HYPERMAS: EVOLVING COLLABORATIVE SKILLS FOR SCALABLE MULTI-AGENT LLM SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

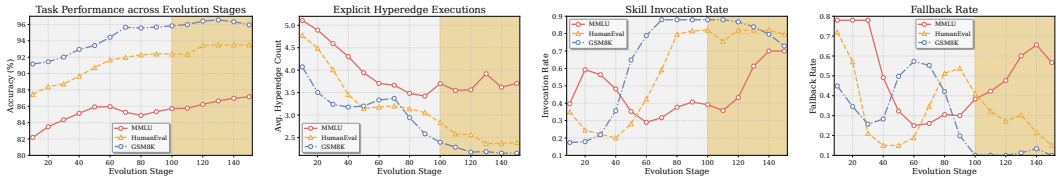
Multi-agent systems (MAS) powered by Large Language Models (LLMs) have demonstrated strong collaborative reasoning capabilities across complex tasks, yet their effectiveness often comes with substantial communication cost. Existing adaptive MAS methods can customize roles, models, and communication structures for each query, but they still rely on an implicit execution assumption: collaboration capability must be unfolded through explicit agent communication whenever a task is solved. As a result, even repeatedly validated coordination processes are re-executed from scratch, causing redundant token consumption and limiting long-term scalability. To bridge this gap, we propose **HyperMAS**, a hypergraph-centered framework that evolves collaborative skills for scalable multi-agent LLM systems. **HyperMAS** represents functional roles as role-level mixture nodes enhanced by heterogeneous LLMs and organizes multi-role coordination through high-order hyperedges. During training, **HyperMAS** explores explicit hypergraph collaboration, identifies reliable and cost-effective sub-hypergraphs from execution traces, and distills them into evolved collaborative skills. During inference, **HyperMAS** adaptively invokes compact skills for familiar routines and expands into explicit hypergraph communication for uncertain cases, balancing efficiency and flexibility. Extensive experiments on mathematical reasoning, code generation, and knowledge-intensive benchmarks show that **HyperMAS** is **(1) effective**, achieving an absolute 2.11% improvement in average accuracy over the strongest adaptive MAS baseline, **(2) efficient**, reducing token cost by 28.4%, and **(3) scalable**, reducing explicit hyperedge executions from 4.86 to 2.28 after skill accumulation.

1 INTRODUCTION

Recent advances in large language model based agents have revealed the strong potential of collaborative intelligence in artificial systems. By assigning agents specialized roles and enabling them to exchange intermediate reasoning results, multi-agent systems have achieved promising performance across code generation, mathematical reasoning, knowledge-intensive question answering, and complex decision making (Du et al., 2024; Liang et al., 2024; Talebirad & Nadiri, 2023; Chen et al., 2024). Beyond manually designed collaboration pipelines, recent studies have started to optimize multi-agent systems along several key dimensions. Communication topology learning determines the information flow among agents (Zhuge et al., 2024; Zhang et al., 2025b;a). Role assignment decides how functional responsibilities are distributed (Qian et al., 2025; Hong et al., 2024; Liu et al., 2024). LLM routing selects suitable backbone models for different roles to balance capability and cost (Ong et al., 2024; Yue et al., 2025). More recent designs further enhance each functional role through internal mixtures of heterogeneous models (Wang et al., 2024; Li et al., 2025a;c; Yao et al., 2026). These efforts collectively suggest that effective multi-agent reasoning depends on how the system organizes role capability, communication, and cost for each task.

This progress naturally moves MAS construction from fixed workflows toward query-conditioned organization (Yue et al., 2025; Zhang et al., 2024; Yao et al., 2026). Given an input query, an adaptive system can select the necessary functional roles, assign suitable models to each role, and arrange inter-role communication under a cost budget. This instance-level construction matters because different tasks demand different collaboration scales. A simple factual question may only need a compact organization with lightweight models. A hard reasoning task may require stronger role capability, broader verification, and richer information exchange among agents. A code generation task may

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



(a) Task Performance (b) Explicit Hyperedges (c) Skill Invocation (d) Fallback Rate

Figure 1: Model analysis of `HyperMAS` across evolution stages. The shaded region indicates the skill-augmented phase. **Left:** Task performance improves overall with task-specific fluctuations. **Middle-left:** Explicit hyperedge executions decline as evolved skills become available. **Middle-right:** Skill invocation increases with accumulated experience. **Right:** Fallback rate decreases but remains non-zero, preserving explicit collaboration for uncertain cases. Together, these trends show that `HyperMAS` reduces repeated structural unfolding without removing the flexibility of full hypergraph communication.

benefit from structured coordination among implementation, testing, and review stages (Hong et al., 2024; Qian et al., 2024). Existing adaptive methods have made meaningful progress in this direction. They learn to configure a suitable collaboration structure for each incoming query and execute multi-agent communication to produce the final answer.

However, these methods still rely on an implicit execution assumption. They assume that collaboration capability must be unfolded through explicit agent communication whenever a task is solved. A controller may learn to select better roles, models, or topologies over time, but the selected collaboration still executes as a fresh interaction process for each query (Yue et al., 2025; Yao et al., 2026; Zhang et al., 2025b;a). This is similar to repeatedly interpreting the same program segment. The system may know the right process to run, yet it still pays the full step-by-step execution cost every time. Once a coordination process has been repeatedly validated, such as derivation followed by computation and verification in mathematical reasoning, or implementation followed by testing and review in code generation, unfolding the same agent communication again brings limited new structure but still incurs the full token cost. The system becomes better at choosing a process, but the process itself does not become a lighter execution capability.

We therefore argue that scalable MAS should specialize repeatedly validated collaboration processes into collaborative skills. A collaborative skill refers to a callable coordination procedure that specifies the required roles, their interaction pattern, and a verification signal for the produced result (Wang et al., 2023a; Xia et al., 2026a). This view complements task-adaptive construction and changes what can be constructed. For familiar processes, the system invokes a compact skill rather than repeatedly unfolding the same full communication process. For uncertain cases, the system still expands into explicit multi-agent collaboration. In this way, accumulated experience reduces redundant structural execution rather than only improving a controller over the same low-level action space.

The key question is how to represent such collaborative skills. A skill in MAS is not fully described by a single model, a single role, or a pairwise communication link. Its effectiveness depends on both role capability and group coordination. A role supported by multiple heterogeneous LLMs can produce diverse proposals and stronger synthesis (Wang et al., 2024; Li et al., 2025a;c; Yao et al., 2026). A group of roles can then coordinate around a shared subgoal, such as checking an intermediate derivation or reviewing a candidate program. Hypergraphs provide a natural structural schema for this two-level dependency. A node represents a functional role enhanced by internal model mixtures, and a hyperedge represents high-order coordination among multiple role-level nodes (Zhang et al., 2025c). Thus, a collaborative skill can be grounded in a sub-hypergraph that preserves both how roles are equipped and how they coordinate.

In this work, we propose `HyperMAS`, a hypergraph-centered framework for evolving collaborative skills in scalable multi-agent LLM systems. `HyperMAS` first performs explicit hypergraph collaboration with role-level mixture nodes and high-order collaboration hyperedges. During training, `HyperMAS` explores diverse hypergraph configurations and records execution traces. It identifies sub-hypergraphs that recur frequently, produce correct outputs reliably, and operate at low communication cost. These qualified sub-hypergraphs are distilled into evolved collaborative skills and added to the available structural set. During inference, the generator builds an active hypergraph by

selecting both explicit hyperedges and evolved skills according to task demand, skill reliability, and communication budget. Familiar routines are executed through compact skills with reduced agent interaction. Unfamiliar routines still trigger full hypergraph collaboration, and their successful traces feed back into the skill evolution process. Figure 2 illustrates this evolution process.

Our contributions are summarized as follows:

- ❶ **New Perspective:** We formulate scalable multi-agent LLM collaboration as an evolving skill construction problem grounded in hypergraph structures. Instead of treating each collaboration structure as a disposable execution plan, `HyperMAS` distills successful sub-hypergraphs into reusable collaborative skills that make the system progressively more efficient.
- ❷ **Practical Framework:** We propose `HyperMAS`, a hypergraph-centered MAS framework integrating role-level mixture nodes, high-order collaboration hyperedges, and evolved collaborative skills. This design preserves both intra-role model collaboration and inter-role group coordination in a unified structural form.
- ❸ **Optimization Method:** We design a cost-aware evolution procedure that learns when to invoke an evolved skill and when to expand explicit hypergraph communication. The generator jointly optimizes task performance, skill reliability, and active communication cost within a single training pipeline.
- ❹ **Experimental Validation:** Experiments on mathematical reasoning, code generation, and knowledge-intensive benchmarks show that `HyperMAS` achieves an absolute 2.11% improvement in average accuracy over the strongest adaptive multi-agent baseline, reduces token cost by 28.4%, and reduces explicit hyperedge executions from 4.86 to 2.28 after skill accumulation.

2 Related Work

LLM-based Multi-Agent Systems. LLM-based multi-agent systems combine role specialization and communication in fixed or adaptive organizations (Li et al., 2023; Hong et al., 2024; Qian et al., 2024; Chen et al., 2024). Fixed systems use manually specified debate, software-engineering, or chain-style protocols (Chan et al., 2023; Liang et al., 2024; Hong et al., 2024; Qian et al., 2024), whereas adaptive systems learn communication topologies, role assignments, workflows, or model-routing policies for each task (Zhuge et al., 2024; Zhang et al., 2025b;a; Liu et al., 2024; Zhang et al., 2024; Shang et al., 2024; Ong et al., 2024; Yue et al., 2025). Recent designs further strengthen roles through heterogeneous model mixtures and coordinate groups through hypergraphs (Wang et al., 2024; Li et al., 2025a;c; Yao et al., 2026; Zhang et al., 2025c). These methods improve task-adaptive construction but generally execute each selected structure anew. `HyperMAS` instead distills repeatedly validated sub-hypergraphs into reusable collaborative skills that reduce repeated explicit communication across related tasks.

Skill Learning for LLM Agents. Skill learning stores successful behavior as reusable programs, instructions, or policies (Wang et al., 2023a; Anthropic, 2025; Xu & Yan, 2026). SkillRL evolves a skill bank during reinforcement learning, whereas ASI induces and verifies program-based skills for web agents (Xia et al., 2026a; Wang et al., 2025b). Related systems study in-context reinforcement, automatic skill bases, self-evolution, GUI skills, and graph-structured composition (Lu et al., 2026; Wang et al., 2026; Zhang et al., 2026; Chen et al., 2026; Xia et al., 2026b). These methods primarily encode what one agent does. Collaborative skills must also preserve role instantiation and information exchange, which `HyperMAS` represents with role-level mixture nodes and inter-role hyperedges.

3 Preliminaries

3.1 Notation and Search Space

Let $\mathcal{D} = \{(q, a^*)\}$ denote the training set, where q is an input query and a^* is the reference answer. Let $\mathbb{M}$ denote the pool of N_m available LLM backbones together with a special `skip` token. Let $\mathbb{R}$ denote the set of N_r candidate functional roles. Let f_ψ denote a frozen sentence encoder used to embed queries, role descriptions, model profiles, and generated outputs. A role-level mixture node v_i represents one functional role equipped with multiple heterogeneous LLMs:

$$v_i = (r_i, \mathbf{m}_i, m_i^{\mathrm{s}}), \quad \mathbf{m}_i = (m_{i,1}, \ldots, m_{i,W}) \tag{1}$$

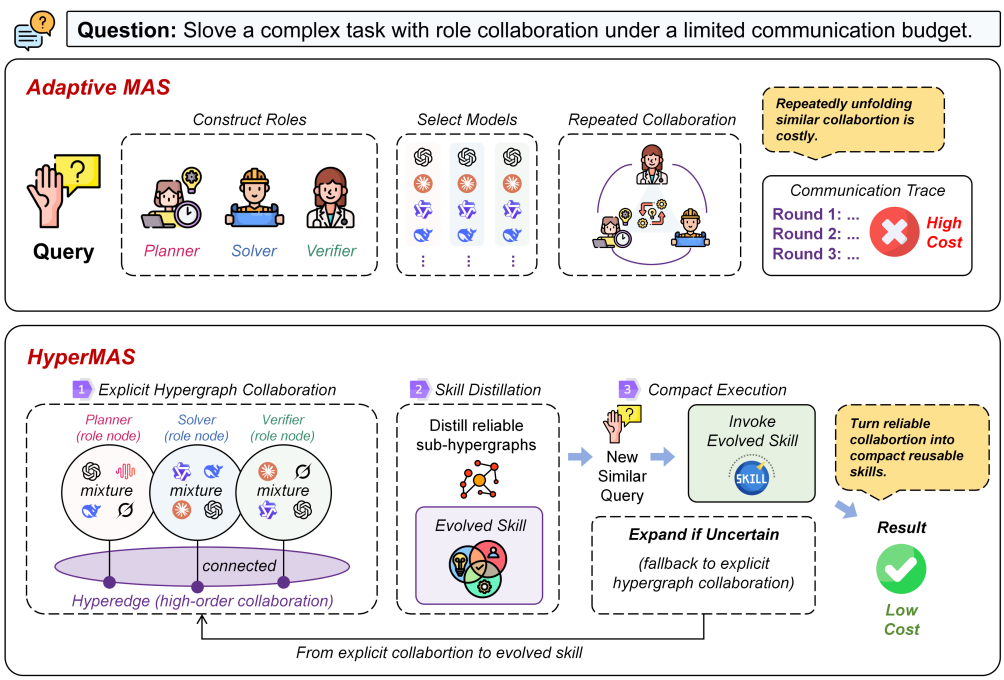


Figure 2: Framework of **HyperMAS**.

Here $r_i \in \mathbb{R}$ is the assigned role, $\mathbf{m}_i$ is the ordered set of proposer LLMs, $m_{i,w} \in \mathbb{M}$ is the proposer at position w, W is the maximum number of proposer positions, and $m_i^{\mathrm{s}} \in \mathbb{M}$ is the synthesizer LLM. Selecting `skip` for a proposer omits that proposer. Selecting `skip` for the synthesizer deactivates the entire node. Given query q, an explicit hypergraph MAS is denoted as $\mathcal{H}_q = (\mathcal{V}_q, \mathcal{E}_q)$, where $\mathcal{V}_q = \{v_i\}_{i=1}^{N_q}$ is the set of active role-level mixture nodes, N_q is the number of active nodes, and $\mathcal{E}_q \subseteq 2^{\mathcal{V}_q}$ is the set of hyperedges. Each hyperedge $e \in \mathcal{E}_q$ connects a subset of role nodes that coordinate around the same intermediate subgoal. Executing $\mathcal{H}_q$ produces answer $\hat{a}$ and incurs token cost $C(\mathcal{H}_q, q)$. A collaborative skill σ is a callable coordination procedure distilled from a repeatedly validated sub-hypergraph:

$$\sigma = (\mathbf{h}_\sigma, \mathcal{B}_\sigma, \mathbf{t}_\sigma, m_\sigma^{\mathrm{exec}}, \bar{U}_\sigma, \bar{C}_\sigma, n_\sigma) \tag{2}$$

Here $\mathbf{h}_\sigma \in \mathbb{R}^D$ is the skill embedding used for activation matching, $\mathcal{B}_\sigma = (\mathcal{V}_\sigma, \mathcal{E}_\sigma)$ is the source sub-hypergraph schema, $\mathbf{t}_\sigma$ is the distilled execution template, $m_\sigma^{\mathrm{exec}} \in \mathbb{M}$ is the executor LLM for compact invocation, $\bar{U}_\sigma$ is empirical utility, $\bar{C}_\sigma$ is empirical invocation cost, and n_σ is the number of reliable occurrences. The available skill set is denoted as $\mathcal{K}$.

3.2 Optimization Objective

HyperMAS learns a construction policy π_θ that builds an active execution structure $\mathcal{A}_q$ for query q. The structure may contain explicit hyperedges, invoked skills, and fallback expansions. The objective is:

$$\max_\theta \mathbb{E}_{(q,a^*)\sim\mathcal{D},\, \mathcal{A}_q\sim\pi_\theta(\cdot|q,\mathcal{K})}\left[R(q, \mathcal{A}_q)\right] \tag{3}$$

Here θ denotes the trainable parameters of the construction policy, $\mathcal{K}$ is the evolving skill set, and $R(q, \mathcal{A}_q)$ balances correctness, token cost, and explicit communication. The skill set is updated through non-parametric distillation from execution traces.

4 Method

4.1 Role-Level Node Construction

HyperMAS instantiates one candidate node for each predefined role in $\mathbb{R}$ and uses the `skip` token of the synthesizer position to prune unnecessary roles. For each role r_i, we encode the query and role

description as:

$$\mathbf{h}_{q,r_i} = f_\psi([q, \mathrm{desc}(r_i)]) \tag{4}$$

Here $\mathbf{h}_{q,r_i}$ is the query-role embedding and $\mathrm{desc}(r_i)$ is the textual description of role r_i. For each LLM candidate $m_\ell \in \mathbb{M}$, we precompute $\mathbf{h}_{m_\ell} = f_\psi(\mathrm{profile}(m_\ell))$, where $\mathrm{profile}(m_\ell)$ describes model capability and cost. For each proposer or synthesizer position p in node v_i, the model selection score is:

$$s_\ell^{(i,p)} = \mathrm{MLP}_{\mathrm{sel}}(\mathbf{h}_{q,r_i} \oplus \mathbf{h}_{m_\ell} \oplus \mathbf{e}_p) \tag{5}$$

Here $s_\ell^{(i,p)}$ is the score of assigning LLM m_ℓ to position p, $\mathrm{MLP}_{\mathrm{sel}}$ is a shared selection network, $\mathbf{e}_p$ is a learnable position embedding, and $\oplus$ denotes concatenation. The selection probability is:

$$\pi_m(m_{i,p} = m_\ell \mid q, r_i) = \frac{\exp(s_\ell^{(i,p)}/\tau)}{\sum_{m' \in \mathbb{M}} \exp(s_{m'}^{(i,p)}/\tau)} \tag{6}$$

Here $\pi_m(m_{i,p} = m_\ell \mid q, r_i)$ is the probability of assigning model m_ℓ to position p, and τ is the temperature. Each active proposer generates a role-specific proposal. The synthesizer aggregates all active proposals into node output o_i. If the synthesizer selects `skip`, node v_i is removed from $\mathcal{V}_q$.

4.2 Hyperedge Construction

After active nodes are determined, **HyperMAS** constructs hyperedges to organize multi-role coordination. Each active node is encoded as:

$$\mathbf{z}_i = \mathrm{MLP}_{\mathrm{node}}(\mathbf{h}_{q,r_i} \oplus \mathrm{Mean}_{p \in \mathcal{P}_i} \mathbf{h}_{m_{i,p}}) \tag{7}$$

Here $\mathbf{z}_i$ is the representation of node v_i, $\mathcal{P}_i$ is the set of active LLM positions inside v_i, and Mean averages the embeddings of selected active LLMs. The affinity between two active nodes is:

$$a_{ij} = \mathrm{MLP}_{\mathrm{aff}}(\mathbf{z}_i \oplus \mathbf{z}_j \oplus f_\psi(q)) \tag{8}$$

Here a_{ij} is the collaboration affinity between nodes v_i and v_j, and $\mathrm{MLP}_{\mathrm{aff}}$ outputs a scalar score. For each anchor node v_i, **HyperMAS** forms one candidate hyperedge by grouping it with its top k related nodes:

$$e_i = \{v_i\} \cup \mathrm{TopK}(\{v_j \mid j \neq i\}, a_{ij}, k) \tag{9}$$

Here e_i is the hyperedge anchored at v_i, TopK returns the k nodes with the highest affinity scores, and k controls the maximum number of partner nodes. Duplicate hyperedges are merged and hyperedges with fewer than two nodes are removed. The affinity scores only form candidate groups. Execution is performed at the hyperedge level, where all member nodes coordinate as one group around a shared subgoal. For each hyperedge e, a designated aggregator node synthesizes the outputs of all member nodes:

$$o_e = m_e^{\mathrm{agg}}(\mathrm{Prompt}_e(q, \{o_i \mid v_i \in e\})) \tag{10}$$

Here o_e is the output of hyperedge e, m_e^{agg} is the synthesizer model of the member node with the highest average affinity to other members, and Prompt_e formats the query and member outputs for group-level coordination.

4.3 Skill Distillation from Execution Traces

During training, **HyperMAS** stores execution traces in a buffer $\mathcal{T}$. Each trace records query q, active nodes $\mathcal{V}_q$, executed hyperedges $\mathcal{E}_q$, node outputs, hyperedge outputs, final answer $\hat{a}$, correctness signal u, and token cost $C(\mathcal{H}_q, q)$. From successful traces with $u = 1$, we extract connected sub-hypergraphs with at most $S_{\max}$ nodes. Each candidate $\mathcal{B}$ is assigned a canonical role hyperedge signature:

$$\kappa(\mathcal{B}) = \mathrm{Canon}(\{r_i \mid v_i \in \mathcal{V}_\mathcal{B}\}, \mathcal{E}_\mathcal{B}) \tag{11}$$

Here $\kappa(\mathcal{B})$ is the canonical signature, $\mathcal{V}_\mathcal{B}$ is the node set, $\mathcal{E}_\mathcal{B}$ is the hyperedge set, and Canon normalizes node order by role labels and hyperedge membership. Candidates with the same signature are grouped into the same pattern entry. For each entry, **HyperMAS** records occurrence count $n_\mathcal{B}$, average utility $\bar{U}(\mathcal{B})$, and average explicit cost $\bar{C}(\mathcal{B})$. The distillation score is:

$$s(\mathcal{B}) = \bar{U}(\mathcal{B}) - \lambda_d \bar{C}(\mathcal{B}) + \mu_d \log(1 + n_\mathcal{B}) \tag{12}$$

Here $s(\mathcal{B})$ is the score of candidate pattern $\mathcal{B}$, λ_d controls the penalty on explicit cost, and μ_d controls the preference for recurring patterns. A candidate is distilled when $s(\mathcal{B}) > \tau_d$, where τ_d is the distillation threshold.

The compact executor is implemented as a single LLM call with a distilled coordination template. The template summarizes the role decomposition, interaction order, verification criteria, and output format shared by successful traces:

$$\mathbf{t}_\sigma = \mathrm{Extract}(\{(q_j, \{o_i^{(j)}\}, \hat{a}_j) \mid \kappa(\mathcal{B}_j) = \kappa(\mathcal{B}), u_j = 1\}) \tag{13}$$

Here $\mathbf{t}_\sigma$ is the template of skill σ, $\mathrm{Extract}$ is an instruction LLM that summarizes common coordination behavior, q_j is a training query, $\{o_i^{(j)}\}$ are intermediate role outputs, and $\hat{a}_j$ is the final answer of trace j. The skill embedding is:

$$\mathbf{h}_\sigma = \frac{1}{n_{\mathcal{B}}} \sum_{j:\kappa(\mathcal{B}_j)=\kappa(\mathcal{B}), u_j=1} f_\psi(q_j) \tag{14}$$

Here $\mathbf{h}_\sigma$ represents the typical queries for which skill σ has been reliable. A new skill is added to $\mathcal{K}$ after the template, embedding, utility, and cost statistics are computed.

4.4 Skill-Guided Active Hypergraph Construction

At inference time, **HyperMAS** constructs an active structure by selecting evolved skills and explicit hyperedges. For each skill $\sigma \in \mathcal{K}$, activation is computed by embedding similarity:

$$\alpha_\sigma(q) = \cos(f_\psi(q), \mathbf{h}_\sigma) \tag{15}$$

Here $\alpha_\sigma(q)$ is the suitability score of skill σ for query q. Skills with $\alpha_\sigma(q) > \tau_a$ are invoked, and at most J skills are selected. The compact execution is:

$$\hat{a}_\sigma = m_\sigma^{\mathrm{exec}}(\mathbf{t}_\sigma, q) \tag{16}$$

Here $\hat{a}_\sigma$ is the skill output and m_σ^{exec} is the executor LLM. Output reliability is estimated by a lightweight verifier:

$$b_\sigma = \mathrm{sigmoid}(\mathrm{MLP}_{\mathrm{ver}}(f_\psi(q) \oplus f_\psi(\hat{a}_\sigma) \oplus \mathbf{h}_\sigma \oplus \bar{U}_\sigma \oplus \bar{C}_\sigma)) \tag{17}$$

Here b_σ is the confidence score, $\mathrm{MLP}_{\mathrm{ver}}$ is a binary classifier, $\bar{U}_\sigma$ is historical utility, and $\bar{C}_\sigma$ is historical cost. If $b_\sigma > \tau_b$, the skill output is accepted. Otherwise, **HyperMAS** expands the source schema $\mathcal{B}_\sigma$ and executes its role nodes and hyperedges through explicit hypergraph collaboration. The final answer is synthesized from accepted skill outputs and explicit hyperedge outputs:

$$\hat{a} = m^{\mathrm{fin}}(\mathrm{Prompt}_{\mathrm{fin}}(q, \{\hat{a}_\sigma\}_{b_\sigma > \tau_b}, \{o_e\}_{e \in \mathcal{E}_q^{\mathrm{ex}}})) \tag{18}$$

Here m^{fin} is the final synthesizer LLM, $\mathrm{Prompt}_{\mathrm{fin}}$ formats all available intermediate results, and $\mathcal{E}_q^{\mathrm{ex}}$ denotes the explicit hyperedges executed for query q.

4.5 Training Procedure

Training proceeds in two phases. In the explicit exploration phase, the skill set is empty and **HyperMAS** learns role-level node construction and hyperedge construction through full hypergraph execution. The reward is:

$$R^{(1)}(q, \mathcal{H}_q) = U(\hat{a}, a^*) - \lambda C(\mathcal{H}_q, q) \tag{19}$$

Here $R^{(1)}$ is the Phase 1 reward, $U(\hat{a}, a^*)$ is correctness utility, and λ controls token cost sensitivity. The construction policy is optimized with policy gradient and entropy regularization:

$$\mathcal{L}^{(1)} = -\log \pi_\theta(\mathcal{H}_q \mid q) R^{(1)}(q, \mathcal{H}_q) - \lambda_H H(\pi_\theta) \tag{20}$$

Here $\mathcal{L}^{(1)}$ is the Phase 1 loss, π_θ is the construction policy, $H(\pi_\theta)$ is the entropy of role, model, and hyperedge selection distributions, and λ_H controls exploration.

In the skill evolution phase, distillation is triggered every T_d iterations. New skills are added to $\mathcal{K}$, and the policy learns to choose between skill invocation and explicit expansion. The reward becomes:

$$R^{(2)}(q, \mathcal{A}_q) = U(\hat{a}, a^*) - \lambda C(q, \mathcal{A}_q) - \mu N_{\mathrm{ex}}(q, \mathcal{A}_q) \tag{21}$$

Table 1: Performance comparison with three types of baselines, including single-agent execution, static multi-agent topologies, and adaptive multi-agent frameworks. The best results are in bold, and the runner-ups are underlined. All multi-agent methods utilize **five** agents drawn from the Qwen2.5-72B-Instruct, Llama-3.3-70B-Instruct, and Qwen2.5-32B-Instruct pool. "Mul.", "Ada.", and "Rob." indicate whether the method supports a multi-agent setting, whether it is task-adaptive, and whether it is adversarially robust, respectively. ✗, ✓ and ✓ signifies no/partial/full support in these aspects.

Method	Mul.	Ada.	Rob.	MMLU	GSM8K	MultiArith	SVAMP	AQuA	HumanEval	Avg.
Single-Agent Methods										
Vanilla	✗	✗	✗	82.14	85.40	93.15	87.18	70.34	71.68	81.65
CoT	✗	✗	✗	82.65$_{\uparrow 0.51}$	87.17$_{\uparrow 1.77}$	94.79$_{\uparrow 1.64}$	88.32$_{\uparrow 1.14}$	73.91$_{\uparrow 3.57}$	75.52$_{\uparrow 3.84}$	83.73
ComplexCoT	✗	✗	✗	83.78$_{\uparrow 1.64}$	87.62$_{\uparrow 2.22}$	95.86$_{\uparrow 2.71}$	90.17$_{\uparrow 2.99}$	77.58$_{\uparrow 7.24}$	74.94$_{\uparrow 3.26}$	84.99
SC (CoT)	✗	✗	✗	82.66$_{\uparrow 0.52}$	87.93$_{\uparrow 2.53}$	96.88$_{\uparrow 3.73}$	88.69$_{\uparrow 1.51}$	75.08$_{\uparrow 4.74}$	77.30$_{\uparrow 5.62}$	84.76
SC (ComplexCoT)	✗	✗	✗	83.65$_{\uparrow 1.51}$	86.14$_{\uparrow 0.74}$	96.94$_{\uparrow 3.79}$	89.72$_{\uparrow 2.54}$	77.69$_{\uparrow 7.35}$	77.94$_{\uparrow 6.26}$	85.35
AutoGPT	✗	✗	✗	83.42$_{\uparrow 1.28}$	87.85$_{\uparrow 2.45}$	96.12$_{\uparrow 2.97}$	89.34$_{\uparrow 2.16}$	76.53$_{\uparrow 6.19}$	79.18$_{\uparrow 7.50}$	85.41
PHP	✗	✗	✗	83.45$_{\uparrow 1.31}$	<u>95.50</u>$_{\uparrow 10.10}$	<u>98.10</u>$_{\uparrow 4.95}$	90.02$_{\uparrow 2.84}$	79.00$_{\uparrow 8.66}$	82.96$_{\uparrow 11.28}$	88.17
ReAct	✗	✗	✗	83.12$_{\uparrow 0.98}$	88.24$_{\uparrow 2.84}$	95.37$_{\uparrow 2.22}$	89.15$_{\uparrow 1.97}$	76.42$_{\uparrow 6.08}$	78.35$_{\uparrow 6.67}$	85.11
ToT	✗	✗	✗	83.89$_{\uparrow 1.75}$	89.06$_{\uparrow 3.66}$	96.52$_{\uparrow 3.37}$	90.24$_{\uparrow 3.06}$	77.95$_{\uparrow 7.61}$	80.12$_{\uparrow 8.44}$	86.30
GoT	✗	✗	✗	84.01$_{\uparrow 1.87}$	89.47$_{\uparrow 4.07}$	96.73$_{\uparrow 3.58}$	90.38$_{\uparrow 3.20}$	78.24$_{\uparrow 7.90}$	81.26$_{\uparrow 9.58}$	86.68
Static Multi-Agent Topologies										
Chain	✓	✗	✗	82.35$_{\uparrow 0.21}$	85.57$_{\uparrow 0.17}$	94.38$_{\uparrow 1.23}$	83.41$_{\downarrow 3.77}$	70.94$_{\uparrow 0.60}$	80.88$_{\uparrow 9.20}$	82.92
Star	✓	✗	✗	80.79$_{\downarrow 1.35}$	85.55$_{\uparrow 0.15}$	93.79$_{\uparrow 0.64}$	88.09$_{\uparrow 0.91}$	68.57$_{\downarrow 1.77}$	75.65$_{\uparrow 3.97}$	82.07
Tree	✓	✗	✗	81.89$_{\downarrow 0.25}$	84.56$_{\downarrow 0.84}$	94.60$_{\uparrow 1.45}$	89.25$_{\uparrow 2.07}$	72.84$_{\uparrow 2.50}$	77.38$_{\uparrow 5.70}$	83.42
Complete Graph	✓	✗	✗	83.15$_{\uparrow 1.01}$	86.49$_{\uparrow 1.09}$	97.20$_{\uparrow 4.05}$	89.48$_{\uparrow 2.30}$	79.21$_{\uparrow 8.87}$	83.75$_{\uparrow 12.07}$	86.55
Random Graph	✓	✗	✗	83.76$_{\uparrow 1.62}$	86.14$_{\uparrow 0.74}$	95.46$_{\uparrow 2.31}$	85.41$_{\downarrow 1.77}$	74.07$_{\uparrow 3.73}$	82.66$_{\uparrow 10.98}$	84.58
Adaptive Multi-Agent Frameworks										
AutoGen	✓	✗	✗	82.13$_{\downarrow 0.01}$	90.06$_{\uparrow 4.66}$	93.80$_{\uparrow 0.65}$	88.44$_{\uparrow 1.26}$	73.65$_{\uparrow 3.31}$	85.41$_{\uparrow 13.73}$	85.58
MetaGPT	✓	✗	✗	83.24$_{\uparrow 1.10}$	89.84$_{\uparrow 4.44}$	95.12$_{\uparrow 1.97}$	89.56$_{\uparrow 2.38}$	76.18$_{\uparrow 5.84}$	85.90$_{\uparrow 14.22}$	86.64
LLM-Blender	✓	✗	✗	81.22$_{\downarrow 0.92}$	89.17$_{\uparrow 3.77}$	94.27$_{\uparrow 1.12}$	88.77$_{\uparrow 1.59}$	77.05$_{\uparrow 6.71}$	84.52$_{\uparrow 12.84}$	85.83
LLM-Debate	✓	✗	✓	83.69$_{\uparrow 1.55}$	90.23$_{\uparrow 4.83}$	96.27$_{\uparrow 3.12}$	90.56$_{\uparrow 3.38}$	77.52$_{\uparrow 7.18}$	83.79$_{\uparrow 12.11}$	87.01
DyLAN	✓	✓	✓	80.16$_{\downarrow 1.98}$	88.16$_{\uparrow 2.76}$	94.27$_{\uparrow 1.12}$	87.40$_{\uparrow 0.22}$	74.16$_{\uparrow 3.82}$	<u>89.70</u>$_{\uparrow 18.02}$	85.64
GPTSwarm	✓	✓	✓	83.98$_{\uparrow 1.84}$	89.74$_{\uparrow 4.34}$	97.84$_{\uparrow 4.69}$	86.42$_{\downarrow 0.76}$	78.16$_{\uparrow 7.82}$	88.49$_{\uparrow 16.81}$	87.44
AgentVerse	✓	✓	✗	83.52$_{\uparrow 1.38}$	90.12$_{\uparrow 4.72}$	96.45$_{\uparrow 3.30}$	89.87$_{\uparrow 2.69}$	77.83$_{\uparrow 7.49}$	86.24$_{\uparrow 14.56}$	87.34
COPPER	✓	✓	✗	83.76$_{\uparrow 1.62}$	91.35$_{\uparrow 5.95}$	96.82$_{\uparrow 3.67}$	90.18$_{\uparrow 3.00}$	78.42$_{\uparrow 8.08}$	87.53$_{\uparrow 15.85}$	88.01
AutoAgents	✓	✓	✗	83.45$_{\uparrow 1.31}$	90.58$_{\uparrow 5.18}$	96.15$_{\uparrow 3.00}$	89.64$_{\uparrow 2.46}$	77.29$_{\uparrow 6.95}$	86.87$_{\uparrow 15.19}$	87.33
G-Designer	✓	✓	✓	84.25$_{\uparrow 2.11}$	92.18$_{\uparrow 6.78}$	97.56$_{\uparrow 4.41}$	91.02$_{\uparrow 3.84}$	78.94$_{\uparrow 8.60}$	88.72$_{\uparrow 17.04}$	88.78
AgentPrune	✓	✓	✓	84.15$_{\uparrow 2.01}$	91.86$_{\uparrow 6.46}$	97.38$_{\uparrow 4.23}$	90.73$_{\uparrow 3.55}$	78.65$_{\uparrow 8.31}$	88.15$_{\uparrow 16.47}$	88.49
AgentDropout	✓	✓	✓	84.08$_{\uparrow 1.94}$	91.52$_{\uparrow 6.12}$	97.21$_{\uparrow 4.06}$	90.45$_{\uparrow 3.27}$	78.51$_{\uparrow 8.17}$	87.68$_{\uparrow 16.00}$	88.24
ARG-Designer	✓	✓	✓	<u>84.72</u>$_{\uparrow 2.58}$	93.47$_{\uparrow 8.07}$	98.04$_{\uparrow 4.89}$	<u>91.38</u>$_{\uparrow 4.20}$	<u>79.36</u>$_{\uparrow 9.02}$	89.41$_{\uparrow 17.73}$	<u>89.40</u>
HyperMAS (Ours)	✓	✓	✓	**86.11**$_{\uparrow 3.97}$	**96.27**$_{\uparrow 10.87}$	**99.06**$_{\uparrow 5.91}$	**93.42**$_{\uparrow 6.24}$	**82.14**$_{\uparrow 11.80}$	**92.06**$_{\uparrow 20.38}$	**91.51**

Here $R^{(2)}$ is the Phase 2 reward, $\mathcal{A}_q$ is the active execution structure, N_{ex} is the number of explicit hyperedge executions, and μ penalizes unnecessary explicit communication. The verifier is trained with binary cross-entropy:

$$\mathcal{L}_{\text{ver}} = -\sum_{\sigma,q} [y_{\sigma,q} \log b_\sigma + (1 - y_{\sigma,q}) \log(1 - b_\sigma)] \tag{22}$$

Here $y_{\sigma,q} = 1$ if skill σ produces a correct output for query q, and $y_{\sigma,q} = 0$ otherwise. During Phase 2, skills with rolling success rate below threshold τ_{off} are deactivated temporarily, and their fallback traces can produce updated versions through re-distillation.

5 Experiments

5.1 Experiment Setup

Datasets and Metrics. We evaluate `HyperMAS` on MMLU (Hendrycks et al., 2021) for general reasoning, on GSM8K (Cobbe et al., 2021), MultiArith (Roy & Roth, 2016), SVAMP (Patel et al., 2021), and AQuA (Ling et al., 2017) for mathematical reasoning, and on HumanEval (Chen, 2021) for code generation. We report accuracy for MMLU and the math benchmarks and pass@1 for Hu-

Table 2: Skill-bank behavior across benchmarks. "Inv." is the fraction of queries invoking a stored skill, "FB" is verifier-triggered fallback, "Edges" is the mean number of explicit hyperedge executions, and "Compact" is the mean token cost of the compact path.

Benchmark	Traces	Skills	Inv.	FB	Edges	Compact (K)	Tok. red.	Score
MMLU	52	22	0.43	31.8%	2.76	5.32	18.7%	86.11
GSM8K	47	9	0.78	20.8%	1.94	1.80	34.2%	96.27
MultiArith	44	6	0.71	22.4%	2.31	2.10	31.1%	99.06
SVAMP	41	7	0.68	23.9%	2.40	2.30	29.6%	93.42
AQuA	38	8	0.59	27.1%	2.57	2.60	25.9%	82.14
HumanEval	39	11	0.64	24.6%	2.14	2.20	29.8%	92.06

manEval. Efficiency and structural metrics include average token cost, explicit hyperedge executions, skill invocation, and fallback rates.

Baselines. We compare with three baseline groups. Single-agent methods include Vanilla, CoT (Wei et al., 2022), ComplexCoT, Self-Consistency (Wang et al., 2023b) with CoT and ComplexCoT, AutoGPT, ReAct (Yao et al., 2023b), ToT (Yao et al., 2023a), GoT (Besta et al., 2023), and PHP. Static MAS structures include Chain, Star, Tree, Complete Graph, and Random Graph. Adaptive MAS methods include AutoGen (Wu et al., 2023), MetaGPT (Hong et al., 2024), LLM-Blender (Jiang et al., 2023), LLM-Debate (Du et al., 2024), DyLAN (Liu et al., 2024), GPTSwarm (Zhuge et al., 2024), AgentVerse (Chen et al., 2024), COPPER, AutoAgents, AgentPrune (Zhang et al., 2025a), AgentDropout (Wang et al., 2025a), G-Designer (Zhang et al., 2025b), and ARG-Designer (Li et al., 2025b).

Implementation Details. We use Qwen2.5-72B-Instruct, Llama-3.3-70B-Instruct, and Qwen2.5-32B-Instruct through a fixed five-agent pool. Temperature is 0 for deterministic single-agent baselines, 0.8 with five paths for Self-Consistency, and 1 for MAS methods. We set $S_{\max} = 3$, use `all-MiniLM-L6-v2` with $D = 384$, and allow $W = 2$ proposers plus one synthesizer per role node. Hyperedge grouping uses $k = 2$, and skills are distilled every $T_d = 200$ iterations with $\tau_d = 0.5$, $\tau_a = 0.55$, and $\tau_b = 0.60$. The verifier is a two-layer MLP with hidden dimension 128. Adam uses a 3×10^{-4} learning rate and $\lambda_H = 0.01$. Experiments use 8 NVIDIA A100 80GB GPUs.

Table 2 shows that GSM8K and MultiArith invoke stored skills most often, whereas MMLU and AQuA retain higher fallback rates as their evidence paths vary more across queries.

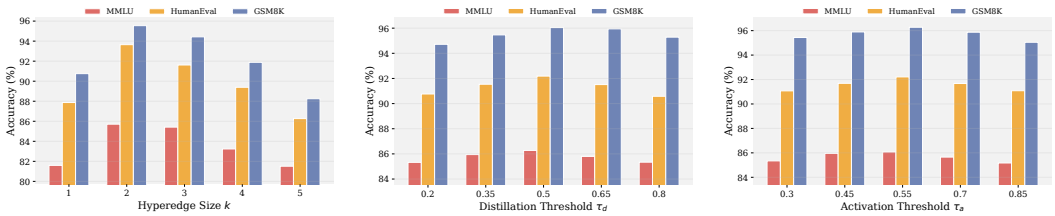
5.2 Main Result (**RQ1**)

RQ1: Can `HyperMAS` outperform single-agent methods, static MAS topologies, and adaptive MAS frameworks?

Table 1 reports the main comparison across six benchmarks. **`HyperMAS`** achieves the best average score of 91.51 and surpasses the strongest baseline, ARG-Designer (89.40), by an absolute 2.11%. The gains span general reasoning, mathematical reasoning, and code generation, with clear improvements on GSM8K, AQuA, and HumanEval where computation, verification, testing, and review recur.

The results also show a progression across method categories. Single-agent methods lack explicit role specialization, static MAS topologies cannot adapt communication to different queries, and adaptive MAS frameworks still treat each constructed structure as a task-local execution plan. The advantage of **`HyperMAS`** suggests that MAS performance can be further improved by accumulating reliable collaboration experience and reusing it through evolved skills, rather than only constructing collaboration from scratch for each query.

Generality and deployment boundary. Additional tests cover an alternative model allocation, distribution shift, less repetitive workloads, and single-agent skill learning. Under an alternative allocation of the same three backbones, **`HyperMAS`** averages 88.64 versus 85.92 for ARG-Designer while reducing tokens by 25.1%. Protocols and additional analyses appear in Sections A, B and D.



(a) Hyperedge Size k (b) Distillation Threshold τ_d (c) Activation Threshold τ_a

Figure 3: Sensitivity analysis of key hyperparameters. We vary hyperedge size k, distillation threshold τ_d, and activation threshold τ_a on MMLU, HumanEval, and GSM8K. All three factors show a peaked response rather than a monotonic trend, indicating that collaborative skill evolution requires controlled hypergraph granularity, selective distillation, and reliable activation.

5.3 Model Analysis (**RQ2**)

RQ2: Does `HyperMAS` truly reduce repeated explicit collaboration through evolved skills?

Figure 1 traces execution across evolution stages. Performance rises overall as compact skills retain task quality. Explicit hyperedge executions decline after skill evolution begins, particularly on GSM8K and HumanEval where verification and review recur, while the decline is slower on MMLU. Skill invocation rises with experience and fallback remains non-zero, showing that **`HyperMAS`** compresses familiar routines while retaining explicit hypergraph expansion for uncertain cases.

5.4 Hyperparameter Analysis (**RQ3**)

RQ3: How do key hyperparameters affect the balance between skill compression and explicit collaboration?

We analyze hyperedge size k, distillation threshold τ_d, and activation threshold τ_a in Figure 3. All three factors show peaked responses, with moderate k and interior threshold values producing the most reliable balance. Small hyperedges miss group-level coordination, whereas large ones add communication noise. Aggressive activation admits unreliable skills, whereas conservative activation retains unnecessary explicit execution. The shared interior optimum supports one fixed configuration across benchmarks.

5.5 Ablation Study (**RQ4**)

RQ4: Which components are necessary for reliable collaborative skill evolution?

Table 3 reports the ablation results. Removing skill evolution raises token cost from 6.37K to 8.71K and explicit hyperedge executions from 2.28 to 4.86. Hypergraph skills, role mixtures, and verifier fallback improve reliability, while cost-aware distillation gives the lowest cost among full systems with the strongest average performance and skill success.

Variant	Perf. Avg. ↑	Comm. Tok. ↓	Comm. Hyp. ↓	Skill Exec. Succ. ↑	Skill Exec. FB ↓
w/o Skill Evo.	90.64	8.71	4.86	–	100.0
w/o Hypergraph Skill	90.37	6.98	2.94	80.3	28.7
w/o Role Mixture	89.96	<u>6.27</u>	2.38	82.1	27.9
w/o Verifier & FB	89.54	**5.83**	**1.76**	74.6	**0.0**
w/o Cost-aware Distill.	<u>91.08</u>	7.19	2.67	<u>87.2</u>	<u>24.1</u>
`HyperMAS`	**91.51**	6.37	<u>2.28</u>	**88.9**	25.1

Table 3: Ablation of `HyperMAS`. Avg. and FB are means over the six benchmarks.

6 Conclusion

We introduced `HyperMAS` to distill reliable collaborative sub-hypergraphs into reusable skills, improving performance and reducing repeated communication across benchmarks.

AI Usage Statement. AI assisted only with language polishing.

REFERENCES

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 2025. Anthropic Engineering Blog.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. arXiv preprint arXiv:2308.09687, 2023.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. arXiv e-prints, August 2023.

Mark Chen. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

Qijia Chen, Andrea Bellucci, Zhida Sun, and Giulio Jacucci. Skilldroid: Compile once, reuse forever. arXiv preprint arXiv:2604.14872, 2026.

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

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

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Forty-first International Conference on Machine Learning, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

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

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. arXiv preprint arXiv:2306.02561, 2023.

Dawei Li, Zhen Tan, Peijia Qian, Yifan Li, Kumar Chaudhary, Lijie Hu, and Jiayi Shen. Smoa: Improving multi-agent large language models with s parse m ixture-o f-a gents. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 54–65. Springer, 2025a.

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

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

Wenzhe Li, Yong Lin, Mengzhou Xia, and Chi Jin. Rethinking mixture-of-agents: Is mixing different large language models beneficial? arXiv preprint arXiv:2502.00674, 2025c.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 17889–17904, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.992. URL https://aclanthology.org/2024.emnlp-main.992/.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. arXiv preprint arXiv:1705.04146, 2017.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration, 2024. URL https://arxiv.org/abs/2310.02170.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. arXiv preprint arXiv:2604.02268, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2024. URL https://arxiv.org/abs/2406.18665.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? arXiv preprint arXiv:2103.07191, 2021.

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. Chatdev: Communicative agents for software development. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, 2024.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In The Thirteenth International Conference on Learning Representations, 2025.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. arXiv preprint arXiv:1608.01413, 2016.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space, 2024. URL https://arxiv.org/abs/2410.06153.

Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of intelligent llm agents, 2023. URL https://arxiv.org/abs/2306.03314.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents. arXiv preprint arXiv:2604.04804, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291, 2023a. doi: 10.48550/arXiv.2305.16291. URL https://arxiv.org/abs/2305.16291.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. URL https://arxiv.org/abs/2406.04692.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023b. URL https://arxiv.org/abs/2203.11171.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 24013–24035, 2025a.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. arXiv preprint arXiv:2504.06821, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155, 2023.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.08234.

Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. Grasp: Graph-structured skill compositions for llm agents. arXiv preprint arXiv:2604.17870, 2026b.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. arXiv preprint arXiv:2602.12430, 2026.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In The Eleventh International Conference on Learning Representations, 2023b.

Tianjun Yao, Zhaoyi Li, and Zhiqiang Shen. Hieramas: Optimizing intra-node llm mixtures and inter-node topology for multi-agent systems. arXiv preprint arXiv:2602.20229, 2026. doi: 10.48550/arXiv.2602.20229. URL https://arxiv.org/abs/2602.20229.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In Annual Meeting of the Association for Computational Linguistics, 2025.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In International Conference on Learning Representations, 2025a.

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

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026. URL https://arxiv.org/abs/2604.01687.

Heng Zhang, Yuling Shi, Xiaodong Gu, Zijian Zhang, Haochen You, Lubin Gan, Yilei Yuan, and Jin Huang. Hyperagent: Leveraging hypergraphs for topology optimization in multi-agent communication. arXiv preprint arXiv:2510.10611, 2025c. doi: 10.48550/arXiv.2510.10611. URL https://arxiv.org/abs/2510.10611.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. arXiv preprint arXiv:2410.10762, 2024.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. In Forty-first International Conference on Machine Learning, 2024.

Appendix

A Generalization and Skill Reuse Analysis **15**
A.1 Alternative Model Allocation 15
A.2 Distribution Shift and Less Repetitive Workloads 15
A.3 Comparison with Single-Agent Skill Learning 16
B Construction Cost and Amortization **16**
C Unified Training and Inference Procedure **17**
C.1 Concrete Hyperedge Walkthrough 17
D Skill Distillation Artifacts **18**
D.1 Verbatim Extract Prompt 18
D.2 Complete Mathematics Skill 18
D.3 Complete Code Skill 18
D.4 Complete Knowledge Skill 18
E Robustness, Skill-Bank Dynamics, and Sensitivity **19**
E.1 Verifier Control under Distribution Shift 19
E.2 Rule-Based and Learned Distillation Scores 19
E.3 Skill-Bank Growth and Retirement 19
E.4 Fixed Hyperparameter Transfer 19
F Statistical Protocol, Compute, and Scope **20**

A Generalization and Skill Reuse Analysis

This section evaluates whether the collaboration primitives remain useful when the backbone ecosystem, task distribution, or reusable unit changes. Each comparison preserves the role inventory, construction budget, answer extraction, and evaluation scripts of the corresponding main experiment. Token reduction is measured relative to ARG-Designer under the same query set and model pool.

A.1 Alternative Model Allocation

We evaluate an alternative fixed allocation of Qwen2.5-72B-Instruct, Llama-3.3-70B-Instruct, and Qwen2.5-32B-Instruct while keeping five active agents and the default thresholds. This allocation makes compact execution less reliable on a subset of tasks, so the verifier expands a larger fraction of queries into explicit collaboration. The accuracy gain remains positive, while the token reduction decreases relative to the default allocation.

Table 4: Alternative model-allocation evaluation. Avg. is computed over the same six benchmarks as Table 1. Token reduction is relative to ARG-Designer.

Method	MMLU	GSM8K	HumanEval	Avg.	Token red.	Skill succ.	Fallback
AgentPrune	80.94	89.61	83.42	85.28	–	–	–
ARG-Designer	81.36	90.24	84.15	85.92	–	–	–
`HyperMAS`	**83.75**	**93.86**	**88.53**	**88.64**	25.1%	84.7%	26.3%

The fallback rate rises from 25.1% under the default allocation to 26.3% under the alternative allocation, while skill success decreases from 88.9% to 84.7%. This shift is consistent with the intended verifier behavior: lower-confidence compact executions are expanded into the stored source sub-hypergraph, trading part of the efficiency gain for a more reliable answer.

A.2 Distribution Shift and Less Repetitive Workloads

For zero-shot transfer, skills are constructed on GSM8K, HumanEval, and MMLU and evaluated without skill-bank updates on MATH, MBPP, and StrategyQA. The model-selection networks and thresholds remain fixed. Table 5 reports task performance together with invocation, fallback, and token reduction.

Table 5: Zero-shot transfer. Invocation and fallback are fractions of evaluated queries.

OOD set	ARG-Designer	`HyperMAS`	Skill invoke	Fallback	Token red.
MATH	58.24	**60.91**	0.42	38.5%	21.3%
MBPP	76.38	**79.62**	0.51	31.2%	24.6%
StrategyQA	71.05	**72.16**	0.19	57.4%	11.7%

The task sequence in Table 6 varies the amount of recurring coordination while retaining a setting in which multi-role reasoning is useful. Skill invocation and token reduction decrease together as workflows become less repetitive. Accuracy remains above the adaptive baseline at each point because role-level mixtures and explicit hyperedge construction remain available when no skill matches.

Table 6: Performance across a spectrum of workflow repetition. The GAIA subset contains 200 level-1 and level-2 tasks.

Benchmark	ARG-Designer	`HyperMAS`	Skill invoke	Token red.
GSM8K	93.47	**96.27**	0.78	34.2%
MATH level 5	42.16	**44.05**	0.31	14.9%
GAIA level 1–2	34.60	**36.20**	0.23	9.8%

A.3 Comparison with Single-Agent Skill Learning

We instantiate three skill-learning baselines within the same model pool and evaluation harness. The Voyager-style library stores executable solution programs, ExpeL stores textual experience, and SkillRL stores policy skills. HyperMAS stores a coordination sub-hypergraph together with a compact template and a fallback schema. This comparison isolates whether representing role handoffs contributes beyond reusing a single-agent procedure.

Table 7: Controlled comparison with single-agent skill-learning methods. Tokens are averaged per query in thousands.

Method	Avg.	Tokens	Reusable unit
Voyager-style skill library	86.55	5.71	solution program
ExpeL	86.83	6.02	textual experience
SkillRL	87.12	5.94	policy skill
`HyperMAS`	**91.51**	6.37	coordination sub-hypergraph

On HumanEval, 61% of the token reduction comes from compressing the propose-test-review handoff. The remaining reduction comes from shorter within-role proposals and avoided repeated verification. This decomposition identifies the additional unit represented by HyperMAS: the skill records how several roles exchange work, rather than only what one model should do.

B Construction Cost and Amortization

The inference results in the main paper exclude the one-time construction phase. We therefore report explicit exploration, skill evolution, and Extract calls separately. API cost uses the same blended rates as the main experiments, and wall-clock time includes remote model latency and local policy updates.

Table 8: End-to-end construction cost for the default experiment suite.

Phase	LLM calls	Tokens	API cost	Wall-clock
Explicit exploration	128,400	41.2M	\$312	18.6 h
Skill evolution	96,700	27.8M	\$198	12.4 h
Extract, 34 triggers	1,088	2.1M	\$17	0.9 h
Total construction	**226,188**	**71.1M**	**\$527**	**31.9 h**

Let C_{build} denote the one-time construction cost and let ΔC_d be the expected per-query saving for deployment domain d. The query break-even point is

$$N_d^{\text{break}} = \lceil C_{\text{build}} / \Delta C_d \rceil \,. \tag{23}$$

Against ARG-Designer, HyperMAS saves an average of 2.53K tokens per query, corresponding to approximately \$0.095 under the reported blended rates. The domain-level calculation in Table 9 makes the deployment boundary explicit.

Table 9: Amortization by workload. Recurrence is the fraction of queries whose selected coordination signature has appeared previously.

Workload	Recurrence	Saving/query	Cost/query	Break-even queries
Mathematics	0.74	3.41K tokens	\$0.170	3,100
Code	0.61	2.86K tokens	\$0.120	4,400
All six benchmarks	0.49	2.53K tokens	\$0.095	5,550
Knowledge-intensive	0.28	1.16K tokens	\$0.047	11,300

The construction phase is therefore most appropriate for recurring workloads. On a one-off or rapidly changing task distribution, the verifier still protects answer quality, but the saved communication may not recover the construction cost within the deployment horizon.

Retrieval and verification overhead. The final bank contains 63 active skills with 384-dimensional embeddings. Exact cosine retrieval takes less than 0.4 ms per query in this setting. Verification accounts for approximately 3% of per-query latency and is larger than retrieval overhead. For a larger bank, approximate nearest-neighbor indexing can replace exhaustive search without changing the skill representation or execution policy.

C Unified Training and Inference Procedure

Algorithm 1 makes the two phases and the inference fallback explicit. The important distinction is that affinity scores construct candidate role groups, while a hyperedge execution is a shared communication round in which member outputs are visible to the designated group aggregator.

Algorithm 1 HyperMAS training, skill evolution, and inference

Require: training set $\mathcal{D}$, model pool $\mathbb{M}$, roles $\mathbb{R}$, thresholds τ_d, τ_a, τ_b
initialize construction policy π_θ, trace buffer $\mathcal{T} \leftarrow \varnothing$, skill bank $\mathcal{K} \leftarrow \varnothing$
for each Phase-1 query q **do**
    construct role-level mixture nodes and explicit hyperedges $\mathcal{H}_q$
    execute every hyperedge as a shared group communication round
    update π_θ using correctness and token-cost reward and append the trace to $\mathcal{T}$
end for
for each Phase-2 query q **do**
    if the iteration is a distillation trigger **then**
        group successful trace sub-hypergraphs by canonical role-hyperedge signature
        distill candidates satisfying reliability, cost, and recurrence criteria into $\mathcal{K}$
    end if
    retrieve candidate skills with $\alpha_\sigma(q) > \tau_a$
    for each retrieved skill σ **do**
        execute the compact template and estimate verifier confidence b_σ
        if $b_\sigma \leq \tau_b$ **then** expand source schema $\mathcal{B}_\sigma$ and execute it explicitly
        end if
    end for
    combine accepted skill outputs with explicit hyperedge outputs, then update π_θ and the verifier
    deactivate skills whose rolling success falls below τ_{off}
end for
return π_θ, $\mathcal{K}$, and the verifier

C.1 Concrete Hyperedge Walkthrough

Consider a GSM8K query for which three role nodes survive pruning: Solver v_1, Verifier v_2, and Synthesizer v_3. Within v_1, two proposer models generate independent derivations and the node synthesizer merges them into o_1. This is intra-role redundancy.

The first hyperedge is $e_1 = \{v_1, v_2\}$. The shared edge prompt contains o_1, the verifier output o_2, and the subgoal description. The member with the highest average affinity supplies the aggregator model, which produces o_{e_1}. The second hyperedge $e_2 = \{v_1, v_2, v_3\}$ receives o_{e_1} together with o_3 and produces the final answer. Members therefore coordinate through a group-level exchange rather than a vote over independent answers.

When this two-edge pattern recurs with high utility and lower compact cost, it becomes the source schema of a skill. Compact execution uses the distilled template in one call. If confidence falls below τ_b, the same stored schema expands into v_1, v_2, v_3, e_1, e_2 and executes explicitly.

D SKILL DISTILLATION ARTIFACTS

D.1 VERBATIM EXTRACT PROMPT

Extract instruction

You are given N successful execution traces that share the role–hyperedge signature {roles, hyperedges}. Each trace contains the query, the per-role intermediate outputs, and the final answer. Write one self-contained instruction that reproduces the coordination behaviour these traces have in common. Include (1) the order in which roles act, (2) what each role checks or produces, (3) the verification criterion that appears in the successful traces, and (4) the required output format. Do not include problem-specific content. Do not exceed 200 words.

The prompt receives only successful traces with the same canonical signature. The resulting template is stored together with the signature, executor, empirical utility, cost, and occurrence count. The runtime verifier, rather than extraction, decides on each query whether a skill is safe to invoke.

D.2 COMPLETE MATHEMATICS SKILL

The mathematics skill is distilled from 47 reliable occurrences of a Solver-Verifier-Synthesizer pattern. Its mean utility is 0.94, explicit cost is 4.9K tokens, and compact cost is 1.8K tokens. The executor is `Qwen2.5-32B-Instruct`. The template is:

> *Solve the problem step by step and state the final numeric answer. Then re-derive the same answer by a different route, using a unit check, a back-substitution, or an order-of-magnitude estimate. If the two derivations disagree, identify the first step where they diverge and redo it. Output three sections: [Derivation], [Independent check], [Final answer].*

The explicit schema contains three nodes and two hyperedges and requires five LLM calls. Compact execution uses one call. A failed verifier check restores the complete schema.

D.3 COMPLETE CODE SKILL

The code skill uses nodes {Implementer, Tester, Reviewer} and hyperedges {(Implementer, Tester), (Implementer, Tester, Reviewer)}. It is distilled from 39 reliable occurrences, with mean utility 0.91, explicit cost 5.6K tokens, and compact cost 2.2K tokens. Its template requests an implementation, two boundary tests, a dry run, and a final code block only after discrepancies are resolved. The verification fields record compilation status, test outcomes, and whether the reviewer identified an unhandled edge case.

D.4 COMPLETE KNOWLEDGE SKILL

The knowledge skill uses nodes {Retriever, Cross-checker, Synthesizer}. It is distilled from 52 occurrences with mean utility 0.89, explicit cost 4.1K tokens, and compact cost 2.0K tokens. The template requires an initial answer, a second evidence route, an explicit conflict check, and one concise final response. In contrast to the mathematics and code skills, this pattern has lower activation on MMLU because many questions do not share the same evidence route.

Table 10: Representative distilled skills. Cost is measured in thousands of tokens.

Domain	Occurrences	Utility	Explicit	Compact	Verification signal
Mathematics	47	0.94	4.9	1.8	independent derivation
Code	39	0.91	5.6	2.2	compile and tests
Knowledge	52	0.89	4.1	2.0	evidence agreement

E Robustness, Skill-Bank Dynamics, and Sensitivity

E.1 Verifier Control under Distribution Shift

We replace the learned verifier with a random acceptor matched to the same acceptance rate. On StrategyQA, the matched random acceptor reaches 68.4, compared with 72.16 for the learned verifier and 71.05 for ARG-Designer. Matching the acceptance rate holds expected compact-execution frequency constant and isolates whether the verifier identifies which outputs require expansion.

Table 11: Verifier control on StrategyQA.

Decision rule	Accuracy	Accept rate	Token red.
ARG-Designer	71.05	–	–
Random acceptor	68.40	42.6%	12.0%
Learned verifier	**72.16**	42.6%	11.7%

E.2 Rule-Based and Learned Distillation Scores

The default distillation score is a monotone combination of empirical utility, explicit cost, and recurrence. We compare it with a two-layer MLP over the pattern embedding and the same statistics, trained on held-out trace outcomes. The two variants produce similar endpoint performance, token use, and skill success. This result supports a separation of responsibilities: the distillation rule gates stable candidates into the bank, while the learned verifier handles query-level uncertainty at execution time.

Table 12: Alternative distillation scorer.

Distillation scorer	Avg.	Tokens (K)	Skill success
Rule-based	91.51	6.37	88.9%
Learned two-layer MLP	**91.58**	**6.31**	**89.4%**

E.3 Skill-Bank Growth and Retirement

The final bank contains 63 active skills: 22 for MMLU, 9 for GSM8K, 6 for MultiArith, 7 for SVAMP, 8 for AQuA, and 11 for HumanEval. During training, 19 skills fall below τ_{off} and are deactivated. Seven later reappear as updated skills distilled from successful fallback traces. Deactivation removes the skill from retrieval but preserves its source traces for diagnostics. Re-distillation creates a new version with separate statistics.

Table 13: Final skill bank and lifecycle events.

Dataset	Stable skills	Deactivated	Re-distilled	Median occurrences
MMLU	22	8	2	31
GSM8K	9	2	1	46
MultiArith	6	1	1	43
SVAMP	7	2	1	39
AQuA	8	3	1	35
HumanEval	11	3	1	37
Total	63	19	7	–

E.4 Fixed Hyperparameter Transfer

One configuration is selected on a 500-query held-out split: $k = 2$, $\tau_d = 0.50$, $\tau_a = 0.55$, and $\tau_b = 0.60$. The same configuration is used across all six benchmarks in Table 1 and in the transfer evaluations. Within $\tau_d \in [0.35, 0.65]$, the standard deviation of average accuracy is 0.42, and within

$\tau_a \in [0.45, 0.70]$ it is 0.51. The endpoint failures in the main sensitivity plot correspond to admitting unreliable candidates or distilling almost no skills, rather than instability throughout the operating range.

F Statistical Protocol, Compute, and Scope

Randomness and intervals. Main results are averaged over three runs whenever policy optimization or stochastic generation is involved. Analysis curves report the mean and sample standard deviation. For skill invocation, fallback, skill success, and explicit hyperedge executions, we additionally compute query-level 95% bootstrap confidence intervals with $1{,}000$ resamples. Hyperparameters are selected once on the held-out split described above and are not tuned separately for test datasets.

Compute resources. The default construction run uses eight NVIDIA A100 80GB GPUs for policy optimization, trace processing, and verifier training. Remote LLM calls dominate wall-clock time. Table 8 separates those calls from local optimization and reports the complete construction total rather than only inference-time communication.

Scope and limitations. HyperMAS relies on recurring and reliable collaboration routines. When queries require unique workflows, few sub-hypergraphs recur often enough to justify distillation, and the method approaches explicit adaptive collaboration. The verifier reduces the risk of applying a mismatched skill by expanding uncertain outputs, although expansion also reduces efficiency. Our evaluation covers tasks with relatively clear correctness signals. Open-ended generation, multimodal interaction, and long-horizon tool use require domain-appropriate verification signals and are outside the present empirical scope.

Broader impacts. Reducing repeated communication can lower the cost of collaborative reasoning systems. The same efficiency can increase the scale at which automated systems are deployed. Skill invocations, verifier decisions, and fallback expansions should therefore be logged and audited together with standard downstream safety evaluation.