

MessageSieve: Runtime Message Pruning for Accurate and Efficient LLM-Based Multi-Agent Communication

Nianhao You¹ Yubo Dong¹ Jun Wang² Hehe Fan¹ Yi Yang^{1*}

¹Zhejiang University, Hangzhou, China

²OPPO Research Institute, Shenzhen, China

{nianhaoyou, dongyubo, hehefan, yangyics}@zju.edu.cn
junwang.lu@gmail.com

Abstract

Large language model (LLM)-based multi-agent systems (MAS) improve reasoning through specialized collaboration, but dense multi-round communication can increase prompt cost and expose agents to outdated, redundant, or noisy context. A prominent line of MAS communication-efficiency work reduces communication through agent-, edge-, or topology-level structural decisions, which operate on graph structure rather than the realized text entering a specific receiver prompt. Such structural decisions are too coarse for runtime communication: messages sent over the same edge may carry useful evidence in one query, redundant content in another, or only a few useful spans. We introduce **MESSAGESIEVE**, a training-free runtime message pruning policy for fixed-graph MAS that operates on message instances rather than agents or edges. Before prompt assembly, it uses lightweight state cards and local message comparisons to preserve original text, retain compact evidence, or omit messages, without a trained selector or additional LLM call. We evaluate **MESSAGESIEVE** on two backbone models and five benchmarks covering expert QA, math reasoning, and code generation. Experiments show that **MESSAGESIEVE** improves the macro-average score over dense FULL MAS by 4.10 points on Qwen3-8B and 2.59 points on Gemini 3.1 Flash-Lite, while reducing evaluation tokens by 19–49%.

1 Introduction

LLM-based multi-agent systems (MAS) have become a practical way to improve difficult reasoning, question answering, and code generation through collaboration among specialized agents (Li et al., 2023a; Chen et al., 2024; Du et al., 2024; Liang

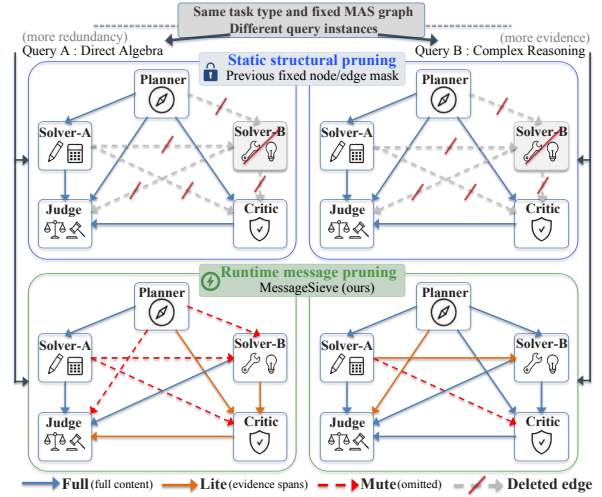


Figure 1: Static structural pruning reuses a fixed node/edge mask across different queries under the same MAS configuration. **MESSAGESIEVE** keeps the graph unchanged and prunes instantiated messages at runtime, choosing FULL to preserve the original message, LITE to retain compact evidence, or MUTE to omit low-value messages before a receiver LLM call.

et al., 2024). Practical frameworks organize agents into task-specific roles such as planners, solvers, critics, and judges, and use multi-turn communication to integrate intermediate results, which can improve final task performance (Hong et al., 2024; Wu et al., 2024; Qian et al., 2024).

This communication also creates a bottleneck. All-to-all communication in dense multi-agent debate causes input contexts and token costs to grow with the number of agents and rounds (Li et al., 2024). More generally, LLMs may fail to use relevant information robustly when it appears among distractors in a long context (Liu et al., 2024a). In multi-agent debate, agents may repeat unchanged answers or copy one another’s answers and explanations; such sycophancy can cause an available correct answer to be ignored and produce inaccurate consensus (Pitre et al., 2025).

*Corresponding author.

Project Page: <https://github.com/NianhaoYou/MessageSieve>

Prior work reduces MAS communication through structural node, edge, or topology pruning (Zhang et al., 2025a; Wang et al., 2025c; Zhang et al., 2026). These methods reveal substantial communication redundancy, but their decisions are made at the structural level. Once an agent or edge is retained, the method does not decide how each realized message on that channel should appear in a specific prompt. This leaves a finer control problem: for each receiver call, the system should decide whether each incoming message is preserved, shortened into compact evidence, or omitted.

We propose **MESSAGESIEVE**, a training-free runtime message pruning policy for MAS with a fixed graph. It keeps the original agents, graph topology, prompt templates, and number of rounds unchanged, and intervenes only before prompt assembly. For each scheduled worker LLM call, the current agent is treated as the receiver, and **MESSAGESIEVE** gathers its current-round spatial messages together with previous-round temporal messages. It then uses four receiver-local components: the Card Builder constructs a receiver-conditioned state card for each incoming message, the Relation Mapper compares messages within the current receiver’s message set, the Admission Gate decides which messages should enter the prompt, and the View Projector exposes admitted messages at an appropriate fidelity. This design lets **MESSAGESIEVE** make different decisions for different message instances even when they come from the same retained edge, reducing noisy prompt context while preserving the original MAS workflow. Figure 1 contrasts static structural pruning with our runtime message pruning under the same MAS graph.

We evaluate **MESSAGESIEVE** in a controlled MAS setting on two backbone models and five benchmarks covering expert QA, math reasoning, and code generation. Compared with single-agent prompting, dense FULL MAS, and structural pruning baselines, **MESSAGESIEVE** achieves the highest macro-average score on both models and improves the performance–token frontier, reducing evaluation tokens by 19–49% relative to dense FULL MAS.

Our contributions are:

1. We formulate fixed-graph MAS communication control as runtime pruning over realized message instances before prompt assembly, rather than node- or edge-level structural pruning.
2. We introduce **MESSAGESIEVE**, a training-free

receiver-local policy that combines message state abstraction, local relation modeling, binary admission, and fidelity-aware view projection.

3. We show that **MESSAGESIEVE** improves the performance–token frontier on two backbone models and five benchmarks, with ablations confirming the value of targeted message selection and compact evidence views.

2 Background and Problem Definition

2.1 Fixed MAS with Spatial and Temporal Messages

Prior work represents agent systems as graphs whose nodes denote agents or agent operations and whose edges describe communication or information flow (Zhuge et al., 2024; Zhang et al., 2025b). We adopt an agent-level version of this representation and model a MAS runtime as a fixed directed graph $G = (\mathcal{V}, \mathcal{E})$ used across discussion rounds, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is the agent set and $\mathcal{E}$ is the set of communication edges.

For a scheduled LLM call, we refer to the agent whose prompt is being assembled as the receiver r . Before the LLM call for receiver r at round t , it has a set of incoming message instances

$$\mathcal{M}_r^{(t)} = \mathcal{S}_r^{(t)} \cup \mathcal{T}_r^{(t)}, \quad (1)$$

where $\mathcal{S}_r^{(t)}$ denotes same-round spatial messages and $\mathcal{T}_r^{(t)}$ denotes previous-round temporal messages. Each message instance is represented as

$$m = (s, r, t, k, x_m), \quad (2)$$

where s is the sender, r is the receiver, t is the receiver round, $k \in \{\text{spatial}, \text{temporal}\}$ is the message kind, and x_m is the textual content carried by the message. This notation separates graph objects, such as nodes and edges, from the message instances that are exposed to a receiver prompt.

2.2 Message Handling in Fixed-Graph MAS

In a dense fixed-graph MAS, the graph determines which senders can communicate with each receiver. If an edge (s, r) is retained, messages instantiated on that channel are forwarded whenever the workflow schedules that communication. Thus, prompt assembly treats realized messages on the same retained channel uniformly.

Structural communication-pruning methods change this behavior by modifying graph objects rather than individual message instances (Zhang

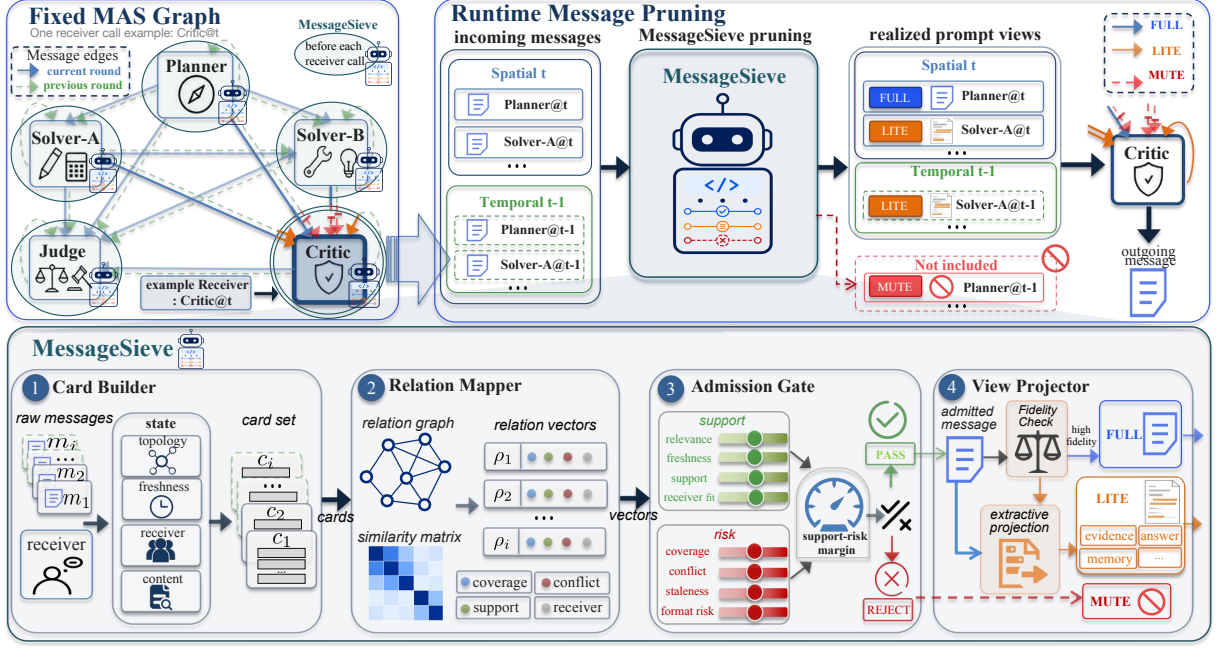


Figure 2: Overview of **MESSAGESieVE** for one receiver call, illustrated with Critic@t. The fixed MAS graph remains unchanged. Before the receiver LLM call, **MESSAGESieVE** gathers spatial and temporal incoming messages, builds receiver-side state cards, maps local relations, and admits or rejects each message. Rejected messages become MUTE; admitted messages are projected as either FULL original text or LITE compact evidence.

et al., 2025a; Wang et al., 2025c; Zhang et al., 2026). At a given structural-pruning state, removed nodes or edges suppress communication carried by those graph objects; these methods do not independently choose a textual view for each realized message. These approaches therefore reduce message volume indirectly, by pruning the graph objects that carry messages rather than deciding over individual message instances.

This is coarser than the prompt context seen by a receiver. A retained edge may carry useful evidence in one query, redundant content in another, or stale temporal information in a later round. We therefore define fixed-graph MAS communication control as a message-instance exposure problem: given G and $\mathcal{M}_r^{(t)}$ for one receiver call, decide which realized view of each message instance should enter the receiver prompt, without changing the graph, agents, or communication schedule.

3 Method

3.1 Runtime Message Pruning

MESSAGESieVE performs runtime message pruning before each scheduled *worker* LLM call, deciding which textual view of each incoming message enters the prompt while leaving the MAS workflow unchanged, as shown in Figure 2.

For a worker receiver call (r, t) , write $\mathcal{M}_r^{(t)} = \{m_i\}_{i=1}^N$, where $m_i = (s_i, r, t, k_i, x_i)$ and x_i is the message text. **MESSAGESieVE** assigns one runtime action to each incoming message instance:

$$a_i \in \{\text{FULL}, \text{LITE}, \text{MUTE}\}. \quad (3)$$

FULL exposes the original runtime message, LITE exposes a compact deterministic evidence view constructed from observable message fields or spans, and MUTE emits no prompt content. These actions are dynamic and local to the current worker call: messages produced by the same sender or edge can receive different actions across queries, receivers, and rounds.

Given the incoming message set $\mathcal{M}_r^{(t)}$ defined in Section 2.1, **MESSAGESieVE** defines a training-free policy over message-instance actions:

$$a_i = \pi_{\text{MS}}(m_i, \mathcal{M}_r^{(t)}, q, \delta; G), \quad m_i \in \mathcal{M}_r^{(t)}. \quad (4)$$

Here q is the problem input, $\delta \in \{\text{QA}, \text{Math}, \text{Code}\}$ is a task-family adapter, and G remains fixed. Across task families, the policy constructs message state cards, maps relations among messages, decides admission, and projects admitted messages into prompt-visible views. The adapter δ parameterizes the protected output unit, representative planning, route

Algorithm 1: Receiver-local worker-message pruning with **MESSAGE**SIEVE

Input: fixed graph G , input q , family δ , incoming pool $\mathcal{M} = \mathcal{M}_r^{(t)}$ for worker call (r, t)
Output: realized prompt views $\mathcal{P}$

```

1 foreach  $m_i \in \mathcal{M}$  do
2    $c_i \leftarrow \text{BuildCard}_\delta(m_i; q, r, G)$ 
3 foreach  $m_i \in \mathcal{M}$  do
4    $\rho_i \leftarrow \text{MapRelations}_\delta(c_i, \{c_j\}_{j \neq i}; r, G)$ 
5 foreach  $m_i \in \mathcal{M}$  do
6    $(g_i, z_i) \leftarrow \text{Admit}_\delta(c_i, \rho_i)$ 
7   if  $g_i = 0$  then
8      $a_i \leftarrow \text{MUTE}; v_i \leftarrow \epsilon$ 
9   else
10     $v_i \leftarrow \text{ProjectView}_\delta(m_i, c_i, \rho_i, z_i)$ 
11     $a_i \leftarrow \text{Action}(v_i)$ 
12  $\mathcal{P} \leftarrow \text{StableAssemble}([v_i]_{m_i \in \mathcal{M}, v_i \neq \epsilon})$ 
13 return  $\mathcal{P}$ 

```

eligibility and exclusions, and family-specific projection and validation: answer/evidence for QA, expression/derivation for Math, and executable artifact/interface/feedback for Code. The policy uses only runtime-visible text, metadata, graph or schedule position, and local comparisons, without a trained selector or additional LLM call. Channel labels persist, and realized views return to their original incoming order.

After the last round, the final judge performs the standard MAS aggregation over all non-empty terminal worker outputs and is not subject to message pruning. The underlying MAS workflow remains unchanged; **MESSAGE**SIEVE modifies only the message views assembled for scheduled worker calls.

Algorithm 1 applies the worker-local policy through four deterministic components: the Card Builder (Section 3.2) builds receiver-conditioned message records, the Relation Mapper (Section 3.3) produces relation vectors, family-level representative tags, and route exclusions within the incoming pool, the Admission Gate (Section 3.4) returns binary gates, and the View Projector (Section 3.5) converts admitted messages into FULL or LITE views. Rejected messages become MUTE.

3.2 Card Builder

The Card Builder converts each message into a state card, because raw text mixes sender context, temporal status, task evidence, and possible noise. For message m_i , the Card Builder constructs a lexical evidence signature $\mathbf{e}_i$ from its text x_i . It also constructs a receiver-demand signature $\mathbf{d}_i$ from the

problem input q and the current receiver’s role and fixed instruction:

$$\begin{aligned} \mathbf{e}_i &= \text{Lex}(x_i), \\ \mathbf{d}_i &= \text{Lex}(q \oplus \text{role}(r) \oplus \text{inst}(r)), \end{aligned} \quad (5)$$

where $\text{inst}(r)$ is the fixed receiver instruction and $\oplus$ denotes textual concatenation before featurization. Lex is a deterministic, length-normalized bag of lexical tokens; task-family answer, artifact, and format markers are recorded separately in ω_i . Their cosine gives receiver relevance, $r_i^{\text{rel}} = \cos(\mathbf{e}_i, \mathbf{d}_i)$.

Temporal freshness compares a temporal message with the most recent signature on the same sender–receiver pair:

$$f_i^{\text{temp}} = \begin{cases} 1 - \cos(\mathbf{e}_i, \mathbf{e}_i^-), & k_i = \text{temporal and history exists} \\ 1, & \text{otherwise.} \end{cases} \quad (6)$$

The card also records topology and channel cues g_i^{topo} , normalized answer and structured fields, and observable content/risk cues ω_i such as evidence, code, feedback, no-update, low-content, format-risk, and fragile-artifact markers:

$$c_i = \text{Card}_\delta \left[\mathbf{e}_i, \mathbf{d}_i, r_i^{\text{rel}}, f_i^{\text{temp}}, g_i^{\text{topo}}, \omega_i \right]. \quad (7)$$

Missing fields remain inactive. We use lexical TF/cosine features because they are deterministic, CPU-local, version-stable, training-free, and require no extra LLM call. Together with sender, role, channel, answer/artifact, and freshness cues, they summarize pre-call evidence for estimating repetition and receiver relevance. Appendix A.1 summarizes the runtime-visible card inputs.

3.3 Relation Mapper

For one worker call, the Relation Mapper compares cards only within the current incoming pool. It asks two complementary questions: how much of a message is already represented, and how much independent evidence it can still contribute to this receiver. For each pair $i \neq j$, lexical similarity and normalized answer/artifact agreement are

$$L_{ij} = \cos(\mathbf{e}_i, \mathbf{e}_j), \quad A_{ij} = \text{Agree}_\delta(\omega_i, \omega_j). \quad (8)$$

Agreement is defined only for comparable visible fields. Because relation mapping is receiver-local, the same-target indicator is active for every compared pair and acts as a shared same-receiver

prior in the default relation kernel, not a pair-discriminative feature.

$$T_{ij} = \text{clip}_{[0,1]} \left(w_s \mathbb{I}[s_i = s_j] \right. \\ + w_{\text{role}} \mathbb{I}[\text{role}(s_i) = \text{role}(s_j)] \\ + w_k \mathbb{I}[k_i = k_j] \\ + w_{\text{tgt}} \mathbb{I}[\text{tgt}(m_i) = \text{tgt}(m_j)] \\ \left. - w_{\text{div}} \mathbb{I}[s_i \neq s_j \wedge k_i \neq k_j] \right). \quad (9)$$

Sender, role, and channel cues still vary across pairs; crossing both sender and channel boundaries receives a small diversity discount.

We combine visible overlap and topology using the following deterministic relation rule:

$$E_{ij} = \text{clip}_{[0,1]}(\alpha_L L_{ij} + \alpha_A A_{ij}), \quad (10) \\ U_{ij} = \text{clip}_{[0,1]}(E_{ij} - \alpha_T T_{ij}).$$

Coverage captures the strongest repeated overlap, while support aggregates the strongest residual contributions:

$$\text{cov}_i = \max_{j \neq i} E_{ij} T_{ij}, \quad (11) \\ \text{sup}_i = \text{TopKAvg}_{j \neq i}(U_{ij}; K).$$

TopKAvg averages the available top $\min(K, N - 1)$ entries; when no $j \neq i$ exists, pairwise coverage, support, and conflict are initialized to zero before applying the stated singleton support floor. Together with visible-answer conflict, receiver compatibility, novelty, an optional family relation tag, and an explicit route-exclusion indicator, these values form

$$\rho_i = [\text{cov}_i, \text{sup}_i, \text{conf}_i, \text{recv}_i, \text{nov}_i, \kappa_i, \text{excl}_i]. \quad (12)$$

Here, recv_i is a composite receiver-compatibility score that extends lexical relevance with observable role and task cues. The binary field $\text{excl}_i \in \{0, 1\}$ records an explicit negative route from the active family plan; it is not a general risk score.

The tag κ_i marks an admissible family representative. Family-specific planning preserves answer-cluster representatives and stable self-memory for QA, derivation anchors for Math, and executable or feedback-bearing artifacts for Code. Appendix A.2 provides additional details on the relation cues, while Appendix A.3 describes the family-specific routing programs. These observable relations guide routing; semantic verification and global coverage remain outside their scope.

3.4 Admission Gate

The Admission Gate maps the state card and relation vector to a binary decision before prompt assembly. It forms

$$z_i = [\psi_\delta(c_i); \rho_i], \quad (13)$$

where ψ_δ converts card fields into scalar observable cues. Fixed support and risk predicate sets then produce

$$S_i^{\text{gate}} = \sum_{h \in \mathcal{H}_\delta^+} \mathbb{I}[h(z_i) = 1], \\ R_i^{\text{gate}} = \sum_{h \in \mathcal{H}_\delta^-} \mathbb{I}[h(z_i) = 1]. \quad (14)$$

Support includes protected artifacts, representative answer/evidence, fresh temporal memory, cross-sender support, useful disagreement, and receiver-relevant feedback. Risk includes low-content or no-update text, stale or covered memory, repeated answers, unsupported conflict, receiver mismatch, and format risk. Duplicate predicate names count once.

Set $\chi_i = 1$ iff no tag is required or the active family plan admits κ_i . Explicit negative routes set $\text{excl}_i = 1$; all other candidates have $\text{excl}_i = 0$. With support–risk margin

$$\Delta_i = S_i^{\text{gate}} - R_i^{\text{gate}}, \quad (15)$$

admission is

$$g_i = \mathbb{I} \left[\begin{array}{l} \text{excl}_i = 0 \wedge \chi_i = 1 \\ \wedge S_i^{\text{gate}} > 0 \wedge \Delta_i \geq \tau \end{array} \right]. \quad (16)$$

The same default threshold τ is used throughout the reported evaluation. A message is admitted when it is not route-excluded, activates an allowed support route, and satisfies the required support–risk balance. If $g_i = 0$, the message becomes MUTE; otherwise it proceeds to view realization. Appendix A.2 provides the gate rules, and Appendix A.4 reports the shared default settings.

3.5 View Projector

The View Projector separates family-required fidelity from default compression. Admission requires an admissible family route, and the corresponding family projector realizes admitted messages. Define the partial family relation projector

$$o_{i,\delta} = \begin{cases} \text{RelProj}_\delta(m_i, c_i, \rho_i), & \kappa_i \neq \emptyset, \\ \perp, & \text{otherwise,} \end{cases} \quad (17)$$

Method	Dynamic pruning	Training-free	GPQA	AQuA	MATH500	LCB	MBPP+	Avg.
Base model: Qwen3-8B								
Single	–	–	32.91	80.31	75.60	25.14	68.34	56.46
CoT	–	–	36.71	84.25	77.00	27.43	68.93	58.86
Full MAS	–	–	43.04	85.83	78.40	32.00	71.89	62.23
AgentPrune (ICLR 2025)	×	×	45.57 _{+2.53}	86.61 _{+0.78}	79.20 _{+0.80}	32.57 _{+0.57}	73.96 _{+2.07}	63.58 _{+1.35}
AgentDropout (ACL 2025)	×	×	48.73 _{+5.69}	86.22 _{+0.39}	80.40 _{+2.00}	30.29 _{−1.71}	72.78 _{+0.89}	63.68 _{+1.45}
SafeSieve (AAAI 2026)	×	×	46.20 _{+3.16}	86.61 _{+0.78}	79.60 _{+1.20}	31.43 _{−0.57}	74.26 _{+2.37}	63.62 _{+1.39}
MESSAGESIEVE	✓	✓	50.63 _{+7.59}	87.40 _{+1.57}	81.80 _{+3.40}	33.14 _{+1.14}	78.70 _{+6.81}	66.33 _{+4.10}
Base model: Gemini 3.1 Flash-Lite								
Single	–	–	65.19	88.19	89.20	48.57	80.77	74.38
CoT	–	–	66.46	90.16	89.60	46.86	81.36	74.89
Full MAS	–	–	72.78	91.34	90.60	54.29	82.84	78.37
AgentPrune (ICLR 2025)	×	×	74.05 _{+1.27}	91.34	91.60 _{+1.00}	55.43 _{+1.14}	83.43 _{+0.59}	79.17 _{+0.80}
AgentDropout (ACL 2025)	×	×	73.42 _{+0.64}	91.73 _{+0.39}	91.40 _{+0.80}	58.29 _{+4.00}	82.54 _{−0.30}	79.48 _{+1.11}
SafeSieve (AAAI 2026)	×	×	74.05 _{+1.27}	90.16 _{−1.18}	90.80 _{+0.20}	57.14 _{+2.85}	83.43 _{+0.59}	79.12 _{+0.75}
MESSAGESIEVE	✓	✓	75.32 _{+2.54}	91.73 _{+0.39}	91.40 _{+0.80}	61.71 _{+7.42}	84.62 _{+1.78}	80.96 _{+2.59}

Table 1: Main results. Values are accuracy for QA/math and pass@1 for code; AQuA denotes AQuA-RAT and LCB denotes LiveCodeBench code_generation_lite. Dynamic pruning and training-free refer to the controlled implementations evaluated here. Subscripts show absolute point differences relative to dense FULL MAS under the same base model and dataset.

A defined output is either x_i or a valid relation-preserving LITE capsule. This projector protects selected Math or Code artifacts and tagged exceptions to the default ratio path.

Let $\ell_{i,\delta} = \text{Lite}_\delta(m_i)$ be the ordinary compact candidate and $\text{Valid}_\delta(\ell_{i,\delta} \mid x_i)$ its family-aware length and format guard. The realized view is

$$v_i = \begin{cases} \epsilon, & g_i = 0, \\ o_{i,\delta}, & g_i = 1 \wedge o_{i,\delta} \neq \perp, \\ \ell_{i,\delta}, & g_i = 1 \wedge o_{i,\delta} = \perp \\ & \wedge \text{Valid}_\delta(\ell_{i,\delta} \mid x_i) = 1, \\ x_i, & \text{otherwise.} \end{cases} \quad (18)$$

Admission controls visibility, and the family projector preserves the selected relation. When no family-specific view is produced, the deterministic ordinary candidate is used only if it passes the family-aware length and format guard; invalid shortening falls back to FULL. QA capsules copy answer/evidence; Math keeps bounded expressions and derivations; Code keeps feedback, interface, or structured notes. No projector invokes another model. Primary Math anchors and admitted non-failed Code artifacts remain FULL; secondary Math anchors and failed-Code feedback become relation-preserving LITE. Failure status takes precedence over independent-source protection. Appendix A.4 provides additional projection and validation details.

4 Experiments

We organize the evaluation around three research questions. RQ1 tests whether runtime message pruning improves task performance over baselines. RQ2 examines the performance–token tradeoff, and RQ3 analyzes why the gains occur through ablations and robustness diagnostics.

4.1 Experimental Setup

We evaluate MESSAGESIEVE in fixed-graph LLM-based MAS for both task performance and communication efficiency. Following recent MAS communication-pruning work (Zhang et al., 2025a; Wang et al., 2025c; Zhang et al., 2026), we compare against single-agent prompting, dense multi-agent communication, and structural-pruning baselines reimplemented under a shared held-out evaluation protocol, starting from the same agent roster and base graph.

Models and benchmarks. We choose two backbone models to cover different deployment regimes: Qwen3-8B, an 8.2B-parameter open-weight model from the Qwen3 family (Yang et al., 2025a), and Gemini 3.1 Flash-Lite, a hosted model from Google’s Gemini family (Gemini 3.1 Flash-Lite, 2026). The evaluation spans scientific question answering, mathematical reasoning, and code generation: GPQA Diamond (Rein et al., 2024); AQuA-RAT (Ling et al., 2017); MATH500 (Hendrycks et al., 2021; Lightman et al., 2024);

LiveCodeBench (Jain et al., 2025); and MBPP+ (Austin et al., 2021; Liu et al., 2023). See Appendix B.1 for details.

Compared methods and protocol. We compare **MESSAGESIEVE** with two single-agent baselines, **SINGLE** and **COT** (Wei et al., 2022). **SINGLE** directly produces an answer or code, whereas **COT** requests brief reasoning or visible-test checks before the final output. We also compare with dense **FULL MAS** and three communication-pruning baselines: **AGENTPRUNE** (Zhang et al., 2025a), **AGENTDROPOUT** (Wang et al., 2025c), and **SAFE-SIEVE** (Zhang et al., 2026). All MAS variants share the agent roster, base graph, prompts, and two-round schedule; **FULL MAS** forwards all messages, whereas structural baselines may alter the active subgraph. For controlled held-out evaluation, the three structural baselines use the same number of adaptation examples, and their resulting structural states are frozen before evaluation. **MES-SAGESIEVE** is applied directly without adaptation examples or fitted pruning state. Appendix B.2 gives the adaptation split and baseline-specific reproduction details.

4.2 RQ1: Main Task Performance

Observation 1: Runtime message pruning improves over dense communication. Across the two model blocks in Table 1, **MESSAGESIEVE** obtains the highest macro-average score. It improves the macro-average score over dense **FULL MAS** by +4.10 points on Qwen3-8B and +2.59 points on Gemini 3.1 Flash-Lite, and is best or tied for best on nine of the ten dataset–model cells. Dense communication can be harmful as well as costly: some message instances distract receivers or reinforce stale consensus, while **MESSAGESIEVE** filters them before they affect downstream calls. Appendix C reports task-level paired uncertainty over the finite evaluation sets and repeated-endpoint diagnostics of endpoint variability.

Observation 2: Instance-level pruning improves over fitted structural pruning. **MES-SAGESIEVE** also improves over the strongest controlled fitted structural-pruning baseline in Table 1. This suggests that the realized message for the current receiver call is a finer and more flexible control unit than an agent’s or edge’s average utility. Structural pruning must make a shared decision for all messages carried by a retained or removed graph object, whereas **MESSAGESIEVE** can make dif-

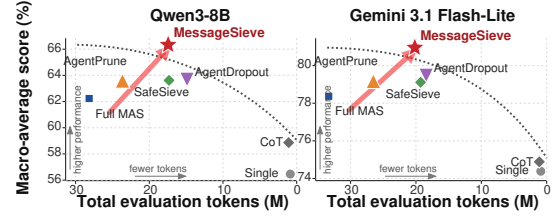


Figure 3: Performance–token tradeoff using the macro-average score across the five benchmarks. Each point reports one method under one backbone model.

ferent decisions for different realized messages on the same communication path. As a result, it can suppress a redundant, stale, or misleading message while still preserving another useful message produced by the same sender or edge in a different query, receiver, or round. These results support the view that communication utility is instance-dependent rather than well captured by a single fixed value assigned to an agent or edge.

4.3 RQ2: Performance–Token Tradeoff

Figure 3 summarizes macro-average score and evaluation-token cost across the five benchmarks. **MESSAGESIEVE** achieves the highest macro-average score on both backbone models. Across the ten model–dataset settings, it reduces total evaluation tokens by 19–49% relative to dense **FULL MAS**, yielding a stronger performance–token frontier. More aggressive structural pruning can save additional tokens but may discard useful evidence; **MESSAGESIEVE** prioritizes selective exposure. Prompt/API communication, output generation, and local routing overhead are distinct measurements. Appendix C.3 reports token accounting, cost estimation, and local routing overhead separately.

4.4 RQ3: Mechanism and Robustness Analysis

We examine which components and routing behaviors contribute to the observed gains, focusing on message-conditioned selection, relation modeling, temporal handling, and compact evidence views. The main diagnostics use Qwen3-8B on GPQA-Diamond and MBPP+; Appendix B.4 provides the protocols, and Appendix E gives trace-level examples. A complementary post-execution audit in Appendix C.4 shows that approximately 89–99% of unique answer or code-artifact sources remain visible, through either **FULL** or **LITE**, to at least one later relevant worker across the five completed

Qwen3-8B trace sets. This suggests that the routing policy usually preserves access to distinct evidence sources rather than collapsing communication to a single consensus path.

4.4.1 Component Ablations

We evaluate each component variant end to end on the same evaluation instances and under the same Qwen3-8B runtime protocol. As shown in Table 2, the Random variant achieves token reductions comparable to the full **MESSAGE**SIEVE policy but performs substantially worse on both datasets, indicating that token reduction alone does not explain the gains. Removing the Relation Mapper causes the largest drop on GPQA, while removing receiver or temporal cues also lowers the observed score relative to the full policy. The No LITE variant underperforms the full policy on both datasets, consistent with compact evidence views contributing beyond token savings. Overall, these results support an empirical role for receiver-aware selection, local relation modeling, temporal cues, and compact evidence views. As a complementary policy-level check, Appendix C.1 separately compares adaptive routing with Always-LITE, Static Edge Drop, and LLMLingua-2 (Pan et al., 2024).

Method	GPQA		MBPP+	
	Accuracy	Token Reduction	Pass@1	Token Reduction
Full MAS	43.04	–	71.89	–
Random	45.57	41.5%	73.96	16.4%
w/o Relation	44.94	42.4%	74.85	20.0%
w/o Receiver	48.10	43.2%	77.81	19.2%
w/o Temporal	48.10	42.6%	77.51	18.9%
No LITE	46.20	42.6%	75.44	15.5%
MESSAGE SIEVE	50.63	42.9%	78.70	19.0%

Table 2: End-to-end Qwen3-8B component ablations on GPQA-Diamond (accuracy) and MBPP+ (pass@1), using the same evaluation instances. Token Reduction is the percentage reduction in evaluation tokens relative to FULL MAS.

4.4.2 Pruning Strength Diagnostic

We evaluate Relaxed and Strict diagnostic configurations end to end on the same full instances and Qwen3-8B setup; neither replays a default trajectory nor selects the default policy. Relaxed removes some redundancy and staleness filters. Strict is broader, also pruning judge messages and, for Code, relaxing executable-code FULL protection; these are exceptions to default worker-message routing and dense final-judge readout. Appendix B.4.2 gives the configuration details.

Among the configurations in Table 3, the default **MESSAGE**SIEVE policy achieves the highest score on both datasets while maintaining a stronger empirical performance–token balance.

Policy	GPQA			MBPP+		
	Acc.	Eval Tok.	F/L/M	Pass@1	Eval Tok.	F/L/M
Full MAS	43.04	3.41M	100/0/0	71.89	3.55M	100/0/0
Relaxed	44.94	2.50M	44/11/45	74.85	3.24M	62/23/15
Strict	45.57	1.73M	7/21/72	74.85	2.79M	32/24/44
MESSAGE SIEVE	50.63	1.95M	10/22/68	78.70	2.88M	44/17/39

Table 3: Pruning-strength diagnostic on Qwen3-8B. F/L/M reports the percentage split of FULL, LITE, and MUTE decisions.

4.4.3 Random Bad-Agent Stress Test

We test stability under a random bad-agent attack. For each query, one worker is randomly selected for corruption, so the degraded sender and resulting harmful messages vary across queries while the topology, agent slots, and communication schedule remain fixed. On Qwen3-8B GPQA-Diamond and MBPP+, Table 4 reports the accuracy or pass@1 drop after the attack. Within this fixed-graph degraded-communication setting, **MESSAGE**SIEVE has the smallest average drop ratio. Using receiver-local observable cues, **MESSAGE**SIEVE can mute or shorten routed worker-to-worker messages without changing the roster or topology. This result measures system-level stability under degraded communication rather than directly identifying every harmful message. Appendix E provides trace-level examples of harmful-message pruning, minority-evidence preservation, LITE projection, and a failure boundary.

Method	GPQA		MBPP+		Avg. ratio
	Points drop	Ratio drop	Points drop	Ratio drop	
AgentPrune	12.66	27.78%	6.21	8.40%	18.09%
AgentDropout	11.39	23.38%	7.99	10.98%	17.18%
SafeSieve	12.03	26.03%	7.40	9.96%	18.00%
MESSAGE SIEVE	9.49	18.75%	6.21	7.89%	13.32%

Table 4: Random bad-agent stress test on Qwen3-8B. Point and ratio drops are relative to each method’s clean accuracy on GPQA and clean pass@1 on MBPP+.

5 Related Work

5.1 LLM-Based Multi-Agent Systems

LLM-based MAS support collaborative task solving through role-based interaction (Li et al., 2023a; Chen et al., 2024), task decomposition and allocation (Dong et al., 2024), and structured discussion or debate (Du et al., 2024; Liang et al., 2024).

General frameworks provide reusable abstractions for organizing agent roles, interactions, and collaborative workflows (Wu et al., 2024; Hong et al., 2024; Qian et al., 2024). Domain-specific systems extend these patterns to video-authenticity detection through adversarial debate and to scientific video understanding and education through iterative role-based collaboration (Qi et al., 2026; Xu et al., 2026). Recent studies further show that collaboration quality depends not only on agent specialization but also on how intermediate information is exchanged, aggregated, and propagated across agents. **MESSAGESIEVE** operates on the messages exchanged in these workflows while leaving agent roles and communication schedules unchanged.

5.2 MAS Communication Efficiency and Topology Design

Work on MAS communication often changes the graph structure or agent participation. GPTSwarm and G-Designer optimize communication topologies (Zhuge et al., 2024; Zhang et al., 2025b), DyLAN selects task-specific agent teams (Liu et al., 2024b), and AnyMAC predicts the next agent and selects relevant prior context (Wang et al., 2025b). Sparse MAD uses a fixed sparse topology; Agent-Prune learns edge masks, AgentDropout removes agents and edges, and SafeSieve updates structural scores during execution (Li et al., 2024; Zhang et al., 2025a; Wang et al., 2025c; Zhang et al., 2026). Beyond topology design, related work addresses long-context collaboration (Zhang et al., 2024), policy adaptation from communication feedback (Wang et al., 2025a), consensus formation (Pitre et al., 2025), cooperation (Madmoun and Lahlou, 2026), and adversarial message propagation (Shahroz et al., 2025). For long-video understanding, MACF follows a different approach in which local multimodal agents send fixed-size sequences of latent tokens to a central coordinator (Chen et al., 2026). **MESSAGESIEVE** keeps the communication graph unchanged and assigns FULL, LITE, or MUTE to each realized natural-language message before prompt assembly.

5.3 Prompt Compression and Memory Retrieval

Prompt compression reduces long inputs by selecting tokens, spans, or passages (Jiang et al., 2023, 2024; Li et al., 2023b; Pan et al., 2024). Long-context memory retrieves query-relevant informa-

tion from earlier context (Wang et al., 2023), while persistent memory systems store and retrieve useful information across interactions (Zhong et al., 2024; Packer et al., 2023). Agent systems also reuse prior experience through distilled insights or skill libraries (Zhao et al., 2024; Wang et al., 2024; Li et al., 2026). Together, these approaches either compress the input for a model call or preserve reusable information across interactions and tasks. **MESSAGESIEVE** instead decides before prompt assembly which inter-agent messages to expose to each receiver during fixed-graph MAS execution.

5.4 Model, Token, and Reasoning Efficiency

Efficiency work also targets model parameters and processed tokens: post-training compression prunes weights, model dimensions, or experts (Frantar and Alistarh, 2023; Ashkboos et al., 2024; Zhou et al., 2025), while visual-token methods reduce token counts or computation in vision and multimodal models (Bolya et al., 2023; Yang et al., 2025b; Sun et al., 2026). At the reasoning level, explicit intermediate traces can improve problem solving (Wei et al., 2022); recent work pursues efficient reasoning through controllable trace compression or structure-aware optimization under constrained response budgets (Xia et al., 2025; Dong et al., 2026), while query refinement provides another way to elicit stronger reasoning (Zhou et al., 2026). These approaches target model parameters, visual tokens, or single-model reasoning traces; **MESSAGESIEVE** instead controls which inter-agent messages enter a receiver prompt.

6 Conclusion

We studied communication efficiency in fixed-graph LLM-based MAS at the level of runtime message instances rather than node- or edge-level structural decisions. **MESSAGESIEVE** keeps the workflow unchanged and decides before prompt assembly whether each incoming message should be preserved, shortened into compact evidence, or omitted. Across two backbone models and five benchmarks, **MESSAGESIEVE** improves the performance–token frontier over dense communication and controlled structural-pruning baselines, showing that dense communication can add cost and distract receivers. These results suggest that efficient MAS design should consider not only who communicates, but which realized messages are worth exposing to each receiver at inference time.

Limitations

This work studies runtime message pruning under a fixed MAS graph. We keep agents, prompts, topology, and the number of rounds unchanged to isolate message-instance control, but this setting does not cover MAS that dynamically create agents, tools, or communication links during execution. **MESSAGESIEVE** uses deterministic LITE views constructed primarily from copied spans and structured visible fields, with a small number of family-safe fixed markers. Our experiments cover two backbone models and five benchmarks across QA, math, and code. Broader model families, longer interaction horizons, and more open-ended agent workflows remain useful directions for future work.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (92570101) and the Earth System Big Data Platform of the School of Earth Sciences, Zhejiang University.

References

- Saleh Ashkboos, Maximilian L. Croci, Marcelo Genari do Nascimento, Torsten Hoefler, and James Hensman. 2024. **SliceGPT: Compress large language models by deleting rows and columns**. In *International Conference on Learning Representations*, volume 2024, pages 11682–11701.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. **Program synthesis with large language models**. *Preprint*, arXiv:2108.07732.
- Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2023. **Token Merging: Your ViT but faster**. In *International Conference on Learning Representations*.
- Kerui Chen, Jinglu Wang, Jianrong Zhang, Ming Li, Yan Lu, and Hehe Fan. 2026. **Scaling video understanding via compact latent multi-agent collaboration**. *Preprint*, arXiv:2605.00444.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. **AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors**. In *International Conference on Learning Representations*, volume 2024, pages 20094–20136.
- Yubo Dong, Hehe Fan, Linchao Zhu, and Yi Yang. 2026. **Structured reasoning for LLMs: A unified framework for efficiency and explainability**. In *International Conference on Learning Representations*, volume 2026, pages 106183–106215.
- Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. **VillagerAgent: A graph-based multi-agent framework for coordinating complex task dependencies in Minecraft**. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16290–16314, Bangkok, Thailand. Association for Computational Linguistics.
- 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.
- Elias Frantar and Dan Alistarh. 2023. **SparseGPT: Massive language models can be accurately pruned in one-shot**. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.
- Gemini 3.1 Flash-Lite. 2026. Gemini 3.1 Flash-Lite - Model Card. <https://deepmind.google/models/model-cards/gemini-3-1-flash-lite/>.
- 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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven 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 *International Conference on Learning Representations*, volume 2024, pages 23247–23275.
- Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024. **AgentCoder: Multi-agent-based code generation with iterative testing and optimisation**. *Preprint*, arXiv:2312.13010.
- 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.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. **LiveCodeBench: Holistic and contamination free evaluation of large language models for code**. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

- Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. [LLMLingua: Compressing prompts for accelerated inference of large language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376.
- Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. [LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677.
- Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. [CAMEL: Communicative agents for “Mind” exploration of large language model society](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008.
- Haoyuan Li, Zhengdong Hu, Jun Wang, Hehe Fan, and Yi Yang. 2026. [Skill-3D: Evolving scene-aware skills for agentic 3D spatial reasoning](#). *Preprint*, arXiv:2606.07436.
- Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023b. [Compressing context to enhance inference efficiency of large language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353.
- Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. 2024. [Improving multi-agent debate with sparse communication topology](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7281–7294.
- 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.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *International Conference on Learning Representations*.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. [Program induction by rationale generation: Learning to solve and explain algebraic word problems](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. [Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 21558–21572.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranajape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. [Lost in the middle: How language models use long contexts](#). *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024b. [A dynamic LLM-powered agent network for task-oriented agent collaboration](#). In *First Conference on Language Modeling*.
- Hachem Madmoun and Salem Lahlou. 2026. [Communication enables cooperation in LLM agents: A comparison with curriculum-based approaches](#). In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 307–321.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. [MemGPT: Towards LLMs as operating systems](#). *Preprint*, arXiv:2310.08560.
- Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. [LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 963–981.
- Priya Pitre, Naren Ramakrishnan, and Xuan Wang. 2025. [CONSENSAGENT: Towards efficient and effective consensus in multi-agent LLM interactions through sycophancy mitigation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22112–22133.
- Hongyuan Qi, Feifei Shao, Ming Li, Hehe Fan, and Jun Xiao. 2026. [DVAR: Adversarial multi-agent debate for video authenticity detection](#). *Preprint*, arXiv:2604.16987.
- 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.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. [GPQA: A graduate-level Google-Proof Q&A benchmark](#). In *First Conference on Language Modeling*.
- Rana Shahroz, Zhen Tan, Sukwon Yun, Charles Fleming, and Tianlong Chen. 2025. [Agents under siege: Breaking pragmatic multi-agent LLM systems with optimized prompt attacks](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9661–9674.

- Zixun Sun, Yubo Dong, Hehe Fan, and Yi Yang. 2026. [Hi-Lo Prune: Look at what you'll lose before pruning with hierarchical token selection](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 31941–31951.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. [Voyager: An open-ended embodied agent with large language models](#). *Transactions on Machine Learning Research*.
- Kuan Wang, Yadong Lu, Michael Santacrose, Yeyun Gong, Chao Zhang, and Yelong Shen. 2025a. [Adapting LLM agents with universal communication feedback](#). In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6105–6122.
- Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. 2025b. [AnyMAC: Cascading flexible multi-agent collaboration via next-agent prediction](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11555–11567.
- Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023. [Augmenting language models with long-term memory](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 74530–74543.
- Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. 2025c. [Agent-Dropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035, Vienna, Austria. Association for Computational Linguistics.
- 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, Ryan W. White, Doug Burger, and Chi Wang. 2024. [AutoGen: Enabling next-gen LLM applications via multi-agent conversations](#). In *First Conference on Language Modeling*.
- Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. [TokenSkip: Controllable chain-of-thought compression in LLMs](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363, Suzhou, China. Association for Computational Linguistics.
- Zhiyu Xu, Weilong Yan, Yufei Shi, Xin Meng, Tao He, Huiping Zhuang, Ming Li, and Hehe Fan. 2026. [SciEducator: Scientific video understanding and educating via Deming-Cycle multi-agent system](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26497–26507.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. 2025b. [VisionZip: Longer is better but not necessary in vision language models](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19792–19802.
- Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. 2025a. [Cut the crap: An economical communication pipeline for LLM-based multi-agent systems](#). In *International Conference on Learning Representations*, volume 2025, pages 75389–75428.
- Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2025b. [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*, pages 76678–76692. PMLR.
- Ruijia Zhang, Xinyan Zhao, Ruixiang Wang, Sigen Chen, Guibin Zhang, An Zhang, Kun Wang, and Qingsong Wen. 2026. [SafeSieve: From heuristics to experience in progressive pruning for LLM-based multi-agent communication](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(35):29892–29900.
- Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arik. 2024. [Chain of agents: Large language models collaborating on long-context tasks](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 132208–132237.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. [ExpeL: LLM agents are experiential learners](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642.
- Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. [MemoryBank: Enhancing large language models with long-term memory](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731.

Yixiao Zhou, Dongzhou Cheng, Zhiliang Wu, Yi Yang, Yu Cheng, and Hehe Fan. 2026. [One refiner to unlock them all: Inference-time reasoning elicitation via reinforcement query refinement](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 38957–38978, San Diego, California, United States. Association for Computational Linguistics.

Yixiao Zhou, Ziyu Zhao, Dongzhou Cheng, Zhiliang Wu, Jie Gui, Yi Yang, Fei Wu, Yu Cheng, and Hehe Fan. 2025. [Dropping experts, recombining neurons: Retraining-free pruning for sparse mixture-of-experts LLMs](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 15169–15186, Suzhou, China. Association for Computational Linguistics.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. [GPTSwarm: Language agents as optimizable graphs](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62743–62767. PMLR.

A Method Details

A.1 Runtime Interface and Card Record

MESSAGESIEVE runs immediately before each scheduled worker call. For receiver r at round t , its pool contains the same-round spatial messages and previous-round temporal messages already available under the fixed schedule. Decisions are keyed by message identifier and surviving views are re-assembled in the original incoming order.

Table 5 summarizes the runtime-visible inputs recorded by the default Card Builder; parser details are collected with the reproduction specification in Appendix A.4.

Signal source	Recorded cues	Routing role
Message and receiver text	Message and task–receiver lexical signatures	Overlap, novelty, and receiver relevance
Routing context	Sender/receiver topology, channel, and temporal freshness	Lineage, route, and temporal comparison
Visible task structure	Answer, evidence, equation, code/interface, feedback, no-update, fragility, and format cues	Family planning and gate reasons

Table 5: Runtime-visible card inputs in the default implementation.

Scope invariant

Routing is receiver-local and pre-call: agents, graph, schedule, rounds, and already-generated messages remain unchanged; only the receiver output and later generation may differ. The dense final judge receives every non-empty terminal worker output as FULL in stable order.

A.2 Relation, Admission, and Default Projection Details

The reported evaluations use one shared relation-and-gate configuration across datasets and backbones; task-family differences enter through the adapter in Appendix A.3. The shared default configuration is provided with the reproduction specification in Appendix A.4.

Because every pair in a receiver-local worker pool has the same target, the w_{tgt} term contributes a shared same-receiver prior in the default relation kernel. Sender, role, channel, and diversity terms provide within-pool pair discrimination.

Equations 8–11 define the shared relation kernel. Agreement is active only when a pair exposes comparable normalized fields: matching visible answers give $A_{ij} = 1$, two code-bearing artifacts receive the coarse artifact value 0.5, and all other pairs contribute zero agreement. Conflict is likewise active only for comparable visible answers that disagree. Novelty is cosine distance from the receiver context, which combines the receiver role/instruction and available self-memory; none of these quantities estimates correctness.

Coverage measures the strongest overlap that also follows a repeated route, whereas support averages the strongest residual contributions. A visible answer that occurs only once receives a default minority-support floor, keeping minority evidence available for gate inspection.

The gate in Equations 15–16 materializes named Boolean reasons, deduplicates their names, and counts distinct support and risk predicates. Here $\chi_i = 1$ iff no relation plan is required or its family plan admits κ_i . The route indicator excl_i is initialized to zero; an explicit negative route such as same-source Code repetition sets it to one and is rejected before the count-based margin is applied. Thus generic cues cannot bypass required family routing. For candidates that survive route eligibility and exclusion, the shared default threshold requires positive support and $S_i^{\text{gate}} \geq R_i^{\text{gate}}$.

Support reasons cover a family representative or protected artifact, a unique or minority answer, useful disagreement, fresh self-memory, cross-

sender contribution, and receiver-relevant evidence or feedback. Risk reasons mark a missing family route, empty, low-content, or no-update text, stale or already-covered evidence, unsupported conflict, receiver mismatch, and format hazard. Representation, temporal, coverage, support, and receiver-mismatch predicates use the same default cutoffs throughout the reported evaluation. Duplicate predicate names count once, and duplication requires both high similarity and low novelty.

Admission requires an admissible family route, and admitted messages are realized by the corresponding family projector. When the projector does not produce a family-specific view, the ordinary compact candidate is retained only if it passes Equation 19; otherwise the original message remains FULL. Compound predicates and stable tie ranks apply the same pre-call observables and definitions.

A.3 Family Routing Programs

All families use the same Card–Relation–Gate–Projector pipeline. They differ in the protected output unit, route eligibility and exclusions, representative planning, and family-specific projection and validation. Algorithm 2 expands the Relation Mapper family plan used inside Algorithm 1.

Algorithm 2 is the canonical representative-selection procedure. Table 6 summarizes only how selected messages are realized and validated.

Code compact-view details. Code fields follow FEEDBACK $\succ$ INTERFACE $\succ$ SUMMARY $\succ$ KEY EVIDENCE $\succ$ DELIVERABLE $\succ$ ANSWER. The first four populated fields are emitted, each capped at 320 characters. Feedback, Summary, and Key Evidence use at most two lines; an unstructured Summary fallback uses at most two sentences. Optional fields with cosine at least 0.9 to an earlier field are skipped. Compact evidence excludes fenced executable bodies and a bare <code omitted> payload.

A.4 Projection Safety and Reproducibility Boundary

View realization follows the canonical branch order in Equation 18: reject to MUTE; otherwise prefer the family relation projector, then retain a valid ordinary compact candidate when needed; invalid shortening falls back to FULL.

For an ordinary candidate v from source x , va-

Algorithm 2: Family-aware representative planning inside the Relation Mapper

Input: receiver pool $\mathcal{M}$, cards $\{c_i\}$, family δ , receiver r

Output: admissible tags κ_i and route-exclusion indicators excl_i

```

1 Initialize every  $\kappa_i \leftarrow \emptyset$  and  $\text{excl}_i \leftarrow 0$ ; use stable
  incoming order for exact ties
2 if  $\delta = \text{QA}$  then
3   cluster messages by visible normalized answer
4   retain one eligible consensus representative, or
5   one representative per disagreement cluster
6   select stable self-temporal answer memory in a
    separate temporal subplan
7 if  $\delta = \text{Math}$  then
8    $\mathcal{A} \leftarrow$  eligible self-temporal or disagreement
9   answer/evidence anchors
10  if  $\mathcal{A} \neq \emptyset$  then
11     $\mathcal{R} \leftarrow$  self-temporal anchors if any exist,
12    otherwise  $\mathcal{A}$ 
13    tag the stable top-ranked item in  $\mathcal{R}$  as the
14    unique FULL anchor
15    tag every other item in  $\mathcal{A}$  as a bounded
16    secondary LITE anchor
17  else
18    retain at most one ranked ordinary
19    consensus context
20 if  $\delta = \text{Code}$  then
21   rank executable/test artifacts before auxiliary
22   context
23   foreach ranked executable artifact  $i$  do
24     if  $i$  repeats a selected artifact from the same
25     immediate sender then
26       if  $i$  is the first failed-feedback carrier
27       then
28         tag  $i$  feedback-LITE
29       else
30          $\text{excl}_i \leftarrow 1$ 
31     else
32       if  $i$  is failed then
33         tag  $i$  feedback-LITE
34       else
35         tag non-failed artifact  $i$  for FULL
36         realization if admitted, including
37         an independent-source artifact;
38   retain at most three receiver-relevant,
39   nonredundant non-code contexts as LITE
40 return tags and route-exclusion indicators

```

lidity is

$$\begin{aligned} \text{Valid}_\delta(v \mid x) = & \mathbb{I}[v \neq \emptyset \wedge v \neq x \\ & \wedge \hat{n}(v) < \hat{n}(x) \\ & \wedge \hat{n}(v) \leq \gamma_\delta \hat{n}(x) \wedge \text{Safe}_\delta(v)]. \end{aligned} \quad (19)$$

where Safe_δ checks the family artifact and output format, $\gamma_{\text{QA}} = \gamma_{\text{Math}} = 0.8$, and $\gamma_{\text{Code}} = 0.9$. The local policy estimate is $\hat{n}(x) = \max(1, \lceil |x|/4 \rceil)$ for non-empty stripped text and zero otherwise. It is used only for projection

Family	Protected content	Default realization and validation
QA	Answer-cluster representative and supporting evidence	Decisive disagreement remains FULL; otherwise LITE copies the visible answer and up to two evidence lines. If tagged extraction fails, a minimal answer–evidence capsule preserves the cluster.
Math	Primary derivation anchor and secondary relation-bearing anchors	The primary anchor remains FULL. Secondary LITE retains the final expression and up to six derivation lines. If no span is extractable, a fixed marker preserves the relation; other invalid shortening falls back to FULL.
Code	Executable/test artifact, interface, and feedback	Failure status takes precedence: failed artifacts, including independent-source ones, become feedback-bearing LITE without the executable body; absent visible feedback uses a fixed failure statement. Admitted non-failed artifacts remain FULL. Up to three auxiliary contexts remain bounded LITE.

Table 6: Default task-family realization of representatives selected by Algorithm 2.

guards; provider-reported input and output tokens remain the evaluation measure.

Secondary Math anchors and failed-feedback Code capsules use dedicated relation-preserving validation rather than the ordinary ratio path, so they remain LITE without restoring a long derivation or failed executable body. All other invalid Math/Code shortening fails closed to FULL. No projector invokes another model.

The reported implementation uses one default policy configuration: main results enable LITE under default strength, with force-FULL intervention and diagnostic ablations disabled. Family selection depends on the QA, Math, or Code output grammar and is independent of benchmark identity, sample identifier, gold answer, hidden score, model identity, named worker, and post-hoc correctness. The implementation fixes parser precedence, complete stable tie ranks, compound-predicate execution, and debug bookkeeping according to the equations and programs above. Separate interfaces handle worker routing and dense judge aggregation, ensuring that every non-empty terminal worker output reaches the judge.

For reproduction, the reported implementation builds the family plan before assigning relation tags, materializes named support and risk reasons before applying the margin gate, and realizes views in the order shown in Equation 18. The same execution sequence is used throughout the reported evaluation.

Default implementation details. *Card parsing.* Lexical signatures lowercase alphanumeric or underscore tokens, discard one-character lexical tokens, and assign zero cosine when either signature is empty. Low content means fewer than 24 stripped characters or at most three lexical types. For Code, visible FEEDBACK/TEST(S) fields pre-

cede an observable feedback span in x_i ; runtime test metadata supplies the field when worker-produced feedback is absent. Missing fields remain inactive.

Shared relation-and-gate defaults. The numerical defaults instantiate conservative routing rules rather than calibrated confidence scores. All settings are fixed globally and shared unchanged across datasets and backbones; no benchmark- or model-specific configuration is introduced.

Relation. Content overlap is treated as the dominant pairwise signal ($\alpha_L = 0.8$), followed by visible answer/artifact agreement ($\alpha_A = 0.6$), while topology provides a smaller correction ($\alpha_T = 0.4$).

Admission. We use $K = 2$ and $\tau = 0$. A message is considered duplicative only when high similarity (≥ 0.8) coincides with low novelty (< 0.6).

Conservative retention. Coverage requires strong overlap ($\text{cov} \geq 0.7$) together with weak residual support ($\text{sup} < 0.3$). Singleton evidence receives a support floor of 0.6 to avoid prematurely suppressing minority information.

Other fixed constants. For exact reproduction, the remaining topology weights are $w_s = 0.3$, $w_{\text{role}} = 0.2$, $w_k = w_{\text{tgt}} = 0.2$, and $w_{\text{div}} = 0.1$. The freshness/staleness cutoffs are 0.4/0.2, with worker-support and receiver-mismatch cutoffs of 0.6 and 0.2, respectively.

B Experimental Details

This appendix reports the protocol behind the main comparisons. Within each dataset–model pair, all MAS variants share the task split, worker roster, judge, prompts, base graph, communication schedule, round budget, decoding setting, and metric. AgentPrune and AgentDropout apply fitted and frozen structural artifacts to this shared workflow,

Base model: Qwen3-8B															
Method	GPQA			AQuA			MATH500			LCB			MBPP+		
	Acc.	In	Out	Acc.	In	Out	Acc.	In	Out	Acc.	In	Out	Acc.	In	Out
Full MAS	43.04	2.99M	0.43M	85.83	3.80M	0.86M	78.40	9.00M	1.84M	32.00	4.94M	0.76M	71.89	3.30M	0.25M
AgentPrune	45.57	2.33M	0.44M	86.61	2.82M	0.87M	79.20	7.22M	2.03M	32.57	4.03M	0.79M	73.96	2.83M	0.27M
AgentDropout	48.73	1.61M	0.32M	86.22	1.53M	0.73M	80.40	3.43M	1.83M	30.29	2.62M	0.64M	72.78	2.02M	0.16M
SafeSieve	46.20	1.31M	0.33M	86.61	1.87M	0.72M	79.60	4.95M	1.48M	31.43	3.26M	0.62M	74.26	2.53M	0.23M
MESSAGE SIEVE	50.63	1.29M	0.66M	87.40	1.93M	0.91M	81.80	3.88M	2.24M	33.14	2.87M	0.79M	78.70	2.62M	0.26M

Base model: Gemini 3.1 Flash-Lite															
Full MAS	72.78	3.76M	0.62M	91.34	3.95M	0.93M	90.60	9.89M	2.20M	54.29	6.20M	1.56M	82.84	3.86M	0.38M
AgentPrune	74.05	2.79M	0.66M	91.34	2.98M	0.95M	91.60	6.89M	2.21M	55.43	5.01M	1.53M	83.43	3.15M	0.36M
AgentDropout	73.42	1.97M	0.56M	91.73	1.99M	0.73M	91.40	4.46M	1.68M	58.29	3.40M	1.26M	82.54	2.14M	0.25M
SafeSieve	74.05	1.48M	0.42M	90.16	1.89M	0.74M	90.80	5.36M	1.71M	57.14	3.65M	1.22M	83.43	2.56M	0.25M
MESSAGE SIEVE	75.32	1.90M	0.75M	91.73	2.03M	0.95M	91.40	3.98M	2.24M	61.71	3.58M	1.52M	84.62	2.88M	0.34M

Table 7: Accuracy and evaluation-token summary for dense FULL MAS, the three pruning baselines, and MESSAGE SIEVE. In and Out are evaluation input and output tokens, respectively, reported in millions.

while the controlled SafeSieve comparison exports and freezes its adapted edge state under a fixed validation/recovery policy. MESSAGE SIEVE preserves the worker roster and base graph and realizes each already-generated incoming message as FULL, LITE, or MUTE before constructing the receiver prompt.

B.1 Model and Dataset Setup

All models use temperature 0.2 and are fixed within each dataset-model block.

Following the 40-example training setting used by AgentPrune (Zhang et al., 2025a), we set a common adaptation budget of 40 examples per benchmark for all three structural baselines. GPQA-Diamond uses a custom 40/158 partition of the complete 198-item Diamond pool. AQuA-RAT uses 40 examples from the official training split and all 254 official test examples. MATH500 uses 40 examples from the original MATH training split and all 500 MATH500 test examples. LiveCodeBench CodeGen Lite uses 40 I/O-stratified test5 examples for adaptation and all 175 test6 examples for cross-release evaluation. MBPP+ uses a custom 40/338 partition of the 378-item EvalPlus MBPP+ pool. In every case, adaptation examples are excluded from the reported evaluation set.

B.2 MAS Runtime and Baseline Protocol

All MAS methods start from the same worker roster, judge, prompts, base graph, graph interface, schedule, and two-round budget within each dataset. Structural baselines may alter the active communication subgraph; for controlled held-

out evaluation, each resulting structural artifact or state is frozen before evaluation. MESSAGE SIEVE retains the base graph and changes only prompt-visible views of already-generated messages. GPQA uses five expert-style workers; AQuA-RAT and MATH500 use four math-oriented workers; LiveCodeBench CodeGen Lite and MBPP+ use five code-oriented workers. We report one run per main-table setting; QA and math tasks use accuracy, while code tasks use pass@1. FULL MAS forwards every runtime message.

Role prompts are task-family-specific. GPQA uses expert roles tailored to scientific QA, informed by role specialization and staged coordination in MetaGPT and ChatDev (Hong et al., 2024; Qian et al., 2024). The math roles are designed for the benchmark tasks. The code roles adapt functional decompositions from MapCoder and AgentCoder (Islam et al., 2024; Huang et al., 2024). Benchmark input/output conventions follow the cited GPQA, AQuA, MATH, LiveCodeBench, MBPP, and EvalPlus sources (Rein et al., 2024; Ling et al., 2017; Hendrycks et al., 2021; Lightman et al., 2024; Jain et al., 2025; Austin et al., 2021; Liu et al., 2023).

AgentPrune and AgentDropout fit their structural pruning artifacts on these disjoint adaptation splits and freeze them for evaluation. Our controlled SafeSieve comparison uses the same examples to adapt its experience-updated edge state. When required, the fixed validation/recovery policy recovers or derives a healthy node-pruned artifact from the stored adaptation history before deterministic export. The held-out runner loads

the resulting fixed method parameters without updating search state, and the artifact type and hash are retained in run provenance. This controlled adaptation/export/freeze protocol provides a frozen counterpart to SafeSieve’s published no-pretraining procedure, which progressively updates edge scores during execution. Both backbones use the same sample IDs, with each baseline adapted separately to each backbone. Adaptation uses gold-answer correctness for GPQA-Diamond, AQUA-RAT, and MATH500 and execution outcomes for LiveCodeBench and MBPP+. **MESSAGESIEVE** does not use the adaptation split and applies its fixed routing policy directly during evaluation.

B.3 Performance–Token Accounting

Table 7 reports evaluation input and output tokens for the methods in Table 1. The tradeoff figure uses total evaluation tokens, computed as input plus output tokens, and reductions are computed relative to dense FULL MAS under the same model and dataset. The figure’s x-axis includes evaluation input and output tokens; structural-baseline fitting and adaptation tokens are logged separately.

B.4 Ablation and Robustness Protocols

B.4.1 Component Ablation Details

The component ablations use the same evaluation instances and Qwen3-8B runtime protocol on GPQA-Diamond and MBPP+ as the main RQ3 analysis. Each variant runs end to end while removing one signal family or action type.

Variant	Changed component	Intended diagnostic
Random	random action assignment	token saving without selection
w/o Relation	local relation vector	redundancy and disagreement cues
w/o Receiver	receiver-conditioned state	role-specific message value
w/o Temporal	temporal freshness cues	previous-round memory handling
No LITE	remove LITE	value of short evidence views

Table 8: Component ablation definitions. Each variant changes one signal family or action type while keeping the same fixed MAS runtime.

RANDOM preserves the approximate FULL/LITE/MUTE action ratios observed in the full policy, but assigns actions without using state-card or relation signals. The relation, receiver, and temporal ablations remove the cues named in Table 8. The **No LITE** ablation disables compact LITE views, so admitted messages remain verbatim. These variants test whether gains come from informed message selection and view realization rather than shorter prompts alone.

B.4.2 Pruning Strength Details

We evaluate the Relaxed and Strict diagnostic configurations end to end on the same full evaluation instances as the default policy, with the same graph, roster, worker and judge prompt templates, Qwen3-8B model, schedule, and decoding setup. Each configuration reruns the workers and judge; neither is a trajectory replay or used to select the default policy. Relaxed removes some redundancy and staleness filters. Strict is a broader routing-strength stress diagnostic: it enables judge-message pruning and, for Code, relaxes executable-code FULL protection. These Strict changes are diagnostic exceptions to the default worker-message routing and dense final-judge readout. The worker gate threshold remains $\tau = 0$.

Preset	Change from default	Effect
Relaxed	remove some redundancy and staleness filters	weaker pruning
Default	use the reported support and risk predicate sets with $\tau = 0$	main policy
Strict	stronger filters plus judge/Code exceptions	stronger pruning

Table 9: End-to-end pruning-strength diagnostic configurations. Strict includes exceptions to the default worker-message routing and dense final-judge readout.

Together, the separately executed Relaxed and Strict configurations quantify sensitivity to weaker and more aggressive routing.

B.4.3 Attack Experiment Details

The degraded-communication stress test evaluates stability when some existing agents produce plausible but low-quality messages. Keeping the topology, receiver schedule, sender identities, public role names, and number of message slots fixed isolates robustness to degraded communication within the same MAS graph.

For each query, one of the five workers is randomly selected for corruption; the final judge is not corrupted. For GPQA-Diamond, the corrupted worker is instructed to behave like a normal domain expert while steering the discussion toward a plausible but wrong answer option. The wrong option is chosen deterministically from the answer choices by moving to the next option after the gold label. For MBPP+, corrupted workers are biased toward compact implementations that satisfy visible examples but under-handle broader hidden-test behavior. In both datasets, corrupted workers are not allowed to reveal that they are corrupted.

The table compares clean and attacked runs for AgentPrune, AgentDropout, SafeSieve, and **MESSAGESIEVE** using absolute and relative perfor-

mance drops. The protocol measures how runtime message filtering moderates degraded worker messages under a fixed MAS graph.

C Further Analysis

This section broadens the evaluation through controlled policy comparisons, statistical reliability analyses, runtime measurements, and trace-level exposure analyses. Together, these analyses examine complementary consequences of the shared routing mechanism.

C.1 Selection and Compression Controls

We compare four worker-message policies in a dedicated Qwen3-8B policy-control suite under the same dense final-judge convention. Adaptive routing denotes the **MESSAGESIEVE** worker-message policy and applies **FULL/LITE/MUTE** per message. Always-LITE retains every non-empty worker message but forces a compact view. Static Edge Drop uses a dataset-specific, content-independent edge mask fixed across queries and keeps retained messages in **FULL**. **LLMLingua-2** (Pan et al., 2024) retains every worker message and applies token-level compression at $r = 0.5$ with the recorded multilingual MeetingBank checkpoint on CPU.

Dataset	Adaptive	Always-LITE	Static Edge	LLMLingua-2
GPQA-Diamond	51.27	44.30	48.10	44.30
MATH500	81.40	81.20	80.20	78.40
MBPP+	77.81	73.96	74.85	72.19

Table 10: Qwen3-8B worker-message policy controls on the complete evaluation sets (accuracy or pass@1), all using dense final-judge readout.

This dedicated policy-control suite is separate from the primary cross-method comparison in Table 1. Across the three datasets, adaptive routing achieves the highest score and outperforms the evaluated static edge-dropping and compression controls; paired uncertainty for the main MessageSieve–Full comparison is analyzed separately in Section C.2.

C.2 Statistical Reliability

Table 11 reports paired bootstrap intervals and McNemar tests for the ten MessageSieve–Full prediction pairs. The bootstrap jointly resamples task items for both methods, while McNemar’s test uses discordant correctness indicators. These analyses quantify paired uncertainty over the finite evaluation sets.

Table 12 summarizes three Qwen endpoint evaluations of Full MAS and adaptive routing under the same decoding configuration, characterizing endpoint-level variability.

C.3 Efficiency and Runtime

Table 7 already separates evaluation input and output tokens. Given dated endpoint prices p_{in} and p_{out} per million tokens, monetary API cost is

$$C = p_{\text{in}} \frac{T_{\text{in}}}{10^6} + p_{\text{out}} \frac{T_{\text{out}}}{10^6}. \quad (20)$$

A deployment can substitute endpoint-specific, dated prices into this expression. We report provider-measured input and output tokens so that monetary cost can be estimated across endpoints, tiers, and pricing dates.

The deterministic policy itself is lightweight. On clean 100-sample Qwen measurements, Python-side pool routing requires 27.38 ms/sample for GPQA and 17.29 ms/sample for MBPP+; total local processing including Card Builder work requires 116.95 and 64.93 ms/sample, respectively. These totals exclude dense final-judge passthrough and remain small relative to hosted task latency.

C.4 Message Exposure Analysis

We measure whether each unique normalized answer source (QA/math) or code-artifact source (code) reaches at least one later relevant worker through a **FULL** or **LITE** view. Table 13 reports source-level availability within completed traces.

A separate post-execution inspection of Qwen3-8B traces identified one GPQA regression in which an already-generated unique target was muted for every later worker. Gold labels and hidden-test outcomes were used only for this after-the-fact categorization and never entered routing.

D Representative Prompt Templates and Runtime Assembly: MBPP+

This section presents the representative MBPP+ prompt templates and their runtime assembly. We use MBPP+ because it exposes the main prompt components used in the experiments: direct single-agent prompting, direct COT, role-specific MAS workers, conditional insertion of incoming messages, and dense final judging. The same MBPP+ worker and judge prompt family is used across **FULL MAS**, structural pruning baselines, and **MESSAGESIEVE**; **MESSAGESIEVE** changes only

Model	Dataset	Gap (pp)	95% paired CI (pp)	MS-only/Full-only	McNemar p
Qwen3-8B	GPQA	+7.59	[+0.63, +15.19]	23/11	0.0576
Qwen3-8B	MBPP+	+6.80	[+3.85, +10.06]	27/4	< 0.0001
Qwen3-8B	LCB	+1.14	[−4.00, +6.29]	11/9	0.8238
Qwen3-8B	AQuA	+1.57	[−0.79, +3.94]	7/3	0.3438
Qwen3-8B	MATH500	+3.40	[+0.80, +6.00]	30/13	0.0137
Gemini 3.1 Flash-Lite	GPQA	+2.53	[−1.90, +6.96]	8/4	0.3877
Gemini 3.1 Flash-Lite	MBPP+	+1.78	[+0.00, +3.55]	8/2	0.1094
Gemini 3.1 Flash-Lite	AQuA	+0.39	[−0.79, +1.57]	2/1	1.0000
Gemini 3.1 Flash-Lite	LCB	+7.43	[+3.43, +12.00]	14/1	0.0010
Gemini 3.1 Flash-Lite	MATH500	+0.80	[−0.40, +2.20]	7/3	0.3438

Table 11: Paired uncertainty for MessageSieve versus Full MAS. Gap is MessageSieve minus Full in percentage points. Paired gaps are computed from exact correctness counts before rounding; they may therefore differ by 0.01 points from differences between displayed two-decimal accuracies. The smallest exact-test value is reported as $p < 0.0001$.

Data	Runs	Full	Adaptive routing	Paired gap
GPQA	3	46.20 $\pm$ 2.19	47.89 $\pm$ 2.92	+1.69 $\pm$ 0.73
MBPP+	3	72.68 $\pm$ 0.62	77.02 $\pm$ 0.45	+4.34 $\pm$ 0.74

Table 12: Repeated-endpoint accuracy/pass@1 (mean $\pm$ sample SD, percentage points) under a shared decoding configuration.

Data	Source	Total	FULL	LITE-only	Exposure
GPQA	unique answer	22	21	0	95.45%
AQuA	unique answer	57	47	4	89.47%
MATH500	unique answer	93	91	0	97.85%
LCB	code artifact	683	149	511	96.63%
MBPP+	code artifact	1,406	606	788	99.15%

Table 13: Post-execution unique-source exposure in completed Qwen MessageSieve traces. Exposure denotes empirical FULL-or-LITE visibility to a later relevant worker.

which incoming worker messages are inserted before a receiver call and whether each inserted message is shown as FULL or LITE. Fixed task instructions, fixed role instructions, and dynamically assembled task/message context are shown separately below.

D.1 Single-Agent Prompt

The single-agent prompt is the direct code-generation baseline. It receives the concrete task instance and returns only the final implementation.

Single-Agent System Prompt

You are a software programmer.
Complete the Python function requested by the problem.
Use the problem statement and public assert tests to identify the expected function name, interface, and return behavior.
Return exactly one Python code block containing the complete function implementation.

D.2 CoT Prompt

CoT uses the same single-agent setting but explicitly instructs the model to reason through the task and visible tests before producing the final implementation.

CoT System Prompt

You are a software programmer.
Before writing the final code, reason through the problem and public assert tests to identify the required function name, argument pattern, return type, and minimal behavior.
Treat the public assert tests as the authoritative contract for the function name, interface, and return behavior. Do not infer extra type support or extra whitespace handling from broad wording alone; only implement what the visible tests and explicit task requirements show.
Prefer the simplest implementation consistent with the task and tests; do not broaden the behavior beyond what the task text and visible tests show.
Return exactly one Python code block containing the complete function implementation.

D.3 MAS Worker Roles

The MBPP+ MAS uses five task-specific worker roles spanning planning, algorithm design, programming, testing, and repair. This decomposition is informed by MapCoder’s retrieval–planning–coding–debugging pipeline and Agent-Coder’s programmer–test-designer–test-executor loop (Islam et al., 2024; Huang et al., 2024). The generic worker template in Section D.4 instantiates each role and collects its role-specific output.

Worker Roles

Project Manager. You are a project manager for a Python function-synthesis task. You are responsible for the overall code structure and for keeping the implementation concise rather than over-engineered. Treat the public assert tests as the authoritative contract for the expected function name and interface. Do not infer extra type support or extra whitespace handling from broad wording alone; only implement what the visible tests and

explicit task requirements show. Specify the minimal function structure needed to complete the task, and do not broaden the behavior beyond the task text and visible tests. Keep your reply concise, preferably within fifty words, and do not list too many points.

Algorithm Designer. You are an algorithm designer for a Python function-synthesis task. Specify the concrete algorithm, helper logic if needed, and key cases implied by the problem and public assert tests. When useful, provide compact pseudocode for the main logic. Do not infer extra type support or extra whitespace handling from broad wording alone; only implement what the visible tests and explicit task requirements show. Prefer the simplest algorithm consistent with the task and tests; avoid speculative input types, hidden edge cases, or behavior expansions not shown by the task and visible tests. Keep your reply concise, preferably within fifty words, and do not list too many points.

Programming Expert. You are a programming expert. You will be given a Python function task and public assert tests. You may receive outputs from other agents; they can be helpful but may be incomplete or wrong. Write a full Python function implementation preserving the expected function name and interface from the public tests. Do not infer extra type support or extra whitespace handling from broad wording alone; only implement what the visible tests and explicit task requirements show. Prefer the simplest implementation consistent with the task and tests; do not broaden the behavior beyond what the task text and visible tests show. Use a Python code block and do not output anything except Python code blocks. Do not switch to stdin/stdout.

Test Analyst. You are a test analyst for a Python function-synthesis task. Use only the task wording and the visible public assert tests to identify interface requirements, boundary conditions, and failure modes that are directly evidenced. Do not invent broader semantics, hidden edge cases, or extra input types that are not supported by the task and visible tests. If a candidate already matches the visible contract, do not propose a wider behavior just to be safe or to anticipate hidden tests. Keep your reply concise, preferably within fifty words, and do not list too many points.

Bug Fixer. You are a bug fixer. You will be given a Python function task, public assert tests, and possibly other agents' outputs. Identify defects in the current design, algorithm, implementation idea, or test reasoning, then provide the smallest evidence-preserving fix. When enough information is available, write a full Python function implementation preserving the expected function name and interface. If the current implementation already passes the visible tests, do not "strengthen" it by adding broader behavior, extra normalization, or extra type support. Prefer the simplest fix consistent with the task and tests; avoid speculative generalization, extra normalization, or hidden-case handling not shown by the task text and visible tests. Use a Python code block for code and do not switch to stdin/stdout.

D.4 Generic Worker Prompt Template

Each worker call uses the same dynamically assembled template. Dense FULL MAS inserts all available peer messages into the two conditional message sections. **MESSAGESIEVE** leaves the task and role text unchanged, but changes which

peer messages appear in those sections and whether each selected message is rendered as FULL or LITE. Each section may contain zero, one, or multiple messages.

Generic Worker Prompt Template

The Python function task is:

{task_input}

Your role is: {role_name}.

You are part of a multi-agent code-generation team using the Project Manager, Algorithm Designer, Programming Expert, Test Analyst, and Bug Fixer role split. Write naturally from your role's perspective. Do not follow a required schema.

Guidelines:

{role_output_instruction}

- be concrete and implementation-oriented
- do not claim an algorithm unless you can justify it from the prompt
- use the public assert tests to infer the expected function name and interface
- do not switch to stdin/stdout
- prefer implementation-relevant details over generic explanation
- prefer the simplest behavior consistent with the task and public tests
- do not invent broader input types, conversions, nested structures, or regex semantics unless explicitly required by the task

Same-round predecessor messages:

{same_round_predecessor_messages}

Previous-round messages:

{previous_round_messages}

The placeholder {role_output_instruction} is replaced by role-specific output guidance. For Programming Expert and Bug Fixer, the instruction asks the worker to include one full Python function in a code block when the implementation is clear, while preserving the function name and interface from the public tests. For Project Manager, Algorithm Designer, and Test Analyst, the instruction asks for a concise role-specific note for the final judge, preferably within fifty words, and discourages a full implementation unless it is necessary.

Each peer message in either conditional message section is rendered as:

Peer Message Rendering

Agent {source} as a {sender_role}:

{optional_public_test_status}

{message_content}

D.5 Dense Final Judge Prompt

The final judge is outside worker-message pruning. Its fixed system instruction asks it to solve the original task first and treat worker outputs as evidence;

its runtime prompt then receives every non-empty terminal worker output as FULL in stable order.

Final Judge System and Runtime Template

System instruction. You are the top decision-maker and final Python code writer for a function-synthesis task. Solve the original task yourself first, then use worker outputs as design, algorithm, implementation, testing, and bug-fixing evidence. Treat the public assert tests as the authoritative contract. Do not infer extra type support or extra whitespace handling from broad wording alone; only implement what the visible tests and explicit task requirements show. Reject code that changes the expected function name or interface, uses stdin/stdout, returns the wrong type, or broadens behavior beyond the task text and visible tests without explicit support. If several candidates pass the visible tests, prefer the one with the narrowest behavior and the fewest assumptions. Choose or repair the most reliable implementation and return exactly one Python code block containing the final function.

Runtime user prompt.

The Python function task is:

```
{task_input}
```

Worker outputs from the team are as follows:

```
{terminal_worker_outputs}
```

Write the final accepted Python solution now. First reconstruct the intended algorithm from the original task and public assert tests, then use the agents as evidence. Prefer explicitly public-test-passing implementations, but repair any violation of the task. Preserve the expected function name, interface, and return type; do not switch to stdin/stdout or broaden unsupported behavior. Return exactly one Python code block containing the full final function implementation.

E Case Studies

We select four recorded Qwen3-8B controlled trajectories from the mechanism analysis. The examples cover two positive routing cases, one LITE projection case, and one failure boundary. We report receiver-local routing actions and condense each trajectory to its decision-relevant messages. Because each routing decision shapes the context for subsequent generation, the cases characterize observed routing pathways. The corresponding task IDs are MBPP+ 572 and GPQA-Diamond reczkBiPPNrNN49Hp, recyPffUDqC2k7ZZO, and recZt4x514UQNsgy.

E.1 Case I: Omitting Misleading Planning Messages Before Repair

Figure 4 illustrates receiver-local routing before code repair. In the logged **MESSAGESIEVE** trajectory, two early planning messages advocate an “exactly twice” interpretation; neither is included in the Bug Fixer’s assembled prompt for that call. A later public-test-supported artifact implements the “exactly once” behavior and remains available

to the terminal decision. The trace demonstrates selective exposure of planning notes and executable feedback.

E.2 Case II: Preserving Minority Scientific Evidence

Figure 5 isolates a GPQA routing decision made before final judging. The logged **MESSAGESIEVE** trajectory keeps an answer-bearing denominator calculation visible to the Scientific Synthesizer, whereas the matched random control loses detailed D-supporting context before synthesis and produces a coherent but invalid C calculation. The retained message is numeric, nonredundant disagreement evidence relevant to that receiver; it can shape the Synthesizer’s terminal output, which then reaches the final judge through dense aggregation.

E.3 Case III: Projecting One Long Message into a LITE View

This case isolates LITE projection for an admitted message. As summarized in Figure 6, all methods reach the correct final answer, while **MESSAGESIEVE** presents the useful worker evidence in a shorter extractive view that retains the uncertainty formula, relevant operator and state context, and final option. The View Projector preserves the downstream evidence cue while removing repeated derivation text and an intermediate false start.

E.4 Case IV: Routing Cannot Recover Absent Evidence

Figure 7 establishes the generated-evidence boundary of runtime routing. Across both worker rounds, all ten responses select option D, yielding a coherent Diels–Alder path that appears plausible under observable support signals. The decisive option-A argument appears in the FULL MAS outcome but is absent from the **MESSAGESIEVE** worker trajectory. Routing prioritizes available evidence, so successful correction depends on workers generating a decisive argument.

Case I: Omitting Misleading Planning Messages Before Repair

Dataset and Task Type

MBPP+; Python function synthesis with public assert tests.

Problem

Write a python function to remove duplicate numbers from a given number of lists.

Public tests:

```
assert two_unique_nums([1,2,3,2,3,4,5]) == [1, 4, 5]
assert two_unique_nums([1,2,3,2,4,5]) == [1, 3, 4, 5]
assert two_unique_nums([1,2,3,4,5]) == [1, 2, 3, 4, 5]
```

Method Outcomes

FULL MAS: incorrect; converges to `freq[num] == 2`.

Random: incorrect; keeps `count[num] == 2` code evidence.

MESSAGESIEVE: correct; preserves the repair path using `count[num] == 1`.

Message Handling Evidence

MUTE Observed routing action. For the Bug Fixer’s first receiver-local pool, the Project Manager and Algorithm Designer messages are logged as MUTE. Representative content states: “*The function should return elements that appear exactly twice in the input list.*” The router assigns MUTE to both messages using observable receiver-local signals; the public tests provide the semantic criterion for the task.

FULL Downstream repair artifact. The Bug Fixer later compares the tests and writes: “*the numbers 2 and 3 appear twice, but the output is [1, 4, 5] ... the function is actually returning numbers that appear exactly once.*” Its public-test-supported implementation is retained in the terminal worker output and reaches the dense judge as FULL.

Figure 4: Receiver-local routing omits two “exactly twice” planning messages from the Bug Fixer’s assembled prompt, while a later public-test-supported repair reaches the dense judge.

Case II: Preserving denominator evidence before synthesis

Dataset and Task Type

GPQA-Diamond; graduate-level science multiple-choice QA.

Problem

We have a solution containing Co(II) ions and thiocyanate, where the total cobalt concentration is $c(\text{Co}) = 10^{-2}$ M and the $[\text{SCN}^-]$ concentration is 0.1 M. The stability constants of the cobalt(II) thiocyanato complexes are $\beta_1 = 9$, $\beta_2 = 40$, $\beta_3 = 63$ and $\beta_4 = 16$. What will be the percentage of the blue dithiocyanato cobalt(II) complexes among all cobalt containing species in the solution?

Options: A 38.1%; B 42.3%; C 25.6%; D 16.9%.

Method Outcomes

FULL MAS: correct; uses the distribution denominator.

Random: incorrect; follows a detailed but invalid C calculation.

MESSAGESIEVE: correct; preserves the D-supporting denominator evidence.

Message Handling Evidence

FULL Observed routing action. MESSAGESIEVE admits the Critic-to-Synthesizer message because its full denominator and answer D provide receiver-relevant numeric disagreement evidence. It remains visible to the Scientific Synthesizer, whose terminal output then reaches the dense final judge.

Downstream effect. With the denominator available, the Synthesizer constructs the D-supporting calculation. In the matched random control, the Synthesizer treats total cobalt as free cobalt and obtains the incorrect C result.

Figure 5: Denominator evidence remains visible to the Scientific Synthesizer before dense terminal judging.

Case III: LITE preserves support without the full trace

Dataset and Task Type

GPQA-Diamond; graduate-level science multiple-choice QA.

Problem

Given spin-1-like matrices P_x , P_y , and P_z , the system is found in the P_x eigenstate with column-vector elements $(-1/2, 1/\sqrt{2}, -1/2)$. What is the uncertainty ΔP_z ?

Options: A $\sqrt{2}\hbar$; B $\hbar$; C $\hbar/2$; D $\hbar/\sqrt{2}$.

Method Outcomes

FULL MAS: right.

Random: right.

MESSAGESIEVE: correct; all nine admitted worker-to-worker views in this trace are LITE.

Message Handling Evidence

LITE Retained spans. One Domain Expert message is reduced from 1716 characters to 411 characters. The LITE view keeps $\Delta P_z = \sqrt{\langle P_z^2 \rangle - \langle P_z \rangle^2}$, the operator/state context, and Final Answer: D, while dropping repeated derivation and an intermediate self-correction.

Omitted spans. The compact view supplies the formula, operator context, and final option to the next receiver; the omitted span comprises expectation calculations, repeated matrix text, and the false start $P_z^2 = P_z$.

Figure 6: A 1716-character response becomes a 411-character extractive LITE view retaining formula, state, and answer.

Case IV: Failure when correct evidence is absent

Dataset and Task Type

GPQA-Diamond; graduate-level science multiple-choice QA.

Problem

Identify the starting material(s) required to synthesize methyl 2-propyl-1,2,4a,5,6,7,8,8a-octahydronaphthalene-1-carboxylate.

Options: A methyl (2E,8E,10E)-tetradeca-2,8,10-trienoate; B 1-vinylcyclohex-1-ene and methyl hex-2-ynoate; C Cyclohexene and methyl 2,3-dimethylenehexanoate; D methyl (2E,4E,10Z)-tetradeca-2,4,10-trienoate.

Method Outcomes

FULL MAS: correct; follows the option-A hydrogenation route.

Random: incorrect; answers B.

MESSAGESIEVE: incorrect; answers D.

Message Handling Evidence

FULL Retained but misleading path. MESSAGESIEVE preserves messages that describe option D as a plausible conjugated-triene or Diels–Alder route. They contain mechanism terms, stereochemistry, and mutually reinforcing D conclusions, so the observable routing signals treat them as useful evidence.

Failure boundary. Every worker in the MESSAGESIEVE trajectory selects D, so the dense judge receives several terminal D rationales and no worker-generated A rationale. The outcome illustrates the generated-evidence boundary described above.

Figure 7: All generated worker evidence in the MESSAGESIEVE trajectory supports a plausible but wrong Diels–Alder route; routing cannot recover an absent option-A argument.