RHYCO: RHYTHM-CONTROLLED COORDINATION FOR MULTI-AGENT LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

In current multi-agent LLM systems, orchestration happens only between turns, so an inactive agent has no structured channel to intervene before being scheduled, even when it could provide decisive evidence that the ongoing reasoning has gone wrong. We name the missing property *communication rhythm*: the ability to listen, stay silent, and proactively intervene at a self-adaptive pace, as effective teams do. We propose RhyCo (Rhythm-Controlled Coordination), a framework that adds communication rhythm to multi-agent systems (MAS) by moving orchestration inside the turn. RhyCo introduces a dual-channel speaker-listener interface that while the speaker streams reasoning chunks, each listener exposes a probabilistic signal at every chunk and decodes a compact structured whisper only on escalation. A two-tier orchestrator then decides whether to stay silent, retain the speaker, or hand control to a listener, with an MLP gate handling routine chunks and an LLM arbitrator engaged only on escalation. Experiments show that unrestricted listener intervention can degrade MAS performance, while RhyCo turns the same proactive capability into consistent gains on math, code, and multi-hop benchmarks. On MATH-L5, RhyCo-Base uses about $1.2\times$ the output tokens of a single agent and RhyCo-RL uses about $3.3\times$, compared to $7.8\times$ for Multi-Agent Debate and $5.7\times$ for Self-Consistency at five samples.

1 INTRODUCTION

Multi-Agent Systems (MAS) have emerged as a promising paradigm for complex reasoning. By integrating specialized agents, memory and retrieval mechanisms (Shinn et al., 2023; Jiang et al., 2023), tool use (Yao et al., 2022; Qin et al., 2023), and workflow-level orchestration (Wu et al., 2023; Hong et al., 2023), MAS often outperform strong single-agent baselines on tasks such as mathematical reasoning, code generation, and multi-hop question answering. The field has also evolved beyond static interaction protocols toward increasingly dynamic workflows that route information and invoke agents on demand (Liu et al., 2023; Kong et al., 2026; Su et al., 2025). Even the most expressive of these dynamic frameworks, including AFlow (Zhang et al., 2024c), MaAS (Zhang et al., 2025), and Puppeteer (Dang et al., 2025), orchestrate agents via top-down invocation: a controller or learned workflow selects which agent acts next, while every other agent stays inactive until scheduled. Across these designs, one coordination property remains largely absent.

We name this missing property *communication rhythm*. In well-functioning human teams, listening, evaluating, and speaking unfold in a rhythm shaped by who has relevant input and when that input becomes useful. First, communication is inherently **multi-channel and proactive**: while one participant is speaking, others listen and evaluate in parallel, and may raise their own judgment without waiting to be invited. Second, contribution timing is **self-adaptive**: the rhythm of interaction speeds up, slows down, or falls silent in response to the evolving reasoning state. Current MAS protocols rarely support these properties. Even in dynamic workflows, agents contribute only when invoked by a top-down orchestrator. Under this paradigm, an inactive agent that could detect an arithmetic error, recognize the need for verification, or see a more promising route still lacks a structured channel to surface that judgment in a timely manner.

Realizing this rhythm would turn inactive agents into an online assessment channel that catches errors and suggests detours while the reasoning is still unfolding. However, instantiating it in MAS is non-trivial for two reasons. **First, enabling multi-channel proactive listening at scale is hard**.

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.

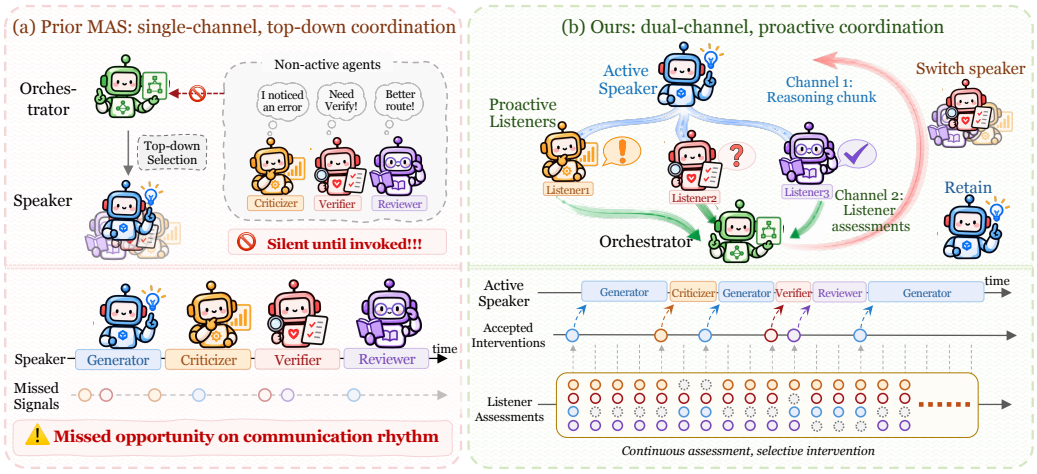


Figure 1: **Prior MAS vs. RhyCo.** Prior MAS (left) uses single-channel, top-down coordination: non-active agents stay silent until invoked, so useful signals are missed. RhyCo (right) uses dual-channel, proactive coordination: listeners continuously assess the speaker, and the orchestrator selectively retains or switches control.

Without being invoked by the orchestrator, every listener in the system continuously monitors the speaker's streaming output and proactively exposes its own judgment, while the decision to intervene is left to the system. This judgment should be concise but informative. If it is not concise, each listener effectively becomes another speaker, inflating the inference cost. If it is concise but uninformative, the resulting signal is too weak to support intervention decisions, and proactive contribution becomes ineffective. **Second, communication rhythm requires coordination that is both global and lightweight.** Rhythm is not a fixed schedule: sometimes the speaker should continue, and sometimes a listener should step in. Choosing between them requires comparing the assessments of listeners, since several may flag different issues or compete to take over at once. Without a principled control, letting each act alone invites premature and conflicting interventions. Although each comparison can be lightweight, it recurs at every chunk, so relying on a heavy centralized controller or multiple rounds of distributed negotiation would quickly erode the efficiency gains of the MAS.

To address these challenges, we propose RhyCo, a framework for rhythm-controlled multi-agent communication. To enable multi-channel communication, RhyCo introduces a dual-channel speaker-listener interface in which speakers produce task-advancing reasoning chunks, while listeners emit probabilistic intervention metrics and decode a compact structured whisper only on escalation. To regulate communication rhythm, RhyCo replaces top-down invocation with a two-tier orchestrator that acts on listener signals. The first tier in the orchestrator is a lightweight MLP gate initialized by rules. It is then trained on counterfactual rollouts that start from the chunk where the reasoning first goes wrong, learning when the system should remain silent and when it should escalate. Once escalation is triggered, the second tier takes over. An LLM arbitrator is invoked to decide whether to retain the current speaker or switch control to a listener. RhyCo has two configurations that share the trained gate and the arbitrator backbone. RhyCo-Base prompts the arbitrator, and RhyCo-RL fine-tunes it with GRPO and LoRA under a reward-minus-cost objective.

Together, these components allow RhyCo to realize adaptive timing and orderly speaker handoff. Experiments show that unrestricted intervention hurts MAS performance, whereas RhyCo turns the same capability into gains: on MATH-L5, RhyCo-Base and RhyCo-RL raise accuracy from 60.9% to 66.9% and 74.1%. They use about $1.2\times$ and $3.3\times$ the output tokens of a single agent, versus $5.7\times$ for Self-Consistency and $7.8\times$ for Multi-Agent Debate. We summarize our contributions as follows:

- **Within-turn orchestration**. We propose RhyCo, a framework that introduces rhythm-controlled communication into MAS and moves orchestration inside the turn. RhyCo enables proactive agent contributions while keeping speaker transitions orderly.
- **Dual-channel speaker-listener interface**. We recast MAS communication so that speakers produce task-advancing reasoning chunks while listeners emit probabilistic intervention metrics and decode a compact structured whisper only on escalation.

- **Lightweight rhythm orchestrator**. We introduce a two-tier orchestrator that dynamically regulates communication rhythm. A first-tier MLP gate, trained on counterfactual rollouts from error onsets, controls self-adaptive communication frequency, and a second-tier LLM arbitrator maintains orderly speaker handoff when listeners compete to intervene.

2 METHOD

Overview. In RhyCo, the agents are partitioned into a *Streaming Speaker* and a pool of *Hybrid Listeners*, both controlled by a *Two-tier Orchestrator*, as shown in Figure 2. At each step, one of four agent roles serves as the speaker and streams its reasoning in 64-token chunks, while the remaining three serve as listeners whose output has two branches. The *upper branch* carries probabilistic metrics that each listener obtains as byproducts of its forward pass, and the *lower branch* carries compact structured whispers that supply the semantic context the metrics lack. The metrics of all listeners, together with a small set of speaker-side metadata, are pooled into a compact 12-dimensional rhythm state that feeds Tier 1, a lightweight MLP gate that runs at every chunk and decides between staying silent and escalating. Whispers are decoded only when the gate escalates, so on routine chunks each listener performs a single prefill pass and generates no tokens. Tier 2, an LLM arbitrator, then reads the pending chunk, the recent history, the listeners' metrics, and these whispers, and either retains the current speaker or switches control to a listener a_i, in which case a_i becomes the speaker for the next chunk. The gate reads only the upper branch because numeric signals suit a cheap classifier, and only the arbitrator reads the lower branch because short structured text suits a language model. Section 2.1 formalizes the runtime loop; Section 2.2 details the speaker-listener interface; Section 2.3 presents the orchestrator and its training procedure.

2.1 RHYTHM-CONTROLLED MAS FORMULATION

Runtime state. We consider a multi-agent reasoning task in which a query q is solved by an agent pool $A = \{a_1, \ldots, a_N\}$. At each step t, one agent acts as the *speaker* spk_t and produces a fixed-budget *pending chunk* c_t of at most B tokens, with $B = 64$ (256 for code). The remaining agents serve as *listeners*, denoted as $L_t = A \setminus \{\text{spk}_t\}$. The system maintains a *history* h_t, which is the sequence of chunks committed so far, and a speaker-local step κ_t, which counts the chunks the current speaker has produced since it last took control. Reasoning terminates when (i) a committed chunk carries a final answer and the orchestrator does not intervene, or (ii) the per-trajectory chunk budget is exhausted, in which case the final answer is aggregated from the committed history.

Communication channels. The system communicates through two channels per step. The *speaker chunk* c_t is streamed by spk_t during decoding and forms the reasoning channel; it advances the trajectory only once committed. Each listener $i \in L_t$ observes (q, h_t, c_t) and produces an *assessment* $p_{i,t}$, which forms the assessment channel and provides evidence for rhythm control.

The upper branches of the listener assessments are aggregated with speaker-side metadata into a compact *rhythm state* that serves as the gate input:

$$z_t = \text{Agg}\big(m_t^{\text{spk}}, \{m_{i,t}\}_{i \in L_t}\big), \tag{1}$$

where m_t^{spk} denotes a small set of speaker-side metadata and $m_{i,t}$ denotes the upper branch of listener i's assessment. Agg is a fixed pooling function, and z_t contains no chunk text, so the gate sees the semantics of the chunk only through the listeners' metrics. The concrete construction of m_t^{spk}, $m_{i,t}$, and z_t is given in Section 2.2 and Appendix B.

Rhythm-control action space. At each step t, the orchestrator selects an action

$$u_t \in \{\text{stay}, \text{retain}\} \cup \{\text{switch}(i) \mid i \in L_t\}, \tag{2}$$

where $\text{switch}(i)$ denotes transferring control to listener $i \in L_t$. Under stay, the system remains silent and commits c_t without invoking the arbitrator. Otherwise, the gate escalates, and the arbitrator selects either retain, which also commits c_t and keeps the current speaker, or $\text{switch}(i)$, which rejects c_t and hands control to listener i. The actions induce the following state transitions:

$$h_{t+1} = \begin{cases} h_t \oplus c_t, & u_t \in \{\text{stay}, \text{retain}\}, \\ h_t, & u_t = \text{switch}(i), \end{cases} \tag{3}$$

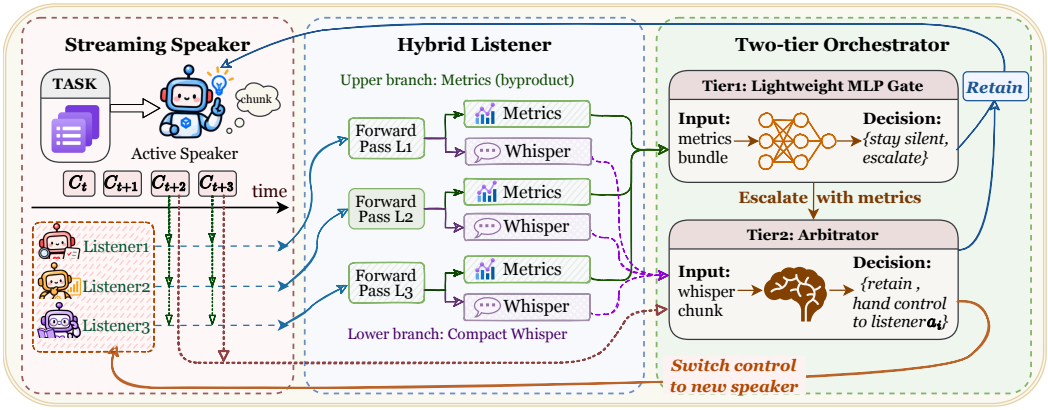


Figure 2: **Method overview.** A Streaming Speaker (left) emits 64-token chunks; three Hybrid Listeners (middle) emit per-chunk metrics (upper branch) and compact structured whispers (lower branch) only on escalation. The Two-tier Orchestrator (right) first passes the metrics through an MLP gate (stay/escalate); on escalation, an LLM arbitrator processes the chunk, recent history, metrics, and whispers and chooses retain or switch to listener a_i.

$$(\text{spk}_{t+1}, \kappa_{t+1}) = \begin{cases} (\text{spk}_t,\ \kappa_t + 1), & u_t \in \{\text{stay}, \text{retain}\}, \\ (a_i,\ 0), & u_t = \text{switch}(i). \end{cases} \tag{4}$$

Here $\oplus$ denotes appending a chunk to the committed history.

Objective. We keep the speaker and listener agents fixed and optimize only the rhythm-control policy Π, which maps what the orchestrator observes at each step to a control action u_t (Section 2.3). Running Π yields a trajectory τ, the sequence of pending chunks, listener assessments, and control actions up to termination. We seek to maximize task quality net of communication and control cost:

$$\max_{\Pi}\ \mathbb{E}_{\tau\sim\Pi}\big[R_{\text{task}}(\tau) - \lambda\, C_{\text{comm}}(\tau)\big], \tag{5}$$

where $R_{\text{task}}(\tau)$ is a task-level reward on the final committed answer, and $C_{\text{comm}}(\tau)$ accounts for listener evaluations, whisper decoding, gate calls, arbitrator calls, and other control overhead. The weight λ governs the efficiency-quality trade-off: heavier orchestration may improve task reward but can erode the efficiency gain from multi-agent coordination.

2.2 Hybrid Speaker-Listener Communication Interface

On the speaker side, reasoning is streamed chunk by chunk through autoregressive decoding. Each chunk boundary serves as an evaluation point for rhythm control, and unless control switches, the speaker resumes decoding from where it stopped. Alongside each chunk, the speaker exposes metadata $m_t^{\text{spk}} = (k_t, p_t^{\min}, r_t, f_t)$: the number of chunks since the last final answer, the lowest token confidence, a self-repetition score against h_t, and a final-answer flag. These features are computed from the generated tokens and their probabilities, without any additional forward pass. On the listener side, at every chunk boundary each listener $i \in L_t$ runs a single forward pass over (h_t, c_t) under a role-specific system prompt that also contains the query q and the whisper format. This pass yields the metric vector $m_{i,t}$, which forms the upper branch of the listener's assessment. If the gate escalates the chunk, the listener additionally decodes a compact structured whisper $w_{i,t}$ as the lower branch, so its assessment is $p_{i,t} = (m_{i,t}, w_{i,t})$ on escalated chunks and $p_{i,t} = m_{i,t}$ otherwise.

The listener's forward pass over c_t yields a role-conditioned predictive distribution at each token, showing how the listener would have continued the same context. The metric vector $m_{i,t}$ exposes this distribution to the orchestrator through three token-level statistics. Let $P_{i,j}$ denote listener i's predictive distribution at the j-th token of c_t, and y_j the token the speaker emitted:

$$\alpha_{i,j} = P_{i,j}(y_j), \qquad H_{i,j} = -\textstyle\sum_v P_{i,j}(v)\log P_{i,j}(v), \qquad \sigma_{i,j} = -\log P_{i,j}(y_j). \tag{6}$$

The agreement $\alpha_{i,j}$, entropy $H_{i,j}$, and surprise $\sigma_{i,j}$ measure how strongly the listener supports the speaker's token, how uncertain the listener itself is, and how unexpected the token is to the listener.

Averaging each statistic over the chunk gives three entries of $m_{i,t}$, and the minimum agreement over the chunk adds a fourth that flags a single sharply disagreeing token. The rhythm state $z_t \in \mathbb{R}^{12}$ combines m_t^{spk} with selected per-listener entries and cross-listener aggregates, arranged in three groups of evidence (Appendix B.3):

$$z_t = \big[\ \underbrace{z_t^{(\mathrm{I})}}_{\text{surface (4)}} \ \big\|\ \underbrace{z_t^{(\mathrm{II})}}_{\text{individual (5)}} \ \big\|\ \underbrace{z_t^{(\mathrm{III})}}_{\text{collective (3)}}\ \big]. \tag{7}$$

Group **I** (*surface*) reflects whether the chunk itself is suspicious on its surface and comprises m_t^{spk}: low token confidence, self-repetition, prolonged absence of a final-answer event, and a final-answer flag. Group **II** (*individual*) reflects whether any single role finds the chunk problematic and contains per-role agreement and entropy values. Group **III** (*collective*) pools the listeners through their disagreement spread, peak entropy, and peak surprise, capturing whether the listeners diverge from one another or any of them raises an alarm.

The two branches serve the concise-but-informative requirement of Section 1. Metrics are cheap enough to compute at every chunk and show whether a chunk looks problematic and which listeners react most strongly, while the whisper adds semantic information as short text that is decoded only on escalation. Each whisper follows the fixed grammar specified in the listener's system prompt:

$$w_{i,t} := [\textsc{role}]\ [\textsc{verdict}]\ [\textsc{confidence}] : [\textsc{reason}], \tag{8}$$

with VERDICT $\in \{\texttt{retain}, \texttt{switch}, \texttt{idle}\}$, CONFIDENCE $\in \{\texttt{high}, \texttt{medium}, \texttt{low}\}$, and REASON a short free-text fragment, such as a step the listener wants to verify. The verdict is only a suggestion to the arbitrator, indicating whether the current speaker should continue (`retain`), the listener should take over (`switch`), or the listener sees no reason to intervene (`idle`). Each whisper has at most 24 tokens, and Appendix B.4 gives its full specification.

2.3 Lightweight Rhythm Orchestrator

We instantiate the rhythm-control policy Π of Section 2.1 as a two-tier orchestrator. At every chunk, the gate reads the rhythm state z_t and chooses between stay and escalation. On escalation, the arbitrator reads the pending chunk, the listeners' metrics and whispers, and the recent committed history, and selects retain or switch(i). Because the gate lets routine chunks pass without invoking the arbitrator, the heavier LLM call is spent only on the chunks the gate flags.

The gate is a small MLP with one hidden layer of 32 units that maps z_t to a probability of escalation, from which the escalation decision is sampled. We first initialize it on simulated data labeled by hand-crafted rules over the rhythm state, for example escalating when self-repetition indicates a stuck trajectory. We then train one gate per benchmark on single-agent trajectories that fail, locating the chunk where the reasoning first goes wrong with an LLM annotator, and labeling the earlier chunks as stay and this onset as escalate. To confirm that intervening at the onset helps, we also branch each trajectory there and let each agent separately continue the reasoning, which additionally supplies training chunks.

We use Qwen3-1.7B as the arbitrator. When the chunk contains a final answer, retaining it ends the reasoning, and switching lets another agent continue. RhyCo-Base uses the pre-trained model with the prompt in Appendix C. RhyCo-RL fine-tunes LoRA adapters of the same model with online GRPO (Shao et al., 2024) while the gate stays frozen. For each training query, we sample $G = 4$ full RhyCo trajectories $\{\tau_k\}_{k=1}^{G}$ under the current arbitrator and minimize

$$\begin{aligned}\mathcal{L}(\theta) = -\frac{1}{G}\sum_{k=1}^{G}\frac{1}{|o_k|}\sum_{t=1}^{|o_k|}\min\Big(\rho_{k,t}\hat{A}_k,\ \mathrm{clip}(\rho_{k,t}, 1-\epsilon, 1+\epsilon)\,\hat{A}_k\Big) \\ + \beta_{\mathrm{KL}}\,\mathbb{D}_{\mathrm{KL}}\big[\pi_\theta \,\|\, \pi_{\mathrm{ref}}\big] - \beta_H\,\mathcal{H}(\pi_\theta),\end{aligned} \tag{9}$$

where o_k denotes the arbitrator tokens generated along τ_k, $\rho_{k,t}$ is the probability ratio of the t-th token under the current and previous policies, and π_{ref} is the pre-trained arbitrator. The advantage $\hat{A}_k$ normalizes the reward of each trajectory within its group,

$$\hat{A}_k = \frac{\mathcal{R}(\tau_k) - \mathrm{mean}_j\,\mathcal{R}(\tau_j)}{\mathrm{std}_j\,\mathcal{R}(\tau_j)}, \qquad \mathcal{R}(\tau) = \mathbb{I}\big[\tau \text{ answers correctly}\big] - \lambda_d\,\frac{T(\tau)}{\bar{T}_d}, \tag{10}$$

Table 1: Performance comparison with prompting, debate, and adaptive workflow baselines. The best and second-best results are shown in **boldface** and underlined, respectively.

Method	Math			Code		Multi-Hop QA	Avg. Gain
	MATH-L5	GSM8K	AIME 2024	MBPP	BigCodeBench	HotpotQA	
Single	0.6091	0.9174	0.1667	0.8035	0.4439	0.456/0.606	0.00pp
CoT	0.6276	0.9181	0.2667	0.8152	0.4404	0.594/0.744	+4.42pp
SC	0.7181	0.9386	0.2667	0.8065	0.4667	0.602/0.748	+6.70pp
MedPrompt	0.6049	0.9234	0.2333	**0.8358**	0.4632	0.580/0.730	+4.07pp
MAD	0.7284	0.9299	0.3333	0.8270	0.4860	0.590/0.747	+8.30pp
AFlow	0.6749	0.9075	0.3000	0.7977	0.4947	0.615/0.764	+6.55pp
MaAS	0.7346	**0.9393**	0.3667	**0.8358**	0.5509	0.598/0.754	+10.48pp
RhyCo-Base	0.6687	0.9371	0.3667	0.8211	0.5491	0.620/0.773	+9.44pp
RhyCo-RL	**0.7407**	**0.9393**	**0.4333**	0.8328	**0.5561**	**0.640/0.798**	**+12.43**pp

where $T(\tau)$ is the token cost of τ, $\bar{T}_d$ is the average token cost of the single-agent baseline on task d, and λ_d is a per-task weight. The token cost $T(\tau)$ approximates C_{comm} in Eq. (5). In Eq. (9), ϵ is the clipping range, β_{KL} weights the KL penalty, and β_H weights the entropy bonus for exploration. Appendix D summarizes the runtime loop in Algorithm 1 and describes the two training stages.

3 Experiments

3.1 Setup

Data and benchmarks. We evaluate RhyCo on three categories of reasoning-intensive benchmarks. For mathematical reasoning, we use GSM8K (Cobbe et al., 2021), AIME 2024 (Mathematical Association of America, 2024), and MATH-L5, a level-5 subset of MATH (Hendrycks et al., 2021) adopted by AFlow and MaAS (Zhang et al., 2024c; 2025). For code generation, we use BigCodeBench (v0.1.0) (Zhuo et al., 2025) and MBPP (Austin et al., 2021). For multi-hop question answering, we use HotpotQA (Yang et al., 2018). We report accuracy for math, pass@1 for code, and EM/F1 for HotpotQA.

Baselines. We compare RhyCo with three groups of baselines. The first group covers single-agent and prompting strategies: direct inference (Single), Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (SC) (Wang et al., 2023b), and MedPrompt (Nori et al., 2023). The second group contains a fixed multi-agent workflow, Multi-Agent Debate (MAD) (Du et al., 2023). The third group covers trainable or adaptive workflow methods, AFlow (Zhang et al., 2024c) and MaAS (Zhang et al., 2025), which construct multi-agent systems via learned workflow search or supernet-style routing. More details on baseline setups are provided in Appendix A.3.

Implementation details. All speakers, listeners, and baseline systems use Qwen3-8B (Qwen Team, 2025) as the base LLM, while the arbitrator uses the smaller Qwen3-1.7B (Qwen Team, 2025). The four agent roles are the generator, criticizer, verifier, and metareviewer (Appendix B.1). Following Zhang et al. (2024c; 2025), most benchmarks use a 1:4 train/test split, and Table 3 lists the exceptions and the training pool sizes. RhyCo-RL uses $3\times$ the chunk budget of RhyCo-Base.

3.2 Main Results on Benchmarks

Table 1 reports results on math, code, and multi-hop QA benchmarks. Overall, RhyCo-RL improves over the Single baseline by 12.43 percentage points on average across the six benchmarks. The gains are concentrated on the harder tasks. On AIME 2024, RhyCo-RL reaches 43.3% versus 16.7% for Single, and on HotpotQA it raises EM from 45.6% to 64.0%. Against MaAS, the strongest adaptive baseline, RhyCo-RL has comparable accuracy and is two problems ahead on AIME 2024; its advantage lies in cost and in its runtime mechanism. On simpler benchmarks, the margin is much smaller. This points to a ceiling effect: a single competent agent already saturates the task, so listeners that share the speaker's inductive biases largely echo it.

Figure 3 compares accuracy and per-query cost on MATH-L5 under two cost measures: output token count (left) and inference cost converted to output-token-equivalent units (right). RhyCo-Base uses about $1.2\times$ the output tokens of Single and RhyCo-RL uses about $3.3\times$, compared with $7.8\times$

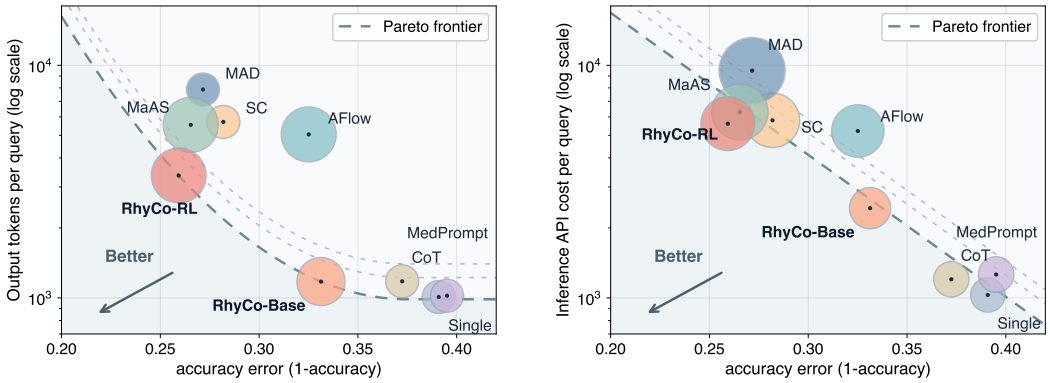


Figure 3: Cost-Accuracy comparison on MATH-L5. **Left**: output tokens per query. Marker size indicates the amount of training each method requires (larger = more training). **Right**: inference cost per query (input and output tokens priced at the 1:8 ratio of commercial APIs and expressed in output-token units); marker size indicates the cost (larger = higher cost).

Table 2: Ablation of RhyCo components on MATH-L5, BigCodeBench (BCB), and HotpotQA (HPQA). All variants are evaluated on the same problems.

Method	Gate	Arbitrator	MATH-L5	BCB	HPQA (EM)	Avg.
Single	None	None	0.6091	0.4439	0.456	0.5030
Unrestricted intervention	None	None	0.3107	0.0000	0.266	0.1922
RhyCo-Base w/o metrics	None	Prompted	0.4321	0.0000	0.381	0.2710
RhyCo-Base w/ rule gate	Rule	Prompted	0.6111	0.4597	0.560	0.5436
RhyCo-Base w/o whispers	Trained	Prompted	0.6646	0.5404	0.614	0.6063
RhyCo-Base	Trained	Prompted	0.6687	0.5491	0.620	0.6126
RhyCo-RL w/o whispers	Trained	Trained	0.7304	0.5561	0.640	0.6422
RhyCo-RL	Trained	Trained	0.7407	0.5561	0.640	0.6456

for MAD and $5.7\times$ for SC at five samples; both RhyCo configurations sit on the Pareto frontier of accuracy versus cost. Since RhyCo-Base involves no RL fine-tuning of the arbitrator, its place on the frontier shows that listener-driven rhythm control yields a favorable accuracy-cost trade-off.

3.3 Framework Analysis

Ablation across components. Table 2 compares RhyCo-Base and RhyCo-RL with two controls and with four variants that each remove or replace one component, and its Gate and Arbitrator columns list the configuration of every row. *Single* is a single-agent baseline with no listener channel, and in *Unrestricted intervention* an LLM orchestrator reads only the transcript at every chunk and interrupts whenever it judges the chunk problematic. The *w/o metrics* variant removes the upper branch together with the gate that reads it, so the arbitrator is consulted at every chunk with whispers alone. The *w/ rule gate* variant replaces the trained gate with the hand-crafted rules used for its initialization, and the two *w/o whispers* variants remove the lower branch from RhyCo-Base and RhyCo-RL so that the arbitrator receives metrics without whispers. We have three findings.

First, unregulated intervention is harmful. Unrestricted intervention and the *w/o metrics* variant both consult an LLM at every chunk and fall far below Single on average. On BigCodeBench, both collapse to 0.000 , where constant switching fragments the program into pieces that fail to parse or execute. **Second, the gate makes intervention selective.** The rule gate already lifts the average above Single to 0.544, and the trained gate raises it to 0.613. **Third, the arbitrator turns structured whispers into gains.** Whispers alone already outperform a transcript-only orchestrator, but they cannot prevent over-intervention as the whispers are sent at every chunk. When the gate selects the chunks and the arbitrator reads each structured whisper together with the metrics, the same whispers improve RhyCo-Base on all three benchmarks and lift RhyCo-RL on MATH-L5 from 0.730 to 0.741. Training the arbitrator with GRPO adds a further 3.3 points on average over RhyCo-Base.

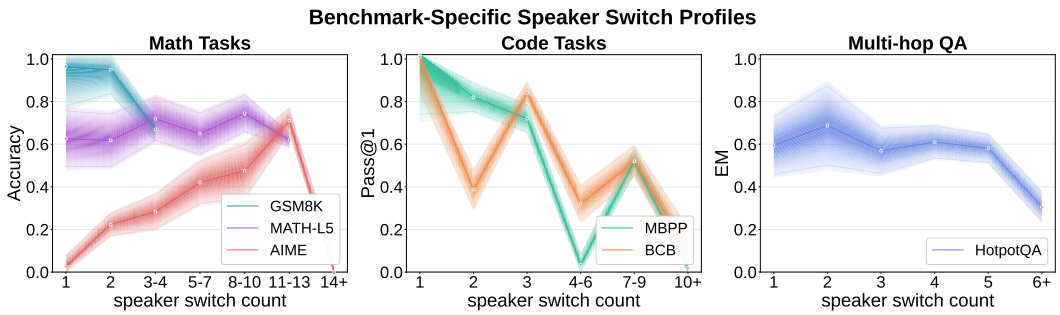


Figure 4: Benchmark-Specific Speaker Switch Profiles. Each point groups examples by the number of speaker switches during inference. The y-axis reports task-specific performance, using accuracy for math, pass@1 for code, and EM for multi-hop QA. Ribbon width denotes the fraction of examples in each switch-count bucket, and thicker regions indicate more common operating regimes.

Switch dynamics across benchmarks. Figure 4 plots task performance as a function of speaker switch count for math, code, and multi-hop QA. We have three observations. **(1) Different benchmarks favor different switch regimes.** On math, accuracy peaks at fewer switches on easier benchmarks, and rises up to 11–13 switches on AIME 2024. On code, MBPP peaks at one switch, while on BigCodeBench, RhyCo allocates more switches for harder programs. Importantly, high-switch buckets contain queries that single-shot baselines often fail on, so RhyCo escalates where baselines hit their ceiling. **(2) Probability mass adapts to task difficulty.** The ribbon encodes the fraction of queries per bucket, denoting that RhyCo observes difficulty differences inside each dataset. Simple benchmarks concentrate mass at low-switch buckets, while harder benchmarks spread it toward higher ones. Many queries are still solved there, such as AIME 2024 queries at 11–13 switches, where 71% of queries are correct. **(3) Multi-switch is useful but risky.** On hard problems, more switches often pay off. However, on simple benchmarks, queries that look easy but hide a pitfall can trap the team in switching without converging, and accuracy drops in the highest bucket of GSM8K, MBPP, and HotpotQA. This over-deliberation is as important a failure mode as the opposite case, where a short and confident trajectory propagates an error after only one or two switches.

3.4 Case Study

Figure 5 traces RhyCo on three problems of increasing difficulty from GSM8K, MATH-L5, and AIME 2024, and contrasts RhyCo with single-agent and multi-agent baselines on the AIME 2024 example. First, the number of switches tracks problem difficulty. On the GSM8K problem, RhyCo triggers only three switches before verifying the answer; on the MATH-L5 problem, four switches occur; on the AIME 2024 problem, all four roles become active, with switches scattered throughout the trajectory. This single-trajectory view mirrors the population-level trend in Figure 4. Second, RhyCo recovers from an error that traps every baseline shown. On the AIME 2024 problem, CoT, Self-Consistency with five samples, and Multi-Agent Debate with three rounds all output 997 by ignoring the triangle inequality, while the Single baseline produces no answer. Majority voting and debate do not resolve the error because every sample shares the same blind spot. RhyCo initially arrives at 997 along the same path, but the verifier detects the triangle-inequality violation, the arbitrator switches control, and the trajectory ends with 349.

4 Related Work

Multi-agent collaboration and orchestration. Multi-agent LLM systems have been explored through a broad range of collaboration paradigms, including role-based collaboration, programmable agent conversations, SOP-driven workflows, debate, and agent societies (Li et al., 2023; Wu et al., 2023; Hong et al., 2023; Chen et al., 2023). Recent work has investigated agent routing and collaboration-mode switching (Liu et al., 2023; Qian et al., 2024; Gao et al., 2025), workflow search and composition (Hu et al., 2024; Zhang et al., 2024c; Li et al., 2025b; Dang et al., 2025), topology adaptation (Li et al., 2024; Zhang et al., 2024b; Ke et al., 2026), structural pruning of agents or communication edges (Zhang et al., 2024a; Wang et al., 2025b; Li et al., 2025a), and query-conditioned orchestration policies (Zhang et al., 2025; Yue et al., 2025). Despite this growing flexibility, the coordination interface in MAS remains largely top-down: agents contribute only when selected by a planner, router, workflow searcher, topology controller, or orchestrator. RhyCo instead

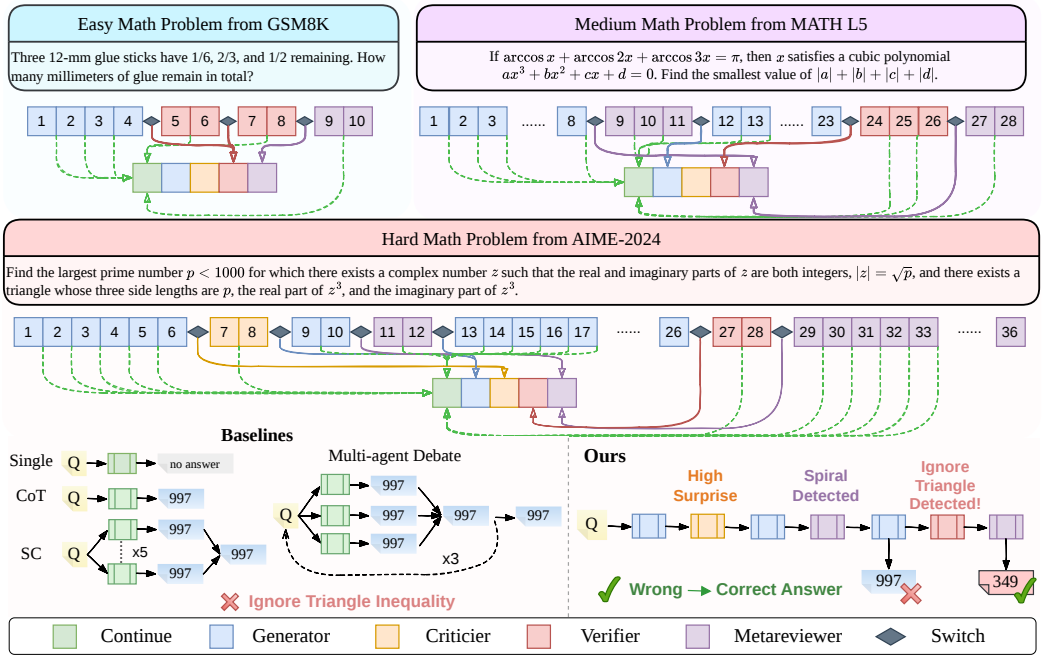


Figure 5: Three representative RhyCo trajectories, with baseline outputs on the AIME 2024 problem.

studies a complementary runtime setting in which non-speaking agents continuously listen and expose compact local assessments.

Reasoning feedback and control signals. A parallel thread of research strengthens reasoning through verification and process supervision (Lightman et al., 2023; Wang et al., 2023a; Setlur et al., 2024; Zheng et al., 2024), process reward modeling (Khalifa et al., 2025), self-refinement and reflection (Madaan et al., 2023; Shinn et al., 2023), and model-based judgment (Zheng et al., 2023). These methods provide judgment signals over intermediate reasoning steps or final answers, and are often used for training, reward modeling, reranking, search, or iterative refinement. Recent agent-oriented work extends this idea to agent trajectories and process-level progress attribution (Choudhury, 2025; Cui et al., 2025; Xi et al., 2025), reflective memory and experiential learning (Yuan et al., 2025; Wang et al., 2025a; Li et al., 2025c; Fan et al., 2026a), MAS process verification (Venkataramani et al., 2026), tool-using agent diagnostics (Cui et al., 2026; Fan et al., 2026b), workflow-level diagnosis and control (Ma et al., 2026; Zhang et al., 2026), and learned listener interruption that cuts off verbose messages to reduce communication cost (Wang et al., 2026). RhyCo instead feeds listener assessments to a lightweight orchestrator that decides at every chunk whether the current speaker continues or another agent takes over.

5 Conclusion

We introduced RhyCo, a framework that adds rhythm-controlled coordination to multi-agent LLM systems. RhyCo recasts MAS communication as a dual-channel speaker-listener interface, in which the speaker streams chunks while listeners expose probabilistic metrics at every chunk and decode a compact structured whisper only when a chunk is escalated. A two-tier orchestrator arbitrates these signals, pairing an LLM arbitrator engaged only on escalation with an MLP gate trained on counterfactual rollouts from error onsets. Across math, code, and multi-hop question answering, both RhyCo-Base and RhyCo-RL sit on the Pareto frontier of accuracy versus communication cost.

Our study has several limitations. We evaluate a fixed four-role agent pool on three task families, and extending RhyCo to long-horizon planning, tool-using agents, and heterogeneous-backbone teams remains important future work. RhyCo also depends on speaker-side metadata such as per-token confidence and on listener-side token distributions, which closed-API models do not expose, so extending it to such backbones will require an alternative signal source. More broadly, we hope this work encourages MAS research to treat the timing of agent contributions as a runtime design choice that complements top-down workflow design.

AI USE STATEMENT

Generative AI tools were used in this paper to assist with identifying and collecting potentially relevant literature for related work, refining the visual presentation of figures, editing the manuscript for readability and clarity, and annotating the first erroneous chunk in failed training trajectories using Claude Opus. Experimental settings, including baseline configurations and evaluation procedures, were determined by the authors. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproducibility in the appendix. Appendix A.1 describes the implementation details, Appendix A.2 reports dataset statistics and splits, and Appendix A.3 specifies the baseline configurations and evaluation setup. Appendix B defines the 12 speaker–listener channel metrics used in our framework and provides their corresponding computation procedures. Appendix C includes the prompt templates used by the LLM-based arbitrator.

REFERENCES

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

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023. doi: 10.48550/arXiv.2308.10848. URL https://arxiv.org/abs/2308.10848.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025. doi: 10.48550/arXiv.2502.10325. URL https://arxiv.org/abs/2502.10325.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv prepring*, abs/2110.14168, 2021.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. Agentic reward modeling: Verifying gui agent via online proactive interaction. *arXiv preprint arXiv:2602.00575*, 2026. doi: 10.48550/arXiv.2602.00575. URL https://arxiv.org/abs/2602.00575.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025. doi: 10.48550/arXiv.2502.01456. URL https://arxiv.org/abs/2502.01456.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. *arXiv preprint arXiv:2505.19591*, 2025. doi: 10.48550/arXiv.2505.19591. URL https://arxiv.org/abs/2505.19591.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *CoRR*, abs/2305.14325, 2023.

Kaixuan Fan, Kaituo Feng, Manyuan Zhang, Tianshuo Peng, Zhixun Li, Yilei Jiang, Shuang Chen, Peng Pei, Xunliang Cai, and Xiangyu Yue. Exploring reasoning reward model for agents. *arXiv*

preprint arXiv:2601.22154, 2026a. doi: 10.48550/arXiv.2601.22154. URL https://arxiv.org/abs/2601.22154. ACL 2026 Findings.

Shengda Fan, Xuyan Ye, Yupeng Huo, Zhi-Yuan Chen, Yiju Guo, Shenzhi Yang, Wenkai Yang, Shuqi Ye, Jingwen Chen, Haotian Chen, Xin Cong, and Yankai Lin. Agentprocessbench: Diagnosing step-level process quality in tool-using agents. *arXiv preprint arXiv:2603.14465*, 2026b. doi: 10.48550/arXiv.2603.14465. URL https://arxiv.org/abs/2603.14465.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025. doi: 10.48550/arXiv.2504.15257. URL https://arxiv.org/abs/2504.15257.

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. *NeurIPS*, 2021.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. doi: 10.48550/arXiv.2308.00352. URL https://arxiv.org/abs/2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. doi: 10.48550/arXiv.2408.08435. URL https://arxiv.org/abs/2408.08435.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*, 2023. doi: 10.48550/arXiv.2305.06983. URL https://arxiv.org/abs/2305.06983.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026. doi: 10.48550/arXiv.2601.14652. URL https://arxiv.org/abs/2601.14652.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025. doi: 10.48550/arXiv.2504.16828. URL https://arxiv.org/abs/2504.16828.

Mingze Kong, Zikun Qu, Zhongquan Zhou, Pengyu Liang, Xiang Li, Zhiwei Shang, Zhi Hong, Kaiyu Huang, Zhiyong Wang, and Zhongxiang Dai. Workflow-r1: Group sub-sequence policy optimization for multi-turn workflow construction. *arXiv preprint arXiv:2602.01202*, 2026. doi: 10.48550/arXiv.2602.01202. URL https://arxiv.org/abs/2602.01202.

Boyi Li, Zhonghan Zhao, Der-Horng Lee, and Gaoang Wang. Adaptive graph pruning for multi-agent communication. *arXiv preprint arXiv:2506.02951*, 2025a. doi: 10.48550/arXiv.2506.02951. URL https://arxiv.org/abs/2506.02951.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023. doi: 10.48550/arXiv.2303.17760. URL https://arxiv.org/abs/2303.17760.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. *arXiv preprint arXiv:2507.18224*, 2025b. doi: 10.48550/arXiv.2507.18224. URL https://arxiv.org/abs/2507.18224.

Yilong Li, Chen Qian, Yu Xia, Ruijie Shi, Yufan Dang, Zihao Xie, Ziming You, Weize Chen, Cheng Yang, Weichuan Liu, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Cross-task experiential learning on llm-based multi-agent collaboration. *arXiv preprint arXiv:2505.23187*, 2025c. doi: 10.48550/arXiv.2505.23187. URL https://arxiv.org/abs/2505.23187.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. *arXiv preprint arXiv:2406.11776*, 2024. doi: 10.48550/arXiv.2406.11776. URL https://arxiv.org/abs/2406.11776.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023. doi: 10.48550/arXiv.2310.02170. URL https://arxiv.org/abs/2310.02170.

Zihan Ma, Zhikai Zhao, Chuanbo Hua, Federico Berto, and Jinkyoo Park. Judgeflow: Agentic workflow optimization via block judge. *arXiv preprint arXiv:2601.07477*, 2026. doi: 10.48550/arXiv.2601.07477. URL https://arxiv.org/abs/2601.07477.

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. *arXiv preprint arXiv:2303.17651*, 2023. doi: 10.48550/arXiv.2303.17651. URL https://arxiv.org/abs/2303.17651.

Mathematical Association of America. AIME 2024: American Invitational Mathematics Examination. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2024. Released by the AI Mathematical Olympiad (AI-MO) and HuggingFace H4.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, et al. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452*, 2023.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*, 2024. doi: 10.48550/arXiv.2406.07155. URL https://arxiv.org/abs/2406.07155.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. doi: 10.48550/arXiv.2307.16789. URL https://arxiv.org/abs/2307.16789.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024. doi: 10.48550/arXiv.2410.08146. URL https://arxiv.org/abs/2410.08146.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366. URL https://arxiv.org/abs/2303.11366.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025. doi: 10.48550/arXiv.2511.21689. URL https://arxiv.org/abs/2511.21689.

Vishal Venkataramani, Haizhou Shi, Zixuan Ke, Austin Xu, Xiaoxiao He, Yingbo Zhou, Semih Yavuz, Hao Wang, and Shafiq Joty. Mas-prove: Understanding the process verification of multi-agent systems. *arXiv preprint arXiv:2602.03053*, 2026. doi: 10.48550/arXiv.2602.03053. URL https://arxiv.org/abs/2602.03053.

Danqing Wang, Da Yin, Ruta Desai, Lei Li, Asli Celikyilmaz, and Ansong Ni. Learning to interrupt in language-based multi-agent communication, 2026. URL https://arxiv.org/abs/2604.06452.

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*, 2025a. doi: 10.48550/arXiv.2505.20732. URL https://arxiv.org/abs/2505.20732.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023a. doi: 10.48550/arXiv.2312.08935. URL https://arxiv.org/abs/2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. *arXiv preprint arXiv:2503.18891*, 2025b. doi: 10.48550/arXiv.2503.18891. URL https://arxiv.org/abs/2503.18891.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, January 01, 2022 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. doi: 10.48550/arXiv.2308.08155. URL https://arxiv.org/abs/2308.08155.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Yajie Yang, Wenxiang Chen, Zhihao Zhang, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. Agentprm: Process reward models for llm agents via step-wise promise and progress. *arXiv preprint arXiv:2511.08325*, 2025. doi: 10.48550/arXiv.2511.08325. URL https://arxiv.org/abs/2511.08325.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2369–2380, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. doi: 10.48550/arXiv.2210.03629. URL https://arxiv.org/abs/2210.03629.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025. doi: 10.48550/arXiv.2501.11425. URL https://arxiv.org/abs/2501.11425.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. *arXiv preprint arXiv:2502.11133*, 2025. doi: 10.48550/arXiv.2502.11133. URL https://arxiv.org/abs/2502.11133.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. *arXiv preprint arXiv:2410.02506*, 2024a. doi: 10.48550/arXiv.2410.02506. URL https://arxiv.org/abs/2410.02506.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024b. doi: 10.48550/arXiv.2410.11782. URL https://arxiv.org/abs/2410.11782.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025. doi: 10.48550/arXiv.2502.04180. URL https://arxiv.org/abs/2502.04180.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024c. doi: 10.48550/arXiv.2410.10762. URL https://arxiv.org/abs/2410.10762.

Mingda Zhang, Haoran Luo, Tiesunlong Shen, Qika Lin, Xiaoying Tang, Rui Mao, and Erik Cambria. Flowsteer: Interactive agentic workflow orchestration via end-to-end reinforcement learning. *arXiv preprint arXiv:2602.01664*, 2026. doi: 10.48550/arXiv.2602.01664. URL https://arxiv.org/abs/2602.01664.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024. doi: 10.48550/arXiv.2412.06559. URL https://arxiv.org/abs/2412.06559.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023. doi: 10.48550/arXiv.2306.05685. URL https://arxiv.org/abs/2306.05685.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2406.15877.

A Experimental Details

A.1 Implementation Details

Backbones. All speaker and listener agents share Qwen3-8B (Qwen Team, 2025) as the base backbone, differentiated only by role-specific system prompts. The arbitrator uses the smaller Qwen3-1.7B (Qwen Team, 2025).

Decoding and budgets. All models run with the Qwen3 thinking mode disabled, and baselines generate at most 4096 tokens per call. RhyCo uses the chunk sizes of Section 2.1, with a per-trajectory budget of 30 chunks for RhyCo-Base (60 on AIME 2024) and 90 chunks for RhyCo-RL (180 on AIME 2024).

Gate (MLP). The gate is a 2-layer MLP with one hidden layer of 32 units, mapping $z_t \in \mathbb{R}^{12}$ to a logit over $\{\text{stay}, \text{escalate}\}$. We use ReLU activations and a weighted binary cross-entropy loss on the labels described below, optimized with Adam.

Gate training data. Before training on real trajectories, the gate is initialized on simulated rhythm states labeled by the hand-crafted rules. For each benchmark, we then run the single-agent baseline on the training pool reported in Table 3, keep the trajectories that fail, and annotate the chunk where the reasoning first goes wrong using Claude Opus 4.6 and checked all annotated chunks manually. Chunks before this onset are labeled stay and the onset chunk is labeled escalate. The later chunks on the original path are also labeled escalate but receive lower weights. From the onset, we create counterfactual continuations in which each of the four agents takes over and continues to completion. These continuations verify that intervening at the onset is useful and provide additional labeled chunks for the gate.

LoRA + GRPO. In the RL configuration, we fine-tune the LoRA with rank 16 while freezing arbitrator's Qwen3-1.7B backbone on the standard attention projection matrices (q_proj, k_proj, v_proj, o_proj). We use Adam with learning rate 5×10^{-5}, a GRPO group size of 4, a per-update batch size of 4, an entropy bonus of 0.05 in the loss, and a KL penalty coefficient of 0.04. GRPO is applied under the reward of Eq. (10). The token cost $T(\tau)$ counts the input and output tokens along τ, excluding listener prefill and including the arbitrator's own outputs, and $\bar{T}_d$ is the average token cost of the single-agent baseline on task d. Training fits within the per-benchmark budget of about 2 hours reported under Compute.

Compute. All training and evaluation are conducted on a single NVIDIA A100 GPU. The full RhyCo pipeline per benchmark (counterfactual rollout collection, gate fitting, and LoRA fine-tuning of the arbitrator) takes approximately 2 hours of wall-clock time. Inference for evaluation also uses a single A100. We did not run preliminary experiments at substantially larger compute beyond what is reported here.

A.2 Dataset Statistics

To build up datasets, we divide each dataset into training and test sets using a TRAIN:TEST ratio of 1:4. GSM8K instead uses 100 training problems and its full test set, and AIME uses AIME 2022 as its training pool and AIME 2024 as its test set. For the MATH benchmark, we follow Zhang et al. (2025), using a subset of harder problems spanning four representative categories, Combinatorics & Probability, Number Theory, Pre-algebra, and Pre-calculus, all at difficulty level 5. The dataset statistics are included in Table 3.

A.3 Baseline Setups

In this section, we provide a detailed description of the configurations for baseline methods:

1. **CoT.** Chain-of-Thought (CoT) prompting guides LLM agents to break down reasoning into sequential steps. We follow the prompting of Wei et al. (2022).
2. **Self-consistency.** To enhance robustness, we aggregate five CoT-generated solutions.
3. **MAD.** We instantiate three LLM-agents, each assigned a distinct role, which participate in up to three rounds of debate, after which the final decision is determined via majority voting, following Du et al. (2023).

Table 3: Dataset statistics.

Domain	Dataset	#Train	#Test	Metric
Code Generation	BigCodeBench[†]	143	570	pass@1
	MBPP	86	341	pass@1
Math Reasoning	GSM8K[§]	100	1319	Accuracy
	MATH-L5	119	486	Accuracy
	AIME 2024[‡]	30	30	Accuracy
Multi-Hop QA	HotpotQA	200	800	EM / F1

[†]The test set is the first 570 tasks of BigCodeBench v0.1.0.
[§]GSM8K uses 100 training problems and its full test set.
[‡]The training pool is drawn from AIME 2022 with 30 problems; the test set is AIME 2024 with 30 problems.

4. **MedPrompt.** MedPrompt (Nori et al., 2023) is a strong prompting strategy that combines dynamic few-shot retrieval, chain-of-thought reasoning, and majority-vote ensembling over multiple sampled completions. We use the publicly described configuration with the same base LLM as the other baselines, and report results under matched ensemble size to keep cost comparisons fair.
5. **AFlow.** In Zhang et al. (2024c), AFlow operates with both `gpt-4o-mini` and `claude-3.5-sonnet`. To maintain fairness under homogeneous conditions, we restrict AFlow to `qwen3-8B` and set MAX_ITERATION=20.
6. **MaAS.** MaAS (Zhang et al., 2025) formulates multi-agent system construction as a search over a probabilistic agentic supernet. For each query, a controller network samples a query-conditional sub-network of agentic operators, which is then executed; the supernet's parameterized distribution and the operators themselves are jointly optimized using empirical Bayes Monte Carlo on task reward and textual gradients on operator configurations. We use the official implementation with default settings, and restrict the operator pool to the same agent backbones used by other baselines for fairness.

B Speaker-Listener Channel Specifications

This section provides the full specification of the speaker-listener interface introduced in Section 2.2, including agent role definitions, the 12-dimensional rhythm-state feature mapping, and the whisper grammar.

B.1 Agent Roles

RhyCo uses four agent roles, each defined by a distinct system prompt. At any time step, exactly one role serves as the active speaker, and the remaining three roles serve as listeners.

- **generator**: produces task-advancing reasoning and is the initial speaker of every query.
- **criticizer**: performs language-level reasoning critique and flags logical gaps.
- **verifier**: tool-grounded checking; may invoke external interpreters or search tools when promoted to speaker.
- **metareviewer**: monitors trajectory-level health (stuck detection, redundancy) and final-answer extraction.

B.2 Speaker Metadata

The speaker metadata $m_t^{\text{spk}} \in \mathbb{R}^4$ captures surface-level chunk properties that are obtained as byproducts of generation:

Index	Symbol	Description
0	`repeat_sim`	Token-level n-gram similarity between c_t and the most-recent prefix of h_t
1	`chunks_since`	Number of chunks since the last final-answer event
2	`conf_min`	Minimum per-token softmax confidence within c_t
3	`has_ans`	Indicator: c_t contains a final-answer marker

The self-repetition score `repeat_sim` is computed as the maximum 4-gram overlap between c_t and any contiguous window of the same length in the recent prefix of h_t, normalized to $[0, 1]$.

B.3 Listener Metric Features

For each listener role, we extract four scalar statistics from the listener's predictive distribution over c_t: `agreement_mean`, `agreement_min`, `entropy_mean`, and `surprise_mean`. These twelve raw scalars are then combined with m_t^{spk} to form $z_t \in \mathbb{R}^{12}$ as follows:

Index	Symbol	Source
0–3	speaker metadata	m_t^{spk} (Group I, surface)
4	`criticizer_agree`	per-role agreement passthrough (Group II)
5	`agree_min_spread`	$\max_i$ agree_min$_i$ $-$ $\min_i$ agree_min$_i$ (Group III)
6	`verifier_ent`	per-role entropy passthrough (Group II)
7	`metareviewer_agree`	per-role agreement passthrough (Group II)
8	`criticizer_ent`	per-role entropy passthrough (Group II)
9	`ent_max`	$\max_i$ ent_mean$_i$ across listeners (Group III)
10	`metareviewer_ent`	per-role entropy passthrough (Group II)
11	`surp_max`	$\max_i$ surp_mean$_i$ across listeners (Group III)

The mapping retains entropy passthroughs for all three listener roles (criticizer, verifier, metareviewer when generator is the speaker) and agreement passthroughs for two of the three roles. The third role's agreement and the per-role surprise are absorbed into the cross-listener aggregations `agree_min_spread` and `surp_max`, which summarize this information at the pool level.

B.4 Whisper Grammar

When the gate escalates a chunk, each listener decodes a structured whisper $w_{i,t}$ following the grammar:

```
whisper     ::=  [ROLE] [VERDICT] [CONFIDENCE]: [REASON]
ROLE         ∈   {criticizer, verifier, metareviewer, generator}
VERDICT      ∈   {retain, switch, idle}
CONFIDENCE   ∈   {high, medium, low}
REASON      ::=  free-text token sequence, length ≤ 16 tokens
```

Total whisper length (header plus reason) is capped at 24 tokens. Whispers are decoded only when the gate selects escalate and are concatenated and passed directly to the arbitrator; the gate does not consume whisper text, and no whisper tokens are generated on chunks the gate lets pass.

C Arbitrator Prompt Templates

This section reproduces the prompt templates used by the RhyCo-Base configuration of the RhyCo arbitrator.

C.1 Math Reasoning

The math template asks the arbitrator to keep the current speaker by default, to interrupt only when its own reading and the listener evidence together indicate a clear need, and to stop once the final answer has appeared and the listener metrics are stable.

Math Reasoning Arbitrator Prompt

You are the Orchestrator of a multi-agent math reasoning team. Your job is NOT to solve math. Your job is to decide WHO should hold the floor.

INPUT 1 - Speaker side. The pending chunk produced by the current speaker, together with the speaker's per-token confidence over that chunk.
INPUT 2 - Listener side. For each idle listener: (a) a one-sentence whisper that conveys the listener's qualitative reaction, and (b) a 12-d feature vector summarizing the listener's internal state:

- has_ans: 1 if the pending chunk contains a \boxed{} answer, else 0.
- chunks_since: chunks since the last \boxed{} appeared (high ⇒ speaker may be stuck).
- repeat_sim: similarity to recent committed chunks (high ⇒ speaker is looping).
- conf_min: speaker's lowest token confidence within this chunk.
- criticizer_agree, metareviewer_agree: per-listener mean agreement with speaker tokens (lower ⇒ more disagreement).
- agree_min_spread: spread between minimum and mean listener agreement (high ⇒ sharp local disagreement).
- criticizer_ent, verifier_ent, metareviewer_ent: per-listener mean entropy (higher ⇒ more confused).
- ent_max: peak listener entropy on any single token.
- surp_max: peak listener surprise on any single token (high ⇒ at least one listener strongly disagrees).

INPUT 3 - Recent committed history. A short window of previously committed chunks, provided so the arbitrator can resolve coreference and detect repetition.

Decision protocol. Combine your own reading of the transcript with the listener evidence. Keep your thinking brief (under 80 tokens). Do NOT restate the problem. Focus on: (1) are the listener metrics concerning, and (2) is the speaker making progress?
Most chunks should result in CONTINUE. Interrupting disrupts the speaker's reasoning chain and is a significant action; use INTERRUPT only when your analysis and the listener evidence together indicate a clear need. Use STOP only when there are no objections from any listener and no further intervention is warranted, e.g. the chunk has emitted the final \boxed{} answer and listener metrics are stable.

Output format. The ACTION line, and the optional HINT line that follows it, must be the LAST lines of your response. The ACTION must take exactly one of the following forms:

- ACTION: CONTINUE
- ACTION: INTERRUPT_NOW(generator)
- ACTION: INTERRUPT_NOW(verifier)
- ACTION: INTERRUPT_NOW(criticizer)
- ACTION: INTERRUPT_NOW(metareviewer)
- ACTION: STOP

If you choose INTERRUPT_NOW, append a HINT line containing a soft suggestion (at most 20 words, no specific diagnosis), e.g.
ACTION: INTERRUPT_NOW(verifier)
HINT: The recent calculation may benefit from numerical verification.

HINT guidance. *Acceptable:* "Consider verifying the recent calculations."; "The current approach may be slow, a different

strategy might help."; "A numerical check could be useful here." *Avoid:* pointing at a specific step ("Step 3 has a sign error in $x^2 + x$.") or revealing the answer ("The answer should be 29."). No other format is accepted.

C.2 Code Generation

The code template routes verifier failures and high listener surprise to the criticizer, sends repeated failures to the metareviewer or the generator, and stops once all tests pass. In the code domain, the verifier executes candidate code in a deterministic subprocess, which the prompt treats specially.

Code Generation Arbitrator Prompt

You are the Orchestrator of a 4-agent Python code team. Your job is NOT to write code. You decide WHO holds the floor next, based on the transcript, the current speaker's pending chunk, listener metrics, and recent execution results.

Agents. For this code task, the four roles specialize as:

- generator: writes or rewrites Python code.
- verifier: runs the most recent code block via subprocess; returns concrete pass/fail and error messages.
- criticizer: reads the execution output, diagnoses failure, and proposes a fix (including revised code).
- metareviewer: selects or merges the best among multiple candidate solutions.

INPUT 1 - Speaker side. The pending chunk produced by the current speaker, your own reasoning about progress, and the speaker's per-token confidence over that chunk.
INPUT 2 - Listener side. For each idle listener: (a) a one-sentence whisper that conveys the listener's qualitative reaction, and (b) a 12-d feature vector summarizing the listener's internal state:

- has_ans: 1 if the pending chunk completes a code block (which automatically triggers execution), else 0.
- chunks_since: chunks since the last code block was completed (high ⇒ speaker may be stuck).
- repeat_sim: similarity to recent committed chunks (high ⇒ speaker is looping on the same approach).
- conf_min: speaker's lowest token confidence within this chunk.
- verifier_agree, metareviewer_agree: per-listener mean agreement with speaker tokens (lower ⇒ more disagreement).
- agree_min_spread: spread between minimum and mean listener agreement (high ⇒ sharp local disagreement).
- verifier_ent, criticizer_ent, metareviewer_ent: per-listener mean entropy (higher ⇒ more confused).
- ent_max: peak listener entropy on any single token.
- surp_max: peak listener surprise on any single token (high ⇒ at least one listener strongly disagrees).

INPUT 3 - Recent committed history and execution results. A short window of previously committed chunks, including the most recent verifier pass/fail summary and any error messages.

Critical semantics.

1. **CONTINUE on verifier is a dead option, never choose it.** The verifier runs a deterministic subprocess; given the same code and tests, re-running yields the same result and wastes one chunk of the trajectory budget.
   - If all tests pass ⇒ STOP.

- If any test fails ⇒ INTERRUPT_NOW(criticizer) to diagnose and fix.
- If the same code has been tested 3+ times with the same failure ⇒ INTERRUPT_NOW(metareviewer), or INTERRUPT_NOW(generator) with a hint to try a fundamentally different algorithmic approach.

2. **CONTINUE makes sense only for LLM speakers** (*generator*, *criticizer*, *metareviewer*) who may need more tokens to finish their thought. When the verifier holds the floor it only executes code and does not "continue writing".
3. **STOP means "I accept the current code as final."** Use only when the verifier has just reported all tests passing and no criticizer concern is open.
4. **Mode collapse.** If the same test has failed the same way 3+ times in a row, the criticizer is stuck patching one approach; route to INTERRUPT_NOW(generator) with a HINT to try a different algorithm class, or INTERRUPT_NOW(metareviewer) if at least two distinct approaches have already been tried.

Soft guidance.

- Verifier FAIL ⇒ usually criticizer, unless the criticizer has tried 2+ times without progress (then metareviewer, or generator with an algorithm-class hint).
- Verifier PASS ⇒ usually STOP, unless partial pass (e.g., 3/5), in which case consider the criticizer for a review pass.
- High surp_max or large agree_min_spread on pending LLM output ⇒ edge cases may be wrong; route to the criticizer.
- Same approach attempted 3+ times without progress ⇒ metareviewer (if diverse candidates exist) or generator with a hard hint to change approach.
- Default when in doubt and the speaker is an LLM ⇒ CONTINUE.

Output format. The ACTION line, and the optional HINT line that follows it, must be the LAST lines of your response. The ACTION must take exactly one of the following forms:

- ACTION: CONTINUE
- ACTION: INTERRUPT_NOW(generator)
- ACTION: INTERRUPT_NOW(criticizer)
- ACTION: INTERRUPT_NOW(metareviewer)
- ACTION: STOP

INTERRUPT_NOW(verifier) is rarely used: the verifier runs automatically after each emitted code block, and is invoked manually only to re-test without new code.
If you choose INTERRUPT_NOW, append a HINT line containing a soft suggestion (at most 20 words, no specific bug fix), e.g. "HINT: The earlier attempt missed the empty-input edge case." or "HINT: Try a different algorithm class; recursion may overflow."
Keep your analysis brief (under 80 tokens).

D ALGORITHM

Algorithm 1 summarizes the RhyCo runtime loop in deployment. Training proceeds in two stages. First, the gate g_ψ is initialized on simulated data labeled by hand-crafted rules and then trained on annotated failed trajectories and their counterfactual continuations (Section 2.3 and Appendix A.1). Second, in the RL configuration the gate is frozen and the arbitrator's LoRA parameters are updated with GRPO under the reward of Eq. (10); RhyCo-Base skips this second stage and uses the pre-trained Qwen3-1.7B directly via prompting.

Algorithm 1 RhyCo runtime

Input: query q, agent pool A, pooling function $\mathrm{Agg}(\cdot)$, gate g_ψ, arbitrator π_θ, chunk budget M
Output: reasoning trajectory h, final answer
Initialize $h_1 \leftarrow \emptyset$; speaker $\mathrm{spk}_1 \leftarrow$ generator; speaker-local step $\kappa_1 \leftarrow 0$
for $t = 1, \ldots, M$ **do**
  $c_t \leftarrow \mathrm{spk}_t.\mathrm{decode}(h_t, q;\ 64 \text{ tokens})$
  $m_t^{\mathrm{spk}} \leftarrow$ extract speaker metadata from c_t and h_t
  for each listener $i \in L_t$ **do**
    $m_{i,t} \leftarrow$ listener i.prefill(q, h_t, c_t) {one forward pass, no decoding}
  end for
  $z_t \leftarrow \mathrm{Agg}(m_t^{\mathrm{spk}}, \{m_{i,t}\}_{i \in L_t})$
  $e_t \leftarrow 1$ with probability $g_\psi(z_t)$, otherwise $e_t \leftarrow 0$
  if $e_t = 0$ **then**
    $u_t \leftarrow$ stay {tier-1 commits}
  else
    $w_{i,t} \leftarrow$ listener i.decode_whisper(q, h_t, c_t) for each $i \in L_t$ {lazy, ≤ 24 tokens}
    $u_t \leftarrow \pi_\theta(c_t, z_t, \{w_{i,t}\}_{i \in L_t}, h_t)$ {tier-2 decides}
  end if
  if $u_t \in \{\text{stay}, \text{retain}\}$ **then**
    $h_{t+1} \leftarrow h_t \oplus c_t;\ \mathrm{spk}_{t+1} \leftarrow \mathrm{spk}_t;\ \kappa_{t+1} \leftarrow \kappa_t + 1$
  else
    $\{u_t = \mathrm{switch}(j)\}; h_{t+1} \leftarrow h_t;\ \mathrm{spk}_{t+1} \leftarrow a_j;\ \kappa_{t+1} \leftarrow 0$
  end if
  if c_t carries a final-answer flag and was committed **then**
    return h_{t+1} and the final answer
  end if
end for
return h_{M+1} and the answer aggregated from it {budget exhausted}