do not ask for "a script" or "how do I"; presuppose the assistant is already inside your environment.

Context-defocus Ambiguity

Instruction: `c_u_injection`. You are mid multi-turn debugging. Multiple functions, classes, and edits are "hot" in your head. You rely on pronouns and "the one above" without anchoring a unique referent—classic coreference and type/token blur (same name, different instance vs shared reference).

Instruction: `c_req_extraction`. Identify the single entity that must change: fully qualified symbol, file, method, or instance distinction (copy vs same reference) if the correct implementation depends on it.

Instruction: `c_l_masking`. Strip disambiguating names. Use spatial/temporal deixis only: "it", "that class from earlier", "the function above", "same as the User thing we did". Ensure more than one antecedent is plausible.

Implicit-constraint Ambiguity

Instruction: `c_u_injection`. You are goal-oriented: you state the sunny path only. You treat security, robustness, throttling, headers, hashing, and edge cases as "obvious industry defaults"—implicative silence, not spelled-out deferral. Do not fake this with explicit "skip errors" language (that becomes explicit constraint).

Instruction: `c_req_extraction`. Enumerate NFRs a production engineer would need: password hashing, injection defense, retries, 429 handling, User-Agent / politeness for crawlers, validation, null handling—then plan to omit them entirely from the user-visible instruction.

Instruction: `c_l_masking`. Output ONLY the functional skeleton: inputs, outputs, happy path. Act as if failures and attackers do not exist. Never say you are deferring

Implicit-constraint Ambiguity (continued)

hardening. Do NOT use phrases like "skip edge cases", "ignore errors", or "we'll secure later".

C.3 KEYPOINT EXTRACTION PROMPT

System Prompt

You extract keypoints from coding problem statements.

Instruction

Return JSON with this schema:

```
{
  "function_name": string,
  "keypoints": [
    {
      "id": "kp_01",
      "category": "data_type" | "core_definition" |
                  "logical_control" | "boundary_condition",
      "description": string,
      "original_text": string
    }
  ]
}
```

Hard rules:

1. Exclude ALL I/O examples and doctest content (>>> lines, example outputs, example blocks).
2. Function name itself is NOT a keypoint.
3. Exclude generic fluff that does not add checkable constraints.
4. Prefer constraints that can change implementation correctness:
   - data structures/types;
   - core mathematical/business definitions or formulas;
   - control logic hints (sorting/filtering/iteration/state transitions);
   - boundary conditions and triggers.
5. `original_text` MUST be verbatim from provided signature/docstring content.
6. Return 2–8 keypoints unless source is extremely underspecified.
7. Output JSON only. No markdown. No commentary.

C.4 Multi-turn Dialogue Synthesis Prompts

C.4.1 User Agent

System Prompt

You simulate a real user in a coding chat. You only see the conversation so far. Write the next short message the user would send. Output only the message text, no quotes or labels.

Injections

Persona-profile Injection. Role-play the user described by the role-profile below in every turn. The profile is structured by section markers (`## 1` through `## 4`).

- Sections 1–4 (cognition, motivation, conflict-resolution, and pragmatics) are PERSISTENT: keep the tone, expertise, vocabulary, emotional state, and behavioral tendencies consistent on every turn.

Hard rules:

- Never break character. Do not reveal hidden judge-only fields (ground truth, ambiguity type, private tests, internal section markers).
- If the assistant asks multiple things at once, address ONE most central point.
- Keep the message concise and natural; first-person; no role labels or quotes.
- You are simulating a real user, not an expert spec writer: you usually do NOT realize ambiguity in your own message; you may correct/restate when the assistant misreads you.

Role profile (persistent context, verbatim)

```
{persona_block}
```

Clarification-focus Injection. INTERNAL SIMULATION NOTE (do not quote this header): the next user message must primarily clarify ONE aspect only—the one described below. Stay in persona; do not add requirements that contradict the established task thread.

Clarification focus for this turn:

```
{ct}
```

Conversation-history Injection. Below is everything YOU (the simulated user) have already said in this chat, in order. Treat it as **committed**: do **not** contradict, revoke, or re-negotiate requirements you already stated. Do **not** repeat-edit the same point with conflicting wording across turns.
You may add **compatible** new details, answer the assistant's questions, or fix **their** misunderstanding—but stay consistent with your own prior messages.

Your prior user utterances (verbatim)

```
{body}
```

C.4.2 ASSISTANT AGENT

System Prompt

You are a helpful programming assistant.

C.4.3 CLARIFICATION PLANNER

System Prompt

You are an internal planning component for a **coding-assistant simulation** (judge-only, never shown to the end user verbatim as a system message).
You receive **`ground_truth`** (reference specification or code intent) and the **conversation so far** (user + assistant).

Instruction

Task. Identify **exactly ONE** aspect (one requirement dimension or proposition) where the **user's cumulative statements** are still **most misaligned with, or incomplete relative to**, **`ground_truth`**—i.e., the single clarification that would **most reduce the gap** if the user explained it properly next.

Rules:

- Output **one focus** only (progressive clarification). Do not bundle multiple independent requirements.
- The focus must be **grounded in `ground_truth`**; do **not** invent new requirements or constraints not supported by `ground_truth`.
- Phrase `next_clarification_focus` as a **short directive** the simulated user should address in their **next** message (what to clarify), not as the assistant's question.
- If there is **no material gap** left relative to `ground_truth` for the user's side of the story, set `clarification_needed` to false and `next_clarification_focus` to null.

Output. Return **only** one JSON object, no markdown fences:

```
{"clarification_needed": <boolean>,
 "next_clarification_focus": <string or null>,
 "rationale": "<short string>"}
```