Communicate While Thinking: CoT-state Communication for Better and Token-Efficient Multi-Agent Reasoning

Anonymous ACL submission

Abstract

Multi-agent LLM systems often coordinate only after agents complete their local Chain-of-Thought (CoT), by exchanging final answers, critiques, or rationales. This output-level communication can arrive too late: intermediate assumptions, evidence choices, or computation errors may already have shaped the final answer before peer information is used.

We introduce CoTIE (CoT-State Inter-Agent Exchange), a multi-agent reasoning approach that shifts communication from final outputs to intermediate CoT states. Instead of waiting for completed answers, a caller agent can request focused help from a peer during ongoing reasoning and integrate the response before producing the final answer. The key idea is not to add a new workflow optimizer, but to change where peer information is attached in the reasoning process.

Across MATH, HumanEval, HotpotQA, and MultiAgentBench, CoTIE improves task performance over output-level communication baselines under matched evaluation settings. On cooperative MultiAgentBench, where token accounting is directly comparable, CoTIE improves the three-scenario average score from 67.5 to 78.3 while reducing observed average token usage from 129,270 to 67,157 compared with MAB, and raises aggregate score per 100K tokens from 52.2 to 116.5. Code is available for at https://anonymous.4open.science/r/CoTIE-2837/.

1 Introduction

Chain-of-Thought (CoT) prompting enables step-by-step reasoning in LLMs (Wei et al., 2022), and multi-agent LLM systems extend this idea by distributing complex tasks across role-specialized agents (Hong et al., 2024; Qian et al., 2024; Wu et al., 2023). However, many multi-agent reasoning systems still postpone communication until each agent has completed its own CoT process (Du et al., 2024; Chen et al., 2024a; Liang et al., 2024). Once an agent has committed to intermediate hypotheses, evidence choices, or computations during reasoning, these decisions may already have shaped the final answer. At that point, messages, critiques, votes, or answer-level reconciliation (Chen et al., 2024a) may arrive too late to repair the reasoning error where it first occurred.

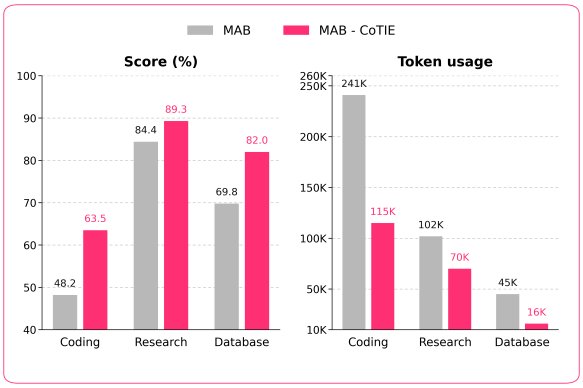

Figure 1: **Better performance with lower token usage on cooperative MultiAgentBench.** Compared with output-level message passing, CoTIE improves scenario scores while reducing token usage across Coding, Research, and Database tasks.

Figure 2 contrasts conventional output-level communication with CoTIE's intermediate CoT-state communication. In output-level communication, a possible local error in an early reasoning step may propagate through the remaining local trace before any peer input is introduced. This contrast motivates the specific design question studied in this paper: under a fixed multi-agent setting, should peer input be introduced only after local CoT traces are completed, or while the caller is still reasoning from an intermediate CoT state? We use *communication attachment point* to denote when peer input is introduced into the caller's CoT process: after a completed output or during an intermediate CoT state. CoTIE adopts the latter attachment-point

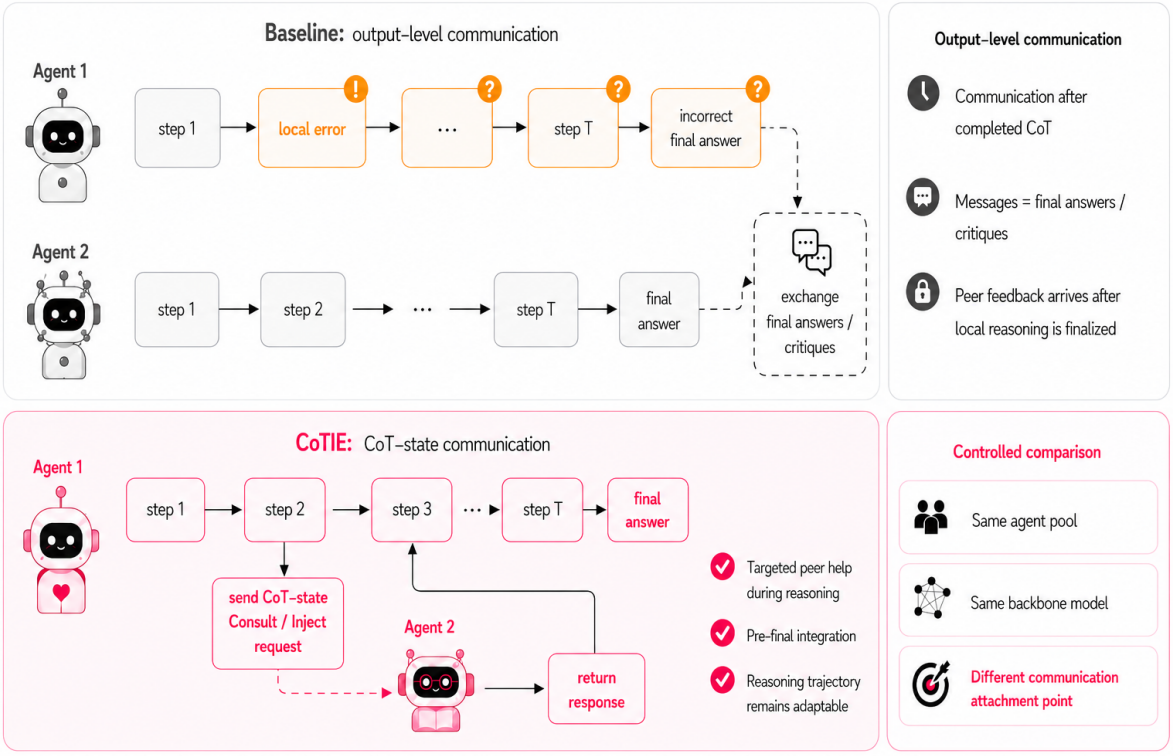

Figure 2: **CoTIE vs. output-level communication.** Output-level communication exchanges peer information only after local CoT completion, whereas CoTIE attaches communication to an intermediate CoT state and integrates the response before finalization.

principle rather than introducing a new workflow optimizer or fixed routing policy. At an intermediate reasoning step, the caller sends a bounded, state-derived request containing the relevant task slice, a short partial trace, the current help need, and the requested help type. The peer returns a targeted response or sub-result, which the caller integrates before producing the final answer. Algorithm 1 formalizes this mechanism through configurable peer selection (line 7), request-option selection (line 8), pre-final message sending (line 10), and response integration (line 12).

Empirically, the main comparison is between final-output-level communication, Late CoT Message Passing, and CoT-state communication. On the general benchmarks, we use the same two-agent setting, backbone model, task input, and final integration rule. On MultiAgentBench, we preserve the benchmark-provided agent topology and replace the output-level coordination opportunity with intermediate CoT-state communication. We do not impose a fixed communication count or total-token budget; token usage is logged after execution as an observed cost of each protocol. On the cooperative MultiAgentBench scenarios, CoTIE improves the three-scenario average score from 67.5 to 78.3 while reducing observed average token usage from 129,270 to 67,157; it also improves aggregate score per 100K tokens from 52.2 to 116.5 compared with MAB.

Contributions.

- **CoTIE as a novel CoT-state communication approach.** We introduce CoTIE, a new approach to multi-agent CoT reasoning in which agents exchange information through intermediate CoT states rather than only through completed outputs. This makes the communication attachment point an explicit design choice in multi-agent LLM reasoning.
- **An efficient step-level CoTIE algorithm.** Beyond the general CoT-state communication principle, we instantiate CoTIE as an efficient step-level algorithm, formalized in Algorithm 1. The algorithm realizes CoT-state exchange through a compact request–response–integration procedure, enabling a caller to obtain focused peer input during reasoning and incorporate it before finalization.
- **Higher task performance.** Across MATH, HumanEval, HotpotQA, and MultiAgentBench, CoTIE improves task performance over output-

level communication baselines. This suggests that introducing peer information during intermediate reasoning can be more useful than exchanging information only after agents have completed their local CoT traces.

- **Higher token utilization efficiency.** The token reduction is consistent with CoTIE's design: peer information is made available earlier in the reasoning process, which can reduce redundant downstream reasoning and final-stage coordination content. On the cooperative MultiAgentBench scenarios, where token accounting is directly comparable, CoTIE reduces token usage and obtains higher task performance per unit of token cost than output-level communication baselines. Since no fixed token budget is imposed, token usage is measured after execution as an outcome of each protocol rather than as a pre-specified constraint.

  Overall, these results show that CoTIE improves reasoning quality and token efficiency by introducing peer input during intermediate reasoning rather than after completed outputs.

2 Method

2.1 Overview

CoTIE changes where cross-agent communication is attached in multi-agent Chain-of-Thought reasoning. Rather than exchanging information only after local reasoning is complete, a caller agent sends a compact request based on its current CoT state during ongoing reasoning, receives a targeted peer response, and integrates it before finalization.

We first define the multi-agent setting, CoT state, and CoT-state request, and then instantiate the principle as a step-level algorithm. The same interface supports CONSULT and INJECT requests, while remaining independent of any particular triggering, routing, or workflow-optimization policy.

2.2 CoTIE Approach

Definition 1 (Multi-agent reasoning system)**.** A multi-agent reasoning instance is a tuple $(\mathcal{N}, \tau, \mathcal{G})$, where $\mathcal{N} = \{a_1, \ldots, a_n\}$ is a set of role-specialized agents, τ is the input task, and $\mathcal{G} \subseteq \mathcal{N} \times \mathcal{N}$ is the allowed communication graph.

Definition 2 (CoT state)**.** The CoT state of agent a_i at reasoning step t is

$$\sigma_i^{(t)} = \big(\mathrm{trace}_i^{(t)}, h_i^{(t)}, u_i^{(t)}, \mathcal{C}_i^{(t)}\big),$$

where $\mathrm{trace}_i^{(t)}$ is the reasoning trace produced so far, $h_i^{(t)}$ is the agent's current intermediate conclusion or partial answer, $u_i^{(t)}$ is the specific point the agent wants help with at step t, such as a missing computation, an unclear condition, a verification need, or a task fragment that it cannot complete confidently, and $\mathcal{C}_i^{(t)}$ records previous cross-agent requests made during the same run.

Definition 3 (CoTIE request)**.** A CoTIE request is a step-level cross-agent message sent from caller a_i to target agent a_j while a_i is still reasoning. It has the form

$$m_{i\to j}^{(t)} = \big(t,\, P_{\mathrm{slice}},\, \mathrm{trace}_{i,\mathrm{short}}^{(t)},\, h_i^{(t)},\, u_i^{(t)},\, r_i^{(t)},\, \mu_i^{(t)},\, \phi\big).$$

where t is the caller's current CoT step, P_{slice} is the part of the task relevant to the request, $\mathrm{trace}_{i,\mathrm{short}}^{(t)}$ is a summarized or compressed view of the reasoning produced so far, $h_i^{(t)}$ is the caller's current intermediate conclusion or partial answer, $u_i^{(t)}$ is the specific help needed at step t, $r_i^{(t)}$ is the requested help type, $\mu_i^{(t)} \in \{\mathrm{CONSULT}, \mathrm{INJECT}\}$ is the request option, and ϕ is the expected reply format.

This distinguishes CoTIE from final-output-level communication, where agents first complete their own CoT processes and only then exchange answers, rationales, critiques, or votes. We make this distinction operational through three criteria.

Definition 4 (CoT-state communication)**.** We treat an exchange as CoT-state communication when it satisfies the following operational criteria, corresponding to the state-message-integration sequence in Algorithm 1:

C1. Active caller state. The caller maintains an intermediate CoT state $\sigma_i^{(t)}$ and has not yet finalized its answer. The exchange is therefore attached to a live reasoning step of the caller, rather than to a completed local output.

C2. State-conditioned message construction. Given $\sigma_i^{(t)}$, the caller selects a reachable peer a_j with $(a_i, a_j) \in \mathcal{G}$, chooses a request option $\mu_i^{(t)} \in \{\mathrm{CONSULT}, \mathrm{INJECT}\}$, and constructs a compact message $m_{i\to j}^{(t)}$ containing the relevant task slice, summarized partial trace, current intermediate conclusion, help need, requested help type, request option, and expected reply format.

C3. Pre-final state update. The peer response $\mathrm{resp}_{j\to i}^{(t)}$ is used to update the caller's next CoT state through an integration decision, $\sigma_i^{(t+1)} \leftarrow \textsc{IntegrateResponse}(\sigma_i^{(t)}, \mathrm{resp}_{j\to i}^{(t)})$, before the caller produces its final answer. If the response is used only for final voting, final debate, or final answer selection, the exchange is final-output-level communication rather than CoT-state communication.

These criteria define the communication attachment point tested in our experiments. They do not require a specific JSON schema, prompt template, or agent architecture; they specify what must be true for communication to occur during the caller's CoT process.

2.3 CoTIE Algorithm

CoTIE operates at the level of reasoning steps. At each step, an agent may continue reasoning alone, or it may issue a cross-agent request. If a request is issued, the caller selects a target agent and chooses one of two request options: consultation or injection. The response is then integrated into the caller's CoT before the final answer is produced.

Algorithm 1 presents CoTIE as a communication principle rather than as a fixed routing or triggering algorithm. The trigger policy, peer selection policy, request-option selection, and integration rule are implementation choices under configuration Ω. What defines CoTIE is the line-level sequence: a caller maintains an intermediate CoT state (line 1), selects a reachable peer and request option (lines 7–8), sends a bounded state message before finalization (lines 9– 10), and integrates the peer response into the next CoT state before producing the final answer (lines 11– 12).

The configuration Ω specifies implementation choices such as the triggering policy, peer-selection rule, request-option selection rule, communication timing, context-management strategy, and integration rule. Different implementations may instantiate these choices differently. In the reported experiments, Ω does not impose an explicit maximum number of cross-agent requests or a fixed total-token budget. Instead, each method runs under its assigned communication protocol, and token usage is measured after execution as an empirical outcome.

CoTIE defines the communication attachment point, not a specific routing or budgeting mechanism. The key distinction is whether peer information enters after a completed output has been produced or while the caller is still reasoning from an intermediate CoT state. Thus, CoTIE can be instantiated with fixed rules, heuristic triggers, role-based matching, or learned routing policies, but the defining property is pre-final CoT-state exchange and integration.

Algorithm 1 CoTIE: CoT-State Inter-Agent Exchange Principle

Require: Task τ; agents $\mathcal{N}$; allowed communication graph $\mathcal{G}$; configuration Ω
Ensure: Final answer $\hat{y}$

1: Each agent $a_i \in \mathcal{N}$ maintains an intermediate CoT state $\sigma_i^{(t)}$ while solving τ.
2: **for** each reasoning agent $a_i \in \mathcal{N}$ **do**
3: $\quad t \leftarrow 0$
4: $\quad$ **while** a_i has not finalized its answer **do**
5: $\quad\quad a_i$ advances or updates its local reasoning state $\sigma_i^{(t)}$.
6: $\quad\quad$ **if** a communication opportunity is invoked under Ω **then**
7: $\quad\quad\quad$ Choose a reachable peer agent a_j with $(a_i, a_j) \in \mathcal{G}$.
8: $\quad\quad\quad$ Choose a request option $\mu_i^{(t)} \in \{\textsc{Consult}, \textsc{Inject}\}$.
9: $\quad\quad\quad$ Construct a compact, history-summarized CoT-state message

$$m_{i\to j}^{(t)} = \big(t,\, P_{\mathrm{slice}},\, \mathrm{trace}_{i,\mathrm{short}}^{(t)},\, h_i^{(t)},\, u_i^{(t)},\, r_i^{(t)},\, \mu_i^{(t)},\, \phi\big).$$

10: $\quad\quad\quad$ Send $m_{i\to j}^{(t)}$ while a_i is still reasoning, before a_i produces its final answer.
11: $\quad\quad\quad$ The peer a_j returns a response $\mathrm{resp}_{j\to i}^{(t)}$ according to the requested help type $r_i^{(t)}$ and reply format ϕ.
12: $\quad\quad\quad$ The caller produces the next CoT state through an integration decision before finalization:

$$\sigma_i^{(t+1)} \leftarrow \textsc{IntegrateResponse}\big(\sigma_i^{(t)}, \mathrm{resp}_{j\to i}^{(t)}\big).$$

13: $\quad\quad\quad$ Record the exchange, including caller, peer, step index, request option, request, response, integration decision, and token usage when available.
14: $\quad\quad$ **end if**
15: $\quad\quad t \leftarrow t + 1$
16: $\quad$ **end while**
17: $\quad s_i \leftarrow$ final answer produced by agent a_i.
18: **end for**
19: **return** $\hat{y}$ using the task-specific final-answer rule.

3 Experiments

We evaluate whether replacing final-output-level communication with intermediate CoT-state communication improves both task performance and observed token efficiency. Table 1 reports the main output-level communication comparison. Table 2

compares the communication attachment point under the same backbone, task inputs, agent roles, and final aggregation rules. The compared protocols use broadly comparable cross-agent communication opportunities, while differing in whether peer information is attached after local CoT completion or during an ongoing CoT state. Token usage is logged after execution as an observed protocol cost rather than imposed as a fixed budget.

3.1 Experimental Setup

We use DeepSeek-V3.2 (DeepSeek-AI, 2025) as the fixed backbone model. We evaluate on MATH (Hendrycks et al., 2021), HumanEval (Chen et al., 2021), HotpotQA (Yang et al., 2018), and three cooperative MultiAgentBench (Zhu et al., 2025) scenarios: Coding, Research, and Database. We use the cooperative scenarios because they match the setting studied here, where agents share information or complete subtasks toward a joint solution; adversarial scenarios are excluded from the main evaluation because they confound assistance with debate-style interaction.

Across communication methods, we align the backbone model, task input, agent roles, communication graph when applicable, decoding settings, history-summarization strategy, and final aggregation rule. The main manipulated variable is the attachment point of peer information: output-level baselines communicate after local CoT completion, Late CoT Message Passing shares completed CoT summaries after local reasoning is finished, and COTIE sends a compact state-conditioned request during ongoing reasoning and integrates the peer response before finalization.

We do not impose a fixed communication count or total-token budget. Each method runs under its assigned communication protocol until it produces the task-level final answer, and token usage is logged after execution as an observed protocol cost. For MultiAgentBench, we write MAB for the benchmark-provided protocol with its output-level coordination stage; MAB-COTIE uses the same benchmark setting but replaces that output-level coordination opportunity with intermediate CoT-state communication. Additional implementation, decoding, context-management, and logging details are provided in Appendix E.

3.2 Compared Methods

We compare against output-level two-agent communication baselines. On MATH, HumanEval, and HotpotQA, Output-level MP exchanges final answers or short rationales after both agents complete local CoT. Output-level MP+SC and Output-level MP+SR use the same output-level communication protocol but strengthen each agent's local reasoning with Self-Consistency or Self-Refine before the exchange. Late CoT Message Passing is used only in the attachment-point comparison: it keeps the same late communication timing but shares a summarized completed-CoT trace. On MultiAgentBench, MAB denotes the benchmark-provided output-level coordination protocol, with MAB+SC and MAB+SR as strengthened variants. Full baseline definitions and prompt templates are provided in Appendices F and G.

3.3 Evaluation Metrics

We report the official task metric for each benchmark: accuracy for MATH, Pass@1 for HumanEval, EM/F1 for HotpotQA, and the official scenario score for MultiAgentBench. We also report total token cost, defined as prompt plus completion tokens, and use score per 100K tokens for aggregate MultiAgentBench efficiency. Since our experiments do not impose a fixed token budget, token cost should be interpreted as an observed outcome of each protocol rather than as a controlled input. For MATH, HumanEval, and HotpotQA, token counts are reported for transparency and implementation-cost comparison. Our formal token-efficiency analysis focuses on cooperative MultiAgentBench, where the compared methods share the same benchmark-level multi-agent setting, broadly comparable cross-agent communication opportunities, and directly comparable token accounting. We therefore use token usage, rather than model-call count, as the primary observed cost metric. Property 1 is supported when COTIE achieves higher or comparable task performance under lower observed token cost than the reference baseline; Property 2 is supported when it also obtains a higher score per 100K tokens. The formal cost-quality definitions and scope of these properties are given in Appendix C.1, and the full MultiAgentBench cost calculations are reported in Appendix D.

3.4 Main Results Across Benchmarks

Table 1 reports the main output-level communication comparison, while Table 2 isolates communication timing. Both tables place score and token cost in adjacent columns for each benchmark, making

the cost-quality comparison visible without adding extra rows. Full cost calculations are provided in Appendix D.

On the general reasoning benchmarks, CoTIE achieves the best task performance across MATH, HumanEval, and HotpotQA. Compared with Output-level MP, CoTIE improves MATH accuracy from 86.8 to 96.0, HumanEval Pass@1 from 90.7 to 95.6, and HotpotQA score from 59.3 to 65.4, while also using fewer tokens on all three benchmarks. CoTIE also outperforms the strengthened output-level variants: compared with Output-level MP+SR, the strongest output-level variant on MATH and HumanEval, CoTIE improves MATH from 90.3 to 96.0 and HumanEval from 94.7 to 95.6 while using fewer tokens; on HotpotQA, it improves over all output-level variants by at least 6.1 points. Because token usage is measured after execution rather than fixed in advance, these results suggest that the observed gains are not obtained simply by increasing token consumption. Instead, CoTIE achieves higher task performance with lower observed token cost in these settings.

On MultiAgentBench, MAB-CoTIE improves over MAB in all three cooperative scenarios: Coding rises from 48.2 to 63.5, Research from 84.4 to 89.3, and Database from 69.8 to 82.0. It also improves the three-scenario average score from 67.5 to 78.3 while reducing average token usage from 129,270 to 67,157. These results support both cost-quality properties on cooperative MultiAgentBench: CoTIE obtains higher task quality with fewer observed tokens and a higher score per 100K tokens than MAB, MAB+SC, and MAB+SR. Appendix D reports the full aggregate and domain-level cost breakdown: CoTIE reaches 116.5 score points per 100K tokens, compared with 52.2 for MAB, 41.1 for MAB+SC, and 27.5 for MAB+SR, while reducing average token usage by 48.0% relative to MAB.

A representative logged MAB-Research trace illustrates this mechanism. At step 4, the caller requests a literature-review subtask from agent3 and integrates the returned gaps into its intermediate research state. At step 5, the caller explicitly conditions a new request to agent2 on that accepted review, asks for research ideas, partially integrates two complete directions, rejects an incomplete third idea, and later develops the robust explainable-planning direction in the final proposal. Appendix B gives the full request, response, integration record, and final-use evidence. The trace shows that the first peer response becomes part of the caller's intermediate reasoning context for the next request, and that the final proposal develops a peer-generated research direction before finalization. This distinguishes CoTIE from final-stage critique or post-hoc answer aggregation.

The broad comparison does not by itself clarify whether the gain comes from sharing reasoning content or from making that content available before finalization. We therefore compare CoTIE with a late reasoning-content control, Late CoT Message Passing, which shares completed CoT summaries only after local reasoning is finished.

3.5 Communication Timing Ablation

Table 2 compares late reasoning-content exchange with CoT-state exchange. Output-level MP exchanges final answers or short rationales after local CoT completion. Late CoT MP keeps the same late timing but shares a summarized completed-CoT trace. CoTIE instead sends a compact CoT-state request during the caller's ongoing reasoning process and integrates the peer response before finalization. The attachment-point criteria are stated in Definition 4; logging details are provided in Appendix H.

The critical comparison is Late CoT MP versus CoTIE: both expose reasoning content to another agent, but only CoTIE attaches the exchange to an ongoing reasoning state. On the general reasoning benchmarks, CoTIE improves over Late CoT MP on MATH (91.8 to 96.0), HumanEval (94.3 to 95.6), and HotpotQA (60.3 to 65.4), while also using fewer tokens on all three tasks. The same score pattern holds on MultiAgentBench: CoTIE improves Coding from 56.7 to 63.5, Research from 87.1 to 89.3, and Database from 78.6 to 82.0. Token usage is lower in all three MultiAgentBench scenarios and lower on average across the three scenarios. This comparison suggests that the advantage is not explained only by exposing agents to more reasoning text. Making peer information available before the caller finalizes its reasoning appears to be an important part of the observed gain.

Appendix H.6 describes the logged diagnostics used to inspect this mechanism, and Appendix B gives a concrete trace in which an intermediate peer response is reused in a subsequent request and then developed into the final answer.

A. General reasoning benchmarks (containing 2 agents)						
Method	**MATH**		**HumanEval**		**HotpotQA**	
	Acc.	**Tok.**	**Pass@1**	**Tok.**	**Score**	**Tok.**
Output-level two-agent communication baselines						
Output-level MP	86.8 ± 2.5	3,407	90.7 ± 0.7	1,254	59.3 ± 2.1	2,936
Output-level MP+SC	84.1 ± 4.0	5,244	92.5 ± 0.4	1,967	58.3 ± 3.1	4,512
Output-level MP+SR	90.3 ± 1.7	5,429	94.7 ± 0.9	2,021	58.0 ± 2.6	4,638
Proposed CoT-state communication						
CoTIE	**96.0 ± 1.3**	**3,271**	**95.6 ± 1.2**	**1,189**	**65.4 ± 2.4**	**2,803**

B. MultiAgentBench cooperative scenarios (containing 3 to 22 agents)						
Method	**Coding**		**Research**		**Database**	
	Score	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
Benchmark output-level protocol						
MAB	48.2 ± 1.1	241,296	84.4 ± 1.7	101,574	69.8 ± 2.1	44,939
MAB+SC	53.9 ± 1.2	291,786	85.7 ± 1.2	148,621	71.3 ± 2.7	72,658
MAB+SR	51.6 ± 0.8	401,574	87.7 ± 0.9	179,541	69.5 ± 1.9	177,656
CoT-state communication						
MAB-CoTIE	**63.5 ± 1.6**	**115,249**	**89.3 ± 1.5**	**70,407**	**82.0 ± 1.9**	**15,814**

Table 1: Main benchmark comparison with DeepSeek-V3.2. Score and total-token columns are shown side by side for each benchmark. Panel A compares two-agent output-level message-passing baselines with CoTIE on general reasoning benchmarks. Output-level MP+SC and Output-level MP+SR strengthen the local reasoning of each agent before the same output-level exchange. Panel B uses MAB for the benchmark-provided MultiAgentBench protocol with its output-level coordination stage; MAB+SC and MAB+SR add local Self-Consistency or Self-Refine before that coordination stage. The communication attachment-point comparison is reported in Table 2.

A. General reasoning benchmarks						
Method	**MATH**		**HumanEval**		**HotpotQA**	
	Acc.	**Tok.**	**Pass@1**	**Tok.**	**Score**	**Tok.**
Output-level MP	86.8 ± 2.5	3,407	90.7 ± 0.7	1,254	59.3 ± 2.1	2,936
Late CoT Message Passing	91.8 ± 1.9	3,302	94.3 ± 0.6	1,297	60.3 ± 1.6	3,027
CoTIE	**96.0 ± 1.3**	**3,271**	**95.6 ± 1.2**	**1,189**	**65.4 ± 2.4**	**2,803**

B. MultiAgentBench cooperative scenarios						
Method	**MAB-Coding**		**MAB-Research**		**MAB-Database**	
	Score	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
MAB	48.2 ± 1.1	241,296	84.4 ± 1.7	101,574	69.8 ± 2.1	44,939
MAB Late CoT MP	56.7 ± 2.7	137,857	87.1 ± 2.1	110,714	78.6 ± 0.8	68,054
MAB-CoTIE	**63.5 ± 1.6**	**115,249**	**89.3 ± 1.5**	**70,407**	**82.0 ± 1.9**	**15,814**

Table 2: Score and total-token columns are shown side by side for each benchmark. Output-level Message Passing and Late CoT Message Passing attach peer information after local CoT completion, whereas CoTIE attaches peer information to an ongoing CoT state and integrates the response before finalization. The protocols use broadly comparable cross-agent communication opportunities; the key difference is the attachment point of peer information. Token usage is logged after execution as an observed protocol cost. In Panel B, MAB is the benchmark-provided coordination baseline, MAB Late CoT MP is the late-CoT control, and MAB-CoTIE is the CoT-state instantiation on MultiAgentBench.

3.6 Mechanism and Cost-Quality Analysis

The MultiAgentBench results support two distinct cost-quality properties. First, MAB-CoTIE improves final task quality while reducing observed token usage. Compared with MAB, it raises the three-scenario average score from 67.5 to 78.3 and reduces average token usage from 129,270 to 67,157, a 48.0% reduction. This supports quality-improving token reduction rather than merely quality preservation.

Second, MAB-CoTIE improves token utilization efficiency. It reaches 116.5 aggregate score points per 100K tokens, compared with 52.2 for MAB, 41.1 for MAB+SC, and 27.5 for MAB+SR.

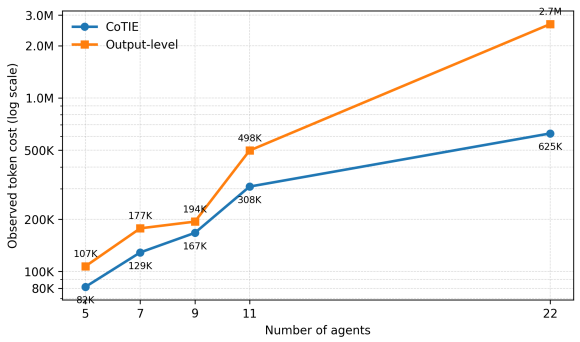

Figure 3: **Agent-count scaling of observed token cost on MultiAgentBench.** We compare MAB and MAB-CoTIE on MultiAgentBench tasks with different numbers of agents. The observed token gap generally widens as the coordination scale increases: in the 22-agent setting, MAB-CoTIE uses 624,580 tokens, while MAB uses 2,664,728 tokens. This trend is consistent with the communication-replacement view of CoTIE: task-relevant information is exchanged and integrated through intermediate CoT states rather than through bulky late-stage exchanges of completed local rationales. Because task identity, topology, and agent count are not fully disentangled, this figure should be interpreted as an observed scaling trend.

Thus, the gain is not obtained by consuming more inference tokens; in the cooperative MultiAgentBench setting, CoTIE obtains more final task performance from each unit of observed token cost.

We do not attribute this effect to fewer model invocations. In the evaluated settings, output-level communication and CoTIE use broadly comparable cross-agent communication opportunities. The relevant cost difference is that the two protocols attach peer information at different points in the reasoning trajectory, which changes how much downstream reasoning and final-stage coordination content is generated before the final answer.

The timing ablation further suggests that the improvement is not explained only by exposing agents to more reasoning text. Late CoT Message Passing also shares completed reasoning summaries, but it does so after local reasoning is finished. CoTIE improves over this late reasoning-content control across all reported task-quality metrics, indicating that making peer information available before finalization is an important part of the observed gain.

Figure 3 further shows the observed token-cost trend across MultiAgentBench tasks with different numbers of agents.

Stage	Logged evidence from a MAB-Research run
Request 1	Agent1 sends an INJECT request to Agent3 for a literature review before finalization.
Integration 1	Agent1 records the response as integrated and updates its intermediate state with accepted research gaps.
Request 2	Agent1 asks Agent2 for research ideas explicitly based on Agent3's returned review.
Integration 2	Agent1 partially integrates two complete ideas and rejects one incomplete idea.
Final use	The final proposal develops the accepted robust explainable-planning direction.

Table 3: Minimal trace evidence for pre-final CoT-state integration. The table reports only the mechanism-critical chain; the complete request, response, integration record, and final-use evidence are provided in Appendix B.

4 Conclusion

We introduced CoTIE, a CoT-state communication approach for multi-agent Chain-of-Thought reasoning. Rather than exchanging information only after agents have produced completed outputs, CoTIE attaches compact peer requests to intermediate CoT states and integrates peer responses before finalization. The contribution is a communication attachment-point principle and its step-level instantiation, rather than a new routing mechanism or workflow optimizer.

Across MATH, HumanEval, HotpotQA, and cooperative MultiAgentBench, replacing output-level communication with CoT-state communication improves final task performance under matched backbone, task inputs, agent roles, and final integration settings. The Late CoT Message Passing control further indicates that the gains are not explained only by sharing more reasoning text, but by making peer information available before the caller finalizes its CoT. On cooperative MultiAgentBench, CoTIE raises the three-scenario average score from 67.5 to 78.3 while reducing observed average token usage from 129,270 to 67,157, a 48.0% reduction. It also improves aggregate token utilization to 116.5 score points per 100K tokens, compared with 52.2 for MAB, 41.1 for MAB+SC, and 27.5 for MAB+SR. These results show that communication timing and attachment point are substantive design choices for improving both reasoning quality and observed token efficiency in cooperative multi-agent LLM systems.

5 Limitations

This work focuses on cooperative multi-agent reasoning settings, where agents share information or complete subtasks toward a joint solution. This setting matches the main motivation of CoTIE: making peer assistance available while a caller is still reasoning. Other interaction patterns, such as adversarial debate, competitive negotiation, or preference aggregation among conflicting agents, may require different communication strategies. Extending CoT-state communication to these settings is an interesting direction for future work.

Finally, the current study evaluates CoTIE primarily at the protocol level: whether moving communication from completed outputs to intermediate CoT states improves task quality and observed token utilization. The logged traces provide evidence that peer responses can be integrated before finalization, but future work can further analyze which request types, integration patterns, and task structures contribute most to the observed gains.

Ethical Considerations

CoTIE aims to make multi-agent LLM reasoning more efficient and more coordinated. Like other methods that strengthen collaborative LLM reasoning, it may support socially beneficial applications, such as scientific synthesis, software-engineering productivity, and accessible education, but it may also increase the scalability of harmful uses, such as automated disinformation generation or phishing workflows. We therefore view transparency and auditability as necessary requirements for responsible deployment. The logging schema described in Appendix H.6 is intended to support auditability by recording the caller, target agent, request content, response content, integration status, and token usage when available. Such logs can help downstream inspection and debugging of cross-agent interactions in production assistant settings.

References

Justin Chen, Swarnadeep Saha, and Mohit Bansal. 2024a. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7066–7085, Bangkok, Thailand. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

DeepSeek-AI. 2025. DeepSeek-V3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11733–11763. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2024. MapCoder: Multi-agent code generation for competitive problem solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4912–4944, Bangkok, Thailand. Association for Computational Linguistics.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904, Miami, Florida, USA. Association for Computational Linguistics.

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. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594.

Md Mahadi Hasan Nahid and Davood Rafiei. 2025. PRISM: Agentic retrieval with LLMs for multi-hop question answering. *arXiv preprint arXiv:2510.14278*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252, Mexico City, Mexico. Association for Computational Linguistics.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

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. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153. Association for Computational Linguistics.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2025a. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, Vancouver, Canada. PMLR.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025b. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8580–8622, Vienna, Austria. Association for Computational Linguistics.

A Related Work

CoT reasoning and local refinement. Chain-of-Thought (CoT) prompting improves LLM reasoning by eliciting intermediate reasoning steps before the final answer (Wei et al., 2022). A large body of work improves this paradigm by sampling, revising, or structuring a model's own reasoning process. Self-Consistency samples multiple independent CoT traces and aggregates their final answers (Wang et al., 2023). Self-Refine revises a completed output using feedback (Madaan et al., 2023), while reflection-style methods ask a model to critique and improve its previous behavior (Shinn et al., 2023). Tree-structured reasoning methods explore multiple possible reasoning paths before selecting an answer (Yao et al., 2023).

These methods are important reasoning-enhancement baselines, but they mainly operate within a single agent or on completed outputs. In our experiments, CoT, Self-Consistency, and Self-Refine are therefore treated as local reasoning enhancers inside output-level multi-agent baselines. By themselves, they do not test the communication attachment point studied in this paper: whether peer information should enter after local CoT completion or during intermediate CoT generation.

Multi-agent LLM collaboration. Multi-agent LLM systems distribute tasks across role-specialized agents and coordinate their outputs through communication, decomposition, voting, debate, or aggregation. Representative systems include AutoGen (Wu et al., 2023), MetaGPT (Hong et al., 2024), ChatDev (Qian et al., 2024), CAMEL (Li et al., 2023), and AgentVerse (Chen et al., 2024b). Task-specific multi-agent systems further apply these ideas to domains such as code generation and retrieval-augmented multi-hop question answering (Islam et al., 2024; Nahid and Rafiei, 2025). These works show that multiple agents can improve task solving by assigning roles, exchanging messages, and combining intermediate products.

Most such systems communicate through messages, drafts, critiques, or final answers after agents have produced substantial outputs. COTIE studies a more specific design choice inside multi-agent reasoning: where peer information enters the caller's CoT process. Instead of waiting for completed outputs, COTIE introduces peer input at an intermediate CoT state and allows the caller to integrate it before finalization.

Debate and output-level communication. Multi-Agent Debate and related debate-style methods let agents produce answers and then exchange arguments, critiques, or votes before selecting a final answer (Du et al., 2024; Liang et al., 2024). Other reconciliation methods similarly focus on combining or correcting completed outputs (Chen et al., 2024a). These methods are strong multi-agent reasoning baselines because they allow agents to challenge each other's conclusions and expose weaknesses in completed answers.

However, debate and reconciliation differ from COTIE in both timing and communication objective. Their main interaction usually occurs after agents have produced candidate answers or substantial reasoning traces. COTIE is cooperative and step-level: a caller asks for help on a current CoT state, and the returned peer response is integrated before the final answer is produced. For this reason, we compare against output-level message passing and debate-style baselines, while using Late CoT Message Passing as a controlled comparison that exposes reasoning content only after local reasoning is complete.

During-reasoning communication and workflow orchestration. The closest prior direction is Exchange-of-Thought, which studies cross-model communication during CoT-like problem solving (Yin et al., 2023). This makes it an important conceptual predecessor. Our focus is narrower: we isolate the communication attachment point at which peer information enters the caller's reasoning process. This lets us separate the effect of sharing reasoning content from the effect of making that content available before finalization.

A separate line of work studies how to construct, select, or optimize agent workflows. Systems such as ADaPT (Prasad et al., 2024), AFlow (Zhang et al., 2025b), and G-Designer (Zhang et al., 2025a) search over workflow structures, agent organizations, or adaptive routing strategies. Our work is complementary: rather than searching for the best overall workflow, we hold the main multi-agent setting fixed and study a specific design variable inside it—whether cross-agent information exchange happens after completed outputs or during intermediate CoT generation.

B Trace Evidence: MAB-Research HARMONIC Workflow

This section presents a trace-level case study from the Research scenario of MultiAgentBench. We select a run in which the agents are asked to develop a research proposal related to the HARMONIC framework for embodied robots. The final CoTIE output proposes a robust strategic-layer planning module for the HARMONIC architecture that maintains causal explainability under adversarial conditions and distribution shifts.

This appendix reports a logged HARMONIC workflow from the MAB-Research scenario. Rather than compressing the case into a schematic, we provide the corresponding logged metadata, request content, peer responses, integration records, and how the returned information is used before the final proposal is produced. The reported fields correspond to the logging step in Algorithm 1, line 13.

Trace summary

Main caller: `agent1`. **Exchange 1:** `agent1` → `agent3`, `request_option=INJECT`, literature review on cognitive architectures, hybrid control systems, explainable robotics, and HARMONIC-related gaps. **Exchange 2:** `agent1` → `agent2`, `request_option=INJECT`, research-idea brainstorming conditioned on the literature review returned by `agent3`. **Logged integration evidence:** both exchanges contain structured integration records, including accepted points, rejected points, conflicts, an updated caller-state summary, and an integration reason. **Observed use:** the final proposal develops the accepted direction of robust, causally explainable strategic-layer planning for HARMONIC under adversarial conditions and distribution shifts.

The purpose of this case study is to show how CoTIE operates inside a research trajectory, rather than only reporting the final answer. The run log records exchanges at discrete scheduler steps rather than at token-level CoT positions. We therefore report the recorded `step_id` values verbatim and interpret each exchange by its request content, peer response, logged integration decision, and subsequent use in the final proposal.

Exchange 1: injecting a literature-review subtask. The first relevant CoTIE exchange has `run_id=718a6798`, `caller_id=agent1`, `target_id=agent3`, `step_id=4`, `request_option=INJECT`, and `help_type=verify`. In this log, `help_type` is an auxiliary implementation tag; the exchange type is identified by `request_option` and the actual request content. At this recorded step, the caller does not ask for a final answer. Instead, it injects a literature-review subtask into a peer agent:

Exchange 1: caller request

"Conduct a comprehensive literature review on cognitive architectures for robotics, hybrid control systems, and explainable AI in robotics. Focus on understanding the current state of research in cognitive architectures for human–robot collaboration, hybrid control systems, explainability and trust in robotic systems, integration of LLMs/VLMs in robotic architectures, and the HARMONIC framework mentioned in the Introduction and similar approaches."

The omitted part of the request specifies that HARMONIC extends hybrid control systems and DIARC-style architectures through independent strategic and tactical layers. The caller asks the peer to identify research gaps and clarify what makes HARMONIC novel. Thus, the request constructs a literature-grounded intermediate state rather than asking the peer to produce the final proposal.

The peer response supplies the requested literature context. It summarizes prior work on cognitive architectures, normative behavior, collaboration systems, hybrid control, explainability, and HARMONIC-style dual-layer architectures:

Exchange 1: peer response

"The literature reveals several approaches to cognitive architectures for human–robot collaboration, including memory-centred cognitive architectures, normative architectures, essential-skills frameworks, and integrated collaboration systems. HARMONIC extends traditional hybrid control systems by making strategic and tactical layers function independently and interactively, while current LLM/VLM-based systems still face limitations in explainability, trust establishment, normative behavior, capability calibration, and human–robot information exchange."

The new log records not only that the response was integrated, but also which peer points were accepted into the caller's intermediate state. This makes the case study stronger than a manual content-level comparison.

Exchange 1: logged integration record

Integration status: `integrated`.
Accepted points include: memory-centred cognitive architectures for social interaction; normative architectures for collaborative decision-making; essential-skills frameworks for joint action; integrated collaboration systems for multi-human–robot teamwork; HARMONIC's independent strategic and tactical layers; HARMONIC's extension of DIARC-style architectures; limitations of generalist LLM/VLM modules for human-level explainability; capability-calibration problems; pacing mismatches in human–robot collaboration; and the need for explicit human–robot information exchange.
Rejected points: none.
Conflicts: none.

Updated state summary: the caller updates its research state with a literature-grounded view of cognitive architectures, hybrid control systems, explainability challenges, and HARMONIC's novel dual-layer contribution. The updated state identifies robot explainability, trust establishment, normative behavior, LLM/VLM integration limitations, and strategic–tactical coordination as the main gaps for later idea generation.

This integration record shows that the first peer response becomes part of the caller-side research state before the final proposal is written. In particular, the caller does not merely receive a final-stage comment; it records a set of accepted literature gaps that are available for subsequent reasoning.

Exchange 2: using the peer review to generate research ideas. The second relevant CoTIE exchange occurs in the same run: `run_id=718a6798`, `caller_id=agent1`, `target_id=agent2`, `step_id=5`, `request_option=INJECT`, and `help_type=verify`. This exchange is especially informative because the caller explicitly conditions the new request on the literature review returned by `agent3`:

Exchange 2: caller request

"Based on the provided Introduction about the HARMONIC framework and the comprehensive literature review conducted by agent3, brainstorm potential research ideas that build upon or address gaps in the Introduction. The Introduction discusses limitations of current robots including lack of cognitive abilities to assess semantics of situations, inability to engage in meaning-oriented dialog, struggles with disturbances and novel situations, and inability to generate causal explanations."

The omitted part of the request lists seven gaps identified in the literature review: limited explainability of LLMs/VLMs in robotics, normative behavior challenges, pacing mismatch in human–robot collaboration, capability calibration problems, contextual-understanding limitations, information exchange complexity, and skill-acquisition difficulties in shared workspaces.

This request shows a concrete information flow across agents. The caller does not issue an independent second request from scratch; it conditions the second request on the literature review returned by `agent3`. In other words, the output of the first CoTIE exchange becomes part of the caller's intermediate reasoning context for the next exchange.

The second peer then proposes research ideas using its robust-AI and safe-machine-learning expertise. The most important accepted direction is the first idea:

Exchange 2: peer response

"Idea 1: Adversarially Robust Explainable Planning for HARMONIC Framework. How can we develop robust strategic-layer planning modules that maintain explainability under adversarial conditions and distribution shifts? The approach combines LLM-based reasoning with adversarial training, a robustness-aware explanation generator, and domain adaptation for novel situations while maintaining causal explainability."

The same peer response also proposes a second complete idea on causal representation learning for cross-domain skill transfer. A third idea on adversarial-aware human–robot communication is started but is incomplete in the returned response.

Unlike a final-stage aggregation protocol, CoTIE records how the caller selectively integrates the peer response before finalization:

Exchange 2: logged integration record

Integration status: `partially_integrated`.
Accepted points: the caller accepts *Idea 1: Adversarially Robust Explainable Planning for HARMONIC Framework*, because it directly addresses explainability limitations under adversarial conditions and distribution shifts. The caller also accepts *Idea 2: Causal Representation Learning for Cross-Domain Skill Transfer*, because it addresses skill acquisition and contextual understanding gaps.
Rejected points: the caller rejects *Idea 3: Adversarial-Aware Human–Robot Communication Protocol* because the peer response is cut off and incomplete.
Conflicts: none.
Updated state summary: the caller now has two complete research directions: robust explainable planning and causal representation learning for skill transfer. The third idea is marked as incomplete and would require a follow-up completion step if it were to be used.

This integration record is important because it shows selective pre-final use of peer information. The caller does not blindly copy the peer response. Instead, it accepts two complete research directions, rejects the incomplete third direction, records no conflicts, and updates its state before producing the final answer.

Integration into the final proposal. The final output develops the accepted robust-planning direction from Exchange 2 into the central research proposal:

Final proposal

"How can we develop a robust strategic layer planning module for the HARMONIC framework that maintains causal explainability under adversarial conditions and distribution shifts while ensuring safe human–robot collaboration?"

The final proposal further specifies four technical components: a hybrid planning architecture combining LLM-based reasoning with formal symbolic planning, adversarial training with explanation-

preserving regularization, a robustness-aware explanation generator, and a domain-adaptation module that maps novel situations to known causal patterns. This final research plan directly extends the accepted robust-planning idea from the second peer response, while using the literature-grounded gaps introduced by the first peer response.

Effect on the reasoning trajectory. This trace provides stronger evidence of pre-final integration than a content-only case study because the integration decisions are explicitly logged. The first exchange, `agent1` → `agent3` at `step_id=4`, injects a literature-review subtask and returns a structured account of cognitive architectures, hybrid strategic–tactical control, explainability gaps, LLM/VLM limitations, and HARMONIC's dual-layer contribution. The integration record explicitly lists accepted points and updates the caller's state summary with these gaps.

The second exchange, `agent1` → `agent2` at `step_id=5`, is then explicitly conditioned on the literature review from `agent3`. The caller asks for research ideas that address the identified gaps using robust-AI and safe-machine-learning expertise. The integration record shows selective use: the caller partially integrates the response, accepts the robust explainable planning idea and the causal representation learning idea, rejects an incomplete communication-protocol idea, records no conflicts, and updates its state to focus on the two complete directions.

The key point is not that another agent comments on a completed answer. Rather, peer information is introduced before the final proposal is formed and is recorded as part of the caller's intermediate state. The logged trajectory shows the following pre-final chain: literature review → accepted gap summary → gap-conditioned idea generation → selective integration → final proposal. This illustrates the mechanism that CoTIE is designed to expose: cross-agent information is attached to an ongoing reasoning process and used before finalization, instead of being appended only as final-stage feedback.

Relation to the aggregate result. This example is consistent with the aggregate Research result reported in the main experiments. In the cooperative MultiAgentBench Research scenario, CoTIE obtains a higher score than the MAB baseline while using fewer observed tokens. The case study gives a qualitative illustration of how this can happen: intermediate peer assistance can make the caller's research trajectory more focused before the final answer is written. In this trace, the caller first compresses a literature review into an accepted gap summary, then uses that summary to request targeted idea generation, and finally develops one accepted direction into the final proposal. This shows how CoT-state exchange can reduce unfocused downstream reasoning and final-stage reconciliation in research-type multi-agent tasks.

C Cost-Quality Analysis and Scope

The main method section defines the operational attachment-point criteria for CoT-state communication in Definition 4. We use this appendix only to specify the cost-quality metrics and the scope of the empirical claim, avoiding a second definition of the communication protocol.

C.1 Token Cost and Reasoning Efficiency

Because CoTIE introduces cross-agent communication during CoT, its benefit should not be evaluated only by final task performance. A useful method should also avoid unnecessary reasoning cost. We therefore evaluate CoTIE using two empirical cost-quality properties. In this paper, the token-efficiency analysis focuses on the three cooperative MultiAgentBench scenarios, where all compared methods use the same benchmark-level multi-agent setting and token accounting is directly comparable.

Property 1: quality-improving token reduction. CoTIE should reduce observed token usage while reaching higher or comparable final task quality than the corresponding output-level communication baseline. Token savings are meaningful only when they do not come at the expense of task performance; in the strongest case, the method improves task quality while reducing observed token cost.

Property 2: higher token utilization efficiency. A stronger property is that CoTIE obtains more task performance per unit of token cost. In this case, the method is not merely lower-cost; it uses its inference cost more effectively.

For each method m, we measure total token cost as

$$T_m = T_m^{\text{prompt}} + T_m^{\text{completion}}.$$

When communication tokens can be separated from ordinary reasoning tokens, we also report them separately. We focus on token usage as the primary cost measure because the compared protocols are

designed to replace, rather than stack on top of, the corresponding output-level coordination opportunity. Consequently, CoTIE has approximately comparable model-call overhead to the corresponding output-level baseline in our implementation. We therefore treat score per 100K tokens as a token-utilization metric, while leaving a full latency and serving-cost analysis to future work.

To measure whether CoTIE improves efficiency, we compare both performance and token cost against final-output-level communication baselines. For a baseline b, we report token reduction as

$$\mathrm{TokenReduction}(m, b) = 1 - \frac{T_m}{T_b}.$$

A positive value means that method m uses fewer tokens than baseline b.

To measure token utilization efficiency, we also report score per 100K tokens:

$$\mathrm{ScorePer100K}(m) = \frac{S_m}{T_m/100K},$$

where S_m is the task score of method m and T_m is its average token cost per task. Higher values indicate that the method obtains more task performance from the same token cost.

These metrics are central to our setting. If CoTIE improves task performance but uses substantially more tokens, the gain may come from a higher inference cost. If CoTIE improves task performance while using fewer observed tokens, the result is consistent with the efficiency hypothesis: communicating earlier during CoT can reduce unnecessary downstream reasoning and may prevent incomplete intermediate states from expanding into longer traces.

This effect is expected to become more visible as the number of agents grows. In small two-agent settings, the amount of cross-agent context and final-stage coordination is limited. In larger multi-agent settings, however, final-stage coordination can accumulate many local histories, repeated rationales, and overlapping reasoning traces. Under such conditions, CoT-state exchange can reduce cost by making useful peer information available before each caller continues expanding its local trajectory.

C.2 Why Token Savings Can Grow with Agent Count

The efficiency hypothesis follows from the communication attachment point. In final-output-level communication, each agent may continue reasoning from an early mistake, incomplete state, or redundant local trajectory. When the agent pool is small, this overhead is limited. When the number of agents increases, however, the system may accumulate many local histories and then spend additional tokens reconciling, summarizing, or revising them only after local CoT traces have already been completed.

CoTIE changes this cost structure. A useful peer response can be integrated before the caller continues, potentially avoiding downstream reasoning from a flawed, incomplete, or redundant state. This does not imply that every individual exchange repairs an error. Rather, it suggests that earlier CoT-state exchange can reduce the amount of unnecessary reasoning that accumulates before final coordination.

This hypothesis is supported by the agent-count scaling analysis reported in Appendix E.3. Across MultiAgentBench tasks with agent counts ranging from 3 to 22, the observed token gap between CoT and CoTIE widens as the number of agents increases. In the 22-agent setting, CoTIE uses slightly over 600K tokens, while CoT exceeds 2.6M tokens. This trend helps explain why token savings are modest in two-agent general benchmarks but substantial in larger cooperative multi-agent settings.

The scaling analysis should be interpreted as a trend rather than a strict causal ablation, because task identity, topology, and agent count are not fully disentangled. A stricter future experiment could vary the number of agents while holding the task fixed. Nevertheless, the observed trend is consistent with the proposed efficiency mechanism: CoT-state communication is most useful when coordination cost itself grows with the size of the agent pool.

C.3 Scope of the Analysis

This analysis supports an empirical claim about the evaluated protocols, not a universal theorem. We do not claim that communication during CoT is always better than final-output-level communication. Final-output-level interaction remains useful for answer selection, critique, debate, and aggregation after agents have completed their own CoT processes.

The claim tested in this paper is narrower: under the same task inputs, backbone model, agent roles, and final integration rule, CoTIE tests whether communication during the CoT process can outperform communication after CoT traces are com-

pleted. Our token-efficiency claim is strongest on cooperative MultiAgentBench, where the compared methods share the same benchmark-level multi-agent setting and token accounting is directly comparable.

A stricter question remains open: whether CoTIE still outperforms final-output-level communication under exactly matched total-token usage. Our current results show that CoTIE can achieve better performance while reducing observed token cost in the evaluated cooperative MultiAgentBench settings. The agent-count scaling analysis further suggests that this observed cost reduction becomes stronger as the number of agents increases, but it should not be interpreted as a fully controlled causal ablation of agent count.

D Additional Results and Token-Cost Tables

D.1 Token Cost

This appendix expands the token-cost metrics introduced in Section 3.3 and reports the full MultiAgentBench cost calculations. We report observed token cost to test whether CoTIE improves performance by simply consuming more tokens, or whether earlier communication can also reduce unnecessary reasoning under the evaluated protocols. Table 4 reports aggregate token utilization efficiency on the cooperative MultiAgentBench scenarios, measured as aggregate score per 100K tokens.

Method	Score	Tokens	Score/100K
MAB	67.5	129,270	52.2
MAB+SC	70.3	171,022	41.1
MAB+SR	69.6	252,924	27.5
CoTIE	**78.3**	**67,157**	**116.5**

Table 4: Aggregate token utilization efficiency on the cooperative MultiAgentBench scenarios. Score and tokens are averaged over the Coding, Research, and Database scenarios. Score/100K denotes average score per 100K tokens.

D.2 MultiAgentBench Token Efficiency

The token-efficiency gain is especially relevant on MultiAgentBench, where the benchmark already includes communication among agents. We therefore analyze cost-quality trade-offs on the three cooperative MultiAgentBench scenarios.

Compared with MAB, CoTIE reduces average tokens per task by 48.0% while improving the three-scenario average score by 10.8 points. Compared with MAB+SC, CoTIE uses 60.7% fewer tokens and improves the average score by 8.0 points. Compared with MAB+SR, CoTIE uses 73.4% fewer tokens and improves the average score by 8.7 points.

These results support Property 1 on cooperative MultiAgentBench: CoTIE reduces observed token usage without sacrificing final quality. They also support Property 2 in the same setting. In terms of score per 100K tokens, CoTIE achieves 116.5, compared with 52.2 for MAB, 41.1 for MAB+SC, and 27.5 for MAB+SR. Thus, CoTIE obtains more task performance from each unit of token usage.

Scenario	MAB	CoTIE	Reduction
Coding	241,296	115,249	52.2%
Research	101,574	70,407	30.7%
Database	44,939	15,814	64.8%
Aggregate	129,270	67,157	48.0%

Table 5: Domain-level token reduction on MultiAgentBench relative to MAB. CoTIE reduces average token usage in all three cooperative scenarios, with the largest reduction on Database.

Method	MATH Acc.	HumanEval Pass@1	HotpotQA Score
MAD	86.0 ± 3.0	90.0 ± 1.5	60.3 ± 3.2
MAD+SC	92.0 ± 3.0	72.0 ± 4.3	57.7 ± 4.7
MAD+SR	85.0 ± 3.0	77.8 ± 2.0	61.0 ± 4.4
COTIE	$\mathbf{96.0 \pm 1.3}$	$\mathbf{95.6 \pm 1.2}$	$\mathbf{65.4 \pm 2.4}$

Table 6: Additional competitive multi-agent baselines on general reasoning benchmarks. SC denotes Self-Consistency and SR denotes Self-Refine. These results are reported as score-only comparisons because the MAD-family baselines were not logged under the same token-accounting format used in the main cost-quality analysis.

D.3 Additional Competitive Multi-Agent Baselines

Table 6 reports additional competitive multi-agent baselines on the three general reasoning benchmarks. These baselines are included as score-only comparisons because their token usage was not logged under the same directly comparable accounting format used for the main cost-quality analysis. Therefore, they are not used for the token-efficiency claims in Section 3; those claims focus on cooperative MultiAgentBench, where the compared protocols share directly comparable token accounting.

E Additional Experimental Details

E.1 Reproducibility and Sampling Protocol

To support reproducibility, we fix the random seeds used for benchmark subsampling and repeated runs. All reported experiments are averaged over three random seeds: 21, 42, and 63. For each of MATH, HumanEval, and HotpotQA, we use each seed to randomly select 150 evaluation examples from the corresponding benchmark split. For the three cooperative MultiAgentBench scenarios, Coding, Research, and Database, we use the same seeding procedure and randomly select 70 examples from each scenario for each seed.

All methods compared within a benchmark use the same sampled examples under the same seed. Thus, for a given seed and benchmark, COTIE and the corresponding baselines are evaluated on identical task instances. This keeps the comparison focused on the communication protocol rather than differences in the evaluated example set. The backbone model, decoding settings, agent roles, final integration rule, and context-management strategy are kept fixed across methods as described in Appendix E.

Backbone and decoding. All reported experiments use DeepSeek-V3.2 as the fixed backbone model. Unless otherwise stated, all compared methods use greedy decoding with temperature 0.0, a maximum generation length of 8192 tokens for local reasoning, and the same stopping criteria across methods. Each reported number is averaged over three runs with seeds 21, 42, and 63. These values are fixed across the two-agent output-level baselines and the CoT-state communication methods.

Benchmarks. For MATH, HumanEval, and HotpotQA, all reported baselines are two-agent communication methods with the same role pair and final integration prompt. Output-level MP exchanges completed local answers or short rationales; Output-level MP+SC and Output-level MP+SR strengthen each agent's local reasoning before the same output-level exchange. Late CoT Message Passing is used as the late reasoning-content control in the timing ablation. For MultiAgentBench, MAB denotes the benchmark-provided protocol with its task roles, interaction topology, and output-level coordination stage. MAB+SC and MAB+SR strengthen local agent reasoning before that coordination stage. MAB-CoTIE uses the same benchmark setting but replaces the output-level coordination opportunity with intermediate CoT-state communication. We evaluate the Research, Coding, and Database scenarios because they are cooperative multi-agent tasks. We exclude adversarial scenarios from the main evaluation because they would mix CoT-state assistance with debate-style interaction.

E.2 Experimental Instantiation Details

This appendix provides additional details on how the abstract CoTIE protocol is instantiated in the reported experiments. In all settings, CoTIE changes the timing and attachment point of cross-agent communication rather than the underlying task, backbone model, or final evaluation rule.

For the general reasoning benchmarks, including MATH, HumanEval, and HotpotQA, we instantiate a two-agent setting. The baseline communication methods exchange information only after an agent has produced a complete local output, whereas CoTIE allows a caller agent to send a structured CoT-state request during intermediate reasoning. This keeps the comparison focused on whether intermediate CoT-state communication provides an advantage over final-output message passing.

For MultiAgentBench, we preserve the benchmark-provided task structure and agent topology. CoTIE does not introduce new task-specific agents or change the benchmark roles. Instead, it changes how an agent can request and integrate peer information during the reasoning trajectory. This design keeps the comparison aligned with the original MultiAgentBench setting while allowing us to test whether pre-final CoT-state exchange improves the cost–quality trade-off.

Token usage is not imposed as a hard constraint during execution. We log token usage after each run and report it as an observed protocol cost. This is why the main experiments compare both task performance and token utilization rather than treating cost as a fixed input constraint.

Unrestricted communication execution. In the reported experiments, communication-based methods are not assigned a manually fixed number of communication rounds or a fixed total-token budget. Each method is executed under its corresponding communication protocol until it produces the task-level final answer. Output-level methods exchange information after local reasoning has produced completed outputs or rationales, whereas CoTIE allows peer requests to be issued during an

ongoing CoT state. We log prompt tokens, completion tokens, communication-related tokens when available, and total tokens after execution. Thus, the reported token cost is an empirical outcome of the method rather than a pre-specified budget.

History summarization. During development, we observed that passing the accumulated reasoning history verbatim can cause the context to grow quickly across reasoning steps and cross-agent exchanges. To keep context handling comparable, the reported CoT and CoTIE runs use the same history-summarization strategy: accumulated reasoning history is summarized before being reused as context. This prevents the comparison from giving CoTIE a special compression advantage and makes token usage an observed result under aligned context-management conditions.

Cost logging. We report token usage as the primary cost metric because prompt and completion tokens are consistently logged across all compared methods. For each run, we log prompt tokens, completion tokens, communication tokens when applicable, and total tokens. We do not use model-call overhead as the primary cost metric because the output-level communication baselines and CoTIE use broadly comparable cross-agent communication opportunities in the evaluated settings. The main cost difference therefore comes from the amount of reasoning and communication content generated under each protocol, which is directly captured by prompt, completion, communication, and total token counts.

Agent-count scaling analysis. For the agent-count scaling analysis in Appendix E.3, we compare CoT and CoTIE on MultiAgentBench tasks with agent counts ranging from 3 to 22. The number of agents is determined by the benchmark task configuration. We do not manually vary the number of agents for the same task; therefore, this analysis should be interpreted as an observed scaling trend across available tasks rather than a strict causal ablation. For each task, we record the observed total token usage of CoT and CoTIE and plot token cost against the number of agents.

E.3 Agent-Count Scaling of Token Cost

The main MultiAgentBench cost tables aggregate over the Coding, Research, and Database scenarios. We further analyze whether the observed token-saving effect changes with the number of agents. This analysis compares CoT and CoTIE across MultiAgentBench tasks whose agent counts range from 3 to 22.

Figure 3 reports the observed total-token cost as a function of the number of agents. The trend shows that the token gap between CoT and CoTIE becomes larger as the number of agents increases. In the 22-agent setting, CoTIE uses slightly over 600K tokens, while CoT exceeds 2.6M tokens. This suggests that CoTIE becomes increasingly token-efficient when the coordination scale grows.

This pattern helps explain the difference between the general reasoning benchmarks and MultiAgentBench. The general benchmarks use two-agent settings, where the amount of cross-agent context and final-stage coordination is limited. In contrast, larger MultiAgentBench tasks create more local histories, more repeated reasoning, and more final-stage coordination cost. CoT-state communication can reduce this accumulated cost by allowing peer information to enter before each caller expands its reasoning trajectory.

This analysis should be interpreted as a scaling trend rather than a strict causal ablation. Agent count, task identity, and benchmark topology are not fully disentangled in the available MultiAgentBench tasks. Nevertheless, the observed widening of the token gap is consistent with the hypothesis that the efficiency advantage of CoTIE is most visible when multi-agent coordination itself becomes costly.

F Method Taxonomy and Baseline Details

This appendix summarizes the method families used in the main experiments, the attachment-point comparison, the MultiAgentBench evaluation, and the agent-count token-scaling analysis. The goal of this taxonomy is to make clear which comparison each method supports. In particular, not every method is used for the same purpose: some methods are output-level baselines for the main results, some are late-reasoning-content controls for the attachment-point comparison, and some are used only for the agent-count scaling analysis.

Table 7 separates four groups:

1. general two-agent output-level communication baselines;
2. attachment-point comparison methods;
3. MultiAgentBench output-level and CoT-state protocols;
4. the CoT baseline used for the agent-count token-

scaling analysis.

Output-level Message Passing is the standard two-agent final-output communication baseline. Output-level MP+SC and Output-level MP+SR keep the same late communication stage but strengthen each agent's local reasoning before that exchange. Late CoT Message Passing is not intended as a broad new baseline; it is a late reasoning-content control. It shares CoT-like reasoning information after local CoT completion, whereas COTIE shares a compact CoT-state request during the caller's ongoing reasoning process. This comparison tests whether the advantage comes from the attachment point of peer information rather than merely from exposing agents to more reasoning text.

For MultiAgentBench, MAB denotes the benchmark-provided protocol with its output-level coordination stage. MAB LATE COT MP denotes the corresponding late-CoT control in the same benchmark setting. MAB-COTIE denotes the same benchmark setting with intermediate CoT-state communication. The agent-count scaling analysis additionally uses a MAB-COT baseline: agents solve the corresponding benchmark task with local CoT-like reasoning and the same history-summarization strategy, but without intermediate CoT-state exchange. This baseline is used only to study how observed token cost scales as the number of agents grows from 3 to 22.

Interpretation of the taxonomy. The taxonomy distinguishes three different experimental questions. First, the main benchmark comparison asks whether replacing final-output-level communication with CoT-state communication improves task performance and observed token efficiency. Second, the attachment-point comparison asks whether the gain persists when the late baseline is allowed to share completed CoT-like reasoning content. Third, the agent-count scaling analysis asks whether the observed token-cost gap between CoT and COTIE becomes larger as the number of agents increases.

The MAB-COT row is included only for the third question. It should not be confused with MAB, which is the benchmark-provided output-level coordination protocol used in the main MultiAgentBench comparison. In the scaling analysis, MAB-COT and MAB-COTIE are compared under the same task-specific agent configuration and history-summarization strategy. The key measured quantity is observed total-token cost, not a pre-specified token budget. As reported in Appendix E.3, the observed token gap widens as the number of agents increases; in the 22-agent setting, COTIE uses slightly over 600K tokens, while CoT exceeds 2.6M tokens.

G Prompt Templates

This appendix reports the prompt templates used for COTIE and the compared output-level baselines. The templates define the reasoning-agent interface, task-specific roles, peer-request actions, pre-final integration records, and result-level exchange prompts used by the baselines.

G.1 CoTIE Peer-Request Interface

CoTIE gives a reasoning agent two peer-request actions. The first action, CONSULT, asks a peer for targeted help on a specific uncertainty in the caller's current reasoning state. The second action, INJECT, asks a peer to solve a separable subtask and return a sub-result that the caller can use in its ongoing reasoning. In both cases, the request is issued before the caller produces its final answer, and the returned response is evaluated before finalization.

In the implementation, CONSULT is exposed as `request_consult`. INJECT is exposed as `request_delegate`, because the returned sub-result is injected into the caller's ongoing reasoning state.

CONSULT request.

Name: `request_consult`

Description: Ask another agent for targeted help during your reasoning. Use this when you are uncertain about a specific claim, calculation, knowledge point, evidence item, interpretation, or implementation detail before you produce your final answer. Describe your question and the kind of help needed.

Arguments:

- `question`: Your specific question. Include enough context so the peer can answer without seeing your full reasoning trace.
- `help_type`: The type of help you need. One of: `verify`, `compute`, `retrieve`, `clarify`, `debug`, or `critique`.
- `preferred_expertise`: Optional. The kind of expertise that would be most helpful, such as `calculation_verification`, `code_review`, `database_analysis`, `literature_review`, or `evidence_checking`.
- `context`: Optional. Additional context about your reasoning so far that will help the peer understand the question.

Method	Agents	Communication timing	Reasoning / message format	Role in evaluation
General two-agent output-level baselines				
Output-level MP	2	After both agents complete local CoT	Each agent first produces a completed local reasoning trace and final answer. The agents then exchange final answers, short rationales, critiques, or concerns.	Standard two-agent output-level communication baseline; also serves as the final-output control in the attachment-point comparison.
Output-level MP+SC	2	After SC-enhanced local CoT	Each agent applies local Self-Consistency before the same output-level exchange. The communication stage remains final-output-level.	Stronger output-level baseline; tests whether local sampling explains the gain.
Output-level MP+SR	2	After SR-enhanced local CoT	Each agent applies local Self-Refine before the same output-level exchange. The communication stage remains final-output-level.	Stronger output-level baseline; tests whether local self-revision explains the gain.
Attachment-point comparison methods				
Late CoT Message Passing	2	After both agents complete local CoT	The peer message contains a summarized completed-CoT trace, the final answer, and optional uncertainty or concern. The message is constructed only after local reasoning is complete.	Late reasoning-content control. It tests whether sharing CoT reasoning content after completion explains the gain, or whether pre-final CoT-state attachment matters.
COTIE	2	During the caller's ongoing CoT process	The caller sends a compact CoT-state request derived from its current reasoning state, receives a targeted peer response or sub-result, and integrates it before finalization.	Proposed two-agent CoT-state communication method; tests pre-final attachment of peer information.
MultiAgentBench output-level and CoT-state protocols				
MAB	Benchmark agents	Benchmark output-level coordination	The benchmark-provided multi-agent protocol with its task-specific roles, interaction topology, history summarization, and output-level coordination stage. In the scaling logs, MAB-CoT denotes this same output-level protocol rather than a separate baseline.	Main MultiAgentBench output-level baseline; also used as the output-level reference in the agent-count token-scaling analysis.
MAB LATE COT MP	Benchmark agents	After local CoT completion	Benchmark-specific late-CoT control: agents share summarized completed-CoT content after local reasoning has finished.	MultiAgentBench late reasoning-content control for the attachment-point comparison.
MAB+SC	Benchmark agents	Benchmark output-level coordination after local SC	Local Self-Consistency is added before the benchmark-provided output-level coordination stage.	Stronger MAB baseline; tests whether local sampling improves the benchmark protocol.
MAB+SR	Benchmark agents	Benchmark output-level coordination after local SR	Local Self-Refine is added before the benchmark-provided output-level coordination stage.	Stronger MAB baseline; tests whether local self-revision improves the benchmark protocol.
MAB-COTIE	Benchmark agents	During CoT	The benchmark setting is preserved, but the output-level coordination opportunity is replaced by intermediate CoT-state communication. A caller sends a CoT-state request, receives a peer response or sub-result, and integrates it before finalization.	COTIE instantiated in the MultiAgentBench cooperative setting; compared against MAB in the main cost-quality analysis and the agent-count token-scaling analysis.

Table 7: Method taxonomy used in the experiments. The table separates output-level baselines, attachment-point comparison methods, and MultiAgentBench-specific protocols. Output-level MP, Output-level MP+SC, Output-level MP+SR, Late CoT Message Passing, and MAB communicate only after local reasoning is complete or through the benchmark-provided output-level coordination stage. COTIE and MAB-COTIE attach communication to an ongoing CoT state and integrate the peer response before finalization. MAB-COT, when it appears in implementation logs for the agent-count scaling analysis, denotes the same output-level MAB protocol and is not treated as a separate baseline.

INJECT request.

Name: `request_delegate`

Description: Ask another agent to solve a separable subtask during your reasoning. Use this when you encounter a sub-problem that can be solved independently, has clear input/output boundaries, and returns a sub-result that can be integrated into your ongoing reasoning before you produce the final answer.

Arguments:

- `subtask_description`: Complete description of the subtask. Include what needs to be done and any constraints.
- `expected_output`: What kind of output you expect back from the peer. Be specific about format and content.
- `context`: Relevant context the peer needs to understand the subtask, such as background, current progress, constraints, partial findings, or intermediate results.
- `preferred_expertise`: Optional. The type of expertise that would be most helpful for the subtask.

G.2 CoTIE Agent Decision Rule

Each CoTIE-enabled reasoning agent receives an instruction that makes the communication attachment point explicit. At each step, the agent may continue reasoning locally, issue a CONSULT request, issue an INJECT request, use a task-provided environment tool when available, or finalize.

You are a collaborative reasoning agent. At each step, choose the next action that best advances the task.

Decision rule:

1. If you are uncertain about a fact, claim, computation, interpretation, evidence item, or implementation detail, use `request_consult`.
2. If you encounter a separable sub-problem that can be solved independently and has clear input/output boundaries, use `request_delegate`.
3. If the task requires environment data, file operations, code execution, database inspection, or other task-provided operations, use an available environment tool.
4. If you are fully confident and ready to answer, produce the final answer in the required task format.

After receiving a peer result, evaluate it before continuing. If the response is incomplete, low quality, only partially useful, or inconsistent with the task constraints, continue reasoning, request additional help if needed, or reject the unsupported part. Receiving a peer response does not mean the task is complete.

Do not guess when targeted peer assistance can resolve a specific uncertainty.

For shared-workspace tasks, such as coding tasks in MultiAgentBench, the agent also receives a workspace-safety instruction:

Before requesting peer improvements, check whether the main workspace file exists and has content. If the file is empty or missing, create the initial content yourself first using the appropriate environment tool. Only after the file exists with real content should you request peer improvements, reviews, or extensions.

G.3 Pre-final Integration Record

After receiving a peer response, the caller records how the response affects its current reasoning state. This record is used for trace analysis and auditability. It documents whether and how the peer response is integrated before finalization.

You just received a response from another agent. Evaluate what you will integrate into your reasoning.

Peer response:

`{peer_response}`

Provide a structured assessment in JSON format:

```
{
  "status":
    "integrated|partially_integrated|
     rejected|deferred",
  "accepted_points": [
    "point 1",
    "point 2"
  ],
  "rejected_points": [
    "point 3"
  ],
  "conflicts": [
    "conflict 1"
  ],
  "updated_state_summary":
    "brief summary of your updated
    reasoning state",
  "reason":
    "why you made these choices"
}
```

Return only the JSON object.

G.4 MultiAgentBench Instantiation

For MultiAgentBench experiments, we preserve the benchmark-provided agents, roles, task setting, tool environment, and default history summarization mechanism. The history summarization mechanism is part of the benchmark execution setting and is kept consistent across the corresponding MAB-based protocols. CoTIE changes the communication attachment point: a caller may issue a CONSULT or INJECT request from an ongoing reasoning state and integrate the peer response before producing the final answer.

MultiAgentBench CoTIE agent instruction.

You are a collaborative reasoning agent working on a MultiAgentBench task. Use your assigned role, available tools, and peer-request actions to solve the task.

You may use `request_consult` when you need targeted help on a specific uncertainty, such as checking a claim, debugging code, retrieving a missing fact, verifying an intermediate result, or clarifying a task constraint.

You may use `request_delegate` when you encounter a separable subtask that can be solved independently and whose result can be integrated into your ongoing solution.

When you receive a peer response, evaluate it before using it. Accept useful points, reject incomplete or unsupported points, record conflicts when they exist, and update your reasoning state before producing the final answer.

When the task is complete, produce the final answer required by the benchmark.

Benchmark history summarization prompt.

You are an advanced summarizer agent designed to condense and clarify the history of conversations between multiple agents. Your task is to analyze dialogues from various participants and generate a cohesive summary that captures the key points, themes, and decisions made throughout the interactions.

Your primary objectives are:

1. Contextual Analysis: Carefully review the entire conversation history to understand the context, including the roles of different agents and the progression of discussions.
2. Identify Key Themes: Extract the main themes, topics, and significant moments in the dialogue, noting any recurring issues or points of contention.
3. Summarize Conversations: Create a clear and concise summary that outlines the conversation's flow, important exchanges, decisions made, and any action items that emerged. Ensure that the summary reflects the contributions of each agent without losing the overall narrative.
4. Highlight Outcomes: Emphasize any conclusions reached or actions agreed upon by the agents, providing a sense of closure to the summarized conversation.

When composing the summary, maintain clarity, coherence, and logical organization. The goal is to provide a comprehensive yet succinct overview that enables the next reasoning step to understand the essential state of the multi-agent dialogue.

G.5 MATH Prompts

For MATH, we use two role-specialized agents: a Solver and a Verifier. The Solver is responsible for step-by-step solution construction, while the Verifier checks calculations, logical consistency, assumptions, and edge cases.

Solver system prompt.

You are a MATH SOLVER specializing in step-by-step reasoning. Your expertise covers algebra, geometry, calculus, number theory, combinatorics, and probability.

Output format: Your final answer must be inside `\boxed{...}`. Example: `\boxed{42}` or `\boxed{x = 5}`.

Workflow:

1. Decompose the problem into sub-steps.
2. Reason through each step carefully.
3. If uncertain about a specific calculation or claim, use `request_consult` to ask the Verifier.
4. If you identify a separable sub-problem outside your core strength, use `request_delegate`.
5. When fully confident, produce the final answer in `\boxed{...}` format.

Decision rule:

- Uncertain about a step? Use `request_consult` with `help_type` `verify` or `compute`.
- Separable sub-problem? Use `request_delegate`.
- Confident and done? Produce the final answer with `\boxed{...}`.

Do not guess. Ask the Verifier when targeted verification is needed.

Verifier system prompt.

You are a MATH VERIFIER specializing in error detection. Your expertise covers calculation verification, logic checking, proof validation, and edge case detection.

Output format: When confirming or correcting, state the correct result clearly. If producing a final answer, use `\boxed{...}`.

Workflow:

1. When consulted, check the specific step or claim presented.
2. Verify calculations, logic, assumptions, and boundary cases.
3. If you find an error, explain what is wrong and suggest the correction.
4. If the reasoning is correct, confirm it clearly.

Be thorough. A missed error can propagate through the entire solution.

MATH agent profiles.

Solver profile: Decomposes math problems, reasons step by step, and proposes solutions. Expertise tags include algebra, geometry, calculus, number theory, combinatorics, probability, stepwise reasoning, and problem decomposition.

Verifier profile: Checks calculation accuracy, finds logic errors, and suggests corrections. Expertise tags include error detection, calculation verification, logic checking, algebra, geometry, calculus, edge cases, and proof validation.

G.6 HumanEval Prompts

For HumanEval, we use a Coder and a Reviewer. The Coder writes the required Python solution, while the Reviewer checks correctness, edge cases, and implementation quality.

Coder system prompt.

You are a PYTHON DEVELOPER. Write clean, correct, well-structured code.

Output format: Return only the required Python code content, indented by four spaces. Do not include markdown fences, extra explanations, or text before or after the code.

Example for a task requiring the body of `has_close_elements(numbers, threshold)`:

```
    for i in range(len(numbers)):
         for j in range(i + 1,
         len(numbers)):
             if abs(numbers[i] -
             numbers[j]) < threshold:
                 return True
     return False
```

Workflow:

1. Read the problem carefully. Understand inputs, outputs, constraints, and edge cases.
2. Design the algorithm and write the required code content.
3. If uncertain about correctness or edge cases, use `request_consult` to ask the Reviewer.
4. If you need a full code review, use `request_delegate`.
5. When confident, output only the required code content.

Decision rule:

- Unsure about correctness? Use `request_consult` with `help_type` `verify` or `debug`.
- Need a full review? Use `request_delegate`.
- Confident? Output the required code content directly.

No markdown fences. No extra explanation. Return only the required code.

Reviewer system prompt.

You are a CODE REVIEWER specializing in correctness and quality.

Output format: When reviewing code, be specific. Point to the exact issue and suggest the fix. If producing the final code, output only the required code content.

Workflow:

1. First, cross-check the problem examples against the code logic. If the problem says `f(2, 8) => [2, 4, 6, 8]`, trace manually through the code and verify whether it actually produces that output.
2. Test mentally with sample inputs and edge cases.
3. Identify specific issues with clear explanations.
4. Provide feedback that the Coder can use to fix the code.

Decision rule:

- Need more context? Use `request_consult`.
- Code is correct? Confirm clearly.
- Found a bug? Explain the exact issue and fix.

Be specific. Point to the exact line or logic issue. Avoid vague feedback.

G.7 HotpotQA Prompts

For HotpotQA, we use a Retriever and a Synthesizer. The Retriever identifies supporting facts from the provided context, while the Synthesizer checks consistency and produces the final multi-hop answer.

Retriever system prompt.

You are an INFORMATION RETRIEVER specializing in multi-hop questions.

Output format: Your final answer must be a JSON object:

```
{"answer": "<short answer string>",
"sp": [["title", sent_id], ...]}
```

Workflow:

1. Decompose the question into sub-questions.
2. For each sub-question, identify the key facts needed.
3. Use the provided context to find supporting sentences.
4. If uncertain about a fact, use `request_consult` to ask the Synthesizer.
5. When all facts are gathered, produce the JSON with answer and supporting fact IDs.

Decision rule:

- Unsure about a fact? Use `request_consult` with `help_type` `verify` or `retrieve`.
- Need synthesis help? Use `request_delegate`.
- Facts gathered? Output JSON with answer and supporting facts.

Do not fabricate facts. Only use evidence from the context. Label supporting sentences by index.

Synthesizer system prompt.

You are an ANSWER SYNTHESIZER specializing in multi-hop QA.

Output format: Your final answer must be a JSON object:

```
{"answer": "<short answer string>",
"sp": [["title", sent_id], ...]}
```

Workflow:

1. Review the facts provided by the Retriever.
2. Check for consistency and contradictions across facts.
3. Synthesize a clear, concise answer addressing all parts of the question.
4. If facts are missing or inconsistent, use `request_consult` to ask the Retriever.
5. When the answer is complete, output the JSON.

Decision rule:

- Missing facts? Use `request_consult` with `help_type retrieve` or `clarify`.
- Need more evidence? Use `request_delegate`.
- Answer complete? Output JSON.

Do not fabricate facts. If evidence is insufficient, say so in the answer. Do not add markdown fences or extra text around the JSON.

G.8 Output-Level Baseline Prompts

This section reports the prompts used for the output-level baselines. These baselines differ from CoTIE in the communication attachment point: agents first complete their local reasoning or local refinement, and only then exchange completed outputs, rationales, critiques, or completed reasoning summaries.

G.8.1 Output-Level Message Passing

Local answer prompt.

You are a role-specialized reasoning agent. Solve the task independently using your assigned role and expertise. Produce your local answer and a concise rationale supporting it.

Do not request peer help during your reasoning. Complete your local reasoning first. Your output will later be compared with another agent's completed output.

Return format:

```
{
  "local_answer": "...",
  "rationale": "...",
  "confidence": "high|medium|low"
}
```

Final output exchange prompt.

You are given completed outputs from multiple agents. Each output contains a local answer, a concise rationale, and a confidence estimate.

Compare the completed outputs. Identify agreements, disagreements, and possible errors. Then produce the final task answer in the required benchmark format.

Inputs:

- Agent 1 completed output: `{agent1_output}`
- Agent 2 completed output: `{agent2_output}`

Instructions:

1. Check whether the completed outputs agree.
2. If they disagree, compare the supporting rationales.
3. Prefer the answer that is better supported by the task evidence, computation, examples, or constraints.
4. Produce only the final answer in the required task format.

G.8.2 Output-Level Message Passing with Self-Consistency

Local self-consistency prompt.

You are a role-specialized reasoning agent. Generate multiple independent solutions to the same task. Each solution should be produced independently, without relying on the others. After generating the candidate solutions, compare their final answers and select the most consistent answer.

Do not request peer help during candidate generation. Complete the local self-consistency process first. Your selected answer will later be compared with another agent's completed output.

Return format:

```
{
  "candidate_answers": ["...", "...",
  "..."],
  "selected_answer": "...",
  "selection_reason": "...",
  "confidence": "high|medium|low"
}
```

Output-level exchange after self-consistency.

You are given completed self-consistency outputs from multiple agents. Each agent has already generated candidate answers and selected its local final answer.

Compare the selected answers and their selection reasons. Identify agreements, disagreements, and possible errors. Produce the final task answer in the required benchmark format.

Inputs:

- Agent 1 self-consistency output: `{agent1_sc_output}`
- Agent 2 self-consistency output: `{agent2_sc_output}`

Instructions:

1. Compare the selected answers.
2. Use candidate-answer consistency as supporting evidence, not as the only decision rule.
3. Resolve disagreements using task evidence, computations, examples, or constraints.
4. Produce only the final answer in the required task format.

G.8.3 Output-Level Message Passing with Self-Refine

Initial local answer prompt.

You are a role-specialized reasoning agent. Produce an initial answer to the task using your assigned expertise. Include a concise rationale.

Return format:

```
{
  "draft_answer": "...",
  "draft_rationale": "..."
}
```

Local feedback prompt.

Review your own draft answer. Identify possible mistakes, missing evidence, incorrect calculations, edge cases, formatting problems, or unsupported claims.

Draft answer: `{draft_answer}`

Draft rationale: `{draft_rationale}`

Return format:

```
{
  "feedback": ["issue 1", "issue 2"],
  "revision_plan": "..."
}
```

Local revision prompt.

Revise your draft answer using the feedback and revision plan.

Draft answer: `{draft_answer}`

Feedback: `{feedback}`

Revision plan: `{revision_plan}`

Return your revised local answer and concise rationale.

Return format:

```
{
  "revised_answer": "...",
  "revised_rationale": "...",
  "confidence": "high|medium|low"
}
```

Output-level exchange after self-refine.

You are given completed revised outputs from multiple agents. Each agent has already produced an initial answer, reviewed it, and revised it locally.

Compare the revised answers and rationales. Identify agreements, disagreements, and possible remaining errors. Produce the final task answer in the required benchmark format.

Inputs:

- Agent 1 revised output: `{agent1_sr_output}`
- Agent 2 revised output: `{agent2_sr_output}`

Instructions:

1. Compare the revised answers.
2. Check whether either revision introduced a new inconsistency.
3. Resolve disagreements using task evidence, computations, examples, or constraints.
4. Produce only the final answer in the required task format.

G.8.4 Late CoT Message Passing

Late CoT Message Passing is the attachment-point control used to distinguish CoTIE from the explanation that agents benefit only from seeing more reasoning text. In this baseline, agents share completed reasoning summaries after local reasoning has finished.

Completed reasoning summary prompt.

You have completed your local reasoning for the task. Summarize the reasoning that led to your local answer. The summary should be concise but include the main assumptions, intermediate results, evidence, computations, and final local answer.

Do not introduce new reasoning steps after the local answer is completed. Summarize only the completed reasoning.

Return format:

```
{
  "completed_reasoning_summary":
  "...",
  "local_answer": "...",
  "confidence": "high|medium|low"
}
```

Late reasoning-content exchange prompt.

You are given completed reasoning summaries from multiple agents. Each summary was produced after the agent completed its local reasoning.

Compare the completed reasoning summaries. Identify agreements, disagreements, missing assumptions, unsupported claims, or possible errors. Then produce the final task answer in the required benchmark format.

Inputs:

- Agent 1 completed reasoning summary: `{agent1_completed_cot}`
- Agent 2 completed reasoning summary: `{agent2_completed_cot}`

Instructions:

1. Compare the final local answers.
2. Compare the reasoning summaries supporting those answers.
3. Determine whether any completed reasoning summary contains a likely error.
4. Produce only the final answer in the required task format.

G.9 MultiAgentBench Output-Level Baseline Prompts

For MultiAgentBench output-level baselines, agents complete their assigned local work under the benchmark task setting and then exchange completed outputs or completed reasoning summaries. The MAB-based baselines preserve the benchmark-provided agents, roles, tool environment, task setting, and default history summarization mechanism.

G.9.1 MAB Output-Level Protocol

Local MAB agent prompt.

You are a role-specialized agent working on a MultiAgentBench task. Complete your assigned part of the task using your role, available tools, and task environment.

Produce your completed output, including the relevant result, concise rationale, and any files, code, database results, research findings, or intermediate artifacts required by the task.

Return format:

```
{
  "completed_output": "...",
  "supporting_rationale": "...",
  "artifacts_or_results": "...",
  "confidence": "high|medium|low"
}
```

MAB output-level exchange prompt.

You are given completed outputs from the agents participating in the task. Review the completed outputs, identify agreements and inconsistencies, and produce the final task-level answer required by the benchmark.

Inputs:

- Completed outputs from participating agents: `{completed_agent_outputs}`

Instructions:

1. Check whether the completed outputs are mutually consistent.
2. Identify missing pieces, unsupported claims, implementation errors, or conflicting conclusions.
3. Use the completed outputs to form the final task-level answer.
4. Return the answer in the required benchmark format.

G.9.2 MAB with Local Self-Consistency

MAB local self-consistency prompt.

You are a role-specialized agent working on a MultiAgentBench task. Generate multiple independent local attempts for your assigned part of the task. Compare the attempts and select the most reliable local output before the output-level exchange.

Return format:

```
{
  "candidate_outputs": ["...", "...",
  "..."],
  "selected_output": "...",
  "selection_reason": "...",
  "confidence": "high|medium|low"
}
```

MAB output-level exchange after local self-consistency.

You are given selected outputs from agents that have each completed local self-consistency. Compare the selected outputs, resolve disagreements, and produce the final task-level answer required by the benchmark.

Inputs:

- Selected outputs from participating agents: `{selected_agent_outputs}`

Instructions:

1. Compare the selected outputs.
2. Use each agent's selection reason as supporting evidence.
3. Resolve conflicts using task constraints, files, code behavior, database results, research evidence, or other benchmark-specific signals.
4. Produce the final answer in the required benchmark format.

G.9.3 MAB with Local Self-Refine

MAB local draft prompt.

You are a role-specialized agent working on a MultiAgentBench task. Produce an initial local output for your assigned part of the task.

Return format:

```
{
  "draft_output": "...",
  "draft_rationale": "..."
}
```

MAB local feedback prompt.

Review your own draft output. Identify possible errors, missing pieces, unsupported claims, inefficient code, incorrect database assumptions, weak research evidence, formatting problems, or violations of task constraints.

Draft output: `{draft_output}`

Draft rationale: `{draft_rationale}`

Return format:

```
{
  "feedback": ["issue 1", "issue 2"],
  "revision_plan": "..."
}
```

MAB local revision prompt.

Revise your draft output using the feedback and revision plan.

Draft output: `{draft_output}`

Feedback: `{feedback}`

Revision plan: `{revision_plan}`

Return format:

```
{
  "revised_output": "...",
  "revised_rationale": "...",
  "confidence": "high|medium|low"
}
```

MAB output-level exchange after local self-refine.

> You are given revised outputs from participating agents. Each agent has already drafted, reviewed, and revised its local output. Compare the revised outputs, resolve disagreements, and produce the final task-level answer required by the benchmark.
>
> **Inputs:**
>
> - Revised outputs from participating agents: `{revised_agent_outputs}`
>
> **Instructions:**
>
> 1. Compare the revised outputs.
> 2. Check whether any revision introduced a new issue.
> 3. Resolve conflicts using task constraints, files, code behavior, database results, research evidence, or other benchmark-specific signals.
> 4. Produce the final answer in the required benchmark format.

G.9.4 MAB Late CoT Message Passing

MAB completed reasoning summary prompt.

> You have completed your local reasoning and local task contribution. Summarize the reasoning that led to your completed output. Include the main assumptions, intermediate findings, tool results, code changes, database conclusions, research evidence, and final local output.
>
> Do not introduce new reasoning steps after the local output is completed. Summarize only the completed reasoning and completed work.
>
> **Return format:**

```
{
  "completed_reasoning_summary":
  "...",
  "completed_output": "...",
  "artifacts_or_results": "...",
  "confidence": "high|medium|low"
}
```

MAB late reasoning-content exchange prompt.

> You are given completed reasoning summaries and completed outputs from the participating agents. Compare the summaries and outputs. Identify agreements, disagreements, missing assumptions, implementation errors, unsupported claims, or conflicting conclusions. Then produce the final task-level answer required by the benchmark.
>
> **Inputs:**
>
> - Completed reasoning summaries and outputs: `{completed_reasoning_summaries}`
>
> **Instructions:**
>
> 1. Compare the completed outputs.
> 2. Compare the completed reasoning summaries supporting those outputs.
> 3. Resolve disagreements using task constraints, files, code behavior, database results, research evidence, or other benchmark-specific signals.
> 4. Produce the final answer in the required benchmark format.

H Protocol Details and Logging

This appendix provides additional details about the CoTIE exchange protocol, the distinction between CoTIE and output-level communication baselines, and the logs used for trace analysis and reproducibility. The purpose of this appendix is to make the evaluated communication attachment point explicit.

H.1 CoTIE Exchange Structure

A CoTIE exchange is a peer-to-peer request issued by a caller agent while its reasoning is still ongoing. The exchange is attached to an intermediate CoT state rather than to a completed answer. The peer response is then evaluated and integrated before the caller produces its final answer.

Each CoTIE exchange contains three conceptual components:

1. **Caller state.** The caller has an ongoing reasoning state, including its current subgoal, partial reasoning, intermediate conclusion, uncertainty, or separable subtask.
2. **Peer request.** The caller sends a bounded request to a peer. The request may ask for targeted assistance or for a separable sub-result.
3. **Pre-final integration.** The caller evaluates the peer response and updates its reasoning state before finalization.

The defining property is the communication attachment point: the peer response can affect the caller's subsequent reasoning, not only a final-stage vote, critique, or aggregation decision.

H.2 Consultation Requests

A CONSULT request is used when the caller wants targeted help on a specific uncertainty in its current reasoning state. The peer is asked to verify, compute, retrieve, clarify, debug, or critique a localized part of the caller's ongoing reasoning.

For example, in a math task, a caller may ask a peer to verify whether an algebraic transformation is valid. In a coding task, a caller may ask whether a specific implementation step, method behavior, edge case, or error message indicates a bug. In a multi-hop QA task, a caller may ask a peer to check whether a retrieved sentence supports a proposed answer.

A CONSULT response is not treated as a separate final answer. It is used to repair, confirm, or strengthen the caller's ongoing reasoning before later steps are generated.

H.3 Injection Requests

An INJECT request is used when the caller identifies a separable subtask whose result can be incorporated into the caller's ongoing reasoning. The caller provides the relevant task fragment, context, constraints, and expected output. The peer returns a sub-result, such as a literature review fragment, debugging analysis, candidate implementation, evidence synthesis, database reasoning result, or other task-specific intermediate artifact.

The returned sub-result is not used as an independent final answer. The caller evaluates it, accepts useful parts, rejects unsupported or incomplete parts, and integrates the accepted content into its ongoing reasoning state before finalization.

Thus, CONSULT and INJECT differ in the size and role of the requested help: CONSULT asks for localized assistance on a current uncertainty, whereas INJECT asks for a structured sub-result that can be incorporated into later reasoning. Both are instances of the same CoTIE principle because both occur before the caller finalizes its answer.

H.4 Request–Response–Integration Record

For trace analysis, each CoTIE exchange is recorded as a request–response–integration record. This record is used to verify that the exchange occurred before finalization and to inspect whether the peer response was actually used by the caller.

A CoTIE record contains the following fields:

```
{
  "run_id": "...",
  "timestamp": "...",
  "caller_id": "...",
  "peer_id": "...",
  "step_id": ...,
  "request_option":
  "CONSULT | INJECT",
  "help_type": "verify | compute |
  retrieve | clarify | debug
  | critique",
  "request": {
    "question_or_subtask": "...",
    "context": "...",
    "expected_output": "..."
  },
  "peer_response": "...",
  "integration": {
    "status": "integrated |
    partially_integrated | rejected |
    deferred",
    "accepted_points": ["..."],
    "rejected_points": ["..."],
    "conflicts": ["..."],
    "updated_state_summary": "...",
    "reason": "..."
  },
  "token_usage": {
    "prompt_tokens": ...,
    "completion_tokens": ...,
    "total_tokens": ...
  },
  "answer_changed_after_response":
  true
}
```

The field `request_option` specifies whether the exchange is a CONSULT or INJECT request. The field `help_type` specifies the requested kind of assistance. The `integration` field records whether the peer response was accepted, partially accepted, rejected, or deferred. The `answer_changed_after_response` field records whether the caller's subsequent answer changed after receiving the peer response.

This unified record makes different CoTIE exchanges comparable. A verification request, a debugging request, a retrieval request, and a structured subtask injection all share the same observable trace structure: caller, peer, step index, request option, request content, peer response, integration status, updated caller state, and token usage.

H.5 Relation to Output-Level Communication

The key difference between CoTIE and output-level multi-agent communication is not whether agents exchange information. Both settings exchange information. The difference is when the exchange occurs and how the returned information can affect the caller.

In output-level communication, each agent first completes its own local CoT process and produces a completed answer, rationale, critique, vote, or summary. The exchange happens only after local reasoning has already been finalized. Such interaction can help with final answer selection or final-stage critique, but it cannot affect the caller's earlier reasoning trajectory.

In Late CoT Message Passing, agents also communicate after local reasoning is complete, but the exchanged content includes completed reasoning summaries. This is a stronger timing baseline because it exposes more reasoning content than final-

answer-only exchange. However, it still attaches peer information after local finalization.

In CoTIE, the caller sends a peer request from an ongoing reasoning state. The returned response is evaluated and integrated before the caller produces the final answer. Therefore, the peer contribution can influence subsequent reasoning steps, not only final aggregation.

Method family	When agents communicate	What is exchanged	Role
Final message passing	After completed local reasoning	Answer, rationale, critique, or vote	Output-level baseline
Late CoT Message Passing	After completed local reasoning	Completed CoT summary and answer	Timing baseline
CoTIE	During ongoing CoT reasoning	CoT-state request and peer response or sub-result	Proposed attachment point

Table 8: Communication attachment points. Output-level message passing and Late CoT Message Passing communicate only after agents complete local reasoning. CoTIE communicates during an ongoing reasoning process and integrates the peer response before finalization.

H.6 Logging and Reproducibility

For each CoTIE exchange, we log the caller agent, peer agent, step index, request option, help type, request content, peer response, integration record, and token usage when available. These logs support both cost accounting and mechanism analysis.

The primary diagnostic fields are:

- whether the request used CONSULT or INJECT;
- whether the response was integrated, partially integrated, rejected, or deferred;
- which points were accepted or rejected by the caller;
- whether conflicts were detected;
- how the caller's intermediate state was updated;
- whether the caller's later answer changed after receiving the peer response;
- the observed token usage of the exchange when available.

These logs are used to inspect whether CoTIE's gains are consistent with the proposed attachment-point mechanism: peer information is introduced into an ongoing reasoning process and can affect subsequent reasoning before the final answer is produced.

I Discussion: Semantic Compression for CoT-State Communication

CoTIE changes the attachment point of cross-agent communication. It does not introduce a new summarization algorithm or a new context-compression method. This distinction is important for interpreting the token-efficiency results.

I.1 Benchmark-Provided History Summarization

In the MultiAgentBench experiments, history summarization follows the benchmark-provided default context-management mechanism. We do not claim this summarization mechanism as part of CoTIE. It is part of the benchmark execution setting and is kept consistent across the corresponding MAB-based protocols.

Therefore, the token-efficiency result should not be interpreted as evidence that CoTIE introduces a better summarizer. The relevant comparison is instead: under the same benchmark-level context-management setting, what happens when the communication attachment point is moved from completed-output exchange to intermediate CoT-state exchange?

In the reported cooperative MultiAgentBench experiments, CoTIE obtains higher task performance while using fewer observed tokens than the corresponding output-level MAB protocols. This supports the claim that, under the benchmark execution setting, pre-final CoT-state communication can improve observed token utilization.

I.2 Why Context Management Matters

Moving communication into intermediate reasoning states creates a practical context-management challenge. A peer request should include enough information for the peer to provide useful help, but it should not repeatedly transmit irrelevant or redundant reasoning history.

A useful CoT-state request should preserve:

- the current subgoal or local uncertainty;
- the relevant task slice or constraints;
- the caller's current intermediate conclusion;
- the specific help needed at the current step;
- any previously accepted peer information that later reasoning depends on;
- the expected response format.

At the same time, it should avoid transmitting unnecessary repeated context, stale intermediate text,

unsupported speculative reasoning, or derivations that are no longer relevant to the current request.

This context-managem, but it becomes more visible when communication happens during reasoning rather than only at the final-output stage. The experiments in this paper keep the benchmark-provided context-management setting fixed when comparing MAB-based protocols, so that the primary manipulated variable remains the communication attachment point.

I.3 Future Work: Reasoning-Aware State Compression

Although CoTIE does not introduce a new summarization method, the results point to an important future direction: reasoning-aware state compression for multi-agent CoT communication.

A general-purpose dialogue summary may not preserve all information needed for downstream reasoning. In contrast, a reasoning-aware state representation should preserve entities, numerical values, constraints, adopted peer information, open questions, intermediate conclusions, and dependencies that later reasoning steps may rely on.

Future work can study adaptive CoT-state representations. For example, a caller may send a short state when asking for local verification, but a richer state when asking a peer to solve an injected subtask. Another direction is to represent the reasoning state as structured fields, such as current subgoal, known facts, unresolved uncertainties, accepted peer contributions, rejected claims, and required output format.

Such compression methods would be complementary to CoTIE. CoTIE specifies when peer information enters the reasoning trajectory; reasoning-aware compression would specify how the intermediate state should be represented so that the exchange remains informative and efficient.