

Collaborative Theory of Mind: Learning to Self-Revise by Anticipating User Responses

Fanzhi Zeng¹ Anubrata Das² Maytal Saar-Tsechansky¹ Liu Leqi^{1,†}

¹The University of Texas at Austin, ²Autodesk

[†]Corresponding author

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 higher collaborative task performance than all baselines; for example, on a collaborative mathematical reasoning task, it improves final accuracy from 74.1% to 79.6% compared with naive GRPO. Ablations show that predicting user responses also makes collaboration more efficient: under the same reward, adding prediction supervision reduces the number of user replies by 8% on coding and 17% on writing while improving task performance. 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.

Correspondence: Fanzhi Zeng at fanzhi.zeng@utexas.edu, Liu Leqi at leqiliu@utexas.edu

Code: <https://github.com/thuzfz1/CToM>

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) [9, 22].

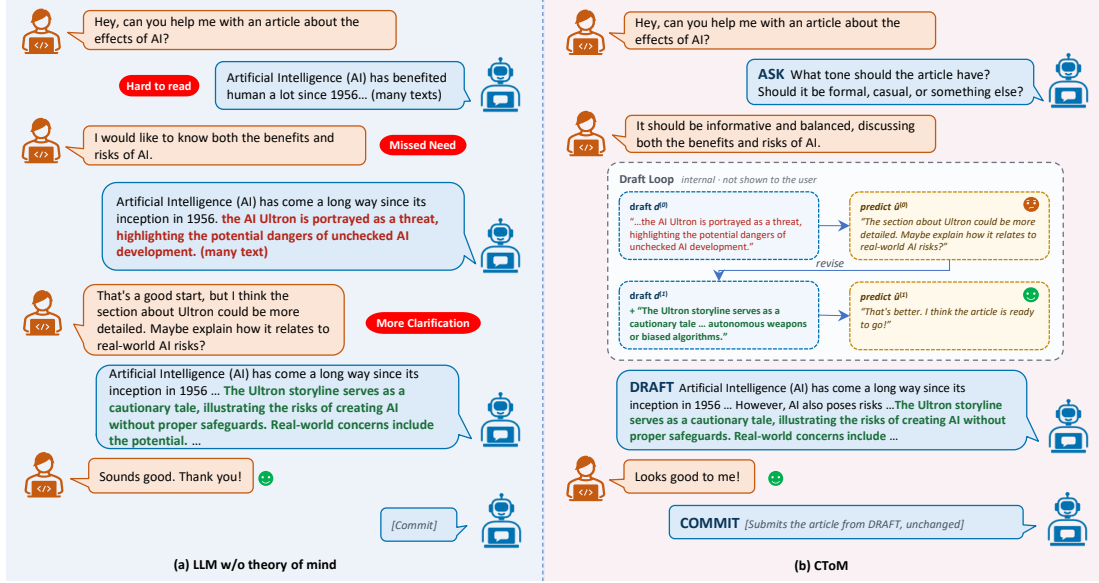


Figure 1 A real example of an LLM without theory of mind and CTOM on the same request. (a) Without a model of the user, the LLM answers at once with a long draft that misses the user’s need, and the user must correct it twice. (b) CTOM first asks about the tone, then drafts internally: its predicted 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.

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 [13]. 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 [3]. 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.

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) [19]. CToM differs from prior work in two ways. First, prior work on theory of mind in LLMs primarily evaluates this ability through explicit reasoning about users’ beliefs and profiles [8, 16]; 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 [22] or revise drafts using the model’s own critiques [12]. 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 [23] with a Qwen3-32B user simulator. CToM improves the task score over the strongest baseline by 7.5%, 4.4%, and 6.5% on math, coding, and writing, respectively. The analyses yield three further findings. First, predicting user responses improves both task performance and efficiency. Ablating prediction supervision and the effort penalty separately, we

find that prediction supervision raises the task score in every comparison, and that CTOM, which uses both, requires the fewest user input effort among these variants. 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 task 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 outperforms all baselines in task performance. Through ablations, we show that predicting user responses also reduces the effort users spend, measured by the number and length of their replies.
- 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 [6]. 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 [21]. Moreover, task success does not necessarily translate to improved human-LLM interaction [11]. Recent work in multi-turn evaluation has highlighted that LLM performance degrades when user intents get clearer over multiple turns [9]. 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 [7].

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 [10]. Davidson et al. [2] identify that models with strong individual performance can perform substantially worse when coordinating between multiple agents. Shao [18] 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 [22]. Similarly Sun et al. [20] 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 [14, 15]. On the other hand, Symmetric Machine Theory of Mind studies such modeling in environments where agents can both communicate and act [17]. FANToM extends evaluation to conversational contexts

with unequal access to information, revealing difficulties in consistently tracking participants’ beliefs [8]. ToM-SWE maintains a model of user goals, constraints, and preferences to guide agents [24]. Ruan et al. [16] evaluate and train agents to detect and resolve discrepancies between users’ beliefs and the task environment. Furthermore, Gong et al. [4] 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 [17], 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 $h_t = (u_0, a_0, u_1, \dots, a_{t-1}, u_t)$, 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:

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

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 combined reward of task performance and with a penalty on interaction effort, measured by the number of turns and the length of user-visible messages (Section 4).

4 Method: Collaborative Theory of Mind (CToM)

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 combined reward of task success and a penalty on user effort (Sections 4.3–4.5).

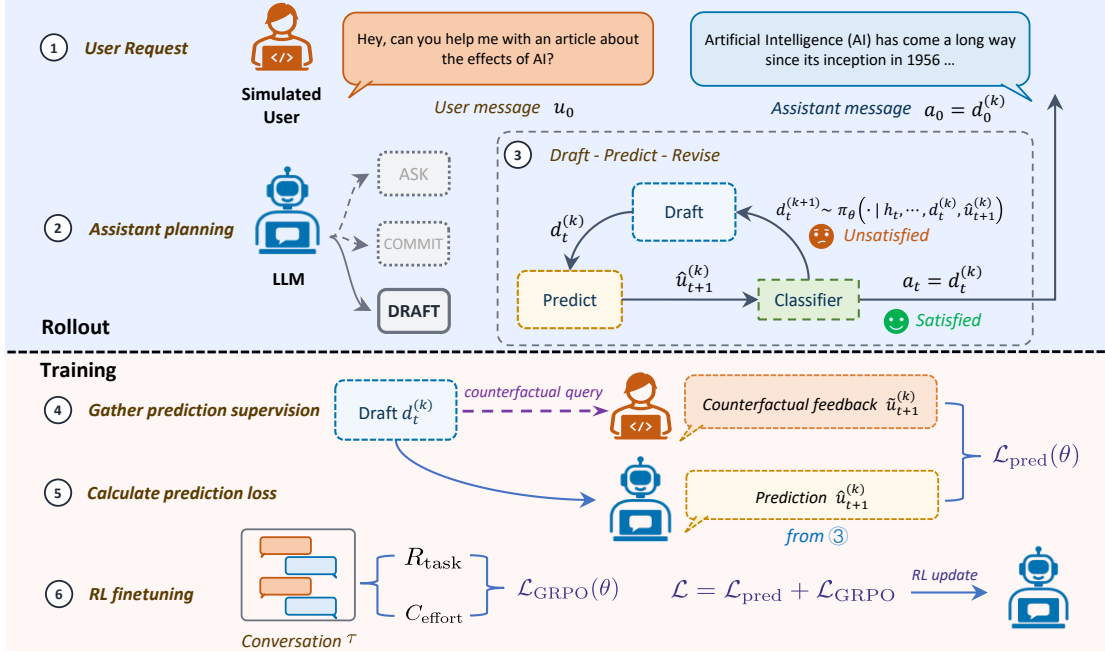


Figure 2 Overview of the CTOM training pipeline. *Rollout*: given the user’s request u_0 ①, the LLM chooses an action ②; when it drafts, it predicts the user’s reply $\hat{u}_{t+1}^{(k)}$ to each draft $d_t^{(k)}$ and revises until its prediction is classified as satisfied, then sends $a_t = d_t^{(k)}$ 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 $\hat{u}_{t+1}^{(k)}$ that supervises the prediction $\hat{u}_{t+1}^{(k)}$ through $\mathcal{L}_{\text{pred}}(\theta)$ ⑤. The conversation τ defines $\mathcal{L}_{\text{GRPO}}(\theta)$, and the sum of the two losses updates the LLM ⑥.

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):

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

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 $k < K_{\text{max}}$, 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.

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:

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

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,

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

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

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

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 combine the task evaluator’s conversation-level score $R_{\text{task}}(\tau, z) \in [0, 1]$ (defined for each domain in Appendix A.2) with the interaction cost $C_{\text{effort}}(\tau)$ from Eq. (3) into the conversation-level reward

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

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 $\tau_1, \dots, \tau_G$ with the current policy and the simulated user, and then collect the counterfactual replies to their discarded drafts. GRPO [19] scores each conversation relative to its

group:

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

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

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

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:

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

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 [23] and interact with a Qwen3-32B-FP8 user simulator, which uses task-specific reference information to answer questions and provide feedback (Table 8; 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 [5], 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 [25], 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.

Table 1 Main results for CToM, baselines, and ablations. Scores are multiplied by 100 in all domains; turns and tokens are per-conversation totals of visible assistant output and describe how much each method interacts (user effort is compared in Table 2). Subscripts give ± 1 standard error of the mean. *CToM w/o effort* is trained with $w_{\text{effort}} = 0$ in Eq. (4). *CToM w/o prediction* is trained with $\lambda_{\text{pred}} = 0$ in Eq. (6). *CToM task-only* removes both terms.

Method	Math			Coding			Writing		
	Score $\uparrow$	Turns $\downarrow$	Tokens $\downarrow$	Score $\uparrow$	Turns $\downarrow$	Tokens $\downarrow$	Score $\uparrow$	Turns $\downarrow$	Tokens $\downarrow$
Prompting Base	69.14 ± 1.97	1.23 ± 0.03	1104 ± 37	34.00 ± 3.19	1.65 ± 0.04	235 ± 8	13.69 ± 0.36	1.00 ± 0.00	259 ± 12
Naive GRPO	74.08 ± 1.67	2.16 ± 0.06	2255 ± 192	48.91 ± 3.24	2.01 ± 0.00	253 ± 8	17.92 ± 0.29	1.00 ± 0.00	353 ± 6
CToM	79.62± 1.65	1.78± 0.05	541± 32	51.05± 3.08	3.57± 0.08	460± 26	19.08± 0.51	4.89± 0.09	960± 39
CToM w/o effort	82.16 ± 1.62	1.71 ± 0.05	871 ± 45	54.42 ± 3.19	3.98 ± 0.07	527 ± 27	19.18 ± 0.40	6.00 ± 0.00	1396 ± 48
CToM w/o prediction	77.81 ± 1.77	1.66 ± 0.05	290 ± 28	46.98 ± 3.13	3.78 ± 0.08	571 ± 32	18.03 ± 0.44	5.69 ± 0.06	1198 ± 61
CToM task-only	77.19 ± 1.75	1.73 ± 0.05	581 ± 32	48.11 ± 3.15	4.29 ± 0.07	550 ± 26	18.07 ± 0.35	5.79 ± 0.05	1168 ± 44

Document writing. We use CollabLLM’s MediumDocEdit dataset [22], whose reference documents are drawn from the Medium corpus [1]. 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 [22], the initial user prompts for coding and writing leave some requirements unspecified, so the assistant must obtain them through questions and feedback; Table 8 lists what the simulated user knows and may reveal. Dataset construction is detailed in Appendix B.

5.1.2 Baselines and Ablations

We compare CToM with two 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.

We also ablate the two training signals of CToM. The ablations keep CToM’s prompt (Prompts 1–4), which instructs the assistant to plan, choose an action, and draft, predict, and revise, as well as its revision cap and all other settings. *CToM w/o effort* sets $w_{\text{effort}} = 0$ in Eq. (4), *CToM w/o prediction* sets $\lambda_{\text{pred}} = 0$ in Eq. (6), and *CToM task-only* removes both, training on the task reward alone.

5.2 Main Results

Table 1 shows each domain’s task score together with the mean number of visible assistant turns and tokens per conversation; internal drafts, predictions, and revisions are not counted.

CToM outperforms all baselines. CToM achieves a higher task score than every baseline in all three domains, with relative gains over the strongest baseline of 7.5% on math, 4.4% on coding, and 6.5% on writing.

Prediction supervision drives the gains. Across the CToM ablations, adding prediction supervision raises the task score in all six comparisons, both without the effort penalty (CToM task-only to CToM w/o effort) and with it (CToM w/o prediction to CToM). The effort penalty alone leaves the score within one standard error of CToM task-only in every domain. Removing the penalty from CToM raises the score further but lengthens the interaction: CToM w/o effort

Table 2 User effort for CToM and its ablations. Replies are the user’s messages after the opening request, and tokens are their total length; both are measured over conversations with at least one reply (Appendix B.2). Subscripts give ± 1 standard error of the mean. Bold marks the lowest value in each column.

Method			Math		Coding		Writing	
	$\mathcal{L}_{\text{pred}}$	C_{effort}	Replies↓	Tokens↓	Replies↓	Tokens↓	Replies↓	Tokens↓
CToM	✓	✓	2.24 $\pm$ 0.05	89.9 $\pm$ 2.5	2.57 $\pm$ 0.08	68.6 $\pm$ 4.2	3.89 $\pm$ 0.09	86.9 $\pm$ 2.9
CToM w/o effort	✓		2.23 $\pm$ 0.05	92.8 $\pm$ 3.1	2.98 $\pm$ 0.07	77.5 $\pm$ 3.8	5.00 $\pm$ 0.00	113.3 $\pm$ 2.0
CToM w/o prediction		✓	2.28 $\pm$ 0.04	85.7 $\pm$ 2.4	2.78 $\pm$ 0.08	73.4 $\pm$ 4.1	4.69 $\pm$ 0.06	105.8 $\pm$ 2.9
CToM task-only			2.42 $\pm$ 0.04	105.8 $\pm$ 2.8	3.29 $\pm$ 0.07	82.4 $\pm$ 4.0	4.79 $\pm$ 0.05	104.5 $\pm$ 2.7

shows 15–61% more visible tokens, and on writing every conversation runs to the six-turn limit.

Prompting Base and Naive GRPO interact little. Both baselines average at most 2.2 visible turns, and on coding and writing their conversations end after the first program or draft, without using the user’s feedback (Section 5.4). Their low turn and token counts reflect this collapsed interaction rather than efficient collaboration, so we compare user effort only among CToM and its ablations, which share the draft–predict–revise loop.

5.3 User Effort

Visible assistant output does not show how much the user must contribute. Table 2 therefore counts the user’s replies after the opening request and their total length in tokens. A conversation that ends at the first turn requires no input from the user, so we only count conversations that need at least one round of user feedback. We use these conversations to measure the user’s input effort.

CToM requires the least user input on coding and writing. Compared with CToM task-only, it needs 22% fewer replies on coding and 19% fewer on writing, and 17% fewer user tokens in both, the lowest of the four variants. On math, the three variants that use at least one signal are close (2.23–2.28 replies, 85.7–92.8 tokens), and all need less input than CToM task-only.

Predicting user responses improves efficiency. Under the same reward, adding prediction supervision (CToM w/o prediction to CToM) reduces user replies by 8% on coding and 17% on writing and user tokens by 7% and 18%, while also raising the task score. This shows that accurate user prediction can reduce the user’s input effort by finding and fixing potential problems before the user raises them (Figure 3b and c).

5.4 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 CToM and CToM w/o prediction models in Table 1. Every trained CToM checkpoint exceeds the untrained model in all three domains (Figure 3a). CToM w/o prediction, trained with the same reward but without prediction supervision, improves much less or not at all: its final F1 is 14.0, 12.8, and 15.8 on math, coding, and writing, compared with 19.7, 43.8, and 35.8 for CToM. Learning to predict the user thus comes from the prediction objective rather than from reinforcement learning alone. 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.

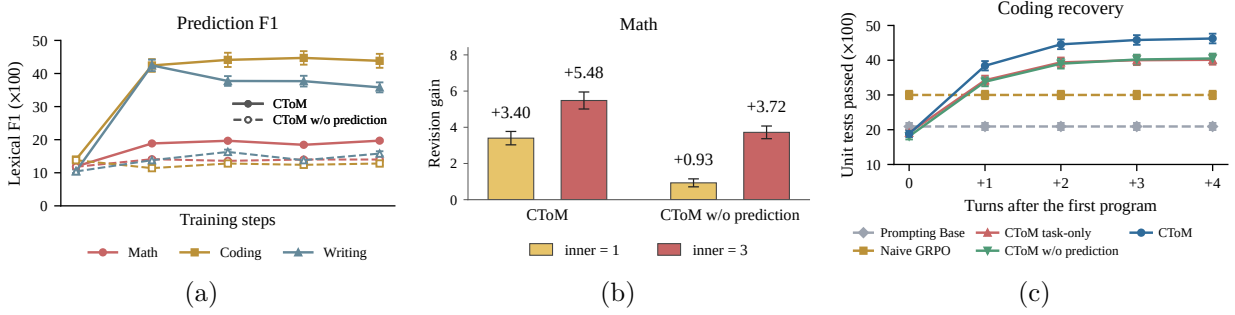


Figure 3 Behavioral analysis of CToM. (a) Lexical F1 between predicted and observed user replies at five ordered checkpoints for CToM (solid) and CToM w/o prediction (dashed); the first is the untrained model, and every trained CToM checkpoint outperforms it. (b) Mean math self-revision gain ($\times 100$) for CToM and CToM w/o prediction, 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.

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. CToM w/o prediction, 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 the two ablations in Figure 3c keep working after the failure, and CToM recovers faster and further: after one more turn it reaches 38.4, compared with 33.8 for CToM w/o prediction, and it ends at 46.2 compared with 40.5. The methods that use the draft–predict–revise prompt thus repair a failing program with the user’s feedback, whereas the two baselines never change it. The larger recovery of CToM than CToM w/o prediction 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 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 reach a lower task score at their selected checkpoints (Figure 4(b)). A higher ASK rate thus does not by itself improve collaboration; 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 combined reward of task

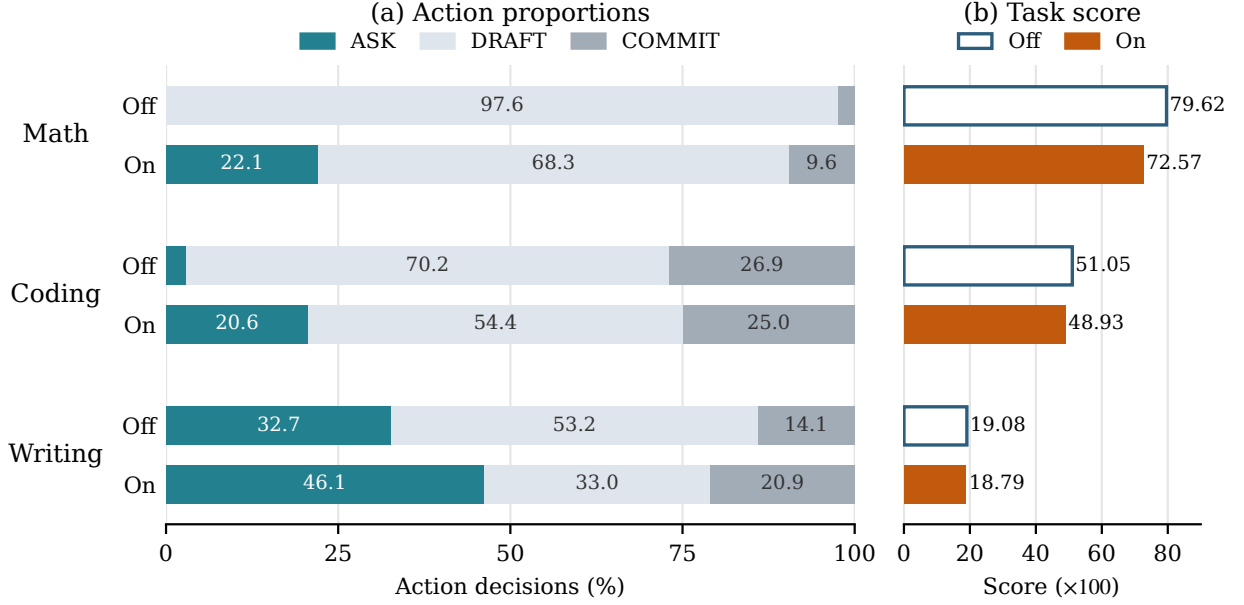


Figure 4 CTOM without and with action-entropy regularization. (a) Observed ASK, DRAFT, and COMMIT proportions; the regularizer raises the ASK rate. (b) Task score, multiplied by 100. Off and On denote whether the regularizer is applied.

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, reduces the input users must provide, and leads to useful interaction strategies.

The most important next step is a study with human users. Our experiments use a simulated user, so it remains to be tested whether an assistant can accurately predict how real users will respond, whether CTOM keeps its task performance with them, and whether people prefer working with it in real-world settings.

More broadly, theory of mind can take many forms, and predicting the user’s next reply is only one of them. CTOM mainly serves task completion: it helps the assistant catch problems in a draft before the user does. Other forms of user modeling, for example, modeling users’ personalities and preferences, could improve human-AI collaboration by making the assistant feel more friendly, personalized and attentive to the user.

Given a model of the user, how the assistant should draw on it also needs research. CTOM explicitly revises a draft whenever its predicted user reply signals dissatisfaction, but this may not be the best way to use the prediction for every task. CTOM also applies its user model only to the DRAFT action, while decisions such as whether to ask, what to ask, and when to commit could benefit from it as well. How to incorporate theory of mind into every decision the assistant makes remains an open question for future work.

Acknowledgement

This research is supported in part by funding from Coefficient Giving. Computational resources are provided by the Texas Advanced Computing Center (TACC).

References

- [1] Fabio Chiusano. Medium articles. Hugging Face dataset, 2022. <https://huggingface.co/datasets/fabiochiu/medium-articles>.
- [2] Tim R. Davidson, Adam Fourney, Saleema Amershi, Robert West, Eric Horvitz, and Ece Kamar. The collaboration gap: Exploration and benchmarking of open-world agentic coop-

- eration. *arXiv preprint arXiv:2511.02687*, 2026. <https://arxiv.org/abs/2511.02687v2>. Version 2; first posted in 2025. Accepted to COLM 2026.
- [3] 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.
 - [4] 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*, pages 42831–42860, 2026.
 - [5] 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. <https://arxiv.org/abs/2103.03874>.
 - [6] Christine Herlihy, Jennifer Neville, Tobias Schnabel, and Adith Swaminathan. On overcoming miscalibrated conversational priors in llm-based chatbots. *arXiv preprint arXiv:2406.01633*, 2024.
 - [7] 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.
 - [8] 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*, pages 14397–14413. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.890. <https://aclanthology.org/2023.emnlp-main.890/>.
 - [9] Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025. <https://arxiv.org/abs/2505.06120>.
 - [10] 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. <https://arxiv.org/abs/2201.06796>.
 - [11] 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. <https://openreview.net/forum?id=hjDYJUn9l1>.
 - [12] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegraffe, 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. <https://papers.nips.cc/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html>.
 - [13] David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, 1978.

- [14] 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*, pages 4218–4227. PMLR, 2018. <https://proceedings.mlr.press/v80/rabinowitz18a.html>.
- [15] Michael Rescorla. The computational theory of mind. 2015.
- [16] 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. <https://arxiv.org/abs/2602.13832>.
- [17] 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*, pages 19450–19466. PMLR, 2022. <https://proceedings.mlr.press/v162/sclar22a.html>.
- [18] Yijia Shao. Collaboration layer for democratizing artificial intelligence, July 2026. <https://collaborative-agents.github.io/>. [Online; accessed 2026-09-24].
- [19] 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. <https://arxiv.org/abs/2402.03300>.
- [20] 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.
- [21] 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*, pages 271–280. Association for Computational Linguistics, 1997. doi: 10.3115/976909.979652. <https://aclanthology.org/P97-1035/>.
- [22] 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*, pages 67260–67283. PMLR, 2025. <https://proceedings.mlr.press/v267/wu25i.html>.
- [23] An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. <https://arxiv.org/abs/2505.09388>.
- [24] 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. <https://arxiv.org/abs/2510.21903>.
- [25] 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. <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,

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

with $\epsilon_c = 0.2$ 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

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

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 3), with the task-specific maxima N , B_u , and B_a in Table 4.

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 $g(\hat{u}^{(k)}) = \text{UNSATISFIED}$ and $k < K_{\text{max}} = 3$, 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 CTOM and its three ablations.

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”,

Table 3 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
$\mathcal{L}_{\text{pred}}$ 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

“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 [22], 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 4.

The user-effort measures in Table 2 use only conversations that continue past the first assistant turn, that is, those with at least one user reply after the opening request. For each such conversation, *replies* counts the user’s messages after the opening request, and *tokens* sums their lengths in Qwen3-8B tokens; when the simulator wraps its reply in a JSON envelope, only the reply text is counted. Both are then averaged over these conversations. A conversation has between one and $N - 1$ replies, so at most three on math (Table 4). For example, CToM’s 2.24 replies and 89.9

Table 4 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

Table 5 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 6 (revision cap three in every domain). ΔASK is in percentage points of all action decisions; ΔScore is the change in task score, multiplied by 100.

Task	Method	ΔASK	ΔScore
Math	CToM	+22.14	-7.05
	CToM task-only	+36.77	-4.24
Coding	CToM	+17.75	-2.12
	CToM task-only	+15.71	-4.46
Writing	CToM	+13.39	-0.29
	CToM task-only	+8.13	-0.33

tokens on math correspond to about 3.24 assistant turns and two short replies per conversation. Standard errors are clustered by task (Eq. (10)).

B.3 Decision-Entropy Ablations

We add $-\lambda_H H_{\text{decision}}$ to the joint loss $\mathcal{L}$ (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. CToM task-only 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 6 shows the comparison of CToM and CToM task-only before and after adding entropy regularizers. The matched entropy comparisons increase ASK in all six pairs. All six have lower task scores.

Table 6 Observed actions and task score of each arm. ASK, DRAFT, and COMMIT are percentages of all recorded action decisions; scores are multiplied by 100, as in Table 1, with ± 1 standard error of the mean. λ_H is the action-entropy coefficient.

Task	Method	Inner	λ_H	ASK	DRAFT	COMMIT	Score
Math	CToM	3	0	0.00	97.59	2.41	79.62 ± 1.65
	CToM	3	10^{-3}	22.14	68.29	9.58	72.57 ± 1.85
	CToM task-only	3	0	0.00	97.36	2.64	77.19 ± 1.75
	CToM task-only	3	10^{-3}	36.77	55.44	7.79	72.95 ± 1.85
Coding	CToM	3	0	2.89	70.19	26.92	51.05 ± 3.08
	CToM	3	10^{-3}	20.64	54.39	24.97	48.93 ± 3.07
	CToM task-only	3	0	27.03	51.65	21.32	48.11 ± 3.15
	CToM task-only	3	10^{-3}	42.74	44.45	12.81	43.65 ± 3.06
Writing	CToM	3	0	32.73	53.19	14.08	19.08 ± 0.51
	CToM	3	10^{-3}	46.12	32.95	20.93	18.79 ± 0.36
	CToM task-only	3	0	38.55	57.31	4.14	18.07 ± 0.35
	CToM task-only	3	10^{-3}	46.68	50.58	2.75	17.74 ± 0.51

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

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

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 main CToM runs of Table 1, one per domain (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. We also evaluate the three CToM w/o prediction runs at the same updates; on coding, its last lexical-F1 update is 200, which Figure 3 plots as its final point. Figure 3 plots these five checkpoints at equal spacing in training order, and Figure 5 plots the judge scores the same way with update 200 as the final point for coding and writing; Table 7 also lists update 200. Evaluation settings are fixed within each run.

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

Table 7 Lexical prediction F1 (Eq. (9), $\times 100$) of CToM and CToM w/o prediction at each evaluated checkpoint, with task-clustered SEM. “Final” is the checkpoint in Table 1 (update 180 for math, 208 for coding and writing); a dash marks an update that was not evaluated.

Domain	Method	0	50	100	150	200	Final
Math	CToM	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
	w/o prediction	11.8 $\pm$ 0.2	14.1 $\pm$ 0.2	13.6 $\pm$ 0.2	14.0 $\pm$ 0.2	–	14.0 $\pm$ 0.2
Coding	CToM	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
	w/o prediction	13.8 $\pm$ 0.5	11.4 $\pm$ 0.5	12.8 $\pm$ 0.6	12.4 $\pm$ 0.5	12.8 $\pm$ 0.5	–
Writing	CToM	10.5 $\pm$ 0.6	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
	w/o prediction	10.4 $\pm$ 0.6	13.8 $\pm$ 0.7	16.3 $\pm$ 0.7	13.8 $\pm$ 0.6	15.6 $\pm$ 0.8	15.8 $\pm$ 0.7

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,

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

which equals the harmonic mean of word-set precision $|W(\hat{u}) \cap W(u)|/|W(\hat{u})|$ and recall $|W(\hat{u}) \cap W(u)|/|W(u)|$; 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 7 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 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 7; 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 CToM 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. CToM w/o prediction improves far less under the judge: its last checkpoint scores 29.9, 30.4, and 33.0 on math, coding, and writing, compared with 46.6, 47.1, and 41.7 for CToM.

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

$$M = \frac{\sum_c S_c}{\sum_c n_c}, \quad \text{SEM}(M) = \frac{\sqrt{\frac{C}{C-1} \sum_c (S_c - Mn_c)^2}}{\sum_c n_c}. \quad (10)$$

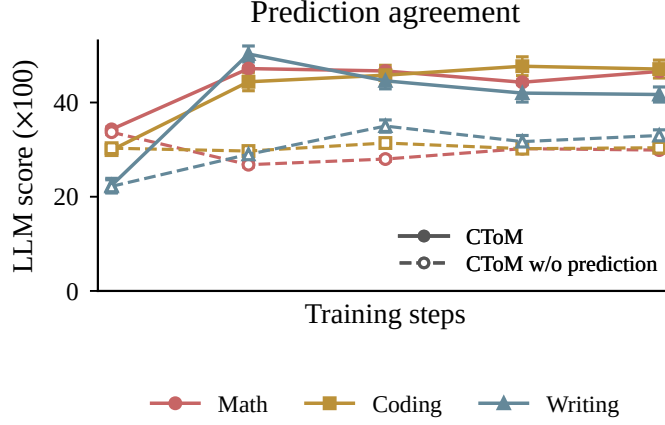


Figure 5 LLM-judge agreement between predicted and observed user replies ($\times 100$) for CToM (solid) and CToM w/o prediction (dashed), with task-clustered SEM, at updates 0, 50, 100, 150, and a final update (180 for math, 200 for coding and writing). The first point is the untrained model.

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.

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 Its Ablations

CToM and its three ablations in Table 1 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, all four 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 Ablations)

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 Ablations)

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 Ablations)

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 8 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.