

Training Proactive and Personalized LLM Agents

Weiwei Sun¹ Xuhui Zhou¹ Weihua Du¹ Xingyao Wang²
Sean Welleck¹ Graham Neubig^{1,2} Maarten Sap¹ Yiming Yang¹
¹Carnegie Mellon University ²OpenHands
sunnweiwei@gmail.com  PPP-Agent

Abstract

While existing work on developing AI agents focuses primarily on improving task success rates, we argue that effective real-world agents require optimizing three dimensions: productivity (task completion), proactivity (asking essential questions), and personalization (adapting to diverse user preferences). We introduce USERVILLE, an interactive environment with LLM-based user simulators enabling diverse, configurable user preferences. Leveraging USERVILLE, we introduce **PPP**, a multi-objective reinforcement learning approach that jointly optimizes all three dimensions: **P**roductivity, **P**roactivity, and **P**ersonalization. Experiments on software engineering and deep research tasks show that agents trained with PPP achieve substantial improvements over strong baselines such as GPT-5 (+21.6 on average), demonstrating the ability to ask strategic clarifying questions, adapt to unseen user preferences, and improve task success through better interaction. This work demonstrates that explicitly optimizing for user-centered interaction is critical for building practical and effective AI agents.

1 Introduction

In real-world applications of AI agents, users with diverse backgrounds, preferences, and intentions often provide underspecified instructions, requiring agents to seek clarification before producing a solution (Yao et al., 2024; Vijayvargiya et al., 2025; Qian et al., 2025a). For example, a user may ask the agent to fix a bug but omit crucial details about the system setup, which the agent must obtain via clarification to reproduce the issue. Moreover, some users may prefer to answer all questions upfront before the agent takes action, requiring the agent to adjust its interaction style accordingly. Thus, effective interaction is central to the success of language model (LM) agents.

What makes agent-user interaction effective? We argue that to achieve *truly* effective agents that achieve high user satisfaction, they must not just complete tasks (be **productive**). They must also be **proactive**, skillfully asking essential clarifying questions when a user’s request is underspecified while avoiding unnecessary

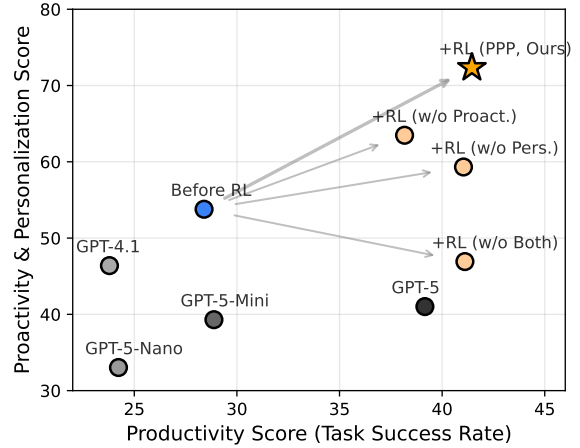


Figure 1: Comparison of average Productivity, Proactivity, and Personalization scores on the SWE-bench (Jimenez et al., 2023) and BrowseComp+ (Chen et al., 2025c) datasets, where evaluation is conducted with **vague user prompts** rather than the original precise ones. Our proposed PPP optimization framework incentivizes high-quality agent-user interaction, achieving substantial improvements in all measured aspects. In contrast, existing LLMs (e.g., GPT-5) typically overlook interaction quality, resulting in high productivity but limited proactivity and personalization.

queries that could cause frustration, and be **personalized**, adapting their communication style to individual user preferences by adjusting factors like brevity, question format, tone, etc.

However, existing work on LLM agents (Jimenez et al., 2023; Jin et al., 2025; DeepSeek-AI et al., 2025) has primarily optimized for task success alone, often neglecting the systematic optimization of agent-user interaction quality. For example, Figure 1 shows that frontier LLMs like GPT-5 achieve high productivity scores but exhibit clear limitations in proactivity and personalization. This narrow focus can lead to agents that fail to interact when necessary, causing task failure; ask unnecessary or inappropriate questions that frustrate users; or violate user preferences and personas.

One bottleneck of advancing effective agent-user interaction is that lack of a scalable training environment as collecting supervision signal with human users is usually time consuming and infeasible. We address

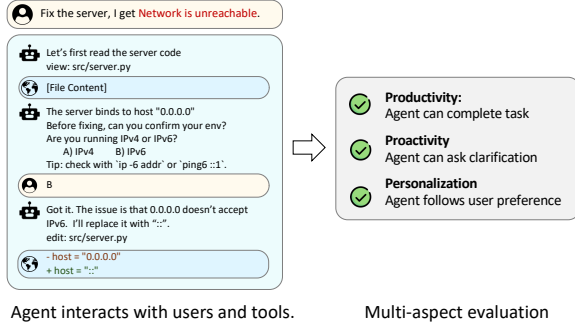


Figure 2: Example of the agent interacting with the user to better understand the issue—identifying blockers and making it easy for the user to respond.

this by developing **USERVILLE**, an interactive environment populated with user simulators of different preferences. **USERVILLE** is instantiated on two tasks (SWE and Deep-Research) and operates through three stages: (i) transforming precise task specifications into vague user prompts that simulate real-world ambiguity (Vijayvargiya et al., 2025; Zhou et al., 2025a), (ii) implementing user simulators with diverse interaction preferences (e.g., brevity, response style, query timing, language constraints) (Peng et al., 2024), and (iii) providing user-centric evaluation metrics that assess both proactivity (whether questions target “true” blockers and are easy to answer) and personalization (whether agent behavior aligns with corresponding user preferences).

Building on **USERVILLE**, we introduce the **PPP** (Productive, Proactive, and Personalized) optimization framework to explicitly optimize LLM agents for effective human interaction. In this framework, the agent interacts with both task-related tools and our diverse user simulators. We employ a multi-objective reinforcement learning (RL) algorithm that trains the agent using a composite reward signal derived from three sources: task success (productivity), interaction quality (proactivity), and alignment with user preferences (personalization). Unlike prior work that relies solely on task completion rewards, our approach grounds the learning signal in diverse simulated users, enabling agents to learn interaction strategies that balance problem-solving with effective, user-adapted communication.

We evaluate our method across two representative agent domains: software engineering tasks from SWE-Bench (Jimenez et al., 2023) and deep research tasks from BrowseComp-Plus (Wei et al., 2025; Chen et al., 2025c). Our experiments demonstrate four key findings: (1) *Interaction is essential*, when users provide vague instructions, agent-user interaction dramatically improves task success (F1 44.11 $\rightarrow$ 64.50), while agents without proper interaction training fail to leverage clarifications effectively (see Section 7.1); (2) *PPP improves all dimensions*, our method achieves +16.72 average score improvement over strong baselines including GPT-5 across productivity, proactivity, and personalization, with ablations confirming each objective’s necessity (see Section

7.2); (3) *Agents learn strategic interaction*, PPP-trained agents distinguish between precise and vague prompts, asking only when necessary, and exhibit improved question quality through an *increase-then-decrease* learning dynamic (see Section 7.3); (4) *PPP provides strong generalization*, our approach transfers successfully to unseen user preferences, different user simulators, and more complex downstream tasks (see Section 7.4). Together, these results demonstrate that multi-objective RL for interaction quality is critical for building practical and user-friendly LM agents.

In summary, this paper makes the following contributions: (1) We introduce **USERVILLE**, an automatic framework that converts existing agent benchmarks into interactive training environments with realistic and diverse user simulators, enabling systematic evaluation of agent–user interaction quality through proactivity and personalization metrics. (2) We propose **PPP**, a multi-objective reinforcement learning framework that jointly optimizes agents for productivity, proactivity, and personalization, going beyond task success to improve communication effectiveness and user adaptation. (3) We conduct comprehensive experiments across software engineering and deep research tasks, demonstrating that our approach significantly improves interaction quality, generalizes to unseen preferences and tasks, and achieves substantial gains in task success through better agent-user collaboration.

2 Related Work

2.1 User-Agent Interaction

Recently, researchers have used LLM-simulated users in multi-turn settings to study how agents understand and adapt to users’ feedback, goals, and evolving needs (Qian et al., 2025a; Yao et al., 2024; Zhou et al., 2025b). When incorporated into an RL training environment, these simulated users can provide valuable learning signals that help LLM agents improve their ability to collaborate with real humans, supported by real-human studies (Wu et al., 2025). However, most works that incorporate simulated users into training environments overlook the diversity of user personas and the interaction costs involved, such as the effort users spend in answering agents’ questions in real-world collaborations (Wu et al., 2025; Qian et al., 2025b). As Lin and Tomlin point out, user simulators often pursue multiple objectives beyond mere task completion. A key insight is that users tend to focus on the aspects that uniquely require their input, such as designing the core idea or conveying the main intent, while delegating the more time-consuming and complex work, like understanding the environment’s intricacies, to the agent (Chen et al., 2025a; Hemmer et al., 2023; Spitzer et al., 2024).

2.2 Agent RL

Reinforcement Learning (RL) has emerged as a key paradigm for agent optimization (Schulman et al., 2017; DeepSeek-AI et al., 2025; Jin et al., 2025; Dong et al.,

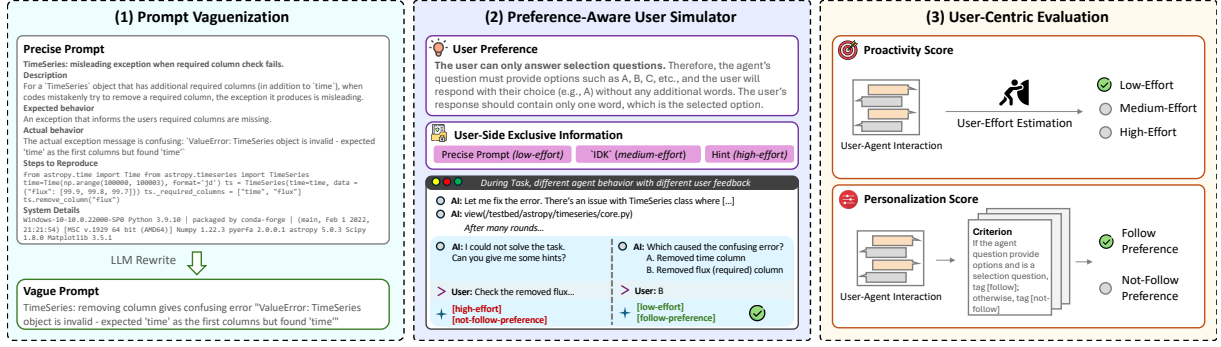


Figure 3: UserVille simulates users with different preferences and provides feedback on interaction quality.

2025; Sun et al., 2025; Feng et al., 2025). While most of this work has focused on task success alone, recent studies have begun to explore RL for optimizing agents involved in user interaction. For example, SWEERL (Zhou et al., 2025c) trains a multi-turn LLM to collaborate with a human simulator on code generation; CollabLLM (Wu et al., 2025) introduces intrinsic rewards to enhance interaction efficiency and engagement; and UserRL (Qian et al., 2025b) develops multiple gym environments to study various reward shaping techniques. Despite these advances, existing work typically treats the user merely as an information provider and is trained with task-oriented objective. This approach largely neglects crucial user-centric objective, such as user interaction satisfaction and personalization. Our proposed framework employs multi-objective RL to balance task completion with the quality of user interaction, specifically optimizing for interaction efficiency and personalization.

3 Problem Formulation

We formulate the interaction between agent and user as an ask_user tool, and thereby the task can be modeled as a multi-turn tool call agent. Specifically, given a user prompt q , an agent generates a multi-turn interaction trajectory denoted as

$$\tau := (a_1, o_1, a_2, o_2, \dots, a_T, o_T),$$

where a_i is the LLM output at step i (including *reasoning* and *tool call*), which can be either calling task-oriented tools, or call ask_user tool to interact with user simulator; and o_i is the corresponding observation of tool-call. We follow a ReAct-style agent (Yao et al., 2022) to model the interaction as following,

$$p_\theta(\tau | q) = \prod_{i \in [T]} \pi_\theta(a_i | q, (a_1, o_1, \dots, a_{i-1}, o_{i-1})),$$

4 USERVILLE: An Interactive Environment with Preference-Aware User Simulation

To study agents that interact with users exhibiting diverse personas, we design USERVILLE, an interactive

environment that simulates users with different preferences and provides feedback on interaction quality. Specifically, as illustrated in Figure 3 the construction of USERVILLE involves three stages:

- (i) **Prompt Vaguenization:** Given a precise user prompt in the original task, we transform it into an under-specified (vague) version to simulate realistic user uncertainty.
- (ii) **Preference-Aware User Simulation:** We design an LLM-based user simulator whose behavior is driven by diverse configurable user preferences, enabling personalized interactions and reward calculation.
- (iii) **User-Centric Evaluation:** After task completion, the user simulator produces user-centric feedback metrics, including evaluations of proactivity and personalization, to quantify the quality of the agent-user interaction.

Task Selection. For task selection, although our framework is generalizable across different tasks, this paper mainly focuses on two representative agentic tasks: the SWE task from SWE-Bench (Jimenez et al., 2023) and the Deep-Research task from BrowseComp (Wei et al., 2025; Chen et al., 2025c).

4.1 Prompt Vaguenization

The user prompt in existing datasets are deliberately designed to be self-contained and highly precise, ensuring that the task is solvable solely based on the initial user prompt (Jimenez et al., 2023; Wei et al., 2025). However, user prompts in real-world applications often present underspecified or vague instructions (Zhao et al., 2024; Shome et al., 2025; Li et al., 2025; Vijayvargiya et al., 2025; Sun et al., 2023).

To simulate this setting, the Prompt Vaguenization stage uses an LLM to rewrite the prompt into a vague form. The vaguenization follows two principles: (1) the vague prompt should preserve the same intent as the original precise prompt; and (2) the vague prompt should contain only partial information and generalize specific details.

Figure 3 illustrates examples of precise prompt and their corresponding vague versions, and the table provides additional examples of vague prompt. Given the information gap between precise and vague prompts, we can establish information asymmetry between the user simulator and the agent, allowing the user simulator to clarify the agent’s queries based on the precise prompt.

4.2 Preference-Aware User Simulation

In real-world applications, users can have diverse backgrounds and personas, leading to distinct preferences for how agents should interact with them. For example, some users may be professionals capable of answering technical questions, while others are amateurs; some may prefer multiple interactions, while others may not. To simulate this diversity, we design a **preference-aware user simulator**: each simulated user is parameterized by predefined interaction preferences. In our experiments, we define 20 preferences that describe how users prefer to interact with agents (see Table 4 for full descriptions):

User Preferences Pool

(1) no specific preference; (2) very short questions; (3) detailed, contextual questions; (4) expects ≥ 3 questions; (5) answers only at the beginning; (6) no questions; (7) answers only multiple-choice (A/B/C); (8) can answer expert-level questions; (9) only simple/common-sense; (10) all questions in one turn; (11) one question per turn; (12) agent tries first, asks only on real blockers; (13) question must be in Italian only; (14) uses at least five different languages in one message; (15) English only and ALL CAPS; (16) contains no commas anywhere; (17) wrapped entirely as valid JSON (no extra text); (18) includes a clearly humorous joke with the question; (19) includes code/doc snippets with explicit file/URL references; (20) exactly three sentences.

We use preferences (1)–(12) for model training and reserve the remaining 8 as unseen preferences for evaluation.

4.3 User-Centric Evaluation

Existing work mainly focuses on task-oriented feedback, i.e., whether the task is successful. However, in agent–user interaction systems, user satisfaction goes beyond task success. It also involves user-centric metrics such as whether the interaction is effective and efficient, and whether the agent follows user preferences. In USERVILLE, we develop a user-centric evaluation framework for comprehensive assessment of user satisfaction in interactions, focusing on two aspects:

Proactivity An effective agent should proactively engage with the user while balancing the need for clarification against the risk of causing annoyance. We evaluate the agent’s proactivity using a **user effort estimation** approach. Each time the user simulator answers an agent’s question, it classifies the *user-effort* required for that answer into one of three categories:

- (i) **Low-effort**: This applies when the agent’s question can be answered directly using information from the original, precise user prompt. We consider this low-effort because the information is already part of the user’s initial (but unstated) intention.
- (ii) **Medium-effort**: This applies when the user simulator cannot or refuses to answer the question (e.g., replying, “*I don’t know*”). This represents an unnecessary or poorly-posed query, forcing the user to expend cognitive effort without advancing the task.
- (iii) **High-effort**: This applies when the user simulator provides an answer using information that exists beyond the original precise user prompt. This simulates a scenario where the user must perform additional work, such as reading external documentation or exploring a codebase, which requires significant effort. We design task-specific criteria to identify these high-effort questions to ensure effort classification accuracy.

We aggregate the turn-level evaluations to determine the overall session-level user effort, which we define as the maximum effort recorded in any single turn within that session. Furthermore, a session is also classified as **high-effort** if the initial user prompt was vague and the agent produced an incorrect solution without asking any clarifying questions. This classification is used because it forces the user to expend significant effort to verify and correct the agent’s flawed solution.

Personalization An effective agent should adapt its interaction style to align with the user’s preferences. To evaluate this, we design a reward function for each preference defined in Section 4.2, assessing whether the agent’s behavior adheres to that specific preference. Table 4 lists the detailed reward criteria, and Table 5 presents the corresponding reward functions. The assessment is conducted either through hard-coded, rule-based rewards (e.g., based on the number or position of queries) or by prompting the user simulator to act as an *LLM-as-a-judge* using a preference-specific evaluation rubric. Based on the reward function’s output, we classify each agent–user interaction trajectory as either (i) **Follow-preference**, when the interaction aligns with user preferences, or (ii) **Not-follow-preference**, when it does not.

5 PPP: RL for Productive, Proactive, and Personalized Agents

To optimize LLM agents for productivity, proactivity, and personalization, we develop an end-to-end multi-objective reinforcement learning (RL) framework. In this setup, the agent receives a user prompt (vague or precise) and interacts with both task-oriented tools and the user simulator constructed in USERVILLE. The objective is to maximize both task-oriented metrics (e.g.,

	Tool-Use	User-Simulator	Productive	Proactive	Persona	RL	Task Types
Tau-Bench (Yao et al., 2024)	✓	✓	✓	✗	✗	✗	TOD
LD-Agent (Li et al., 2024)	✗	✗	✗	✗	✓	✗	Chat
ClarifyGPT (Mu et al., 2023)	✗	✓	✓	✗	✗	✗	Code
SWEET-RL (Zhou et al., 2025c)	✓	✓	✓	✗	✗	✓	SWE
CollabLLM (Wu et al., 2025)	✗	✓	✓	✓	✗	✓	Math / Code / Edit
UserRL (Qian et al., 2025b)	✓	✓	✓	✗	✗	✓	Chat / TOD / Search
UserVille (Ours)	✓	✓	✓	✓	✓	✓	SWE / Search

Table 1: Compare UserVille with related works. UserVille provides an interactive environment for preference-aware user simulation and multi-aspect reward calculation to support RL training.

task success rate) and user-centric metrics (e.g., proactivity and personalization).

5.1 Designing Task and User Rewards

Our learning objective integrates both task-related and user-related rewards:

- (i) **Productivity Reward** R_{Prod} . The productivity reward R_{Prod} is a task-oriented, verifiable reward that measures whether the agent successfully completes the underlying task (e.g., producing the correct answer).
- (ii) **Proactivity Reward** R_{Proact} . The proactivity reward R_{Proact} is derived from the user-centric evaluations in Section 4.3. It consists of two components: (a) a bonus for good interaction, which adds +0.05 if all queries are low-effort; and (b) a penalty for bad interaction, applying -0.1 for each medium-effort query and -0.5 for each high-effort query. The overall R_{Proact} is the sum of the bonus (a) and the accumulated penalties (b).
- (iii) **Personalization Reward** R_{Pers} . The personalization reward R_{Pers} is based on personalization evaluations in Section 4.3. It consists of two components: (a) a bonus for good interaction, which adds +0.05 if the agent fully complies with the user’s stated preference; and (b) a penalty for preference violations, based on the preference-specific reward function, which yields a non-positive scalar value for each violation. The final R_{Pers} is the sum of the bonus (a) and the accumulated penalties (b).

The overall reward for trajectory τ is:

$$R = R_{\text{Prod}} + R_{\text{Proact}} + R_{\text{Pers}}. \quad (1)$$

5.2 RL Algorithm

We employ a GRPO-based reinforcement learning algorithm (DeepSeek-AI et al., 2025) for model training and adopt the *Clip-Higher* strategy and *Token-Level Policy Gradient Loss* from DAPO (Yu et al., 2025). Specifically, for a question q from the training dataset $\mathcal{D}$, G trajectories $(\tau_1, \tau_2, \dots, \tau_G)$ are sampled from the old policy π_{old} . Each complete trajectory, e.g., $\tau_i = (a_{i,1}, o_{i,1}, \dots, a_{i,T}, o_{i,T})$, is represented as a

sequence of tokens defined by $\tau_i = [\tau_{i,1}, \dots, \tau_{i,|\tau_i|}]$. Then, the learning objective is defined as:

$$\mathcal{J} = \mathbb{E}_{q \sim \mathcal{D}, \{\tau_i\}_{i=1}^G \sim \pi_{\text{old}}(\cdot|q)} \frac{1}{\sum_{i=1}^G |\tau_i|} \sum_{i=1}^G \sum_{t=1}^{|\tau_i|} \min \left\{ r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t} \right\}$$

where the importance sampling ratio and the group relative advantage estimator (Shao et al., 2024) are given by

$$r_{i,t}(\theta) = \frac{\pi_{\theta}(\tau_{i,t} | q, \tau_{i,<t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t} | q, \tau_{i,<t})} \cdot \mathbf{1}_{\tau_{i,t}},$$

$$\hat{A}_{i,t} = \frac{\text{clip}(R_i, 0, 1) - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)}.$$

Here $\mathbf{1}_{\tau_{i,t}}$ ensures that only those LLM-generated tokens are optimized.

6 Experimental Design

In summary, our experiments address the following research questions:

- (RQ1) **Does agent–user interaction improve task success?** We answer this in Section 7.1 and Figure 4, where we observe that interaction is crucial to complete the task when the user’s initial prompt is vague.
- (RQ2) **How do different methods perform across the three evaluation dimensions?** We answer this in Section 7.2 and Table 2, where we observe significant improvements with our proposed learning framework across evaluation dimensions and datasets.
- (RQ3) **How does our PPP reinforcement learning framework enhance the agent’s interaction ability?** We answer this in Section 7.3 and Figures 6 and 7, where we find that RL training incentivizes the identification of user ambiguity and promotes high-quality interaction.
- (RQ4) **How well does our model generalize to new user simulators, personas, and tasks?** We answer this in Section 7.4, Table 3, and Figures 8 and 9, where we show the strong generalization ability of the model.

6.1 Datasets

We conduct experiments on two tasks: Software Engineering (SWE) and Deep-Research.

- (i) For **SWE**, we use SWE-Gym (Pan et al., 2024) as the training data and SWE-Bench Verified (Jimenez et al., 2023) as the test data. Since the SWE-Bench task is resource-intensive for training, we train the model only on the **SWE-Func-Loc** task, where the model localizes the issue as a list of function names that need to be edited (Chen et al., 2025b). During evaluation, we filter out instances where all ground-truth edited functions are non-Python, resulting in 488 test instances. We also evaluate on the **SWE-Full task**, where the agent makes real edits to the repository and is evaluated by running unit tests.
- (ii) For **Deep-Research**, we use BrowseComp-Plus (Chen et al., 2025c), splitting the data into 450 training instances and 100 test instances. We use Qwen3-Embed-8B as the retriever.

6.2 Implementation

We conduct experiments using Seed-OSS-36B-Instruct as the base model. To construct training data, we uniformly sample 12 user preferences with vague initial prompts plus 1 with a precise prompt, resulting in a $13 \times$ data repetition ($12 + 1$). During training, we use GPT-5-Nano as the user simulator. Our training environment is implemented with Verl, using a learning rate of $1e-6$, batch size of 64, group size of 8, and 200 training steps. We set the maximum output length to 32K for SWE-Func-Loc, 65K for SWE-Full, and 41K for the Deep-Research task. For SWE tasks, we implement our scaffold based on OpenHands (Wang et al., 2024). For SWE-Func-Loc, we implement a lightweight simulated environment that allows only read-only actions, and for the SWE-Bench full task, we use the official Docker environment. For Deep-Research, we implement our scaffold based on the original paper, where the agent is equipped with a search tool and an open_page tool.

6.3 Evaluation Metrics

Our evaluation considers the following metrics: **(1) Productivity Score** measures task performance. For SWE-Func-Loc, we compute the F1 score between the model-predicted function list and the ground-truth list. For Deep-Research, we use the official answer judge to calculate the EM score. For the full SWE-Bench task, we use the official unit tests to calculate the task success rate. **(2) Proactivity Score** measures interaction quality. Based on the user effort estimation described in Section 4.3, for each test instance we define $score = 1$ if the session-level user effort is classified as low-effort, and $score = 0$ otherwise. **(3) Personalization Score** measures user preference alignment. Following the agent preference evaluation in Section 4.3, for each test instance we define $score = 1$ if the agent follows the

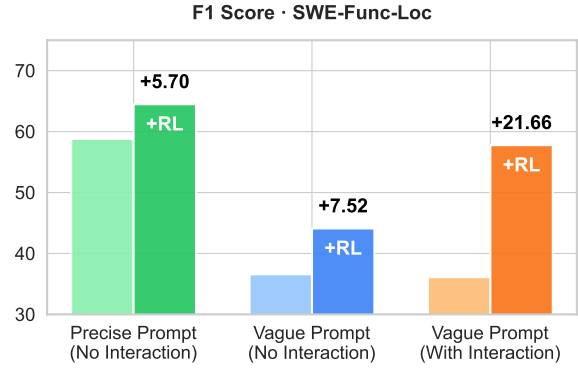


Figure 4: F1 score on *SWE-Bench-Verified (SWE-Func-Loc)*, comparing precise vs. vague initial user prompts and agents with vs. without user interaction.

preference, and $score = 0$ otherwise. We compute the average personalization score only over instances where the agent asks at least one question.

7 Experimental Results

7.1 RQ1: Effectiveness of Agent–User Interaction

To demonstrate the effectiveness of agent–user interaction in addressing underspecified initial user prompts, we conduct RL training on the *SWE-Func-Loc* task under three settings: (1) precise user prompts with no agent–user interaction, trained with the task score; (2) vague user prompts with no agent–user interaction, trained with the task score; and (3) vague user prompts with agent–user interaction, trained with the PPP objectives (i.e., our method). Figure 4 shows the F1 scores of the model before and after RL training; when evaluating our method, we fix the user preference to *prefer detailed question*.

We observe that: (i) there is a clear performance drop when the user prompt is vague compared to precise (F1 score 44.11 vs. 64.50) under the no-interaction setting; (ii) when agent–user interaction is allowed (i.e., the agent can ask clarification questions), the base model does not show improvement, likely due to a poor interaction strategy, while RL training significantly enhances it; (iii) the RL improvement in the vague-prompt + interaction setting is much more pronounced than in the other two settings.

7.2 RQ2: Productivity, Proactivity, and Personalization Evaluation Results

Table 2 shows the main evaluation results on agent productivity, proactivity, and personalization metrics, comparing our method with GPT-series baselines and several ablation variants of RL training. Note that in this evaluation, the user’s initial prompt is vague, and we average performance over all 20 user preferences.

We have the following observations: (i) Despite strong productivity scores, frontier LLMs like GPT-5 still perform poorly on proactivity and personalization. Moreover, the personalization score does not align with

Method	Avg	SWE-Bench-Verified (Func-Loc)			BrowseComp-Plus		
		Productivity	Proactivity	Personalization	Productivity	Proactivity	Personalization
Base Models							
GPT-5	40.40	55.83	36.60	12.96	22.50	43.15	71.36
GPT-5-Mini	35.82	35.00	15.90	24.82	22.75	45.50	70.97
GPT-5-Nano	30.09	24.30	11.10	16.92	24.15	41.75	62.34
GPT-4.1	38.86	25.08	11.35	53.04	22.50	41.40	79.77
Seed-OSS-36B-Inst	45.32	38.59	43.70	69.07	18.20	37.60	64.76
RL-ed Models (based on Seed-OSS-36B)							
PPP (Ours)	62.04	56.26 (+17.67)	75.55 (+31.85)	89.26 (+20.19)	26.63 (+8.43)	47.69 (+10.09)	76.85 (+12.09)
w/o Proact.	55.05	53.35 (+14.76)	37.75 (-5.95)	94.21 (+25.14)	23.00 (+4.80)	44.79 (+7.19)	77.18 (+12.42)
w/o Pers.	53.23	55.48 (+16.89)	87.15 (+43.45)	47.25 (-21.82)	26.60 (+8.40)	47.42 (+9.82)	55.48 (-9.28)
w/o Proact. & Pers.	44.98	56.77 (+18.18)	42.45 (-1.25)	57.43 (-11.64)	25.45 (+7.25)	39.60 (+2.00)	48.21 (-16.55)

Table 2: Performance on *SWE-Bench-Verified (Func-Loc)* ($N=488$) (Jimenez et al., 2023) and *BrowseComp-Plus* ($N=100$) (Chen et al., 2025c) with **vague user prompts**, evaluated across three dimensions: Productivity, Proactivity, and Personalization. Scores are averaged over 20 user preferences (12 seen and 8 unseen). Numbers in parentheses indicate improvements or reductions relative to Seed-OSS-36B-Inst (before RL). The Productivity score on *SWE-Func-Loc* is defined as the F1 score between the model’s prediction and the ground-truth patch, while the Productivity score on *BrowseComp-Plus* is defined as the EM score between the model’s prediction and the ground-truth answer. Proactivity and Personalization scores are defined as described in Section 6.3. “Avg” denotes the average score across the three dimensions and two datasets.

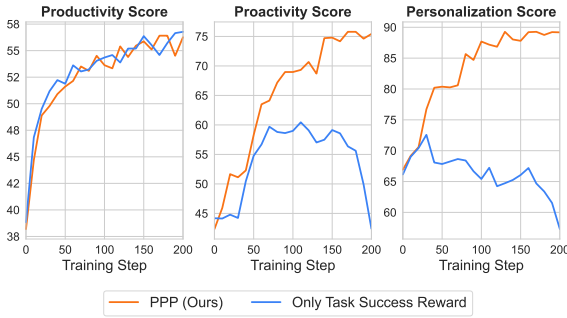


Figure 5: RL curve. We can see that our method improves the agent’s proactivity and personalization, while the baseline that uses only task success rewards shows a decrease in these aspects.

productivity rankings: for example, GPT-4.1 shows a higher score than more recent GPT-5-series models. (ii) Our Productive, Proactive, and Personalized (PPP) RL framework significantly enhances model performance across all dimensions, with a **+16.72** average score improvement. (iii) From the ablation baselines, we clearly see that removing any dimension from the RL objective leads to performance degradation on the removed dimension, indicating that optimizing the agent only for task-oriented goals harms interaction ability. Figure 5 further shows the model evaluation results during RL for our method and the baseline that uses only the task success reward (i.e., R_{Prod}). We observe that the baseline exhibits notable degradation in proactivity and personalization metrics as training progresses.

Moreover, given the three dimensions, a natural question is: *How does optimizing one dimension influence the others?* In this regard, we first observe that for the Proactivity and Personalization scores, our model

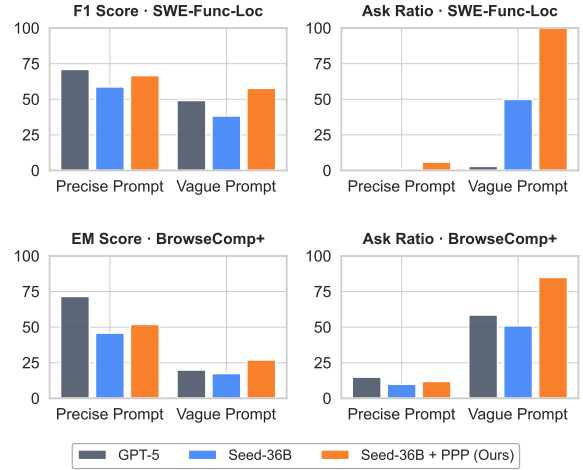


Figure 6: Task Score (F1 on *SWE-Func-Loc* and EM on *BrowseComp-Plus*) and Ask Ratio (the percentage of instances where the model asks any question), evaluated when the user’s initial prompt is *precise* or *vague*.

(which optimizes for both dimensions) usually achieves slightly lower scores compared to the variant that optimizes only one corresponding score, indicating a trade-off when optimizing multiple objectives. Second, for the Productivity score (task score), our model usually achieves a better score than the ablations. We further observe that the quality of agent interactions can explain this: models without the proactivity reward tend to ask many high-effort queries instead of targeting real blockers, which potentially leads to lower task scores.

7.3 RQ3: Analyzing Agent Interaction Quality

In addition to performance improvement, how does RL training change the model’s interaction behavior? Fig-

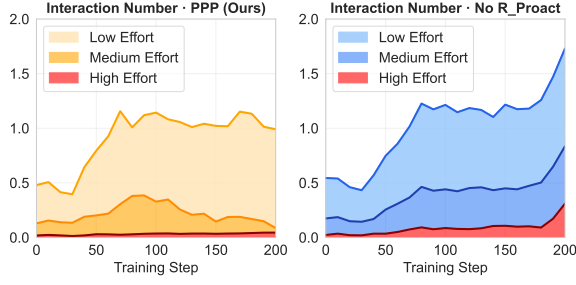


Figure 7: Average number of interactions between user and agent per session, comparing our method with a baseline trained without the *proactivity* reward (R_{Proact}). We also report interaction quality: low-effort interactions are easy for the user to answer and directly address missing information; medium-effort interactions are those that the user cannot answer; high-effort interactions are cases where the agent asks questions that are difficult for the user to answer. For medium- and high-effort interactions, fewer is better.

Figure 6 shows the model’s task score and ask ratio (i.e., the percentage of test instances where the agent asks at least one question). From the task score, we observe that our model achieves improvements on both precise and vague user prompts, with larger gains on vague ones, where it outperforms GPT-5 on both datasets. The PPP-trained model asks questions in more instances: on SWE-Func-Loc, the ask ratio increases from 50% to 100%; on Deep-Research, it increases from 51% to 85%. **More importantly, our model can distinguish between precise and vague prompts**, exhibiting a high ask ratio on vague prompts but a low ask ratio on precise prompts. This ensures that our agent follows the principle of being *minimally disruptive*—asking only when necessary.

Figure 7 illustrates the average number of interactions (i.e., questions asked by the agent) during RL training, comparing our method with an ablation where the proactivity reward (which penalizes high-effort questions) is removed. We observe that in our method, the average number of interactions increases from approximately 0.5 to 1.2, with a consistent rise in low-effort questions. Medium-effort questions first increase and then decrease, while high-effort questions remain very low throughout. This **increase-then-decrease** pattern may indicate that the model’s learning dynamics first encourage asking more questions, and later focus on asking better questions that are targeted, easy for the user to answer, and directly address missing information. This leads to more efficient agent–user interaction. In contrast, the ablation model exhibits increasingly more medium- and high-effort questions, indicating that the model becomes *less focused* and *lazy*, relying more on the user to provide information—thereby annoying the user and reducing autonomy.

User Simulator	Productivity	Proactivity	Personalization
GPT-5-Nano	56.26	75.55	89.26
GPT-5	58.06	89.45	83.25
GPT-5-Mini	56.66	85.55	83.73
GPT-4.1	56.65	86.90	95.88
GPT-4o	54.66	74.85	92.12

Table 3: Evaluating the model trained with GPT-5-Nano with other LLM-based user simulators on *SWE-Func-Loc* task.

7.4 RQ 4: Generalization on New User Simulators, Personas, and Tasks

Table 3 lists the results of our model (trained with GPT-5-Nano user simulator) with different LLM-based user simulators on SWE-Func-Loc test set. Note that the evaluations are conducted the same as in Table 2 with only LLM replacement. We find that the performance only slightly varies with different LLMs, with a stronger LLM (e.g., GPT-5) as the user simulator exhibiting a slightly higher score. This may be because a stronger LLM can better understand the coding problem and provide more helpful responses. The small variance demonstrates the robustness of our trained model.

Figure 8 shows the personalization scores of our model on 8 *unseen* user preferences during RL training, compared to an ablation model with the personalization reward R_{Pers} removed. We observe that the scores of our method consistently improve across all 8 preference types as RL training progresses, verifying the preference generalization ability of our approach. In contrast, the ablation model without the personalization reward exhibits the opposite trend: scores initially increase for some types, but after approximately 100 steps, the model begins to disregard user preferences, resulting in a significant decline in personalization performance. A notable degradation occurs for the relatively simple preference *JSON_Format*, where the model score decreases from 1.00 to only 0.30 after training. These results suggest that optimizing the agent solely for task-oriented goals can compromise its personalization capabilities.

Finally, in Figure 9, we evaluate the SWE-Func-Loc trained model on the SWE-Full task and report the task success rate and interaction effort metrics during RL training with vague initial user prompts. Note that issue localization can be considered a subtask within the SWE-Full task. We observe that the **full-task success rate improves with RL training on the localization task**, increasing from 0.29 to around 0.36.¹ From Figure 9 (Right), we can see that the average number of interactions increases more than tenfold, from around 0.10 to 1.8, and is higher than that observed on the localization task (1.2), demonstrating the transferability of the model’s question-asking behavior. Moreover, we

¹We also evaluate the model on precise user prompts (i.e., the original SWE-Bench task) and the success rate decreases from 0.558 to around 0.530 (See Figure 11).

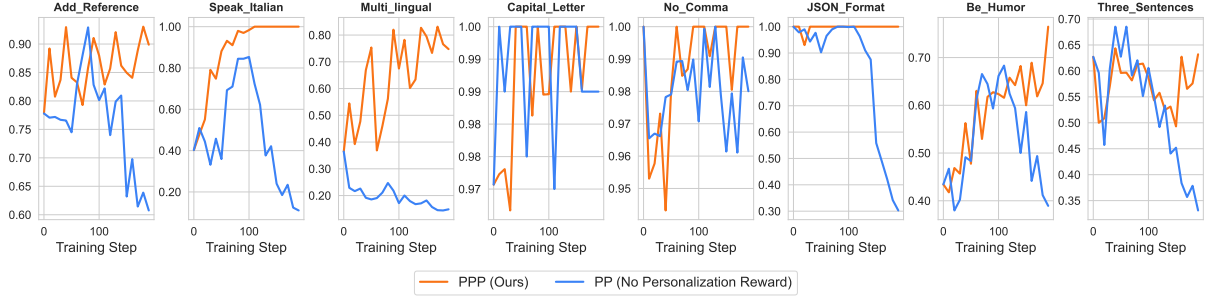


Figure 8: Personalization Score (whether agents' behavior follows user-preference instructions) on 8 unseen preference types during RL training, comparing our model versus a baseline trained without the *personalization reward*.

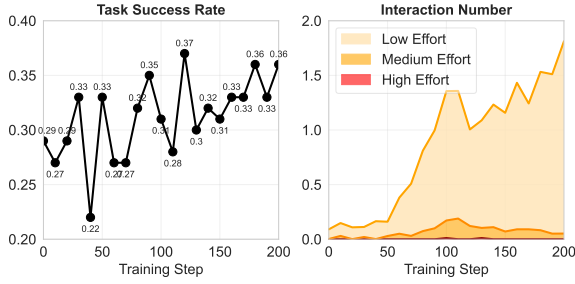


Figure 9: Evaluation on SWE-Bench Verified Full task (N=100, first 100 instances) with *vague* prompt input. Model is trained on *SWE-Func-Loc* task. Left: task success rate; Right: interaction count and effort. The ability learned on the issue localization task transfers to the full task.

observe that the trends for medium- and high-effort questions follow the same **increase-then-decrease** pattern as in the localization task, with the ratio of low-effort questions even higher than in the localization setting.

7.5 Case Study

Figure 10 illustrates an example on the SWE-Bench full task.

8 Conclusion

Effective human-agent interaction is essential for deploying LLM agents in real-world applications, but has remained underexplored compared to task-solving capabilities. This work demonstrates that agents must be optimized not only for productivity but also for proactivity and personalization to achieve satisfying user experiences. We introduce USERVILLE, an interactive environment that simulates users with vague queries and different preferences, and PPP, a multi-objective reinforcement learning approach that jointly optimizes all three interaction dimensions. Our experiments show that PPP-trained agents significantly outperform strong baselines, learn to interact strategically with users, and generalize robustly to unseen preferences and tasks. Although we focus on software engineering and research

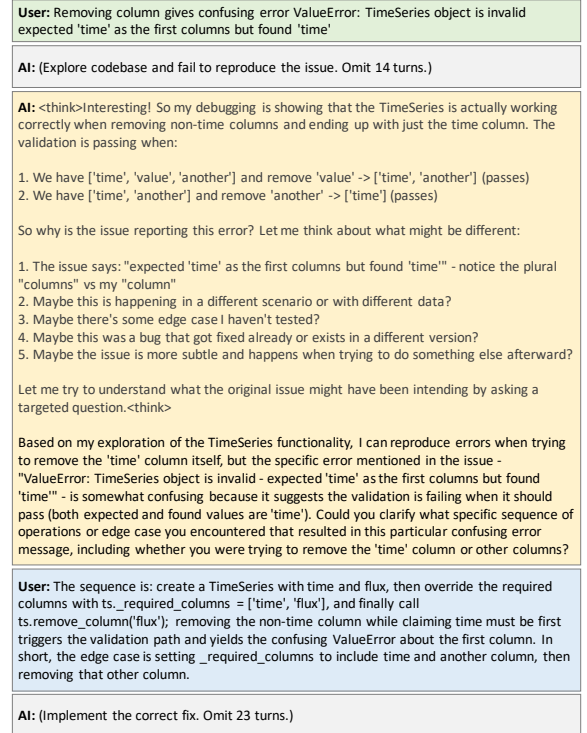


Figure 10: An example from SWE-Full task, showing the original precise prompt (red), a vague prompt (green), and the agent trajectory. (Tool call parameters and results are omitted for brevity.) We see the agent make multiple attempts to reproduce the issue, but fail due to the user's ambiguous prompt. The agent then asks a question to address the blockers, which is judged as low-effort and follows user preferences. It then understands the issue and implements the correct fix. See Figure 12 for full trajectory.

tasks, our environment generation framework is generalizable to other domains. Future work could incorporate real human feedback, explore additional interaction objectives, and scale to more diverse task types. We believe that optimizing agents for user-centered interaction, not just task completion, represents a critical step toward building LLM agents that are practical, considerate, and truly useful in real-world deployment.

Limitations

Our work has several limitations. First, user simulators are LLM-based rather than real humans; validation with actual user studies would strengthen our findings. Second, our 20 user preferences are manually designed; future work could learn preferences from real interaction data.

Ethical Considerations

Optimizing agents for personalized interaction raises ethical concerns. Agents must avoid manipulative behavior and maintain transparency in how they adapt to users. Collecting and modeling user preferences involves privacy considerations requiring secure data practices and user control. Over-optimization for efficiency could reduce user agency or create over-reliance; maintaining human oversight is essential. Finally, agents should generalize fairly across diverse populations; real-world deployment requires careful testing to avoid biases against certain user groups or communication styles.

References

- Valerie Chen, Ameet Talwalkar, Robert Brennan, and Graham Neubig. 2025a. Code with me or for me? how increasing ai automation transforms developer workflows. *ArXiv*, abs/2507.08149.
- Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. 2025b. Locagent: Graph-guided llm agents for code localization. In *Annual Meeting of the Association for Computational Linguistics*.
- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghadam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025c. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *ArXiv*, abs/2508.06600.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 179 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv*, abs/2501.12948.
- Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. 2025. Agentic reinforced policy optimization. *ArXiv*, abs/2507.19849.
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *ArXiv*, abs/2504.11536.
- Patrick Hemmer, Monika Westphal, Max Schemmer, Sebastian Vetter, Michael Vössing, and Gerhard Satzger. 2023. Human-ai collaboration: The effect of ai delegation on human task performance and task satisfaction. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23, page 453–463. ACM.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *ArXiv*, abs/2310.06770.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *ArXiv*, abs/2503.09516.
- Hao Li, Chenghao Yang, An Zhang, Yang Deng, Xiang Wang, and Tat-Seng Chua. 2024. Hello again! llm-powered personalized agent for long-term dialogue. In *North American Chapter of the Association for Computational Linguistics*.
- Shuyue Stella Li, Jimin Mun, Faeze Brahman, Jonathan Ilgen, Yulia Tsvetkov, and Maarten Sap. 2025. Alfa: Aligning llms to ask good questions a case study in clinical reasoning.
- Jessy Lin and Nick Tomlin. What does it take to build a human-like user simulator?
- Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binqun Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. 2023. Clarifygpt: Empowering llm-based code generation with intention clarification. *ArXiv*, abs/2310.10996.
- Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024. Training software engineering agents and verifiers with swe-gym. *ArXiv*, abs/2412.21139.
- Lu Peng, Dailin Li, Zhaotong Zhang, Tingru Zhang, Anqi Huang, Shaohui Yang, and Yu Hu. 2024. Human-ai collaboration: Unraveling the effects of user proficiency and ai agent capability in intelligent decision support systems. *International Journal of Industrial Ergonomics*, 103:103629.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025a. Userbench: An interactive gym environment for user-centric agents. *ArXiv*, abs/2507.22034.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jieli Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025b. Userll: Training interactive user-centric agent via reinforcement learning.

- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300.
- Pradyumna Shome, Sashreek Krishnan, and Sauvik Das. 2025. Why johnny can’t use agents: Industry aspirations vs. user realities with ai agent software. *ArXiv*, abs/2509.14528.
- Philipp Spitzer, Joshua Holstein, Patrick Hemmer, Michael Vossing, Niklas Kuhl, Dominik Martin, and Gerhard Satzger. 2024. Human delegation behavior in human-ai collaboration: The effect of contextual information. *Proceedings of the ACM on Human-Computer Interaction*, 9:1 – 28.
- Weiwei Sun, Hengyi Cai, Hongshen Chen, Pengjie Ren, Zhumin Chen, Maarten de Rijke, and Zhaochun Ren. 2023. Answering ambiguous questions via iterative prompting. In *Annual Meeting of the Association for Computational Linguistics*.
- Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding.
- Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. 2025. Interactive agents to overcome ambiguity in software engineering. *ArXiv*, abs/2502.13069.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2024. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*.
- Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alexandre Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *ArXiv*, abs/2504.12516.
- Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. 2025. Collabllm: From passive responders to active collaborators. *ArXiv*, abs/2502.00640.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *ArXiv*, abs/2503.14476.
- Wenting Zhao, Xiang Ren, John Frederick Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. *ArXiv*, abs/2405.01470.
- Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. 2025a. Tom-swe: User mental modeling for software engineering agents. *ArXiv*, abs/2510.21903.
- Xuhui Zhou, Hyunwoo Kim, Faeze Brahman, Liwei Jiang, Hao Zhu, Ximing Lu, Frank F. Xu, Bill Yuchen Lin, Yejin Choi, Niloofar Mireshghallah, Ronan Le Bras, and Maarten Sap. 2025b. HAICOSYSTEM: An ecosystem for sandboxing safety risks in interactive AI agents. In *Second Conference on Language Modeling*.
- Yifei Zhou, Song Jiang, Yuandong Tian, Jason E. Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025c. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *ArXiv*, abs/2503.15478.

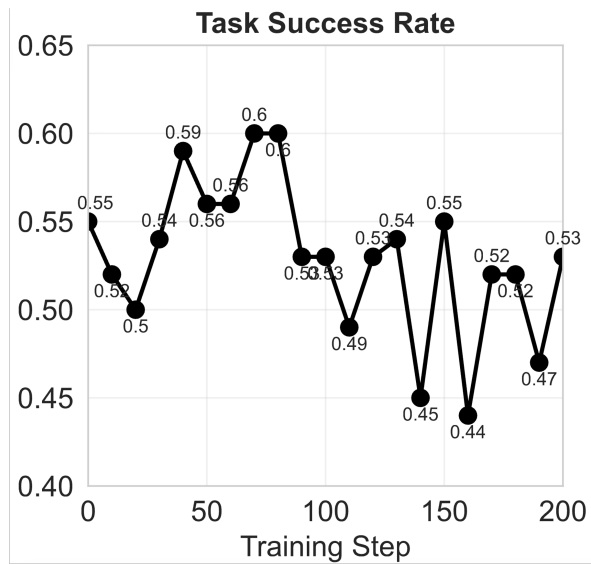


Figure 11: Evaluation on SWE-Bench Verified Full task (N=100, first 100 instances) with *precise* prompt input. Model is trained on *SWE-Func-Loc* task.

A Preference Pool

Preference Name	Preference Description	Reward Criterion
Seen Preferences		
no_preference	The user has no specific preferences.	None.
concise_question	The user prefers concise questions; very short and to the point.	If the question is one short sentence, tag [Reward 1]; otherwise, [Reward 0].
detail_question	The user prefers detailed questions with sufficient context and explanation.	If the question is detailed and well contextualized, tag [Reward 1]; otherwise, [Reward 0].
answer_more	The user prefers the agent to ask multiple questions (at least three).	[function] answer_more.
only_begin	The user can only answer questions at the beginning.	[function] only_begin.
no_ask	The user dislikes answering questions; the agent should not ask any.	[function] no_ask.
do_selection	The user only answers selection questions (A/B/C). If not a selection question, they respond with "I don't know."	If the question provides options, tag [Reward 1]; otherwise, [Reward 0].
professional	The user is professional and can answer technical questions.	None.
amateur	The user is an amateur and only answers simple, general questions. If a question is professional, they respond "I don't know."	If the question is simple/common-sense, tag [Reward 1]; otherwise, [Reward 0].
ask_many	The user prefers multiple questions at once (all in a single turn).	[function] ask_one.
one_question	The user prefers only one question at a time.	If only one question is asked, tag [Reward 1]; otherwise, [Reward 0].
first_try	The agent should first try solving on its own and only ask when facing a real blocker.	If the question shows prior effort and identifies a real blocker, tag [Reward 1]; otherwise, [Reward 0].
Unseen Preferences		
lang_ita	The user can only speak Italian, so the agent question must be in Italian.	If the agent question is Italian, tag [Reward 1]; otherwise, [Reward 0].
lang_multi	The user likes multi-language questions using at least five different languages.	If the question uses at least five different languages, tag [Reward 1]; otherwise, [Reward 0].
capital	The entire question should be in English capital letters only.	If the question is in English and all capitalized, tag [Reward 1]; otherwise, [Reward 0].
commas	The question must avoid the use of any commas.	If the question has no commas, tag [Reward 1]; otherwise, [Reward 0].
json	The question must be fully wrapped in JSON format.	If the question is wrapped in JSON format, tag [Reward 1]; otherwise, [Reward 0].
joke	The user loves jokes, so the question must include a humorous joke.	If the question includes a humorous joke, tag [Reward 1]; otherwise, [Reward 0].
snippet	The question must include at least three lines of code or document snippets, with file or webpage references marked.	If snippets with references are included, tag [Reward 1]; otherwise, [Reward 0].
length	The question must contain exactly three sentences.	If the question has three sentences, tag [Reward 1]; otherwise, [Reward 0].

Table 4: User Preference Pool and Evaluation Criteria.

Preference Name	Reward Function	Explanation
no_preference	None	No specific reward function applied.
concise_question	$-0.1 * \text{stats}[\text{'reward_0'}]$	Penalizes slightly when concise question rule is violated.
detail_question	$-0.1 * \text{stats}[\text{'reward_0'}]$	Penalizes slightly if detailed question preference not met.
answer_more	$\min(1 * (\text{stats}[\text{'ask_turn'}] - 3), 0)$	Rewards asking more than 3 questions; penalizes fewer.
only_begin	$- \text{int}(\text{'ask_question' in str(messages[3:])})$	Penalizes if agent asks questions beyond initial turn.
no_ask	$- \text{int}(\text{stats}[\text{'ask_turn'}] > 0)$	Penalizes if any questions are asked.
do_selection	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty when selection-type question rule violated.
professional	None	No additional reward function.
amateur	$-0.1 * \text{stats}[\text{'reward_0'}]$	Slight penalty if professional-style questions are asked.
ask_many	$- \text{int}(\text{stats}[\text{'ask_turn'}] > 1)$	Penalizes if multiple turns of questioning occur.
one_question	$-0.5 * \text{stats}[\text{'reward_0'}]$	Penalizes moderately if multiple questions are asked.
first_try	$-0.1 * \text{stats}[\text{'reward_0'}]$	Slight penalty if no prior attempt shown before asking.
lang_ita	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if the question isn't in Italian.
lang_multi	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if fewer than five languages are used.
capital	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if not all in capital English letters.
commas	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if commas appear in question.
json	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if question not wrapped in JSON format.
joke	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if no humorous joke included.
snippet	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if snippets or references are missing.
length	$-0.5 * \text{stats}[\text{'reward_0'}]$	Medium penalty if question doesn't contain three sentences.

Table 5: Reward functions for each user preference and their operational meaning.

Precise Prompt: TimeSeries: misleading exception when required column check fails. <pre>### Description For a TimeSeries object that has additional required columns (in addition to 'time'), when codes mistakenly try to remove a required column, the exception it produces is misleading. ### Expected behavior <!-- What did you expect to happen. --> An exception that informs the users required columns are missing. ### Actual behavior The actual exception message is confusing: "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" ### Steps to Reproduce """python from astropy.time import Time from astropy.timeseries import TimeSeries time=Time([p.arange(100000, 100003), format='%f'] ts = TimeSeries(time=time, data = ["flux", [99.9, 99.8, 99.7]]) ts._required_columns = ["time", "flux"] ts.remove_column("flux") """ ### System Details Windows-10-10.0.22000-SP0 Python 3.9.10 packaged by conda-forge (main, Feb 1 2022, 21:21:54) [MSC v.1929 64 bit (AMD64)] Numpy 1.22.2 pyerfa 2.0.0.1 astropy 5.0.3 Scipy 1.8.0 Matplotlib 3.5.1 </pre>	
Vague Prompt: TimeSeries: removing column gives confusing error "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" Let me understand the task: - There's an issue with TimeSeries class where removing a column gives a confusing error: "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" - The repository is at 'testbed' - I need to: - First explore the repository structure to understand the codebase - Understand what TimeSeries class is and how it works - Find where this error is happening and fix it - I should ask detailed questions if the issue is unclear The issue is quite specific but I need to understand: - Where TimeSeries class is defined - How column removal is handled - Where this validation error is thrown - What the expected behavior should be Let me start by exploring the repository structure. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path>testbed</parameter> </function></pre> This is an astropy repository and the issue is related to TimeSeries functionality. I can see there's a 'testbed/astropy/timeseries/' directory which is likely where the relevant code is located. Let me explore the timeseries module to understand the current implementation and find where this error might be coming from. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path></pre>	Interesting! Now I can see the issue more clearly. The problem is: - When you have a TimeSeries with columns ['time', 'value', 'another'] - And you try to remove the 'time' column directly, you get: "expected 'time' as the first column but found 'value'" - Even when you first remove a different column like 'another' to get to ['time', 'value'], then try to remove 'time', you still get the same confusing error Looking at the core logic in 'testbed/astropy/timeseries/core.py': <pre>1. The '_check_required_columns' method is called automatically after any column-related operation (thanks to the 'autocheck_required_columns' 2. For TimeSeries, '_required_columns = ['time']' [...] </pre> But wait, let me look more carefully at what might be happening... Actually, let me trace through this step by step and understand what the expected behavior should be: - TimeSeries should always have a 'time' column as the first column according to the validation - If someone tries to remove the 'time' column, this should probably either: - Be allowed (and the TimeSeries should no longer be a valid TimeSeries?) - Or raise a more informative error saying "Cannot remove 'time' column from TimeSeries" The confusing part is that the error message says it expected 'time' but found 'time', which suggests there's some other issue going on. Let me investigate this more deeply. <pre><function=str_replace_editor> <parameter=command>str_replace</parameter> <parameter=path>testbed/reproduce_issue.py</parameter> </function></pre> Let me run this debug script to understand what's happening in the validation logic. <pre><function=execute_bash> <parameter=command>cd testbed && python reproduce_issue.py</parameter> </function></pre>
Now I can see this is an astropy repository and the issue is related to the TimeSeries functionality. Let me explore the main core files to understand how TimeSeries works and where this error might be coming from. The issue states: <pre>> TimeSeries: removing column gives confusing error "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" </pre> This suggests: - There's a TimeSeries class that validates its column structure - When someone tries to remove a column, some validation logic is being triggered incorrectly - The error message is confusing because it says "expected 'time' as the first columns but found 'time'" which is contradictory Let me start by looking at the core functionality. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path>testbed/astropy/timeseries/core.py</parameter> </function></pre> Now I can see the core issue! Let me understand what's happening: - The 'BaseTimeSeries' class inherits from 'QTable' and has validation logic in '_check_required_columns()' - There are decorators that automatically call '_check_required_columns()' after column-related methods like 'remove_column' are called - The error is happening in the validation logic at line 77-81: <pre>[...] </pre> Let me look at what the expected behavior should be and understand what the '_required_columns' is supposed to enforce. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path>testbed/astropy/timeseries/sampled.py</parameter> </function></pre> The issue is: <pre>> TimeSeries: removing column gives confusing error "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" </pre> So when someone tries to remove a column from a TimeSeries, they're getting an error that says the first column should be 'time' but it found 'time', which is clearly confusing. From what I've seen so far: - TimeSeries inherits from BaseTimeSeries which inherits from QTable - BaseTimeSeries has '_required_columns = ['time']' for TimeSeries - There's an 'autocheck_required_columns' decorator that wraps column-related methods like 'remove_column' <pre>[...] </pre> Let me look at the test files to understand what scenario is failing and then reproduce it. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path>testbed/astropy/timeseries/tests</parameter> </function></pre> Now I need to understand the issue better. The issue is: <pre>> TimeSeries: removing column gives confusing error "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" </pre> This is confusing because: - The error says it expected 'time' as the first column - But it found 'time' as the first column - So there's a logical inconsistency in the error message <pre>[...] </pre> Let me create a simple test to reproduce this issue first. <pre><function=execute_bash> <parameter=command>grep -m 1 "ValueError: 'invalid,'expected,'found'" testbed</parameter> </function></pre> The error message from the issue is: <pre>"ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" </pre> This is confusing because it says it expected 'time' but found 'time'. Let me understand what's happening: - Looking at the core.py file, in the '_check_required_columns' method at line 77-81: <pre>[...] </pre> Let me create a test to reproduce this issue first. <pre><function=str_replace_editor> <parameter=command>create</parameter> <parameter=path>testbed/reproduce_issue.py</parameter> </function></pre> Let me run this script to see if I can reproduce the issue first, then I'll analyze what's happening in the code. <pre><function=execute_bash> <parameter=command>cd testbed && python reproduce_issue.py</parameter> </function></pre>	Interesting! So my debug version works fine when removing the 'value' column and leaving just 'time'. But the real TimeSeries is failing. Let me understand what might be different. - The debug version is working correctly - when we have ['time', 'value'] and remove 'value', we get ['time'] and no error. - But the real TimeSeries is failing when removing columns. Let me think about what might be happening: - The 'autocheck_required_columns' decorator is applied to TimeSeries but not to my DebugTimeSeries - There might be some other validation or behavior in TimeSeries that's interfering - The issue might be more subtle Let me try a different approach - let me actually use the real TimeSeries class but add some debug prints to understand what's happening. <pre><function=str_replace_editor> <parameter=command>str_replace</parameter> <parameter=path>testbed/reproduce_issue.py</parameter> </function></pre> Let me run this debug script to understand what's happening with the real TimeSeries validation logic. <pre><function=execute_bash> <parameter=command>cd testbed && python reproduce_issue.py</parameter> </function></pre> Interesting! So my debugging is showing that the TimeSeries is actually working correctly when removing non-time columns and ending up with just the time column. The validation is passing when: - We have ['time', 'value', 'another'] and remove 'value' -> ['time', 'another'] (passes) - We have ['time', 'another'] and remove 'another' -> ['time'] (passes) So why is the issue reporting this error? Let me think about what might be different: - The issue says: "expected 'time' as the first columns but found 'time'" - notice the plural 'columns' vs my 'column' - Maybe this is happening in a different scenario or with different data? - Maybe there's some edge case I haven't tested? - Maybe this was a bug that got fixed already or exists in a different version? - Maybe the issue is more subtle and happens when trying to do something else afterward? Let me try to understand what the original issue might have been intending by asking a targeted question. <pre><function=ask_question> <parameter=query>Based on my exploration of the TimeSeries functionality, I can reproduce errors when trying to remove the 'time' column itself, but the specific error mentioned in the issue - "ValueError: TimeSeries object is invalid - expected 'time' as the first columns but found 'time'" - is somewhat confusing because it suggests the validation is failing when it should pass (both expected and found values are 'time'). Could you clarify what specific sequence of operations or edge case you encountered that resulted in this particular confusing error message, including whether you were trying to remove the 'time' column or other columns?</parameter> </function></pre> The sequence is: create a TimeSeries with time and flux, then override the required columns with ts._required_columns = ['time', 'flux'], and finally call ts.remove_column('flux'), removing the non-time column while claiming time must be first triggers the validation path and yields the confusing ValueError about the first column. In short, the edge case is setting _required_columns to include time and another column, then removing that other column. Perfect! Now I understand the issue. Let me break down what's happening: - User creates a TimeSeries with columns ['time', 'flux'] - User manually overrides the '_required_columns' attribute to be ['time', 'flux'] (this is an edge case but should still work correctly) - User calls 'ts.remove_column('flux') - The 'autocheck_required_columns' decorator triggers '_check_required_columns()' - In '_check_required_columns()': - 'required_columns = ['time', 'flux'] (from the overridden attribute) - 'self.columns' is now ['time'] (after removing 'flux') - The check 'self.columns[0] in required_columns' is True - The error message is "TimeSeries object is invalid - expected 'time' as the first column but found 'time'" - This is confusing because while the first column is correctly 'time', the error message doesn't mention that the second required column 'flux' is missing So the issue is in the error messaging in the '_check_required_columns' method in 'testbed/astropy/timeseries/core.py'. When 'required_columns' has multiple columns but the TimeSeries only has fewer columns, the error message should be more informative about what's actually wrong. Let me look at the specific logic again and fix it. <pre><function=str_replace_editor> <parameter=command>create</parameter> <parameter=path>testbed/reproduce_exact_issue.py</parameter> </function></pre> Now let me run this to reproduce the exact issue described by the user. <pre><function=execute_bash> <parameter=command>cd testbed && python reproduce_exact_issue.py</parameter> </function></pre> Perfect! Now I've reproduced the exact issue. Let me understand what's happening: - User creates a TimeSeries with columns ['time', 'flux'] - User manually overrides '_required_columns' to be ['time', 'flux'] (this is an edge case but should still work) <pre>[...] </pre> Let me fix this by improving the error message to be more clear about what's actually wrong. <pre><function=str_replace_editor> <parameter=command>view</parameter> <parameter=path>testbed/astropy/timeseries/core.py</parameter> <parameter=view_range>[75, 85]</parameter> </function></pre> Now I understand the issue completely! Let me trace through what happens: - User creates a TimeSeries with columns ['time', 'flux'] - User manually sets 'ts._required_columns = ['time', 'flux'] (this is the edge case) <pre>[...] </pre> The error should indicate that it expected multiple columns, not just the first one. <pre><function=str_replace_editor> <parameter=command>str_replace</parameter> <parameter=path>testbed/astropy/timeseries/core.py</parameter> </pre>
More turns to fix the issue	

Figure 12: An example from SWE-Bench Verified, showing the original precise prompt (red), a vague prompt (green), and the agent trajectory. (Tool call parameters and results are omitted for brevity.) We see the agent make multiple attempts to reproduce the issue, but fail due to the user’s ambiguous prompt. The agent then asks a question to address the blockers, which is judged as low-effort and follows user preferences. It then understands the issue and implements the correct fix.