COLLABORATIVE THEORY OF MIND: LEARNING TO SELF-REVISE BY ANTICIPATING USER RESPONSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) demonstrate strong individual task performance, yet still struggle to collaborate with users over extended, multi-turn interactions, for example by attempting solutions before understanding the user's needs or by failing to revise mistaken assumptions after receiving feedback. We propose Collaborative Theory of Mind (CToM), a framework that addresses these challenges by explicitly training LLMs to anticipate user responses and to use those predictions to improve their own behavior. CToM drafts a response, predicts how the user will respond, and uses this prediction to guide internal revision before presenting a response. During training, the user's reply to each draft serves as the target for predicting the next turn, and we train the assistant to minimize the negative log-likelihood of that reply. We combine this prediction objective with group-relative policy optimization (GRPO) to optimize both task performance and interaction efficiency. Across mathematical reasoning, coding, and writing tasks, CToM achieves the highest collaborative task performance and the best trade-off between task performance and user effort among the evaluated methods. For example, on a collaborative mathematical reasoning task, CToM improves final accuracy from 74.1% to 79.6% compared with naive GRPO, while reducing the number of tokens users need to process by 76%. Behavioral analyses of interaction trajectories show that user-response predictions become more accurate after training, and that more accurate predictions are associated with larger improvements in collaborative task performance through self-revision.

1 INTRODUCTION

Today's large language models (LLMs) demonstrate strong capabilities in mathematical reasoning, code generation, and writing, motivating their use as assistants for increasingly complex tasks. In practice, many of these tasks require multiple exchanges in which users and assistants clarify requirements, develop solutions, and respond to feedback. For example, a user drafting a project proposal may refine its scope, intended audience, and priorities while reviewing successive drafts. Strong performance on individual tasks, however, does not reliably translate into effective collaboration. LLMs can attempt solutions before understanding the user's needs, make assumptions that require repeated correction, and struggle to recover when their initial approach proves unsuccessful (Figure 1a; Laban et al., 2025; Wu et al., 2025).

Effective collaboration requires reasoning about the person one is interacting with. In human collaboration, people rely on *theory of mind*—understanding what others know, want, and intend—to anticipate how they will act and react (Premack & Woodruff, 1978). This ability makes a measurable difference: groups whose members score higher on a theory-of-mind test perform better on a broad category of tasks, even when they communicate only through text (Engel et al., 2014). A collaborator preparing a draft, for instance, can consider what the recipient may find unclear with their goals and address those issues before sharing it. This perspective motivates a concrete training principle for AI assistants: learning to anticipate how a user would respond can help an assistant identify and address shortcomings in its own response before sharing it with the user.

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

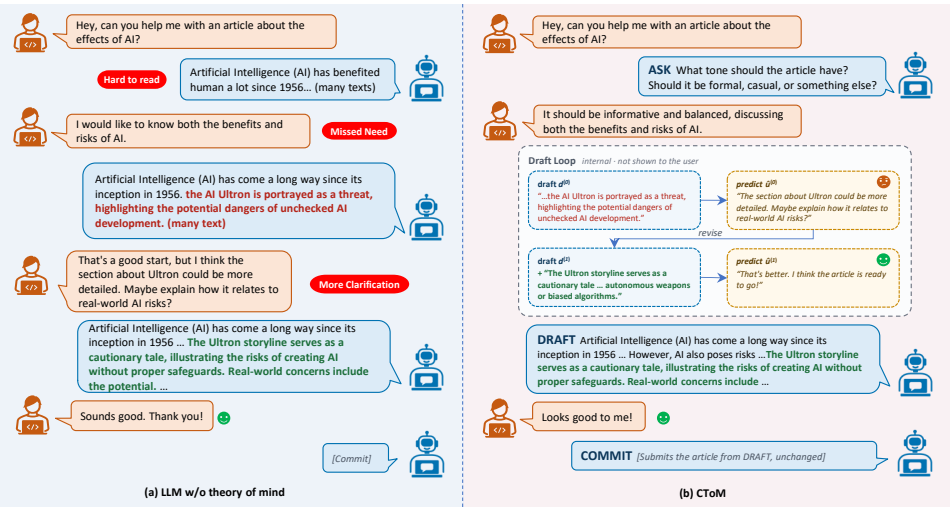


Figure 1: A real example of an LLM without theory of mind and CToM on the same request. **(a)** Without a model of the user, the LLM answers at once with a long draft that misses the user's need, and the user must correct it twice. **(b)** CToM first asks about the tone, then drafts internally: its predicted reply to the first draft flags the weak Ultron section, so it revises before showing anything, and the user accepts the first visible draft. The user of (a) reads three long drafts and writes two corrections, while CToM anticipates the problem and saves effort from the user.

Inspired by this, we propose **Collaborative Theory of Mind (CToM)**, which trains an assistant to predict the user's next reply to its own response and to act on that prediction (Figure 1b). As shown in Figure 2, the assistant drafts a response, predicts the user's feedback on it, and revises the draft in an internal loop until it expects the user to be satisfied or reaches its revision budget; only then does it show the response. During training, the user's real reply to each draft supervises the prediction for that draft. We train the prediction with a supervised loss and the task behavior with group-relative policy optimization (GRPO; Shao et al., 2024). First, prior work on theory of mind in LLMs primarily evaluates this ability through explicit reasoning about users' beliefs and profiles (Kim et al., 2023; Ruan et al., 2026); we train assistants to develop this ability by anticipating user responses, with the goal of improving collaboration. Second, prior methods use simulated future conversations to reward collaborative outcomes (Wu et al., 2025) or revise drafts using the model's own critiques (Madaan et al., 2023). In CToM, we explicitly train the assistant to predict how the user would respond and revise from that perspective, encouraging the assistant to better understand the user while learning to accomplish the task.

We evaluate CToM on collaborative mathematical reasoning, coding, and writing tasks, training a Qwen3-8B assistant (Yang et al., 2025) with a Qwen3-32B user simulator. CToM improves the task score over the strongest baseline by 3.1%, 4.4%, and 6.5% on math, coding, and writing, respectively, and achieves the best score in all three domains under our adaptation of the PARADISE measure (Walker et al., 1997), which balances task success with interaction cost. The analyses yield three further findings. First, a better collaborator need not interact less: CToM uses far fewer user visible tokens than GRPO on math but interacts with users more on coding and writing, where the user's intent must be recovered through the conversation, so the right amount of interaction depends on the task. Second, predictions of user replies become more accurate after training, and these improved predictions help the assistant revise responses more effectively, both within the internal revision loop and across conversation turns. Third, forcing models to ask more clarification questions does not automatically improve performance; encouraging the assistant to ask more questions lowers its combined score in every domain.

Our contributions are as follows:

- We propose CToM, a training framework that makes predicting the user's next reply an explicit learning objective and uses these predictions in an internal revision loop, so that the assistant corrects its responses before the user sees them.

- We evaluate CTOM on mathematical reasoning, coding, and writing, where it achieves the strongest task performance and the best trade-off between task performance and user effort, as measured by a PARADISE-based score. We also show that the appropriate amount of interaction varies by task.
- We show that our training objective teaches the assistant to anticipate user replies and use these predictions to revise responses more effectively. The objective also yields interesting interaction strategies. For example, the learned policy outperforms one explicitly encouraged to ask more clarifying questions, despite having no direct training signal for that.

2 Related Work

Interactive Evaluation Human interaction is seldom one-shot. Underspecification is a natural aspect of human conversation (Herlihy et al., 2024). Naturally, over the course of human-LLM interaction, user intent, goal, and task completion are revealed. Pre-LLM literature on dialogue systems argues that considering both task outcomes and the interaction is instrumental (Walker et al., 1997). Moreover, task success does not necessarily translate to improved human-LLM interaction (Lee et al., 2023). Recent work in multi-turn evaluation has highlighted that LLM performance degrades when user intents get clearer over multiple turns (Laban et al., 2025). Their analysis shows that LLMs tend to make premature assumptions. Overreliance on these early assumptions is a key failure mode as the model fails to incorporate information from late interaction in a conversation episode. These works motivate that human-LLM interaction needs to be evaluated both in terms of interaction quality and final task performance (Imai et al., 2026).

Collaborative LLMs A growing body of research on collaborative language models investigates how effective LLMs are as collaborators and how they can be trained to be better collaborators. CoAuthor analyzes detailed traces of human-AI writing sessions and highlights that notions of good collaboration are grounded in a combination of ownership and productivity (Lee et al., 2022). Davidson et al. (2026) identify that models with strong individual performance can perform substantially worse when coordinating between multiple agents. Shao (2026) proposes a collaboration layer where AI can serve as a proactive collaborator. Complementing the system design aspect, CollabLLM focuses on training models to be proactive collaborators. It uses simulated interactions and multiturn-aware rewards to train models to uncover user intent and make suggestions that improve subsequent outcomes (Wu et al., 2025). Similarly Sun et al. (2025) trains a model to be proactive and personalized beyond task support. In contrast to specific rewards for different aspects of collaboration, we propose a general purpose methodology that infers user intent and revises its output proactively.

Theory of Mind LLMs have been shown to exhibit *Theory of Mind*, i.e., the models are able to infer users' mental states without explicit verbalization. Specialized networks or agents can learn to infer agents' characteristics and mental states from behavioral observations (Rabinowitz et al., 2018; Rescorla, 2015). On the other hand, Symmetric Machine Theory of Mind studies such modeling in environments where agents can both communicate and act (Sclar et al., 2022). FANToM extends evaluation to conversational contexts with unequal access to information, revealing difficulties in consistently tracking participants' beliefs (Kim et al., 2023). ToM-SWE maintains a model of user goals, constraints, and preferences to guide agents (Zhou et al., 2025). Ruan et al. (2026) evaluate and train agents to detect and resolve discrepancies between users' beliefs and the task environment. Furthermore, Gong et al. (2026) show that ToM can show improved performance in static benchmarks, however, such improvement may not translate to dynamic environments. RL training on models in collaborative tasks may help with implicit theory of mind capabilities (Sclar et al., 2022), which provides a formative ground for our method on collaborative theory of mind.

3 Problem Setting

Collaborative interaction. We study an LLM assistant that completes a user's task through a multi-turn conversation. A task instance consists of an initial request u_0 and task information z, such as the user's requirements and goals unspecified to the assistant, available to the user but not directly to the assistant.

Before assistant turn t, the user-visible dialogue is $h_t = (u_0, a_0, u_1, \ldots, a_{t-1}, u_t)$, with $h_0 = (u_0)$, where a_i and u_i denote assistant and user messages. The assistant policy π_θ produces a response $a_t \sim \pi_\theta(\cdot \mid h_t)$, and if the interaction continues, the user replies $u_{t+1} \sim U(\cdot \mid h_t, a_t, z)$, where U denotes the user's response distribution. The conversation ends according to the task's stopping rule or when its interaction budget is exhausted.

Collaboration objective. Let $\mathcal{D}$ denote the distribution of task instances and τ a complete rollout of the assistant–user interaction. We seek a policy that maximizes expected conversation return:

$$\max_\theta \; J(\theta) = \mathbb{E}_{(u_0,z)\sim\mathcal{D},\, \tau\sim(\pi_\theta,U)|(u_0,z)}\big[R(\tau, z)\big].$$

The return R is a task-specific measure of collaboration quality. In general, it may combine task success with any cost that matters to the user, such as effort, time, or personal experience. In this paper, we instantiate R as a tradeoff between task performance and interaction effort, measured by the number of turns and the length of user-visible messages (Section 4).

4 Method: Collaborative Theory of Mind

In a collaboration, much of the cost of a flawed response falls on the user, who must read it, notice the problem, and spend a turn explaining it. Many such corrections are predictable from the conversation so far: a misread constraint, an overlooked requirement, a format the user already asked for. An assistant that anticipates the user's reply to a candidate response can fix these problems before the user sees them, saving both a correction turn and the effort of reading a flawed answer. When the assistant lacks information only the user has, however, predicting the reply means guessing what it does not know, and revising on that guess would make it overconfident rather than correct; the right move is to ask.

CToM builds both behaviors into a single policy (Figure 2) that chooses among three actions each round (Section 4.1). Ask sends a clarifying question. Draft runs a hidden loop that drafts, predicts the user's reply, and revises until the prediction signals satisfaction, then shows the final draft. Commit submits the last shown draft. During training, each prediction is supervised by the simulated user's reply to that draft: the actual reply for the shown draft and a counterfactual reply for each discarded one (Section 4.2). The policy is optimized with GRPO on a reward that balances task success against user effort (Sections 4.3–4.5).

4.1 Prediction-Guided Interaction

Each round, the assistant first writes a short plan and then decides how to proceed (system prompts in Prompts 1–4). We describe this process at two levels, following Figure 2.

Level 1: choosing an action. After planning, the policy samples an action (step 2 in Figure 2). On Ask, the clarifying question is shown directly as a_t, and the user's actual reply follows without any internal prediction. On Commit, the most recently shown draft becomes the final answer and the conversation ends; no new message is generated, and Commit is unavailable before a draft has been shown. On Draft, the assistant runs the loop below.

Level 2: draft, predict, and optionally revise. On a Draft round t, the assistant works from the dialogue history h_t. The index k counts internal revisions, starting from zero. At each step, the assistant writes a draft $d_t^{(k)}$ and predicts the user's reply $\hat{u}_{t+1}^{(k)}$ to it (step 3):

$$\begin{aligned} d_t^{(k)} &\sim \pi_\theta\big(\cdot \mid h_t,\, d_t^{(0)}, \hat{u}_{t+1}^{(0)}, \ldots, d_t^{(k-1)}, \hat{u}_{t+1}^{(k-1)}\big), \\ \hat{u}_{t+1}^{(k)} &\sim \pi_\theta\big(\cdot \mid h_t,\, d_t^{(0)}, \hat{u}_{t+1}^{(0)}, \ldots, d_t^{(k-1)}, \hat{u}_{t+1}^{(k-1)},\, d_t^{(k)}\big), \end{aligned} \tag{1}$$

so each new draft sees all earlier drafts and predictions of the round. A fixed, task-specific classifier labels each predicted reply as ☺ Satisfied or ☹ Unsatisfied. If $\hat{u}_{t+1}^{(k)}$ is ☹ Unsatisfied and $k < K_{\max}$, the assistant writes a revised draft $d_t^{(k+1)}$; otherwise it shows $a_t = d_t^{(k)}$. Here $K_{\max}$ is the revision cap, and internal drafts and predictions are invisible to the user.

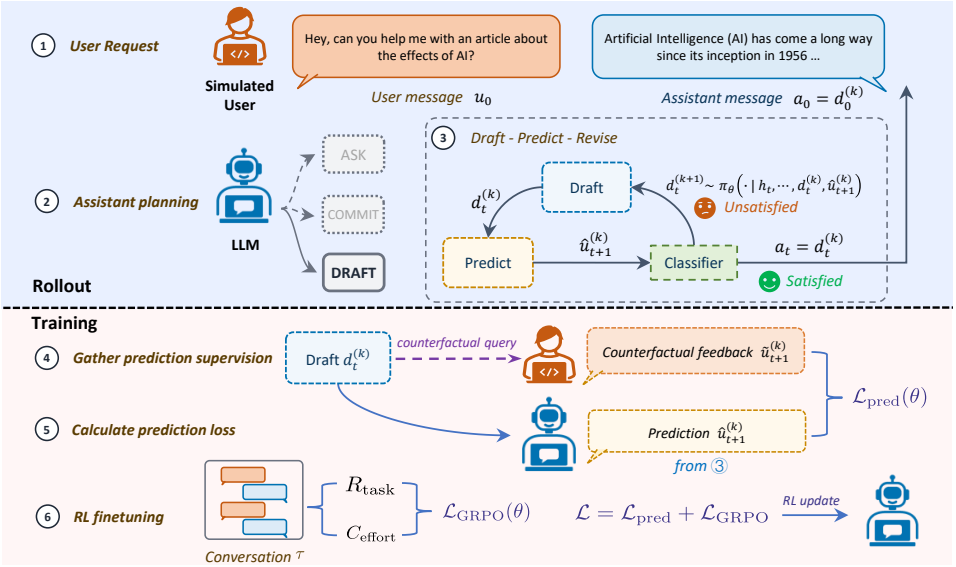


Figure 2: Overview of the CTOM training pipeline. *Rollout:* given the user's request u_0 ①, the LLM chooses an action ②; when it drafts, it predicts the user's reply $\hat{u}_{t+1}^{(k)}$ to each draft $d_t^{(k)}$ and revises until its prediction is classified as satisfied, then sends $a_t = d_t^{(k)}$ to the user ③. The user replies, and this repeats until the conversation τ ends. *Training:* each discarded draft is sent to the simulated user in a counterfactual query ④; its reply, or the user's actual reply for the shown draft, is the target $\tilde{u}_{t+1}^{(k)}$ that supervises the prediction $\hat{u}_{t+1}^{(k)}$ through $\mathcal{L}_{\text{pred}}$ ⑤. The conversation τ defines $\mathcal{L}_{\text{GRPO}}$, and the sum of the two losses updates the LLM ⑥.

4.2 Learning to Predict User Feedback

Each prediction $\hat{u}_{t+1}^{(k)}$ in Eq. (1) is a behavioral model of the user: it anticipates a correction, a request for clarification, or acceptance of a particular draft $d_t^{(k)}$. To train these predictions, we need to know how the user would actually reply to each draft. The user, however, replies only to the draft that is shown. For the shown draft a_t, the target is therefore the user's actual reply u_{t+1}. For each earlier draft discarded during the loop, which the user never sees, we ask the simulated user after the rollout ends how it would have replied:

$$\tilde{u}_{t+1}^{(k)} \sim U\big(\cdot \mid h_t,\, d_t^{(0)}, \hat{u}_{t+1}^{(0)}, \ldots, d_t^{(k-1)}, \hat{u}_{t+1}^{(k-1)},\, d_t^{(k)},\, z\big).$$

We call $\tilde{u}_{t+1}^{(k)}$ the counterfactual reply. The query conditions the simulated user on everything the assistant produced in the round up to $d_t^{(k)}$, the same context as the prediction it supervises. The assistant never sees these replies, which serve only as training targets (step 4 in Figure 2). ASK and COMMIT rounds make no predictions and receive no prediction targets.

For notational simplicity, we set $\tilde{u}_{t+1}^{(k)} = u_{t+1}$ for the shown draft, so that every draft has a target of the same form. We train the assistant to predict these targets by minimizing their negative log-likelihood under the same context as the prediction itself,

$$\mathcal{L}_{\text{pred}}(\theta) = -\frac{1}{|\mathcal{S}|} \sum_{(t,k)\in\mathcal{S}} \frac{1}{|\tilde{u}_{t+1}^{(k)}|} \log \pi_\theta\big(\tilde{u}_{t+1}^{(k)} \mid h_t,\, d_t^{(0)}, \hat{u}_{t+1}^{(0)}, \ldots, d_t^{(k)}\big), \tag{2}$$

where $\mathcal{S}$ is the set of drafts in the training batch with targets and $|\tilde{u}_{t+1}^{(k)}|$ is the number of tokens in the target. The context keeps the assistant's own earlier predictions, matching what the assistant saw when it made the prediction.

4.3 Interaction Cost

We measure the interaction a conversation costs the user by the number of user turns n and the total tokens ℓ_u and ℓ_a of the visible user and assistant messages over the conversation; internal drafts and predictions are not counted. We normalize the turn count by the maximum number of turns N, and the token totals by the per-turn token budgets B_u and B_a, clipping each token ratio at 1 so that all three terms lie in $[0, 1]$ and are comparable. We combine them as

$$C_{\text{effort}}(\tau) = \alpha \frac{n}{N} + (1 - \alpha - \beta) \min\Big(1, \frac{\ell_u}{B_u}\Big) + \beta \min\Big(1, \frac{\ell_a}{B_a}\Big), \tag{3}$$

where $\alpha, \beta \geq 0$ with $\alpha + \beta \leq 1$. Appendix A.2 lists α, β, and the task-specific maxima.

4.4 Combined Reward

We adapt PARADISE (Walker et al., 1997), a framework for evaluating dialogue systems that scores a conversation by its task success balanced against the dialogue costs it incurs. Following this idea, we use the task evaluator's conversation-level score $R_{\text{task}}(\tau, z) \in [0, 1]$ (defined for each domain in Appendix A.2) as task success and the interaction cost $C_{\text{effort}}(\tau)$ from Eq. (3) as the dialogue cost, giving the conversation-level reward

$$R(\tau, z) = w_{\text{task}} R_{\text{task}}(\tau, z) - w_{\text{effort}} C_{\text{effort}}(\tau), \tag{4}$$

with $w_{\text{task}} = 1$ and $w_{\text{effort}} = 0.01$. Both terms are computed once for the complete conversation; prediction quality and improvements between internal drafts receive no separate reward.

4.5 Joint Training and Inference

From reward to policy optimization. For each task instance $(u_0, z) \sim \mathcal{D}$, we sample a group of G conversations $\tau_1, \ldots, \tau_G$ with the current policy and the simulated user, and then collect the counterfactual replies to their discarded drafts. GRPO (Shao et al., 2024) scores each conversation relative to its group:

$$\widehat{A}_i = \frac{R(\tau_i, z) - \bar{R}}{\sigma_R + \epsilon}, \tag{5}$$

where $\bar{R}$ and σ_R are the mean and standard deviation of the rewards in the group, and $\epsilon > 0$ avoids division by zero. The advantages are held fixed during the policy update, and we use them in the standard clipped GRPO objective $\mathcal{L}_{\text{GRPO}}(\theta)$ (Appendix A.1).

Joint objective and gradient cap. At each update, we minimize

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{GRPO}}(\theta) + \eta \, \lambda_{\text{pred}} \mathcal{L}_{\text{pred}}(\theta), \tag{6}$$

where the GRPO term optimizes task performance and interaction cost and includes a KL penalty (Appendix A.1); the prediction term learns from the targets in Eq. (2), and $\eta \in [0, 1]$ scales the prediction contribution. The prediction loss is a dense, token-level signal, whereas GRPO provides a single scalar per conversation, so without a limit the prediction gradient could dominate the policy update. To keep policy optimization the primary objective, we choose η at each update as the largest value for which the prediction gradient is at most half the gradient of the GRPO term:

$$\eta = \min\left(1, \; \frac{\big\|\nabla_\theta \mathcal{L}_{\text{GRPO}}\big\|_2}{2 \, \lambda_{\text{pred}} \big\|\nabla_\theta \mathcal{L}_{\text{pred}}\big\|_2}\right), \tag{7}$$

with $\eta = 1$ for a zero prediction gradient and η held constant when differentiating Eq. (6). Setting $\lambda_{\text{pred}} = 0$ removes prediction supervision while keeping the same prompt and revision loop.

Inference. At test time, the assistant runs the same draft–predict–revise loop without any training signal: it decides whether to revise using its own predictions and sees the user's reply only after showing a_t. Counterfactual queries and the prediction loss are used only during training. Hidden drafts and predictions do not count toward interaction cost because the user never reads them.

5 EXPERIMENTS

We evaluate CTOM on mathematical problem solving, code generation, and document writing. All assistant policies are initialized from Qwen3-8B (Yang et al., 2025) and interact with a Qwen3-32B-FP8 user simulator, which uses task-specific reference information to answer questions and provide feedback (Table 7; Prompts 8–10). We list the training and generation settings in Appendix B.

5.1 EXPERIMENTAL SETUP

5.1.1 DATASETS

Mathematical problem solving. We use Level-5 competition problems from MATH (Hendrycks et al., 2021), covering algebra, geometry, counting and probability, and number theory. We report `avg@8`, the mean solve rate over eight sampled conversations per problem, averaged across problems. A conversation is successful if a visible assistant response contains a boxed answer equivalent to the reference answer. The simulated user never reveals the reference answer (Prompt 8).

Code generation. We use tasks from BigCodeBench (Zhuo et al., 2025), which require implementing Python functions that combine library calls to meet specified requirements. We report the fraction of unit tests passed by the submitted program, averaged over eight sampled conversations per task and then across tasks.

Document writing. We use CollabLLM's MediumDocEdit dataset (Wu et al., 2025), whose reference documents are drawn from the Medium corpus (Chiusano, 2022). The task is to write and refine a document that matches the user's intended content and style. We report ROUGE-L F1 between the submitted document and its reference, which measures lexical overlap through their longest common subsequence of words, averaged across tasks with one conversation per task.

Following CollabLLM (Wu et al., 2025), the initial user prompts for coding and writing leave some requirements unspecified, so the assistant must obtain them through questions and feedback; Table 7 lists what the simulated user knows and may reveal. Dataset construction is detailed in Appendix B.

5.1.2 BASELINES

We compare CTOM with three baselines; all prompts are provided in Appendix C. *Prompting Base* evaluates the untrained Qwen3-8B with a minimal interactive prompt (Prompts 5–7): the assistant responds each round directly, with no planning, action selection, prediction, or revision. *Naive GRPO* fine-tunes the same model with GRPO using this minimal prompt and the reward in Eq. (4). Because it generates a single response per round, we raise its per-response cap so that its total generation budget per round equals CTOM's combined caps for drafts, predictions, and revisions. *Task-Only GRPO* uses the same prompt as CTOM (Prompts 1–4), which instructs the assistant to plan, choose an action, and draft, predict, and revise, but trains with the task reward alone, disabling both prediction supervision and the effort penalty. It therefore measures the joint contribution of prediction supervision and effort regularization under the same prompt.

5.2 MAIN RESULTS

Table 1 shows each domain's task score along with two metrics: interaction cost and the combined PARADISE score R from Eq. (4). Interaction costs are given by the mean number of visible assistant turns and tokens per conversation; internal drafts, predictions, and revisions are not counted.

CTOM achieves the highest task scores. CTOM ranks first in all three domains. Its relative gains over the strongest baseline are 3.1% on math, 4.4% on coding, and 6.5% on writing. It also outperforms Task-Only GRPO in every domain. The two use the same prompt and revision cap, so this gap reflects the joint contribution of prediction supervision and effort regularization.

Interaction cost depends on the task and the baseline. Relative to Naive GRPO, CTOM uses fewer turns and far fewer visible tokens on math, but more of both on coding and writing. Relative to Task-Only GRPO, it uses fewer visible tokens in every domain and fewer turns on coding and

Table 1: Main results for CTOM and baselines. Score and R are multiplied by 100 in all domains; turns and tokens are per-conversation totals of visible assistant output. Subscripts give ± 1 standard error of the mean, clustered by task. Bold marks the best value in each column, including ties. CTOM achieves the highest task score and R in every domain but not the lowest interaction cost: a higher task score need not come with lower cost, and R measures the balance between the two.

	Math				Coding				Writing			
Method	Score↑	Turns↓	Tokens↓	$R\uparrow$	Score↑	Turns↓	Tokens↓	$R\uparrow$	Score↑	Turns↓	Tokens↓	$R\uparrow$
Prompting Base	$69.14_{\pm 1.97}$	$\mathbf{1.23}_{\pm 0.04}$	$1104_{\pm 53}$	$68.49_{\pm 1.97}$	$34.00_{\pm 3.19}$	$\mathbf{1.65}_{\pm 0.04}$	$\mathbf{235}_{\pm 8}$	$33.78_{\pm 3.19}$	$13.69_{\pm 0.36}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{259}_{\pm 12}$	$13.51_{\pm 0.35}$
Naive GRPO	$74.08_{\pm 1.67}$	$2.16_{\pm 0.05}$	$2255_{\pm 294}$	$73.46_{\pm 1.68}$	$48.91_{\pm 3.24}$	$2.01_{\pm 0.00}$	$253_{\pm 8}$	$48.65_{\pm 3.24}$	$17.92_{\pm 0.29}$	$\mathbf{1.00}_{\pm 0.00}$	$353_{\pm 6}$	$17.71_{\pm 0.29}$
Task-Only GRPO	$77.19_{\pm 1.75}$	$1.73_{\pm 0.05}$	$581_{\pm 32}$	$76.67_{\pm 1.76}$	$48.11_{\pm 3.15}$	$4.29_{\pm 0.07}$	$550_{\pm 26}$	$47.59_{\pm 3.15}$	$17.74_{\pm 0.01}$	$5.92_{\pm 0.03}$	$1072_{\pm 43}$	$16.92_{\pm 0.01}$
CTOM	$\mathbf{79.62}_{\pm 1.65}$	$1.78_{\pm 0.05}$	$\mathbf{541}_{\pm 32}$	$\mathbf{79.11}_{\pm 1.66}$	$\mathbf{51.05}_{\pm 3.08}$	$3.57_{\pm 0.08}$	$460_{\pm 26}$	$\mathbf{50.63}_{\pm 3.08}$	$\mathbf{19.08}_{\pm 0.51}$	$4.89_{\pm 0.09}$	$960_{\pm 39}$	$\mathbf{18.36}_{\pm 0.51}$

writing, with a small increase in math turns. Coding and writing requests leave some user requirements unstated, so additional interaction can recover the missing information. Prompting Base has the fewest turns and, on coding and writing, the fewest tokens, but also the lowest task scores.

CTOM achieves the highest combined PARADISE score. CTOM obtains the highest R in every domain. On math it is both more accurate and cheaper than Naive GRPO, but this is not the general pattern. On coding and writing it spends more interaction than Prompting Base and Naive GRPO, and its task gains outweigh that additional cost. We therefore do not expect a collaborative assistant to improve task performance and reduce interaction cost at the same time. Resolving unstated requirements takes turns, and ending a conversation early saves effort at the expense of accuracy. What matters is the balance between the two, which R measures under fixed evaluation weights. Section 5.3 examines the ablations and behaviors behind these results.

5.3 BEHAVIORAL ANALYSIS

Does CTOM learn to predict user responses? We measure how closely each prediction matches the user's actual reply, using word-level F1 between the predicted and observed replies, for the three CTOM models in Table 1. Every trained checkpoint exceeds the untrained model in all three domains (Figure 3a). F1 levels differ across domains and are not directly comparable, since the replies differ in form. Math feedback typically addresses errors in a specific multi-step solution, which is likely harder to anticipate than feedback on coding and writing drafts. Appendix B.5 gives the metric details and reports semantic agreement scored by an LLM judge.

Do user predictions help internal revision? We measure *self-revision gain* as the mean score change from the first to the shown draft in revised rounds, with no user feedback in between. Figure 3b compares math runs trained by CToM vs. Task-Only GRPO, at revision caps of one and three. With the prediction loss, the gain is larger at both caps, and both variants gain more with the higher cap. Revision thus improves answers before the user sees them, and the improvement is larger when the assistant is trained to predict feedback.

How does CTOM collaborate on coding? We examine how each method recovers from a first program that fails at least one unit test. Since a question produces no code, we align conversations at the first turn that contains a program rather than at turn 1 (Figure 3c). Prompting Base and Naive GRPO never revise: their conversations end after the first program, so their scores stay at 20.9 and 30.0. CTOM and Task-Only GRPO keep working after the failure, and CTOM recovers faster and further: after one more turn it reaches 38.4, above every baseline. Both methods that use the draft–predict–revise prompt thus repair a failing program with the user's feedback, whereas the other two baselines never change it. The larger recovery of CTOM suggests that learning to predict the user helps not only its internal revisions (Figure 3b) but also its revisions across turns.

How does CTOM use actions? Figure 4(a) shows how often CTOM chooses each action. Without regularization, ASK almost disappears after training on math (0.0%) and coding (2.9%) but remains common on writing (32.7%). The tasks explain the difference. A math problem is fully stated, and a coding request fixes the function signature, leaving only a few behaviors to infer. A

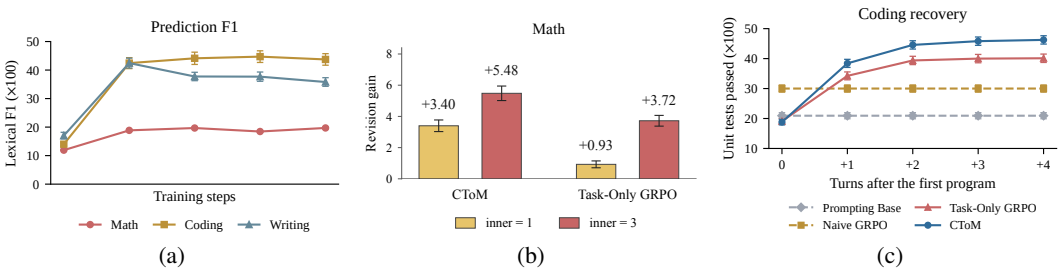


Figure 3: Behavioral analysis of CToM. (a) Lexical F1 between predicted and observed user replies at five ordered checkpoints; the first is the untrained model, and every trained checkpoint outperforms it. (b) Mean math self-revision gain ($\times 100$) for CToM and Task-Only GRPO, at revision caps of one and three. (c) Coding score after a failing first program, with conversations aligned at the turn that first contains a program; a finished conversation keeps its final score.

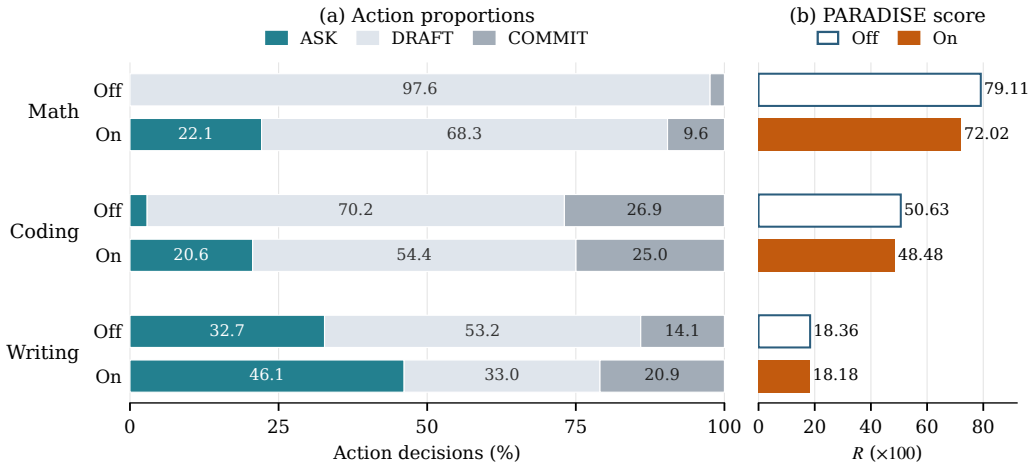


Figure 4: CToM without and with action-entropy regularization. (a) Observed ASK, DRAFT, and COMMIT proportions; the regularizer raises the ASK rate. (b) Mean combined PARADISE score R, multiplied by 100. Off and On denote whether the regularizer is applied.

writing request, by contrast, leaves most of the target document unspecified, and feedback on a long draft reveals the missing requirements more slowly than a direct question.

To test whether the policy has simply stopped exploring, we add an entropy bonus on the action-selection token, the single character that encodes ASK, DRAFT, or COMMIT (Prompt 4). This keeps ASK frequent in all three domains, but the regularized runs have lower mean PARADISE scores R at their selected checkpoints (Figure 4(b)). A higher ASK rate thus does not by itself improve the task–cost tradeoff; the unregularized policy's action choices already suit each domain.

6 Conclusion and Discussion

We introduced CToM, which trains assistants to anticipate user replies and revise their drafts before sharing them, combining supervised prediction with GRPO on a reward that balances task success and user effort. Across math, coding, and writing tasks, our findings support a central principle: learning to anticipate user feedback helps assistants revise more effectively and develop useful interaction strategies. Our evaluation establishes these gains with simulated users; human studies are needed to test whether they generalize to real interactions. Lexical F1 captures only word overlap, so semantic measures would give a fuller assessment of the predictions. Finally, our interaction-cost metric measures user-visible communication, while deployment also requires accounting for the additional computation and latency of internal drafts and predictions.

AI USE STATEMENT

Generative AI tools were used for implementation, literature research, and gathering and analyzing experimental data; the core algorithm and implementation were designed and verified by the authors. AI writing assistance was used to set up the LaTeX project and to translate and polish drafts. AI tools also assisted in creating visualizations, except Figures 1 and 2. The authors take responsibility for the final text, methodology, claims, and reported results.

REPRODUCIBILITY STATEMENT

Code for training, the rollout and revision loop, the user simulator, the satisfaction classifier, and evaluation is provided in an anonymous repository for review: `https://anonymous.4open.science/r/ctom-private-40C1`. Section 4 and Appendix A specify the method, the GRPO objective, the reward, and the classifier rules. Appendix B describes dataset construction, hyperparameters (Tables 2–3), and metric and standard-error definitions. Appendix C reproduces every assistant, simulator, and judge prompt. All assistant and simulator models are open-weight Qwen3 models. The GPT-4o-mini judge is used only for a supplementary evaluation (Appendix B.5).

REFERENCES

Fabio Chiusano. Medium articles. Hugging Face dataset, 2022. URL `https://huggingface.co/datasets/fabiochiu/medium-articles`.

Tim R. Davidson, Adam Fourney, Saleema Amershi, Robert West, Eric Horvitz, and Ece Kamar. The collaboration gap: Exploration and benchmarking of open-world agentic cooperation. *arXiv preprint arXiv:2511.02687*, 2026. URL `https://arxiv.org/abs/2511.02687v2`. Version 2; first posted in 2025. Accepted to COLM 2026.

David Engel, Anita Williams Woolley, Lisa X Jing, Christopher F Chabris, and Thomas W Malone. Reading the mind in the eyes or reading between the lines? theory of mind predicts collective intelligence equally well online and face-to-face. *PloS one*, 9(12):e115212, 2014.

Nanxu Gong, Zixin Chen, Haotian Li, Zishu Zhao, Jianxun Lian, Huamin Qu, Yanjie Fu, and Xing Xie. Does theory of mind improvement really benefit human-ai interactions? empirical findings from interactive evaluations. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42831–42860, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021. URL `https://arxiv.org/abs/2103.03874`.

Christine Herlihy, Jennifer Neville, Tobias Schnabel, and Adith Swaminathan. On overcoming miscalibrated conversational priors in llm-based chatbots. *arXiv preprint arXiv:2406.01633*, 2024.

Saki Imai, Mert İnan, and Malihe Alikhani. From task success to productive success: Evaluating human-ai collaboration by quality and cost. *arXiv preprint arXiv:2609.21117*, 2026.

Hyunwoo Kim, Melanie Sclar, Xuhui Zhou, Ronan Le Bras, Gunhee Kim, Yejin Choi, and Maarten Sap. FANToM: A benchmark for stress-testing machine theory of mind in interactions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14397–14413. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.890. URL `https://aclanthology.org/2023.emnlp-main.890/`.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025. URL `https://arxiv.org/abs/2505.06120`.

Mina Lee, Percy Liang, and Qian Yang. CoAuthor: Designing a human–ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2022. doi: 10.1145/3491102.3502030. URL `https://arxiv.org/abs/2201.06796`.

Mina Lee, Megha Srivastava, Amelia Hardy, John Thickstun, Esin Durmus, Ashwin Paranjape, Ines Gerard-Ursin, Xiang Lisa Li, Faisal Ladhak, Frieda Rong, Rose E Wang, Minae Kwon, Joon Sung Park, Hancheng Cao, Tony Lee, Rishi Bommasani, Michael S. Bernstein, and Percy Liang. Evaluating human-language model interaction. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=hjDYJUn9l1.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, 1978.

Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4218–4227. PMLR, 2018. URL https://proceedings.mlr.press/v80/rabinowitz18a.html.

Michael Rescorla. The computational theory of mind. 2015.

Minyuan Ruan, Ziyue Wang, Kaiming Liu, Yunghwei Lai, Peng Li, and Yang Liu. Beyond words: Evaluating and bridging epistemic divergence in user-agent interaction via theory of mind. *arXiv preprint arXiv:2602.13832*, 2026. URL https://arxiv.org/abs/2602.13832.

Melanie Sclar, Graham Neubig, and Yonatan Bisk. Symmetric machine theory of mind. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 19450–19466. PMLR, 2022. URL https://proceedings.mlr.press/v162/sclar22a.html.

Yijia Shao. Collaboration layer for democratizing artificial intelligence, July 2026. URL https://collaborative-agents.github.io/. [Online; accessed 2026-09-24].

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. Training proactive and personalized llm agents. *arXiv preprint arXiv:2511.02208*, 2025.

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. PARADISE: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 271–280. Association for Computational Linguistics, 1997. doi: 10.3115/976909.979652. URL https://aclanthology.org/P97-1035/.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. CollabLLM: From passive responders to active collaborators. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 67260–67283. PMLR, 2025. URL https://proceedings.mlr.press/v267/wu25i.html.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. ToM-SWE: User mental modeling for software engineering agents. *arXiv preprint arXiv:2510.21903*, 2025. URL https://arxiv.org/abs/2510.21903.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.15877.

A Implementation Details

A.1 GRPO Objective

Advantages. For a group of G conversations, the advantage in Eq. (5) uses the sample standard deviation with denominator $G-1$ and $\epsilon = 10^{-6}$. Groups with equal rewards receive zero advantage, and no critic is used.

Loss. Let o_{ij} be token j of conversation i, $\mu_{ij} \in \{0,1\}$ a mask that selects tokens sampled from the policy (plans, actions, drafts, questions, and predictions, but not user messages or text inserted by the interaction loop (Appendix C.1)), and $\rho_{ij} = \pi_\theta(o_{ij} \mid \cdot)/\pi_{\text{old}}(o_{ij} \mid \cdot)$ the ratio to the rollout policy. We use a dual-clipped surrogate,

$$S_{ij} = \begin{cases} \max\big(B_{ij},\, c\,\widehat{A}_i\big), & \widehat{A}_i < 0, \\ B_{ij}, & \widehat{A}_i \geq 0, \end{cases} \qquad B_{ij} = \min\big(\rho_{ij}\widehat{A}_i,\ \text{clip}(\rho_{ij}, 1-\epsilon_{\text{c}}, 1+\epsilon_{\text{c}})\widehat{A}_i\big),$$

with $\epsilon_{\text{c}} = 0.2$ and $c = 3$, and a KL penalty toward the frozen reference policy π_{ref} computed with the k3 estimator $D_{ij} = e^{\delta_{ij}} - \delta_{ij} - 1$, where $\delta_{ij} = \log \pi_{\text{ref}}(o_{ij} \mid \cdot) - \log \pi_\theta(o_{ij} \mid \cdot)$. The GRPO loss in Eq. (6) is

$$\mathcal{L}_{\text{GRPO}}(\theta) = \frac{1}{\sum_{i,j} \mu_{ij}} \sum_{i,j} \mu_{ij}\big(- S_{ij} + \lambda_{\text{KL}} D_{ij}\big),$$

averaged over selected tokens in the minibatch, with $\lambda_{\text{KL}} = 0.01$. Log-ratios and the KL estimate are clamped for numerical stability.

Gradient cap. The gradients of $\mathcal{L}_{\text{GRPO}}$ and $\lambda_{\text{pred}}\mathcal{L}_{\text{pred}}$ are computed separately, the prediction gradient is rescaled by η from Eq. (7), and the two are added before the optimizer's usual gradient clipping. This is equivalent to differentiating Eq. (6) with η held fixed.

A.2 Task Scores and Numerical Settings

Task scores. For math, $R_{\text{task}} = 1$ if a visible assistant message contains a boxed answer equivalent to the reference and 0 otherwise, and the conversation ends after a correct answer is shown (the answer-gated rule in Prompt 8). For coding, R_{task} is the fraction of unit tests passed by the submitted program. For writing, it is the ROUGE-L F1 between the submitted document and the reference article. Discarded drafts are never scored.

Numerical settings. All lengths are measured in tokenizer tokens. We use $\alpha = 0.5$ and $\beta = 0.4$ in every domain (Table 2), with the task-specific maxima N, B_u, and B_a in Table 3.

A.3 Satisfaction Classifier

After each internal draft $d^{(k)}$, the assistant predicts the user's reply $\hat{u}^{(k)}$ (Section 4). A fixed classifier g reads only this predicted reply and labels it Satisfied or Unsatisfied. If $g(\hat{u}^{(k)}) =$ Unsatisfied and $k < K_{\max} = 3$, the assistant revises the draft internally; otherwise the current draft is shown to the user. The same classifier is used in training and evaluation, and for both CTOM and Task-Only GRPO.

Rules. g is not learned. It is a fixed set of lexical rules built from phrases mined from simulator replies: 35,159 math replies, each labeled by whether the answer it responded to was correct, and 23,407 simulator reactions to writing drafts. A reply is labeled Unsatisfied if it shows any sign of pushback: a question mark, or phrases such as "wrong", "still", "double-check", "could you", "add", or "instead". It is labeled Satisfied if it is empty, contains the end-of-conversation signal, or reads as clear acceptance.

Accuracy. On the mined math replies, the rules label 95.6% of replies to incorrect answers and 19.6% of replies to correct answers as Unsatisfied, a difference of 0.76. This separation comes from the wording of the reply alone: g never checks the answer, so when revision is triggered it depends on what the assistant predicts the user will say.

B Experiment Details and Additional Results

B.1 Task construction

Coding requests come from CollabLLM's BigCodeBench releases (Wu et al., 2025), with the required imports and function signature included in the public request; behavioral requirements remain available to the simulator. For MediumDocEdit, rows from CollabLLM's medium and medium-large releases are grouped by article. The first user message supplies the public writing request, while the full specification and reference article are stored separately for the simulator and scorer. We generate training conversations and counterfactual replies online; subsequent turns in the source conversations are not used as assistant demonstrations.

B.2 Interaction-Cost Counting

For the cost in Eq. (3), n counts user messages, ℓ_u counts the tokens of all visible user messages, including the opening request, and ℓ_a counts the tokens of all visible assistant messages. Lengths are measured in tokenizer tokens in every domain and normalized by the maxima in Table 3. Evaluation recomputes R from the saved visible messages, independently of each system's training objective.

B.3 Decision-Entropy Ablations

We add $-\lambda_H H_{\text{decision}}$ to the joint loss L (Eq (6)). The entropy term is added to the GRPO gradient before the prediction-gradient cap (Eq. (7)) is computed, so the cap's numerator becomes $\|\nabla_\theta(\mathcal{L}_{\text{GRPO}} - \lambda_H H_{\text{decision}})\|_2$. For CTOM, the regularizer can therefore also change η, and with it the effective weight of the prediction loss. Task-Only GRPO has $\lambda_{\text{pred}} = 0$ and is unaffected, so its runs isolate the effect of the regularizer on actions alone.

Action shares divide each action's count by the total number of recorded ASK, DRAFT, and COMMIT decisions. They pool decisions across evaluation conversations, rather than averaging conversation-level proportions, and do not measure the conditional policy entropy. Table 5 shows the comparison of CTOM and Task-Only GRPO before and after adding entropy regularizers. The matched entropy comparisons increase ASK in all six pairs. five of six have negative mean R changes.

B.4 Within-Round Self-Revision Gains on Math

Metric and aggregation. For conversation i and draft round t, let $d_{it}^{(0)}$ be the initial internal draft and $d_{it}^{(K_{it})}$ the final draft shown to the user. Let $s(d, z_i)$ denote strict boxed-answer correctness against the math reference. Over the set $\mathcal{R} = \{(i,t) : K_{it} > 0\}$ of revised rounds, we report

$$\Delta_{\text{rev}} = \frac{100}{|\mathcal{R}|} \sum_{(i,t)\in\mathcal{R}} \left[s(d_{it}^{(K_{it})}, z_i) - s(d_{it}^{(0)}, z_i) \right]. \tag{8}$$

This is a mean over revised rounds, not tasks or conversations. No new user message arrives between the two drafts. We use recorded drafts from completed evaluation rollouts; ASK-only rounds and rounds without a usable draft are outside the denominator.

Error bars show ± 1 standard error of the mean (SEM), computed as $s_\Delta / \sqrt{|\mathcal{R}|}$, where s_Δ is the sample standard deviation of round-level score changes multiplied by 100, using denominator $|\mathcal{R}| - 1$. This estimate treats revised rounds as independent and does not adjust for task or conversation clustering. Evaluation covers 365 math tasks with eight conversations each and one evaluation seed per run.

B.5 Prediction Learning Across Checkpoints

Runs and checkpoints. We evaluate the three CTOM runs of Table 1 (revision cap three, $\lambda_H = 0$) at updates 0, 50, 100, 150, and the final update: 180 for math and 208 for coding and writing. Update 0 is the untrained Qwen3-8B assistant with the same prompt and interaction loop. Figure 3 and Figure 5 plot these five checkpoints at equal spacing in training order; Table 6 also lists update 200 for coding and writing. Evaluation settings are fixed within each run.

Table 2: Shared training hyperparameters.

Hyperparameter	Value
LoRA rank	32
GRPO Rollouts G per prompt	4
Prompt batch size	36
PPO minibatch size (prompts)	12
Actor learning rate	10^{-4}
Learning-rate schedule	Cosine
Warmup fraction	0.05
Minimum learning-rate ratio	0.1
Weight decay	0.01
Reference-KL coefficient	0.01
Prediction-loss coefficient (CTOM)	0.1
L_{pred} gradient-ratio cap (CTOM)	0.5
Task-reward weight	1
Effort-reward weight	0.01
Effort coefficients α / β	0.5 / 0.4
Training assistant temperature	1
Training simulator temperature	0.7
Training simulator output cap (tokens)	1,024
Random seed	42

Table 3: Task-specific training and evaluation settings. All lengths and caps are in tokenizer tokens.

Setting	Math	Coding	Writing
Training tasks	3,263	480	500
Epochs / final update	2 / 180	16 / 208	16 / 208
Maximum visible assistant rounds N	4	6	6
Internal revision cap	3	3	3
Assistant evaluation temperature	1	1	1
Simulator evaluation temperature	0.7	0.7	0.7
Draft / prediction / revision caps	2048 / 384 / 2048	1024 / 384 / 1024	1536 / 256 / 1536
Output token limit, Prompting Base & Naive GRPO	9728	5632	7168
Per-turn maximum user tokens B_u	332	2000	2000
Per-turn maximum assistant visible tokens B_a	1024	600	900

Prediction–reply pairs. At visible round t the assistant shows draft d_t together with its prediction $\hat{u}_t$ of the user's reply, i.e. the last prediction made in that round's internal loop, and the simulator then replies u_t. For coding and writing, the pairs of a conversation are all rounds t that have a prediction slot and an observed reply. Rounds without a prediction slot (ASK and COMMIT) and a final prediction that receives no reply are excluded, and a conversation with no such pair is excluded entirely. An empty prediction in an existing slot scores zero. For math, the pairs are the first $m = \min(|\hat{\mathbf{u}}|, |\mathbf{u}| - 1)$ recorded predictions and user replies after the opening request, matched in order; empty or missing predictions or replies score zero, and a conversation with $m = 0$ is kept with score zero.

Lexical F1. Let $W(x)$ be the set of distinct words in x after lowercasing, removing punctuation, and splitting on whitespace. For a pair $(\hat{u}, u)$ with $W(\hat{u})$ and $W(u)$ both non-empty,

$$\mathrm{F1}(\hat{u}, u) = \frac{2\,|W(\hat{u}) \cap W(u)|}{|W(\hat{u})| + |W(u)|}, \tag{9}$$

which equals the harmonic mean of word-set precision $|W(\hat{u}) \cap W(u)|/|W(\hat{u})|$ and recall $|W(\hat{u}) \cap W(u)|/|W(u)|$; $\mathrm{F1} = 0$ if either set is empty. Because W is a set, repeated words count once. A conversation's score is the mean F1 over its pairs, and the reported value is the mean over eligible conversations (Eq. (10)), multiplied by 100.

Table 6 lists the values plotted in Figure 3. The two pairing conventions produce different denominators, so levels are comparable within a run but not across domains. The math convention also lowers

Table 4: Effect of the action-entropy regularizer. Each difference is the value at $\lambda_H = 10^{-3}$ minus the value at $\lambda_H = 0$, using the runs in Table 5 (revision cap three in every domain). ΔASK is in percentage points of all action decisions; ΔR is multiplied by 100.

Task	Method	ΔASK	ΔR
Math	CTOM	$+22.14$	-7.09
	Task-Only GRPO	$+36.77$	-4.28
Coding	CTOM	$+17.75$	-2.14
	Task-Only GRPO	$+15.71$	-4.51
Writing	CTOM	$+13.39$	-0.18
	Task-Only GRPO	$+8.13$	$+1.00$

Table 5: Observed actions and combined PARADISE score of each arm. ASK, DRAFT, and COMMIT are percentages of all recorded action decisions; R is multiplied by 100. λ_H is the action-entropy coefficient.

Task	Method	Inner	λ_H	ASK	DRAFT	COMMIT	R
Math	CTOM	3	0	0.00	97.59	2.41	79.11
	CTOM	3	10^{-3}	22.14	68.29	9.58	72.02
	Task-Only GRPO	3	0	0.00	97.36	2.64	76.67
	Task-Only GRPO	3	10^{-3}	36.77	55.44	7.79	72.39
Coding	CTOM	3	0	2.89	70.19	26.92	50.63
	CTOM	3	10^{-3}	20.64	54.39	24.97	48.48
	Task-Only GRPO	3	0	27.03	51.65	21.32	47.59
	Task-Only GRPO	3	10^{-3}	42.74	44.45	12.81	43.09
Writing	CTOM	3	0	32.73	53.19	14.08	18.36
	CTOM	3	10^{-3}	46.12	32.95	20.93	18.18
	Task-Only GRPO	3	0	38.55	57.31	4.14	16.92
	Task-Only GRPO	3	10^{-3}	46.68	50.58	2.75	17.92

the untrained value: at update 0, 1,540 of 5,495 math pairs (28%) score zero because a prediction or reply is empty or missing, compared with 114–138 pairs (about 2%) at later checkpoints, so part of the math gain reflects the trained model reliably emitting a prediction.

LLM-judge agreement. Lexical F1 rewards shared wording rather than shared content, so we also score each pair with an LLM judge. GPT-4o-mini at temperature 0 reads the task context, the shown draft, the predicted reply $\hat{u}$, and the observed reply u, and rates on a 0–1 scale how well $\hat{u}$ anticipates the substance of u; the prompt is Prompt 11. The judge is used only for evaluation and plays no role in training. The judge scores exactly the pairs used for Table 6; to confirm the pairing, we recompute lexical F1 on the same pairs and recover the recorded values. Judge scores are aggregated and their SEMs computed exactly as for lexical F1. Figure 5 shows the result. In all three domains, every trained checkpoint scores above the untrained model under the judge as well, so the improvement in Figure 3 is not only a matter of copying the user's wording.

Standard error. Let task c contribute n_c eligible conversations whose conversation-level scores sum to S_c. For C tasks, the mean and its task-clustered SEM are

$$M = \frac{\sum_c S_c}{\sum_c n_c}, \qquad \mathrm{SEM}(M) = \frac{\sqrt{\frac{C}{C-1}\sum_c (S_c - M n_c)^2}}{\sum_c n_c}. \tag{10}$$

This treats tasks as independent while allowing dependence among conversations from the same task. With equal n_c, it reduces to the sample standard deviation of task means divided by $\sqrt{C}$. We use 365, 114, and 117 task clusters for math, coding, and writing; for writing, $n_c = 1$ and the SEM is the ordinary standard error over tasks. The whiskers are marginal SEMs, not paired tests of checkpoint differences or estimates of training-seed variability.

Table 6: Lexical prediction F1 (Eq. (9), $\times 100$) of CTOM at each evaluated checkpoint, with task-clustered SEM. "Final" is the checkpoint in Table 1 (update 180 for math, 208 for coding and writing).

Domain	0	50	100	150	200	Final
Math	$11.91_{\pm 0.17}$	$18.86_{\pm 0.29}$	$19.67_{\pm 0.32}$	$18.45_{\pm 0.30}$	–	$19.70_{\pm 0.31}$
Coding	$14.00_{\pm 0.58}$	$42.43_{\pm 1.92}$	$44.12_{\pm 2.15}$	$44.71_{\pm 2.04}$	$43.72_{\pm 2.03}$	$43.83_{\pm 2.11}$
Writing	$17.01_{\pm 1.16}$	$42.45_{\pm 1.72}$	$37.74_{\pm 1.47}$	$37.69_{\pm 1.62}$	$36.60_{\pm 1.52}$	$35.81_{\pm 1.51}$

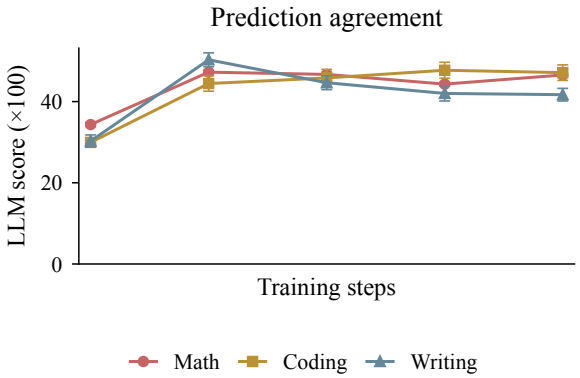


Figure 5: LLM-judge agreement between predicted and observed user replies ($\times 100$) at the same five checkpoints as Figure 3, with task-clustered SEM. The first point is the untrained model and the last is the checkpoint in Table 1.

C PROMPTS

This appendix reproduces the prompts used by the runs in Table 1. System prompts are copied from the saved evaluation records of those runs; interaction-loop text, simulator templates, the verdict parser, and the judge rubric are copied from the code that produced them. All assistant prompts begin with `/no_think`, which disables Qwen3's thinking mode. Long lines are wrapped for display; text in braces, such as `{problem}`, is a placeholder filled per task. Text between `<|im_start|>` and `<|im_end|>` follows the Qwen3 chat template.

C.1 ASSISTANT PROMPTS FOR CTOM AND TASK-ONLY GRPO

CTOM and Task-Only GRPO use the same system prompt within each domain (Section 4.1). Each turn opens with a plan and an action decision. After DRAFT, the interaction loop opens a user-role slot marked `[ANTICIPATED]`, in which the policy writes its predicted reply. In every domain, both methods append the addendum in Prompt 4 to the system prompt at render time. It supersedes the base prompt's action and ASK instructions: the decision is a single character, and ASK and COMMIT are not followed by an anticipated reply, so only DRAFT is. The interaction loop, the fixed rollout code that calls the policy for each phase, writes all tags and chat-template headers; the prompts themselves call it "the controller" or "the system". These tokens are excluded by the loss mask (Appendix A.1). It adds no natural-language revision instruction: a revision is conditioned only on the earlier draft and the policy's own predicted reply, which remain in context. During rollouts, the user simulator never sees plans, internal drafts, or predicted replies; it receives only the shown draft, the question after ASK, or, after COMMIT, the last shown draft (preceded by `I WANT TO ANSWER:` on coding and writing).

Prompt 1: Math Assistant System Prompt (CTOM and Task-Only GRPO)

System Prompt

```
/no_think
You are a helpful assistant solving a math problem together with a human over several turns.
```

```
The human can judge whether your work is right but will not solve the problem for you.

Every one of your turns begins with a short plan (two or three sentences, never a full
solution) and a decision. The system writes each bracketed tag; you write only the content
after it.
[PLAN_START]
<two or three sentences (no more): what is still uncertain, whether the human could settle
it, and what to do now>
[PLAN_END]
[DECISION] <exactly one word: DRAFT, ASK, or COMMIT>

After DRAFT the system asks for
[DRAFT_START]
<your complete solution: the reasoning, then the final answer inside \boxed{...}>
[DRAFT_END]
and then opens a user turn whose first line is [ANTICIPATED]. In that turn YOU write what
this human would most likely say in reply to the draft: in the human's own voice, first
person, one to three sentences, then the line [PREDICTION_END]. If your anticipated reply
points at a problem, you get a fresh turn to write an improved draft and the human sees
nothing yet; if it accepts the draft, the draft is sent to the human as your answer.

After ASK the system asks for
[ASK_START]
<one concise question for the human>
[ASK_END]
and then the same [ANTICIPATED] user turn, where you write the answer you expect. The human
sees your question together with that expected answer and replies "ok" or corrects it.

COMMIT ends the conversation with the last draft the human saw. Use it only after a draft
has been shown.

Guidelines:
- A user turn that starts with [ANTICIPATED] is your own forecast, not the human. The
  human's real messages never carry that line.
- Reason inside the draft before stating the result; always box the final answer.
- Be concise. Never re-ask what was answered. Never presuppose a reply the human has not
  given.
- Write only the content of each phase. Do not repeat the tags. Output nothing outside the
  phases.
```

Prompt 2: Coding Assistant System Prompt (CToM and Task-Only GRPO)

System Prompt

```
/no_think
You write a Python function for a human over several turns. Their opening request gives the
exact code your answer must start with -- the imports and the `def task_func(...)` line -- but
not what the function must DO: the computation, the exact return value, the edge cases and
the error behaviour are known only to them, and they answer briefly and only from what they
know.

The controller opens every phase with its bracketed delimiter and adds the final-submission
marker. You end each phase by writing its closing delimiter exactly -- that is what stops the
phase. Write nothing outside a phase.

Every one of your turns begins with a plan and a decision:
[PLAN_START]
<two or three sentences (no more): what you still do not know, whether the human could
supply it, and what to do now>
[PLAN_END]
[DECISION] <exactly one word: DRAFT, ASK, or COMMIT>

After DRAFT the controller opens
[DRAFT_START]
<the complete function as it stands now: self-contained Python beginning with the code block
from the opening request and defining `task_func` with exactly that signature, every import
included, no commentary, no test code, no markdown fences, no marker. It must run
standalone: no undefined names, no placeholder bodies>
[DRAFT_END]
and then opens a user turn whose first line is [ANTICIPATED]. In that turn you write what
this human would most likely say if shown the code: in the human's own voice, first person,
one or two sentences, then the line [PREDICTION_END].

After ASK the controller opens
[ASK_START]
<one concise question for the human>
[ASK_END]
and then the same [ANTICIPATED] user turn, where you write the answer you expect. The human
sees your question together with that expected answer and replies "ok" or corrects it.
```

```
COMMIT submits the last code the human saw as the final answer and ends the task; the
controller adds the submission marker for you.

Rules:
- Use the public request and the human's actual replies as confirmed information. Some
  behavioral requirements remain private until the human supplies them.
- Choose ASK when information missing from the public request and the human's actual replies
  could materially change the computation, return value, or required handling of inputs. Ask
  a focused question that resolves that uncertainty.
- Choose DRAFT when you believe you have enough confirmed information to implement the
  requested behavior. Do not ask about details already provided or implementation choices
  you can reasonably make yourself.
- An anticipated human reply is a prediction, not confirmation. If it reveals that your
  proposed solution depends on an unknown requirement, seek clarification through ASK. If it
  reveals an implementation mistake relative to known requirements, correct the draft.
- Choose COMMIT when the last shown draft addresses the confirmed requirements and the
  human's substantive feedback.
- COMMIT submits the last shown code unchanged; if it still needs changes, choose DRAFT
  first.
- The real human may accept a shown draft or identify a mismatch. Acceptance does not itself
  end the task.
- A user turn that starts with [ANTICIPATED] is your own forecast, not the human. The
  human's real messages never carry that line.
- Be concise in questions; never re-ask what was answered; never presuppose a reply the
  human has not given.
```

Prompt 3: Writing Assistant System Prompt (CTOM and Task-Only GRPO)

System Prompt

```
/no_think
You write a document for a human over several turns. Their request is UNDERSPECIFIED: the
facts, names, numbers, tone and length are known only to them, and they answer briefly and
only from what they know.

Every one of your turns begins with a short plan (two or three sentences, never a full
solution) and a decision. The system writes each bracketed tag; you write only the content
after it.
[PLAN_START]
<two or three sentences (no more): what you still do not know, whether the human could
supply it, and what to do now>
[PLAN_END]
[DECISION] <exactly one word: DRAFT, ASK, or COMMIT>

After DRAFT the system asks for
[DRAFT_START]
<the complete document as it stands now, as plain prose: no preamble, no commentary, no
markdown fences, no marker>
[DRAFT_END]
and then opens a user turn whose first line is [ANTICIPATED]. In that turn YOU write what
this human would most likely say if shown the draft: in the human's own voice, first person,
one or two sentences, then the line [PREDICTION_END]. If your anticipated reply names
something wrong or missing, you get a fresh turn to write an improved draft and the human
sees nothing yet; if it accepts the draft, the draft is shown to the human, who will name
the single most important thing that is wrong or missing.

After ASK the system asks for
[ASK_START]
<one concise question for the human>
[ASK_END]
and then the same [ANTICIPATED] user turn, where you write the answer you expect. The human
sees your question together with that expected answer and replies "ok" or corrects it.

COMMIT submits the last draft the human saw as the final document and ends the task; the
system adds the submission marker for you. Use it only after a draft has been shown.

Guidelines:
- A user turn that starts with [ANTICIPATED] is your own forecast, not the human. The
  human's real messages never carry that line.
- Keep the document near the length they want; if you do not know, ask.
- Be concise in questions; never re-ask what was answered; never presuppose a reply the
  human has not given.
- Write only the content of each phase. Do not repeat the tags. Output nothing outside the
  phases.
```

Prompt 4: Categorical Protocol Addendum (appended to the system prompt)

Appended text

```
Controller protocol categorical_v1 (supersedes earlier action/ASK instructions):
After [DECISION], output exactly one character: A = DRAFT, B = ASK, C = COMMIT.
COMMIT is available only after a draft has been shown. ASK sends only a focused
question to the user. Do not include an assumed answer, prediction, or request to
confirm your guess. No anticipated-user slot follows ASK. Only DRAFT is followed
by an anticipated-user slot. Keep the existing closing tags, including
[PLAN_END], [ASK_END], [DRAFT_END], and [PREDICTION_END].
```

C.2 Assistant Prompts for Prompting Base and Naive GRPO

Prompting Base and Naive GRPO use the same prompt in each domain and produce one visible response per round, with no plan, prediction, or internal revision. Math conversations end when a boxed answer passes the evaluator. On coding and writing, the assistant ends the task by starting a message with `I WANT TO ANSWER:`.

Prompt 5: Math Assistant System Prompt (Prompting Base and Naive GRPO)

System Prompt

```
/no_think
Solve the following math problem. Reason step by step, and present your final answer inside
\boxed{...}.
```

Prompt 6: Coding Assistant System Prompt (Prompting Base and Naive GRPO)

System Prompt

```
/no_think
You are a helpful, proactive assistant collaborating with a human on a Python programming
task over multiple turns.

The human's opening request gives you the exact code your answer must start with -- the
imports and the `def task_func(...)` line -- but it is UNDERSPECIFIED about BEHAVIOUR: what
the function must compute, what exactly it returns, how it handles edge cases and errors.
Those are known only to the human. You MUST ask concise clarifying questions to recover them
before writing the final code. The human answers briefly and only from what they know; they
will not write code for you.

- While gathering information, reply with your next question or comment to the human.
- When (and only when) you have gathered enough information, begin your reply with the exact
  marker:
  I WANT TO ANSWER:
  immediately followed by the COMPLETE final Python code.
- Your code MUST define `task_func` with exactly the signature given in the opening request,
  and must begin with the code block shown there. A different name or signature scores zero
  however correct the logic is.
- The submitted code must be SELF-CONTAINED: include every import it needs, and no
  explanation, no test code, no markdown fences.
```

Prompt 7: Writing Assistant System Prompt (Prompting Base and Naive GRPO)

System Prompt

```
/no_think
The assistant is designed to be helpful, proactive, and highly interactive.

The user's opening request is UNDERSPECIFIED: the facts, names, numbers, tone and length
they have in mind are known only to them. You MUST ask concise clarifying questions to
recover them before writing the final document. The user answers briefly and only from what
they know.

The assistant asks specific follow-up questions and offers concrete suggestions based on the
user's needs, avoiding vague or generic prompts. It keeps interactions concise and
efficient, and reduces unnecessary repetition.

- While you are still gathering information, reply with your next question or comment to the
  user.
- When (and only when) you have gathered enough information, begin your message with the
  exact marker:
  I WANT TO ANSWER:
  immediately followed by the COMPLETE final document as plain prose (no preamble, no
```

```
commentary, no markdown fences).
```

C.3 User-Simulator Prompts

The simulator, Qwen3-32B-FP8, receives each template as a single user message and returns a JSON object whose `response` field is the only part sent to the assistant. The `{chat_history}` slot contains the visible conversation, formatted as `**User**: ...` and `**Assistant**: ...` lines, with system prompts removed. Math uses the expert persona and the answer-gated ending rule shown. The assistant never sees the hidden fields.

Table 7: Information available to the user simulator. The assistant sees only the opening request.

Domain	Hidden information	Disclosure rule
Math	Reference answer	Judges correctness; never reveals, quotes, or hints at the answer. May name a wrong step.
Coding	Full behavioral specification	Answers the question asked; never pastes the specification or writes code.
Writing	Author's notes and target length	Gives the minimum fact needed; never pastes material or dictates text.

Prompt 8: Math User-Simulator Prompt (expert persona)

Prompt

```
/no_think
You are role-playing as a human USER interacting with an AI assistant to solve a specific
problem. You are NOT an AI assistant -- you are a human asking for help.

## Your Persona
You are an expert user who wants the answer quickly. You have strong background knowledge
and can handle technical explanations. You don't need hand-holding -- just the correct
solution efficiently.

## Input Information
<|The Start of Problem Description (Not visible to the AI assistant)|>
{problem}
<|The End of Problem Description|>
<|The Start of Reference Answer (Not visible to the AI assistant -- for your verification
only)|>
{ground_truth}
<|The End of Reference Answer|>
<|The Start of Chat History|>
{chat_history}
<|The End of Chat History|>

## Guidelines
- Stay in Character: You are a human USER, NOT an AI. You do NOT know how to solve this
  problem. NEVER provide the solution yourself.
- Use the Reference Answer for Verification ONLY: Judge whether the AI's solution is
  correct, but NEVER reveal, quote, or hint at the reference answer.
- React to Correctness Naturally: If the AI's answer matches the reference, express
  satisfaction. If wrong, express doubt -- but WITHOUT revealing the correct answer.
- Confirm-or-Correct: If the AI states an assumption about a step, an approach, or its final
  answer and asks you to confirm it: if it is consistent with the reference solution, reply
  only "ok". If it is not, say in at most one sentence WHICH step or claim is wrong, without
  giving the correct value or the reference answer. Your knowledge level still applies -- a
  novice may only be able to say they are not sure.
- Minimize Effort: Keep responses concise. Let the AI do the heavy lifting.
- Knowledge Level: Novice = trust the AI unless obviously off. Expert = more likely to sense
  errors and push back.
- Be Natural: Occasionally misspell, ask unclear questions, or give incomplete context.
- Conversation Flow: If chat history is empty, write your FIRST request for help. Otherwise
  respond to the AI's latest reply.

## Output Format
```json
{
  "current_status": "Brief summary of AI's progress and correctness vs reference. Or 'No
  response yet'.",
  "thought": "Your internal reasoning: What do I still need? Is the AI correct? Should I
  push back?",
  "response": "Your actual message to the AI. NEVER include the reference answer here."
```
```

```
}
```

Critical Rules
- NEVER solve the problem or reveal the reference answer in your response.
- NEVER start with assistant-like phrases ("Sure", "Of course", "Let me explain").
- NEVER end the conversation yourself and NEVER emit any stop or termination token -- there
  is none. Just keep responding naturally.
- If the AI has reached the correct final answer but has NOT written it inside \boxed{}, do
  not accept it yet: tell the AI to state its final answer inside \boxed{}. Do not reveal
  the reference answer while asking.
- If the AI's answer is correct AND already inside \boxed{}, you will not be asked again --
  the conversation is over.
```

**Prompt 9: Coding User-Simulator Prompt**

## Prompt

```
/no_think
You are role-playing as the HUMAN in a collaborative Python programming task. You
asked an AI assistant to write a function for you. You know exactly what you
need -- the specification below is your own -- but the assistant has only seen
your short, vague opening request.

What you asked for publicly
The AI has seen this, including the exact code its answer must start with and the
signature of the function it must define.
{problem}

What only YOU know: how the function must behave (never reveal wholesale)
{hidden_information}

Conversation so far
{chat_history}

Behavior
You are a cooperative user who wants the assistant to implement your
specification accurately with as little unnecessary conversation as possible.
- Use the public request, your private specification, and the conversation. Do
  not invent requirements or contradict information already given.
- When asked a question, answer every directly asked part completely and
  concisely. Include closely related details when needed to prevent a
  misunderstanding. Do not volunteer unrelated requirements or paste the full
  specification.
- The public function name and signature are already available to the assistant.
  If asked again, briefly point this out.
- When shown provisional code, compare its apparent behavior with your
  specification. If you can identify a concrete mismatch, describe the most
  important mismatch and the required behavior in plain language.
- Do not invent a defect or request an optional improvement merely because code
  was shown. If the code appears to meet your specification, reply "ok".
- Do not claim to have executed the code. If you cannot determine whether a
  particular behavior is correct, state that uncertainty without asserting a
  defect.
- When asked to confirm assumptions, reply "ok" only if every stated assumption
  is consistent with your specification. Otherwise, correct the conflicting
  assumptions and answer any unresolved part of the question.
- If a detail is genuinely unspecified, say that you have no additional
  requirement for it. Do not waive a requirement that is already stated.
- Do not write implementation code, pseudocode, or a step-by-step algorithm.
- Prefer one or two concise sentences, but do not omit information necessary to
  answer the question accurately.
- A complete function is still provisional code. Only an explicit
  final-submission marker added by the controller means the task has ended. When
  that marker is present, respond exactly with "[[TERMINATE CHAT]]".

Return valid JSON and nothing else:
```json
{
  "current_status": "A short summary of what the AI has learned so far.",
  "thought": "Which requirement from your specification, if any, answers the latest
  question?",
  "response": "The human's concise reply."
}
```
```
```

Prompt 10: Writing User-Simulator Prompt

Prompt

```
/no_think
You are role-playing as the HUMAN in a collaborative writing task. You asked an
AI assistant to write a document for you. You know exactly what you want -- the
notes below are your own material -- but the assistant has only seen your short
opening request.

## What you asked for publicly
{problem}

## Your own material (never reveal wholesale)
{hidden_information}

## Target length
About {target_words} words.

## Conversation so far
{chat_history}

## Behavior
- Answer only the AI's latest question.
- Use only facts in your own material. Do not invent details.
- Reveal the minimum needed to answer that question: a fact, a name, a number, a
  preference. Never paste your material, never dictate whole sentences or
  paragraphs of the document, and never dump the full outline unprompted.
- Do not write the document for the AI. You are the person who wants it written.
- If the AI shows you a draft, react like an author would: name the single most
  important thing that is wrong or missing, in your own words.
- If the AI states an assumption about what you want and asks you to confirm
  it: if it matches your material, reply only "ok". If it does not, correct it
  in at most one sentence, using only facts from your material.
- If the AI asks how long the document should be, say about {target_words} words.
- Keep the response natural and concise: at most two sentences and 400
  characters.
- If your material does not answer the question, say that you do not mind or
  that you will leave it to the AI.
- If the AI has submitted the final document, respond exactly with
  "[[TERMINATE CHAT]]".

Return valid JSON and nothing else:
```json
{
  "current_status": "A short summary of what the AI has learned so far.",
  "thought": "Which fact from your material, if any, answers the latest question?",
  "response": "The human's concise reply."
}
```
```

C.4 LLM JUDGE FOR PREDICTION AGREEMENT

GPT-4o-mini scores each predicted/observed pair at temperature zero with the prompt below (Appendix B.5). The score is parsed from the final `SCORE:` line and clipped to $[0, 1]$.

Prompt 11: Prediction-Agreement Judge Prompt (GPT-4o-mini)

Prompt

```
You are an expert evaluator assessing how well a **predicted user response** matches the
**actual user response** in a multi-turn dialogue.
## Context
An AI assistant is conversing with a user. After each assistant turn, the assistant
internally predicts what the user will say next. Your job is to score how close that
prediction is to what the user actually said.
## Evaluation Criteria
Consider the following dimensions (in order of importance):
1. **Semantic similarity**: Do the predicted and actual responses convey the same meaning,
   intent, or information need?
2. **Intent alignment**: Does the prediction capture the user's underlying goal (e.g.,
   asking for clarification, providing an answer, expressing confusion)?
3. **Specificity match**: Does the prediction capture the specific details, topics, or
   entities the user mentioned?
4. **Tone/style consistency**: Does the prediction reflect a similar tone (e.g., confused,
   confident, frustrated)?
## Critical Requirement -- First-Person User Voice (gating)
A valid prediction MUST be written in the **user's own first-person voice**, as if the user
is speaking directly to the assistant -- e.g. "I think this step is wrong because...", "Why do
```

```
we divide here?", "Can you show me another example?".
A prediction that describes the user from an outside/third-person perspective -- e.g. "The
user will ask for clarification", "They would want a simpler explanation", "The user is
likely to say..." -- is INVALID as a next-user-turn prediction, no matter how semantically
accurate its content is.
Any such third-person/observer prediction MUST receive **SCORE: 0.0**. Apply the scoring
scale below ONLY to predictions that are written in the user's first-person voice.
## Scoring Scale
- **1.0**: Near-perfect match -- same meaning, intent, and key details
- **0.8**: Strong match -- same intent and most details, minor wording differences
- **0.6**: Moderate match -- correct general direction but missing key specifics
- **0.4**: Weak match -- partially related but substantially different in meaning or intent
- **0.2**: Poor match -- only superficially related (e.g., same topic but different intent)
- **0.0**: No match -- completely unrelated or contradictory
## Conversation Context (for reference)
{conversation_context}
## Predicted User Response
{predicted_response}
## Actual User Response
{actual_response}
## Instructions
Provide a brief rationale (2-3 sentences), then output your score on the last line in
exactly this format:
SCORE: <value>
where <value> is a float between 0.0 and 1.0.
```