SIGMA: Skill-Incidence Graphs for Compositional Multi-Agent Design

Anonymous ACL submission

Abstract

Existing graph-based multi-agent system (MAS) designers mainly improve collaboration by optimizing communication topologies over predefined agents, roles, or groups. However, because each node remains a closed-set entity, these methods struggle to generalize to tasks that require unseen combinations of capabilities. We propose SIGMA, a skill-incidence graph framework that constructs agents as task-conditioned bundles of reusable skills. Given a task and a skill library, SIGMA predicts a skill-agent incidence matrix, composes agent node embeddings from selected skills, and decodes a communication topology over the constructed agents. During execution, skill-specific mailboxes route messages to the relevant assigned capabilities, making the incidence structure directly operational. Across six reasoning and coding benchmarks with three base LLMs, SIGMA achieves the best average performance and improves over CARD, the strongest non-compositional topology-based baseline, by 2.06, 2.36, and 1.75 points, respectively. It also shows stronger robustness to unseen skill libraries, with an average performance drop of only 0.96 points. These results suggest that compositional node construction is a complementary and important axis for multi-agent design beyond communication topology optimization. Code is available at `https://anonymous.4open.science/r/SIGMA-2338/`.

1 Introduction

Large language model based multi-agent systems (**MAS**) solve complex tasks by assigning agents specialized roles and communication structures (Li et al., 2023a; Wu et al., 2024; Hong et al., 2024; Qian et al., 2024; Chen et al., 2024). Recent graph-based designers further automate this process by learning task-adaptive topologies over agent nodes (Liu et al., 2024; Zhuge et al., 2024; Zhang et al., 2025b,a). Yet most of them still regard each node as a closed-set role, operation, or predefined group: they optimize how existing agents communicate, while leaving what each agent is able to do largely fixed.

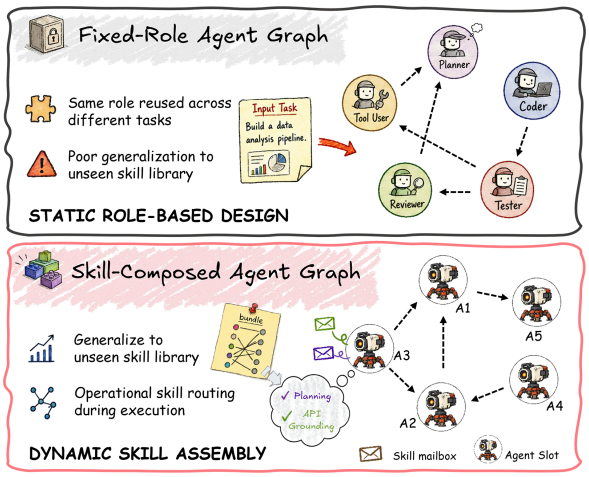


Figure 1: Comparison between fixed-role and skill-composed multi-agent graph design.

This node-level rigidity limits compositional generalization. Real tasks often require capabilities that cross role boundaries, including tool use and intermediate actions (Karpas et al., 2022; Yao et al., 2022), API grounding and structured tool access (Li et al., 2023b; Patil et al., 2023; Qin et al., 2024), and reusable procedural skills (Wang et al., 2023). Such capabilities are not always aligned with a single role. Although fixed roles are useful abstractions, they do not explicitly construct new agent identities from reusable capabilities. Consequently, a topology designer can learn a better graph over predefined agents, but it cannot make the same slot become a retrieval-grounded implementer for one task and a verification-heavy reasoner for another.

In light of this dilemma, we propose the **LLM-based Skill-Incidence Multi-Agent Protocol (SIMAP)**, which provides guidance for compositional LLM-MA graph design:

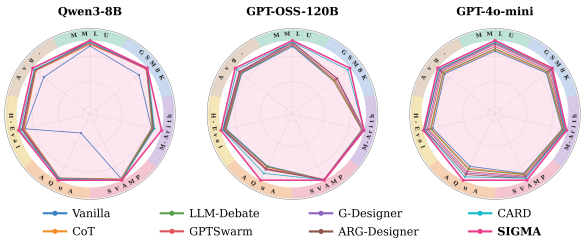


Figure 2: Performance plot across six benchmarks and three base LLMs.

Skill-Incidence Multi-Agent Protocol (SI-MAP): *Given a task q and skill library $\mathcal{L}$, a compositional LLM-MA designer should satisfy:* ***(1) Compositionality****, constructing each agent as a task-conditioned skill bundle;* ***(2) Executability****, making assigned skills affect prompts, tools, memory, or routing;* ***(3) Topology compatibility****, exposing constructed node representations to a graph decoder; and* ***(4) Expandability****, allowing new tool/API skill cards at test time without retraining the topology decoder.*

To instantiate SI-MAP, we introduce SIGMA, a compositional and executable LLM-powered multi-agent graph designer. SIGMA represents reusable capabilities as skill cards, predicts a task-conditioned skill-agent incidence matrix, composes node embeddings from the selected cards, and decodes a communication topology over the constructed agents. During execution, the same incidence structure is exposed through skill-specific mailboxes, so skill assignments are operational rather than latent annotations. This design deliberately separates two decisions that are coupled in role-based systems: *what* each agent can do is decided by skill incidence, whereas *how* agents exchange information is decided by topology decoding. The separation lets SIGMA keep the matched-crew evaluation protocol of prior graph-based MAS work while testing a different modeling axis, namely whether reusable capabilities can construct more transferable agent nodes. Figure 2 provides a compact preview of our empirical findings. The formal objects and full algorithms are given in Section 3, Section 4, and Appendix B.

Our contribution can be summarized as follows:

❶ ***Protocol Proposal.*** We propose SI-MAP, a protocol that shifts graph-based MAS design from connecting fixed agents to constructing executable agent nodes from reusable skills. It formalizes compositionality, executability, topology compatibility, and expandability as requirements for skill-aware multi-agent graph design.

❷ ***Practical Solution.*** We present SIGMA, which couples incidence-based node construction, skill-aware topology decoding, and mailbox-based execution under matched multi-agent settings. Unlike methods that only prune or reconnect predefined nodes, SIGMA changes the capability composition of each node before topology generation.

❸ ***Experimental Validation.*** We evaluate SIGMA against single-agent, debate-based, and topology-based multi-agent baselines under compositional generalization settings. The evaluation isolates whether skill incidence improves compositional generalization under the same crew size, context budget, and graph-decoding interface. As summarized by Figure 2, SIGMA achieves consistently strong performance across diverse benchmarks, suggesting that the gains are not tied to a specific backbone or task family.

2 Related Work

LLM-based multi-agent systems. LLM-based multi-agent systems decompose complex tasks into interacting agents with specialized roles, workflows, and communication protocols. Role-playing frameworks assign agents complementary identities, enabling them to collaborate through instruction following, perspective taking, and iterative communication (Li et al., 2023a; Chen et al., 2024). Workflow-oriented systems further organize agents into structured pipelines, where different agents are responsible for planning, coding, reviewing, testing, or project management (Wu et al., 2024; Hong et al., 2024; Qian et al., 2024). Another work improves reasoning by introducing debate, discussion, or self-reflection among multiple LLM agents, allowing agents to expose errors and refine intermediate answers through interaction (Du et al., 2024; Liang et al., 2024). Multi-agent frameworks have also been applied to simulation and decision-making scenarios (Park et al., 2023).

Graph-based Multi-agent Design. Graph-based MAS methods mainly improve collaboration by adapting the communication topology among predefined agents. One line of work designs task-adaptive or dynamic connections between agents (Liu et al., 2024; Zhuge et al., 2024). Another line treats the graph itself as an optimizable or learnable object, using graph search or

graph decoders to produce task-conditioned topologies (Zhang et al., 2025b; Wu et al., 2026). Recent work further studies conditional topology design, where the communication graph adapts to changing environmental signals such as model capability, tool availability, or knowledge-source variation (Wu et al., 2026). Pruning-based methods instead reduce redundant communication by removing unnecessary agents or edges (Zhang et al., 2025a). These methods make communication more flexible than hand-written chains or fully connected graphs, but the agent nodes are still largely fixed.

Tool-augmented and skill-based agents. Tool-augmented agents extend LLMs with external executable procedures (Karpas et al., 2022; Yao et al., 2022; Schick et al., 2023; Shen et al., 2023; Li et al., 2023b; Patil et al., 2023; Qin et al., 2024). Skill-library agents further show that reusable skills can be stored and adapted across tasks (Wang et al., 2023).

3 Preliminaries

3.1 Graph-Based Multi-Agent Design

For a task query q, a graph-based MAS with K agent slots is represented as a directed graph

$$\mathcal{G}_q = (\mathcal{A}, \mathcal{E}_q),$$

where $\mathcal{A} = \{a_1, \ldots, a_K\}$ denotes agent slots and $\mathcal{E}_q$ denotes directed communication edges. Its adjacency matrix $\mathbf{A}_q \in \{0,1\}^{K\times K}$ satisfies $\mathbf{A}_q[i,j] = 1$ if messages from a_i are passed to a_j. Conventional role-based systems initialize each agent as an atomic entity,

$$a_k = \{\texttt{Base}_k, \texttt{Role}_k, \texttt{State}_k, \texttt{Plugin}_k\}, \quad (1)$$

where the role/profile and tool configuration determine the node semantics. This interface supports topology learning, but it leaves the node vocabulary closed-set. Given a generated graph $\mathcal{G}_q$, agents communicate for T rounds. At round t, an agent receives the task query and messages produced by its in-neighbors in the previous round:

$$\begin{aligned} \mathcal{R}_k^{(t)} &= a_k\left(\mathcal{P}_{k,\text{sys}}^{(t)}, \mathcal{P}_{k,\text{usr}}^{(t)}\right), \\ \mathcal{P}_{k,\text{usr}}^{(t)} &= \left\{q, \{\mathcal{R}_j^{(t-1)} : a_j \in \mathcal{N}_{\text{in}}(a_k)\}\right\}, \end{aligned} \quad (2)$$

where $\mathcal{N}_{\text{in}}(a_k)$ is the in-neighborhood of a_k and $\mathcal{R}_j^{(0)}$ is initialized as an empty message. After the final round, an aggregation function returns the system answer:

$$\hat{y}_q \leftarrow \text{Aggregate}\left(\mathcal{R}_1^{(T)}, \ldots, \mathcal{R}_K^{(T)}\right). \quad (3)$$

SIGMA keeps this graph-execution interface, summarized in Appendix B.2, while replacing atomic role nodes with skill-composed nodes.

3.2 Skill-Agent Incidence

Let $\mathcal{L} = \{s_1, \ldots, s_M\}$ be a library of reusable skills, and let $\mathcal{S}$ denote the corresponding set of skill-card nodes when we form skill-agent bipartite graphs.

Definition 1 (Skill Card)**.** A skill card is a reusable capability descriptor

$$s_m = (n_m, d_m, \eta_m, \kappa_m, g_m), \quad (4)$$

where n_m is the skill name, d_m is its natural-language description, η_m is an executable affordance such as a tool, API endpoint, retrieval operation, or reasoning procedure, κ_m is an optional input-output contract, and g_m records the grounding source. Purely cognitive skills are still instantiated as concrete prompt-level procedures or mailbox routing targets, preventing vague role labels from being treated as executable skills. Each card is embedded by a frozen encoder as $\mathbf{e}_m = \text{Enc}(s_m) \in \mathbb{R}^d$.

For a query q, SIGMA constructs the K agent slots through a skill-agent incidence matrix

$$\begin{aligned} &\mathbf{B}_q \in \{0,1\}^{M\times K}, \\ &\mathbf{B}_q[m,k] = 1 \iff s_m \mapsto a_k. \end{aligned} \quad (5)$$

The k-th column defines the executable skill bundle

$$\mathcal{L}_{q,k} = \{s_m \in \mathcal{L} \mid \mathbf{B}_q[m,k] = 1\}. \quad (6)$$

Thus, a_k is no longer a single role token; it is a task-conditioned bundle of selected capabilities.

Definition 2 (Skill-Incidence MAS)**.** Given a query q, a skill library $\mathcal{L}$, and K agent slots $\mathcal{A} = \{a_1, \ldots, a_K\}$, a skill-incidence multi-agent system is

$$\mathcal{X}_q = (\mathcal{L}, \mathcal{A}, \mathbf{B}_q, \mathbf{U}_q, \mathcal{G}_q, \mathcal{M}_q), \quad (7)$$

where $\mathbf{B}_q$ is the executable skill-agent incidence matrix, $\mathbf{U}_q$ contains bundle embeddings used by the graph decoder, $\mathcal{G}_q$ is the decoded communication graph, and $\mathcal{M}_q$ is the set of skill-specific mailboxes. A deployed skill-incidence MAS requires that each nonempty bundle is grounded in

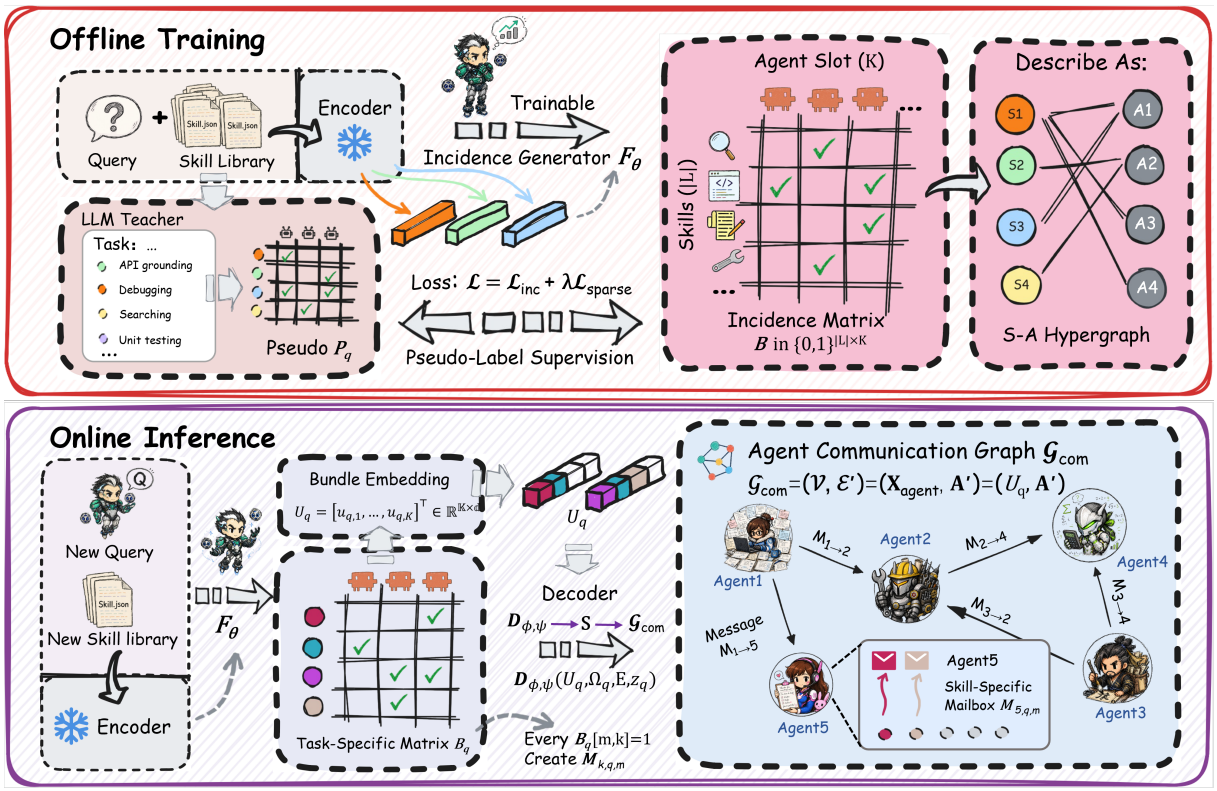


Figure 3: Framework of SIGMA. Offline, an incidence generator is trained with deterministic pseudo-label supervision to assign reusable skills to agent slots. Online, the predicted task-specific incidence matrix is converted into skill-composed agent embeddings, which are used to decode an agent communication graph with skill-specific message routing.

skill cards, each node embedding is computed from the selected cards, and every incident pair (m, k) owns an execution-time mailbox.

This separates capability composition from topology generation: $\mathbf{B}_q$ determines what each agent can do, while $\mathcal{G}_q$ determines how the constructed agents communicate. The definition describes the discrete system used at inference and execution. During training, SIGMA optimizes a continuous relaxation whose hard projection yields $\mathbf{B}_q$; this preserves differentiability without changing the executable object evaluated at test time.

4 Methodology

4.1 Overview

Figure 3 illustrates the overall framework of SIGMA. SIGMA treats reusable skills as first-class design objects. Given a query q and a skill library $\mathcal{L}$, it first predicts a skill-agent incidence structure:

$$\begin{aligned}(\mathbf{z}_q, \mathbf{E}) &= \mathrm{SkillEncode}(q, \mathcal{L}),\\ (\mathbf{P}_q, \mathbf{B}_q) &= F_\theta(\mathbf{z}_q, \mathbf{E}),\\ \mathcal{H}_q &= (\mathcal{S}, \mathcal{A}, \mathbf{B}_q).\end{aligned} \tag{8}$$

It then constructs skill-composed agents, decodes their communication topology, and executes the resulting graph:

$$\begin{aligned}(\mathbf{U}_q, \mathbf{\Omega}_q) &= \mathrm{Construct}(\mathcal{H}_q, \mathbf{z}_q, \mathbf{E}),\\ \mathcal{G}_q &= D_{\phi,\psi}(\mathbf{U}_q, \mathbf{\Omega}_q, \mathbf{E}, \mathbf{z}_q),\\ \hat{y}_q &= \mathrm{Execute}(\mathcal{G}_q, \mathbf{B}_q, q).\end{aligned} \tag{9}$$

Here $\mathbf{P}_q$ is the soft incidence matrix used for learning, $\mathbf{B}_q$ is the hard executable incidence matrix, $\mathcal{H}_q$ is the induced skill-agent bipartite graph, and $\mathbf{\Omega}_q$ stores skill-attention weights. The learnable components are the incidence generator F_θ and the skill-aware topology decoder $D_{\phi,\psi}$. The full decoding algorithm and implementation details are provided in Appendix B.

4.2 Incidence Generation and Agent Construction

Each skill card is serialized using the fields in Definition 1 and encoded together with the query by a frozen text encoder:

$$\begin{aligned}\mathbf{z}_q &= \mathrm{Enc}(q),\\ \mathbf{e}_m &= \mathrm{Enc}(s_m),\\ \mathbf{E} &= [\mathbf{e}_1, \dots, \mathbf{e}_M]^\top.\end{aligned} \tag{10}$$

The frozen encoder lets new tool/API cards enter at test time by extending $\mathcal{L}$, rather than changing the model architecture.

For each skill s_m and slot a_k, the incidence generator computes

$$\alpha_{m,k} = F_\theta([\mathbf{z}_q, \mathbf{e}_m, \mathbf{p}_k, \mathbf{z}_q \odot \mathbf{e}_m]), \tag{11}$$

where $\mathbf{p}_k$ is a learned embedding for the k-th canonical agent slot and $\odot$ denotes element-wise product. The soft incidence probability is

$$\mathbf{P}_q[m,k] = \sigma(\alpha_{m,k}). \tag{12}$$

For notational brevity, we write $p_{m,k} = \mathbf{P}_q[m,k]$ below. Soft incidence provides supervision and a differentiable training path, while the hard matrix $\mathbf{B}_q$ determines executable bundles. We obtain $\mathbf{B}_q$ through a deterministic sparse projection. For each slot k, SIGMA first forms the thresholded candidate set

$$\mathcal{C}_{q,k} = \{m : \sigma(\alpha_{m,k}) > \tau_{\text{skill}}\}. \tag{13}$$

It then selects at most k_s skills, with an argmax fallback when no skill exceeds the threshold:

$$\mathcal{I}_{q,k} = \begin{cases} \mathrm{Top}_{k_s}(\mathcal{C}_{q,k}; \alpha_{\cdot,k}), & \mathcal{C}_{q,k} \neq \emptyset, \\ \{\arg\max_m \alpha_{m,k}\}, & \text{otherwise.} \end{cases} \tag{14}$$

We set $\mathbf{B}_q[m,k] = 1$ iff $m \in \mathcal{I}_{q,k}$. If two slots produce the same bundle, the later slot is repaired by replacing the lowest-margin duplicated skill with the highest-scoring unused skill. This avoids repeated agents under a fixed crew size; Appendix B.4 gives the complete algorithm.

The hard incidence matrix defines a skill-agent bipartite graph $\mathcal{H}_q = (\mathcal{S}, \mathcal{A}, \mathbf{B}_q)$. For each slot, SIGMA computes query-conditioned attention over selected skills:

$$\beta_{m,k} = \frac{\mathbf{B}_q[m,k]\exp(\alpha_{m,k})}{\sum_{\ell=1}^{M} \mathbf{B}_q[\ell,k]\exp(\alpha_{\ell,k})}. \tag{15}$$

During training, an analogous soft support is used for differentiability. Let $\mathcal{J}_{q,k} = \mathrm{Top}_{r_c}(\{1,\ldots,M\}; \alpha_{\cdot,k})$ be the top-ranked candidate set with $r_c \geq k_s$. The soft attention is

$$\tilde{\beta}_{m,k} = \frac{\mathbf{1}[m \in \mathcal{J}_{q,k}] p_{m,k}\exp(\alpha_{m,k})}{\sum_{\ell \in \mathcal{J}_{q,k}} p_{\ell,k}\exp(\alpha_{\ell,k})}. \tag{16}$$

Below $\omega_{m,k}$ denotes $\tilde{\beta}_{m,k}$ during training and $\beta_{m,k}$ during inference, and $\mathbf{\Omega}_q = [\omega_{m,k}]$. The node embedding combines slot identity, task context, and selected skill cards:

$$\mathbf{u}_{q,k} = \mathrm{Norm}\left(\mathbf{p}_k + \mathbf{z}_q + \sum_{m=1}^{M} \omega_{m,k}\mathbf{e}_m\right). \tag{17}$$

The resulting node matrix is

$$\mathbf{U}_q = [\mathbf{u}_{q,1}, \ldots, \mathbf{u}_{q,K}]^\top \in \mathbb{R}^{K \times d}. \tag{18}$$

It is consumed by the topology decoder under the same crew size K as role-based graph baselines. Thus, a node is not a pooled role embedding; its semantics are determined by the task and by executable skills assigned to the slot.

4.3 Skill-Aware Topology Decoding

SIGMA decodes communication edges from both node-level affinity and skill-bundle compatibility. For each ordered pair (i,j), the agent-level affinity is

$$r_{ij} = \mathrm{MLP}_\phi([\mathbf{u}_{q,i}, \mathbf{u}_{q,j}, \mathbf{z}_q, \mathbf{u}_{q,i} \odot \mathbf{u}_{q,j}]). \tag{19}$$

The skill-bundle compatibility aggregates pairwise complementarity between selected skills:

$$\begin{aligned} g_{mnq} &= \mathrm{MLP}_\psi([\mathbf{e}_m, \mathbf{e}_n, \mathbf{z}_q, \mathbf{e}_m \odot \mathbf{e}_n]), \\ c_{ij} &= \sum_{m,n} \omega_{m,i}\omega_{n,j} g_{mnq}. \end{aligned} \tag{20}$$

The compatibility term captures task-specific complementarity such as retrieval skills feeding API grounding, planning skills feeding code writing, or testing skills feeding debugging. The sum is evaluated only over sparse active supports: $\mathcal{J}_{q,i} \times \mathcal{J}_{q,j}$ during training and the Cartesian product of hard-selected skills during inference. Thus, the active compatibility cost depends on bundle size rather than the full library. The final edge logit combines both terms with a weak workflow prior:

$$\ell_{ij} = r_{ij} + \lambda_c c_{ij} + \lambda_0 \mathbf{A}_0[i,j], \quad i \neq j, \tag{21}$$

where $\mathbf{A}_0$ is a simple anchor topology such as a chain, star, or dataset-provided workflow prior. Setting $\lambda_c = 0$ recovers a matched decoder that ignores explicit skill-pair compatibility. At inference time, edge probabilities are thresholded as

$$\begin{aligned} \mathcal{G}_q &= (\mathcal{A}, \mathcal{E}_q), \\ \mathcal{E}_q &= \{(a_i, a_j) : \sigma(\ell_{ij}) > \gamma_e,\ i \neq j\}. \end{aligned} \tag{22}$$

If a benchmark requires a directed acyclic schedule, a canonical slot-order mask is applied before thresholding. This decoder remains topology-compatible

with prior graph-based MAS methods because it consumes the constructed node matrix $\mathbf{U}_q$ and outputs the same type of directed communication graph.

4.4 Execution and Learning

SIGMA uses $\mathbf{B}_q$ during execution as well as representation learning. For each incident pair (m, k), the executor maintains a skill-specific mailbox; incoming messages are routed to the most relevant assigned skills using frozen embedding similarity, and only the corresponding mailbox summaries enter the agent prompt. Thus, skill incidence affects prompts, tools, memory organization, and message routing instead of remaining a hidden variable. Formally, for each incident pair (m, k) with $\mathbf{B}_q[m, k] = 1$, SIGMA creates

$$\mathcal{M}_{q,k,m}^{(t)} = \{r : r \to (a_k, s_m),\ \mathrm{time}(r) < t\}, \tag{23}$$

where $r \to (a_k, s_m)$ indicates that message r is routed to skill s_m inside slot a_k. A frozen router scores incoming message r against each incident skill by

$$\gamma_m(r) = \mathrm{sim}(\mathrm{Enc}(r), \mathbf{e}_m), \tag{24}$$

and routes it to the top-r_s assigned skills. At round t, the system prompt contains the selected skill cards and state, while the user prompt contains the task and mailbox summaries:

$$\begin{aligned} \mathbf{s}_{q,k}^{(t)} &= \mathrm{Summarize}(\mathcal{M}_{q,k}^{(t)}), \\ \mathcal{P}_{k,\mathrm{sys}}^{(t)} &= \{\mathcal{L}_{q,k}, \mathtt{State}_k^{(t)}\}, \\ \mathcal{P}_{k,\mathrm{usr}}^{(t)} &= \left\{q, \mathbf{s}_{q,k}^{(t)}\right\}. \end{aligned} \tag{25}$$

Each mailbox summary is capped under the same total context budget used by the baselines. Appendix C gives the full prompt format and fallback routing.

Training uses pseudo-labels $\mathbf{B}_q^\star$ built only from training-split role descriptions, tool/API documents, and execution traces. For each training slot, a deterministic labeler retrieves candidate skills and selects at most k_s cards, yielding canonical incidence columns without using held-out queries, answers, traces, or inserted tool/API cards. The incidence generator is trained with binary cross-entropy:

$$\mathcal{L}_{\mathrm{inc}} = \sum_q \sum_{m=1}^{M} \sum_{k=1}^{K} \mathrm{BCE}\left(\mathbf{B}_q^\star[m, k], p_{m,k}\right). \tag{26}$$

When annotated communication graphs are available, the topology decoder is trained with an edge reconstruction loss:

$$\mathcal{L}_{\mathrm{edge}} = \sum_q \sum_{i \neq j} \mathrm{BCE}(\mathbf{A}_q^\star[i, j], \sigma(\ell_{ij})). \tag{27}$$

In our benchmark setting, datasets do not provide ground-truth communication edges; therefore $\mathbf{A}_q^\star$ is instantiated from the same weak anchor topology $\mathbf{A}_0$ used across controlled graph baselines, rather than from held-out answers or evaluation traces. We further use a soft bundle-size cap and graph sparsity penalty,

$$\begin{aligned} \mathcal{L}_{\mathrm{sparse}} = &\sum_q \sum_{k=1}^{K} \max\left(0, \sum_{m=1}^{M} p_{m,k} - k_s\right) \\ &+ \lambda_g \sum_q \sum_{i \neq j} \sigma(\ell_{ij}). \end{aligned} \tag{28}$$

The final objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{inc}} + \lambda_e \mathcal{L}_{\mathrm{edge}} + \lambda_s \mathcal{L}_{\mathrm{sparse}}. \tag{29}$$

If annotated target graphs are unavailable, $\mathcal{L}_{\mathrm{edge}}$ is computed against this shared structural prior or omitted in the matched-decoder control. Pseudo-label construction and complexity analysis are detailed in Appendix D and Appendix F. At test time, SIGMA can use the frozen training library or append new cards derived from held-out tool/API documents, because new skills enter through the same text-card encoder.

5 Experiments

5.1 Setup

Benchmarks. We evaluate SIGMA on six representative benchmarks covering code generation, general knowledge, mathematical reasoning, and symbolic problem solving: ❶ **Code Generation**: HumanEval (Chen et al., 2021); ❷ **General Reasoning**: MMLU (Hendrycks et al., 2020); ❸ **Mathematical Reasoning**: GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MultiArith (Roy and Roth, 2015), and AQuA (Ling et al., 2017). For all benchmarks, we report task accuracy as the primary metric. To examine whether the proposed method is robust across model scales and families, we conduct experiments with three base LLMs: *Qwen3-8B* (Yang et al., 2025), *GPT-OSS-120B* (Agarwal et al., 2025), and *GPT-4o-mini* (Hurst et al., 2024). The average score across all benchmarks is also

Table 1: Performance comparison with three different base LLMs. **Mul.**, **Topo.**, and **Comp.** indicate whether the method supports multi-agent execution, communication-topology design, and skill-based composition, respectively. Dark blue cells denote the **best** and light blue cells **second-best** results. Cross, mixed check–cross, and check marks indicate no, partial, and full support.

Method	Mul.	Topo.	Comp.	MMLU	GSM8K	MultiArith	SVAMP	AQuA	HumanEval	Avg.
					Qwen3-8B					
Vanilla	✗	✗	✗	57.71	80.76	87.37	96.68	35.83	82.23	73.43
CoT	✗	✗	✗	58.74 ↑1.03	90.75 ↑9.99	87.83 ↑0.46	97.01 ↑0.33	81.10 ↑45.27	82.85 ↑0.62	83.05 ↑9.62
LLM-Debate	✓	✗	✗	59.61 ↑1.90	90.12 ↑9.36	88.46 ↑1.09	97.62 ↑0.94	81.14 ↑45.31	83.85 ↑1.62	83.47 ↑10.04
GPTSwarm	✓	✓̸	✗	59.58 ↑1.87	89.78 ↑9.02	90.27 ↑2.90	97.85 ↑1.17	81.52 ↑45.69	84.14 ↑1.91	83.86 ↑10.43
G-Designer	✓	✓	✗	60.07 ↑2.36	90.85 ↑10.09	90.14 ↑2.77	**98.45** ↑1.77	82.08 ↑46.25	84.73 ↑2.50	84.39 ↑10.96
ARG-Designer	✓	✓	✗	**60.92** ↑3.21	91.14 ↑10.38	89.38 ↑2.01	98.21 ↑1.53	**82.47** ↑46.64	85.47 ↑3.24	84.60 ↑11.17
CARD	✓	✓	✗	60.13 ↑2.42	**91.87** ↑11.11	**90.54** ↑3.17	98.13 ↑1.45	81.92 ↑46.09	**87.28** ↑5.05	**84.98** ↑11.55
SIGMA	✓	✓	✓	**61.44** ↑3.73	**91.56** ↑10.80	**98.39** ↑11.02	**98.68** ↑2.00	**83.46** ↑47.63	**88.71** ↑6.48	**87.04** ↑13.61
					GPT-OSS-120B					
Vanilla	✗	✗	✗	81.58	84.49	95.90	95.07	74.77	90.04	86.98
CoT	✗	✗	✗	83.39 ↑1.81	84.81 ↑0.32	95.63 ↓0.27	95.19 ↑0.12	76.02 ↑1.25	90.87 ↑0.83	87.65 ↑0.68
LLM-Debate	✓	✗	✗	82.92 ↑1.34	85.53 ↑1.04	95.72 ↓0.18	94.89 ↓0.18	75.38 ↑0.61	91.43 ↑1.39	87.65 ↑0.67
GPTSwarm	✓	✓̸	✗	83.59 ↑2.01	86.88 ↑2.39	96.50 ↑0.60	94.77 ↓0.30	77.41 ↑2.64	92.30 ↑2.26	88.58 ↑1.60
G-Designer	✓	✓	✗	85.06 ↑3.48	88.62 ↑4.13	96.86 ↑0.96	95.21 ↑0.14	78.13 ↑3.36	92.88 ↑2.84	89.46 ↑2.49
ARG-Designer	✓	✓	✗	85.63 ↑4.05	91.08 ↑6.59	**98.67** ↑2.77	94.81 ↓0.26	78.62 ↑3.85	93.47 ↑3.43	90.38 ↑3.41
CARD	✓	✓	✗	**85.78** ↑4.20	**93.44** ↑8.95	98.21 ↑2.31	**95.48** ↑0.41	**82.76** ↑7.99	**93.98** ↑3.94	**91.61** ↑4.63
SIGMA	✓	✓	✓	**87.28** ↑5.70	**96.25** ↑11.76	**98.34** ↑2.44	**95.94** ↑0.87	**90.16** ↑15.39	**95.83** ↑5.79	**93.97** ↑6.99
					GPT-4o-mini					
Vanilla	✗	✗	✗	67.61	85.33	93.94	86.12	69.80	77.29	80.02
CoT	✗	✗	✗	68.84 ↑1.23	86.51 ↑1.18	94.46 ↑0.52	86.89 ↑0.77	71.85 ↑2.05	78.64 ↑1.35	81.20 ↑1.18
LLM-Debate	✓	✗	✗	69.66 ↑2.05	87.29 ↑1.96	94.97 ↑1.03	87.67 ↑1.55	72.93 ↑3.13	79.48 ↑2.19	82.00 ↑1.99
GPTSwarm	✓	✓̸	✗	71.40 ↑3.79	88.88 ↑3.55	96.01 ↑2.07	89.21 ↑3.09	75.06 ↑5.26	81.39 ↑4.10	83.66 ↑3.64
G-Designer	✓	✓	✗	73.19 ↑5.58	90.46 ↑5.13	96.78 ↑2.84	90.37 ↑4.25	76.58 ↑6.78	82.67 ↑5.38	85.01 ↑4.99
ARG-Designer	✓	✓	✗	74.23 ↑6.62	91.84 ↑6.51	**97.56** ↑3.62	91.53 ↑5.41	78.16 ↑8.36	84.54 ↑7.25	86.31 ↑6.30
CARD	✓	✓	✗	**75.07** ↑7.46	**93.13** ↑7.80	96.94 ↑3.00	**91.62** ↑5.50	**80.40** ↑10.60	**85.11** ↑7.82	**87.05** ↑7.03
SIGMA	✓	✓	✓	**76.47** ↑8.86	**93.20** ↑7.87	**99.11** ↑5.17	**93.85** ↑7.73	**83.86** ↑14.06	**86.29** ↑9.00	**88.80** ↑8.78

reported as an overall measure of general task-solving ability. All methods use the same benchmark splits, deterministic decoding, and dataset-specific answer extractor; Appendix H.2 reports the concrete Vanilla and CoT prompt templates used in the evaluation. For MMLU, we follow the cost-controlled evaluation protocol used by G-Designer and ARG-Designer: all methods are evaluated on the same fixed 153-question shuffled subset, while training/optimization uses separate examples and never uses held-out answers.

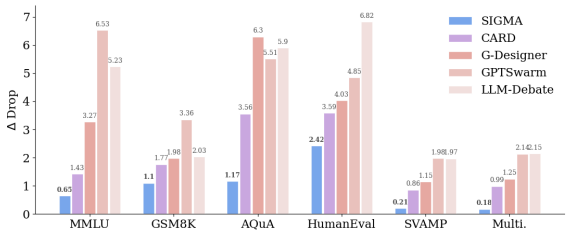


Figure 4: Performance drop from source skill libraries to unseen skill libraries. Lower values indicate stronger robustness to library changes.

Baselines. We compare SIGMA with both single-agent and multi-agent baselines. For single-agent settings, we use **Vanilla** prompting and **CoT** (Wei et al., 2022). For multi-agent settings, we include **LLM-Debate** (Du et al., 2024), **GPTSwarm** (Zhuge et al., 2024), **ARG-Designer** (Li et al., 2026), CARD (Wu et al., 2026), and G-Designer (Zhang et al., 2025b). These baselines evaluate whether SIGMA's gains come from skill-composed node construction rather than stronger prompting, additional agents, or topology optimization alone. Table 6 also displays more training and experiment details of SIGMA.

5.2 Main Results

Superior Overall Performance. The results in Table 1 show that SIGMA consistently improves multi-agent collaboration across different model scales and task domains. Across *Qwen3-8B*, *GPT-OSS-120B*, and *GPT-4o-mini*, SIGMA achieves the highest average scores of 87.04, 93.97, and 88.80, respectively. Compared with CARD, the strongest non-compositional topology-based baseline in terms of average performance, SIGMA

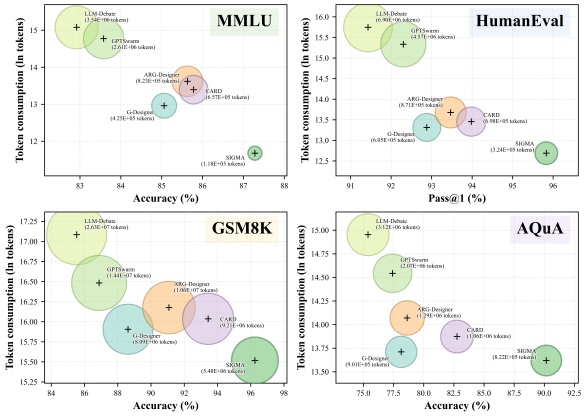


Figure 5: Visualization of the accuracy and token trade-off across benchmarks.

brings consistent average improvements of 2.06↑, 2.36↑, and 1.75↑ points. It ranks first on 16 of the 18 benchmark–model combinations, with especially clear gains on AQuA and HumanEval, where structured reasoning and executable construction matter. These results indicate that the gains of SIGMA are robust across baselines and benchmarks. To further isolate whether the improvements come merely from exposing task-relevant skill cards to the model, we also introduce SINGLE-AGENT+SKILLS, a controlled baseline that receives the same union of selected skill cards as SIGMA but removes multi-agent execution, topology decoding, and mailbox routing. As a controlled baseline, SINGLE-AGENT+SKILLS improves over CoT but remains below SIGMA on average and on most datasets in Appendix Table 9, suggesting that the gains come not only from exposing skill cards, but also from topology-aware multi-agent execution and mailbox routing.

Generalization to unfamiliar skill libraries. When the source skill library is replaced with unseen skill cards at test time, SIGMA suffers the smallest average degradation. As shown in Table 14 and Figure 4, its performance drops by only 0.96 points, lower than CARD (2.03), `G-Designer` (3.00), GPTSwarm (4.06), and LLM-Debate (4.02). This robustness is especially evident on MMLU, AQuA, and HumanEval, where topology-centric baselines exhibit much larger performance drops under library shifts. The result indicates that skill incidence can compose unseen cards into agent slots instead of relying on a fixed capability inventory.

Efficiency. Figure 5 shows that SIGMA reaches the best accuracy with the fewest tokens among compared multi-agent methods on MMLU, HumanEval, GSM8K, and AQuA. Compared with GPTSwarm, it reduces token consumption by 95.5%, 61.9%, 92.9%, and 60.3% on these benchmarks, respectively; compared with LLM-Debate, the maximum reduction reaches 96.7%. Thus, SIGMA improves collaboration efficiency by composing task-specific agents rather than increasing communication volume; detailed statistics are in Appendix G.2.

5.3 Ablation Study

Table 2 ablates the main components of SIGMA. ■***w/o Inc.*** replaces the learned skill-agent incidence predictor with flat skill assignment. ■***w/o Skill Dec.***, ■***w/o Mailbox***, and ■***w/o Anchor Prior*** remove skill-aware decoding, mailbox routing, and the weak workflow prior, respectively. The consistent drops show that SIGMA's generalization comes from the joint effect of compositional node construction, skill-aware topology decoding, and executable skill routing.

Table 2: Ablation study under source and unseen skill libraries, using *GPT-OSS-120B*

Variant	MMLU		HumanEval	
	Source	Unseen	Source	Unseen
SIGMA	**85.62**	**84.31**	**95.97**	**95.16**
w/o Inc.	80.39	80.39	91.23	90.77
w/o Skill Dec.	83.01	82.35	92.33	92.41
w/o Mailbox	81.04	80.39	93.07	92.18
w/o Anchor Prior	83.01	83.66	93.74	92.93

6 Conclusion

We presented SIGMA, a skill-incidence framework for compositional multi-agent design. Rather than treating agents as fixed role nodes, SIGMA constructs each agent as a task-conditioned bundle of reusable skills and decodes communication over these skill-composed nodes. Skill-specific mailboxes further make the incidence structure executable during multi-agent interaction. By separating capability composition from topology generation, SIGMA enables fixed agent slots to express diverse task-specific identities and generalize to unseen skill combinations. Our results highlight that effective multi-agent design should consider not only how agents communicate, but also how their internal capabilities are composed.

Limitations

We note several limitations. ❶ SIGMA depends on the quality and coverage of the skill library, making noisy or incomplete skill cards a potential source of error. ❷ Its deterministic pseudo-labels and canonical slot ordering may inherit biases from labeling heuristics. ❸ The current framework assumes a fixed number of agent slots, leaving dynamic crew-size prediction as future work. ❹ Our evaluation focuses on benchmark-style reasoning and coding tasks, while real-world tool-use environments may involve noisier APIs, longer horizons, and skills with side effects. ❺ Finally, although skill-specific mailboxes make skill incidence executable, adaptive routing and conflict resolution remain open directions.

SIGMA does not attempt to solve the globally optimal skill-agent assignment problem by exhaustive search. Instead, it learns a task-conditioned incidence generator and applies a sparse deterministic projection, trading global optimality guarantees for scalability under a fixed bundle-size budget.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, and 1 others. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, and 1 others. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. 2026. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 23142–23150.

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.

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.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 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.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1743–1752.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.

Tongtong Wu, Yanming Li, Ziye Tang, Chen Jiang, Linhao Luo, Guilin Qi, Shirui Pan, and Gholamreza Haffari. 2026. Card: Towards conditional design of multi-agent topological structures. *arXiv preprint arXiv:2603.01089*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

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*.

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 *International Conference on Machine Learning*, pages 76678–76692. PMLR.

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

A Appendix Roadmap

This appendix provides additional details for SIGMA, including method design, skill-card serialization, incidence decoding, mailbox execution, pseudo-label construction, evaluation protocols, implementation settings, and additional robustness results.

Appendix Roadmap

Appendix B: Method Details
Additional method details, including graph execution, skill-card serialization, hard incidence decoding, continuous relaxation, and end-to-end inference.

Appendix C: Mailbox Execution
Skill-mailbox execution, mailbox summary format, and fallback routing.

Appendix D: Pseudo Labels
Deterministic pseudo-label construction and canonical slot ordering.

Appendix E: Experimental Protocol
Matched-crew evaluation, held-out skill composition, test-time skill insertion, baselines, and metrics.

Appendix F: Implementation Details
Hyperparameters, training configuration, training objective, and complexity analysis.

Appendix G: Additional Results
Additional results, including robustness to unfamiliar skill libraries.

Appendix H: Prompt Templates
Prompt templates for SIGMA agent execution, Vanilla/CoT baselines, and dataset-specific answer extraction.

Appendix I: Additional Details
More details about datasets, skill libraries, and models used in experiments.

Appendix J: Failure Cases
Analysis of some failure cases during experiments.

Appendix K: Impact Statement
Impact of SIGMA.

Appendix L: Case Study
Detailed, card-style replay of representative HumanEval cases from the raw execution logs.

Table 3: Operational checklist for the SI-MAP protocol. Each check corresponds to a required property of executable skill-incidence based multi-agent design, with its concrete realization in SIGMA summarized on the right.

Property	Instantiation in SIGMA
Compositionality	Builds each agent slot from a task-conditioned bundle of reusable skill cards via the incidence matrix B_q.
Executability	Exposes assigned skills through prompts, tool affordances, retrieval procedures, and skill-specific mailboxes.
Topology Compatibility	Feeds skill-composed agent embeddings U_q into the same graph decoder used for communication topology generation.
Expandability	Allows new tool/API skill cards to be appended at test time and encoded by the frozen skill encoder.

B Additional Method Details

B.1 SI-MAP Design Checklist

The Skill-Incidence Multi-Agent Protocol (SI-MAP) requires that a graph-based multi-agent designer satisfies four operational properties. Table 3 summarizes how SIGMA instantiates these properties.

B.2 Graph Execution Interface

The main text keeps only the compact graph definition. For completeness, we state the generic multi-agent execution interface used by SIGMA and the baselines. Given a generated graph $\mathcal{G}_q$, agents communicate for T rounds. At round t, agent a_k receives the task query and messages produced by its in-neighbors:

$$\begin{aligned} \mathcal{R}_k^{(t)} &= a_k\left(\mathcal{P}_{k,\text{sys}}^{(t)}, \mathcal{P}_{k,\text{usr}}^{(t)}\right), \\ \mathcal{P}_{k,\text{usr}}^{(t)} &= \left\{q, \{\mathcal{R}_j^{(t-1)} : a_j \in \mathcal{N}_{\text{in}}(a_k)\}\right\}, \end{aligned} \tag{30}$$

where $\mathcal{N}_{\text{in}}(a_k)$ is the in-neighborhood of a_k and $\mathcal{R}_j^{(0)}$ is initialized as an empty message. After the final round, an aggregation function returns the system answer:

$$\hat{y}_q \leftarrow \text{Aggregate}\left(\mathcal{R}_1^{(T)}, \ldots, \mathcal{R}_K^{(T)}\right). \tag{31}$$

SIGMA keeps this interface but replaces atomic role nodes with skill-composed nodes and routes messages through skill-specific mailboxes.

B.3 Skill Card Serialization

Each skill card is serialized into a fixed textual template before being encoded. We use the same template for training, validation, and test-time skill insertion.

SKILL CARD TEMPLATE `tag`

```
Skill_id      "{unique skill identifier}"
Name          "{skill name}"
Description   "{capability description}"
Benchmark     "{target benchmark or task
              domain}"
Inputs        ["{input field 1}", "..."]
Outputs       ["{output field 1}", "..."]
Tools         ["{optional tool/API/retrieval
              resource}", "..."]
Tags          ["{retrieval tag 1}", "..."]
```

SKILL CARD EXAMPLE I `mmlu`

```
Skill_id      "option_elimination"
Name          "Option elimination"
Description   Rule out implausible
              multiple-choice answers before
              selecting the final answer.
Benchmark     "MMLU"
Inputs        ["query", "messages"]
Outputs       ["analysis", "answer"]
Tools         []
Tags          ["mmlu", "multiple-choice",
              "elimination"]
```

SKILL CARD EXAMPLE III `aqua`

```
Skill_id      "synthesize"
Name          "Synthesize"
Description   Produce the final response by
              consolidating the reasoning
              trajectory into a concise
              answer.
Benchmark     "AQuA"
Inputs        ["query", "messages"]
Outputs       ["analysis", "answer"]
Tools         []
Tags          ["aqua", "synthesis",
              "final-answer"]
```

SKILL CARD EXAMPLE III `humaneval`

```
Skill_id      "algorithm_design"
Name          "Algorithm design"
Description   Choose an algorithmic strategy
              and complexity target before
              implementing the solution.
Benchmark     "HumanEval"
Inputs        ["query", "messages"]
Outputs       ["analysis", "answer"]
Tools         []
Tags          ["humaneval", "algorithm",
              "complexity"]
```

For tool or API skills, the executable affordance field contains the callable endpoint, retrieval operation, or tool-use instruction. For cognitive skills, the affordance is instantiated as a concrete prompt-level procedure or mailbox-routing target rather than a vague role description. This prevents purely latent role labels from being treated as reusable executable skills.

B.4 Hard Incidence Decoding

The incidence generator predicts a soft assignment probability $P_q[m, k]$ for every skill-agent pair. The executable incidence matrix B_q is obtained by deterministic sparse projection. For each agent slot k, we first keep skills whose probability exceeds τ_{skill} and then retain at most k_s skills. If no skill exceeds the threshold, the highest-scoring skill is used as an argmax fallback. To avoid repeated agent identities under a fixed crew size, duplicated bundles are repaired by replacing the lowest-margin duplicated skill with the highest-scoring unused skill.

Algorithm 1 SIGMA incidence decoding

Require: Query embedding z_q, skill embeddings E, slot embeddings $\{p_k\}_{k=1}^{K}$, incidence generator F_θ, threshold τ_{skill}, bundle size k_s
Ensure: Hard incidence matrix $B_q \in \{0,1\}^{M\times K}$

```
Initialize B_q ← 0
for k = 1 to K do
    for m = 1 to M do
        α_{m,k} ← F_θ([z_q, e_m, p_k, z_q ⊙ e_m])
        p_{m,k} ← σ(α_{m,k})
    end for
    C_{q,k} ← {m : p_{m,k} > τ_skill}
    if C_{q,k} ≠ ∅ then
        I_{q,k} ← Top_{k_s}(C_{q,k}; α_{·,k})
    else
        I_{q,k} ← {arg max_m α_{m,k}}
    end if
    for m ∈ I_{q,k} do
        B_q[m, k] ← 1
    end for
end for
Repair duplicated columns of B_q by replacing the lowest-
margin duplicated skill with the best unused skill for the
affected slot.
return B_q
```

B.5 Continuous Relaxation and Bundle Attention

During training, SIGMA uses a sparse continuous relaxation aligned with the hard bundle. For each slot k, let $\mathcal{J}_{q,k} = \text{Top}_{r_c}(\{1,\dots,M\}; \alpha_{\cdot,k})$ be the top-ranked candidate support with $r_c \geq k_s$. The soft bundle attention is

$$\tilde{\beta}_{m,k} = \frac{\mathbf{1}[m \in \mathcal{J}_{q,k}] p_{m,k} \exp(\alpha_{m,k})}{\sum_{\ell \in \mathcal{J}_{q,k}} p_{\ell,k} \exp(\alpha_{\ell,k})}, \tag{32}$$

where $p_{m,k} = \sigma(\alpha_{m,k})$. We use $\tilde{\beta}_{m,k}$ during differentiable training and the hard-selected attention

$\beta_{m,k}$ during inference. The same sparse support is used when computing skill-bundle compatibility, so pairwise skill interactions are evaluated over $\mathcal{J}_{q,i} \times \mathcal{J}_{q,j}$ during training and over hard-selected skill products during inference.

B.6 End-to-End Inference Procedure

Algorithm 2 gives the complete inference procedure. SIGMA first constructs the task-specific skill-agent incidence matrix, builds incidence-aware agent embeddings, decodes the communication graph, and then executes the graph with skill-specific mailboxes.

Algorithm 2 SIGMA inference

Require: Query q, skill library $\mathcal{L}$, number of agent slots K, encoder Enc, incidence generator F_θ, topology decoder D_ϕ
Ensure: Final answer $\hat{y}_q$
$z_q \leftarrow \mathrm{Enc}(q)$
$e_m \leftarrow \mathrm{Enc}(s_m)$ for each $s_m \in \mathcal{L}$
$B_q \leftarrow \textsc{IncidenceDecode}(z_q, E, F_\theta)$
for $k = 1$ to K **do**
  Compute skill attention $\beta_{m,k}$ over selected skills with $B_q[m,k] = 1$
  $u_{q,k} \leftarrow \mathrm{Norm}(p_k + z_q + \sum_m \beta_{m,k} e_m)$
end for
$U_q \leftarrow [u_{q,1}, \ldots, u_{q,K}]^\top$
for each ordered pair (i,j) with $i \neq j$ **do**
  Compute agent affinity r_{ij}
  Compute skill-bundle compatibility c_{ij}
  $\ell_{ij} \leftarrow r_{ij} + \lambda_c c_{ij} + \lambda_0 A_0[i,j]$
end for
$E_q \leftarrow \{(a_i, a_j) : \sigma(\ell_{ij}) > \gamma_e, i \neq j\}$
Create one mailbox $M_{q,k,m}$ for every incident pair (m,k) with $B_q[m,k] = 1$
for $t = 1$ to T **do**
  for each agent a_k in the communication schedule **do**
    Route incoming messages to the top-r_s incident skill mailboxes.
    Build the system prompt from selected skill cards and state.
    Build the user prompt from q and mailbox summaries.
    Generate response $R_k^{(t)}$.
  end for
end for
$\hat{y}_q \leftarrow \mathrm{Aggregate}(R_1^{(T)}, \ldots, R_K^{(T)})$
return $\hat{y}_q$

C Skill-Mailbox Execution

The mailbox mechanism makes the incidence matrix executable during multi-agent interaction. For each incident pair (s_m, a_k), SIGMA maintains a skill-specific mailbox $M_{q,k,m}$. When a message arrives at agent a_k, the router compares the message embedding with the embeddings of skills assigned to a_k and routes the message to the top-r_s relevant mailboxes. The agent prompt then receives compact summaries of these mailboxes instead of a flattened context containing all messages.

Formally, for every incident pair (m,k) with $B_q[m,k] = 1$, the mailbox at round t is

$$\mathcal{M}_{q,k,m}^{(t)} = \{r : r \to (a_k, s_m),\ \mathrm{time}(r) < t\}, \tag{33}$$

where $r \to (a_k, s_m)$ means that message r is routed to skill s_m inside agent a_k. A frozen router scores each incident skill by

$$\gamma_m(r) = \mathrm{sim}(\mathrm{Enc}(r), \mathbf{e}_m), \tag{34}$$

and routes r to the top-r_s skills among those assigned to a_k. At round t, the prompt is constructed as

$$\begin{aligned} \mathbf{s}_{q,k}^{(t)} &= \mathrm{Summarize}(\mathcal{M}_{q,k}^{(t)}), \\ \mathcal{P}_{k,\mathrm{sys}}^{(t)} &= \{\mathcal{L}_{q,k}, \mathtt{State}_k^{(t)}\}, \\ \mathcal{P}_{k,\mathrm{usr}}^{(t)} &= \left\{q, \mathbf{s}_{q,k}^{(t)}\right\}. \end{aligned} \tag{35}$$

Each mailbox summary is capped at b tokens, and all baselines are evaluated under the same total context budget.

C.1 Mailbox Summary Format

We use a fixed summary format for each mailbox:

SIGMA MAILBOX `routed evidence`

```
Box        Mailbox[{skill}]
Src        slot {id}, sim={score}
Signal     {answer / evidence / tool output}
Summary    {routed message summary}
Hint       {next-step usage}
```

SIGMA MAILBOX EXAMPLE I `problem_decomposition`

```
Box        Mailbox[problem_decomposition]
Src        slot 0, sim=0.087
Signal     answer = C
Summary    The predecessor agent identifies
           option C as the best answer. Its
           analysis suggests that the
           question asks for a skill that is
           not central to planning. It
           contrasts conceptual, analytical,
           and communication skills with IT
           and computing skills, arguing
           that the latter are useful but
           less fundamental in the planning
           context.
Hint       Use this routed message to
           decompose the decision into core
           planning skills and non-essential
           supporting skills.
```

The summary budget is capped by b tokens per mailbox. Baselines are given the same total context

budget, so the mailbox design changes only message organization rather than increasing available context.

SIGMA MAILBOX EXAMPLE II `elimination`

```
Box        Mailbox[elimination]
Src        slot 0, sim=0.102
Signal     answer = C
Summary    The predecessor agent selects
           option C and supports it by
           eliminating less plausible
           alternatives. The analysis argues
           that IT and computing skills,
           although useful, are not the most
           fundamental management skills for
           effective planning.
Hint       Use this routed evidence to rule
           out options describing useful but
           non-core planning skills, while
           keeping option C as the strongest
           candidate.
```

C.2 Fallback Routing

If all router scores are low or the selected agent has only one assigned skill, the message is routed to the highest-scoring incident skill. In implementation, we also keep a lightweight general state for each agent slot, but only skill-specific mailbox summaries are exposed as capability-grounded execution context. This fallback prevents message loss while preserving the operational meaning of the incidence matrix.

D Pseudo-Label Construction

D.1 Training Skill Library

The training skill library $\mathcal{L}_{\text{train}}$ is constructed only from training-split sources, including role descriptions, tool/API documents, retrieval procedures, and execution traces. Held-out queries, held-out answers, held-out execution traces, and held-out tool/API cards are not used during pseudo-label construction.

D.2 Deterministic Incidence Pseudo-Labels

For each training task, we construct a deterministic pseudo-label incidence matrix $B_q^\star$ using `teacher_light_label`. These labels are not human annotations and are not generated by an LLM. The labeler retrieves the top $R = 8$ candidate skills from $\mathcal{L}_{\text{train}}$ and assigns at most $k_s = 3$ skills to each of the $K = 5$ agent slots. The selected skills define the columns of $B_q^\star$ and provide supervision for the incidence predictor.

The label files do not contain ground-truth communication edges. Therefore, when edge supervision is needed during training, we use a chain prior as structural supervision rather than assuming annotated edge labels.

PSEUDO-LABEL EXAMPLE

```
{
  "id": "mmlu-val-abstract_algebra-0",
  "query": "The cyclic subgroup of Z_24
    generated by 18 has order
    Option A: 4
    Option B: 8
    Option C: 12
    Option D: ...",
  "S": [
    ["analogy", "counterexample", "
    domain_translation"],
    ["counterexample", "domain_translation
    ", "problem_decomposition"]
  ],
  "B": [
    [0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 1, 1, 1],
    [1, 1, 0, 0, 0],
    [0, 0, 0, 1, 1],
    [0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0]
  ]
}
```

Figure 6: Example pseudo-label for skill-agent incidence supervision. The field `S` records task-relevant skill groups, while `B` is a binary incidence matrix indicating which skills are assigned to each agent slot.

D.3 Pseudo-Label Format

The deterministic pseudo-labeler outputs selected skill names for each slot. The output is stored in a lightweight JSON-style format:

PSEUDO-LABEL TEMPLATE `incidence supervision`

```
Id         {benchmark split and example id}
Query      {task input or question text}
S          {task-relevant skill groups or
           candidate skill sets}
B          {binary skill-agent incidence
           matrix, where B_{i,k} = 1 assigns
           skill i to agent slot k}
```

Here is an example: Figure 6

D.4 Canonical Slot Ordering

Pseudo-label columns follow the canonical slot order used by the training pipeline. This avoids introducing an additional bipartite matching objective

between predicted slots and target agents. When multiple assignments are equivalent, ties are resolved deterministically by the labeler.

E Experimental Protocol

E.1 Matched-Crew Evaluation

All compared methods use the same number of agent slots K, the same communication-round budget T, and the same total context budget. This ensures that the comparison isolates how agent nodes are constructed and how messages are routed, rather than allowing gains from larger crews or longer contexts.

E.2 Held-Out Skill-Composition Setting

The held-out skill-composition setting evaluates whether a method can solve tasks requiring unseen combinations of skills. Training tasks and test tasks may share individual skill cards, but the specific multi-skill bundles required at test time are held out from training. This setting measures compositional generalization over reusable capabilities. As shown in I.4

E.3 Test-Time Skill-Insertion Setting

The test-time skill-insertion setting evaluates whether a trained designer can use newly appended tool/API skills without retraining the topology decoder. At test time, new skill cards are serialized with the same template as training cards and encoded by the frozen encoder. The incidence generator and skill-aware topology decoder then operate over the expanded library $\mathcal{L}_{\text{train}} \cup \mathcal{L}_{\text{new}}$.

A method succeeds in this setting only if it can assign the newly inserted skills to appropriate agent slots and use them during execution. This distinguishes genuine expandability from merely memorizing training-time role or tool configurations.

E.4 Baselines and Controlled Variants

We compare SIGMA with role-based, topology-based, and skill-based controls. The key controlled variants are summarized in Table 4.

E.5 Evaluation Metrics

The primary metric is downstream task accuracy under the same evaluator used by each benchmark. We also report the source-to-unseen performance drop:

$$\Delta = \text{Acc}_{\text{source}} - \text{Acc}_{\text{unseen}}. \tag{36}$$

Table 4: Baselines and controlled variants used in our experimental protocol.

Category	Method	Controlled Design Factor
Multi-agent baselines	LLM-Debate	Uses role-specialized agents for argument exchange, without explicit reusable skill assignment.
Topology baselines	GPTSwarm	Optimizes communication structures over predefined agents, but does not predict skill-agent incidence.
	`G-Designer`	Generates task-adaptive topologies over fixed agent nodes, isolating topology design from capability composition.
Skill assignment variants	Flat-skill designer	Retrieves a global skill set for the whole team without assigning skills to individual agent slots.
	Embedding-only incidence	Assigns skills using frozen embedding similarity, removing the learned incidence generator.
Execution variant	SIGMA w/o mailbox	Keeps predicted skill assignments but flattens incoming messages into a shared agent context.
Upper-bound diagnostic	Oracle incidence	Uses gold or pseudo-label incidence assignments at test time; included only as a diagnostic upper bound.

A smaller Δ indicates stronger robustness when the available skill library changes.

When target incidence labels are available, we report micro-averaged precision, recall, and F1 over skill-agent assignments. Let T_{inc}, P_{inc}, and G_{inc} denote the true-positive, predicted-positive, and gold-positive incidence counts over all (q, m, k), respectively:

$$\text{Prec}_{\text{inc}} = T_{\text{inc}}/P_{\text{inc}}, \tag{37}$$

$$\text{Rec}_{\text{inc}} = T_{\text{inc}}/G_{\text{inc}}, \tag{38}$$

$$\text{F1}_{\text{inc}} = 2/(\text{Prec}_{\text{inc}}^{-1} + \text{Rec}_{\text{inc}}^{-1}). \tag{39}$$

When annotated communication graphs are available, we report edge precision, recall, and F1 over directed edges:

$$\text{Prec}_{\text{edge}} = \frac{|E_q \cap E_q^\star|}{|E_q|}, \tag{40}$$

$$\text{Rec}_{\text{edge}} = \frac{|E_q \cap E_q^\star|}{|E_q^\star|}, \tag{41}$$

$$\text{F1}_{\text{edge}} = \frac{2 \cdot \text{Prec}_{\text{edge}} \cdot \text{Rec}_{\text{edge}}}{\text{Prec}_{\text{edge}} + \text{Rec}_{\text{edge}}}. \tag{42}$$

Table 5: Main hyperparameters used by SIGMA.

Symbol	Meaning	Setting
K	Number of agent slots	5
T	Communication rounds	Matched to benchmark budget
τ_{skill}	Skill selection threshold	0.25
k_s	Maximum skills per agent slot	3
R	Candidate support size for pseudo-labeling	8
r_s	Routed skills per incoming message	2
γ_e	Edge prediction threshold	0.5
b	Mailbox summary budget	Fixed under total context budget
λ_e	Edge loss weight	1.0
λ_c	Compatibility weight	1.0
λ_g	Graph sparsity weight	1×10^{-3}

Table 6: Core training configuration of SIGMA.

Item	Setting
Training target	Joint training of F_θ and $D_{\phi,\psi}$
Optimizer	AdamW
Epochs	50 for MMLU experiments
Learning rate	1×10^{-3}
Text encoder	`HashingTextEncoder`
Embedding dimension	384
Hidden dimension of F_θ	256
Hidden dimension of $D_{\phi,\psi}$	128
Number of agent slots K	5
Top-r candidate skills	8
Candidate support size	8
Maximum skills per slot	3
Edge label mode	`chain`
Anchor mode	`chain`
Acyclic graph constraint	`true`
Edge loss weight	1.0
Graph sparsity weight	1×10^{-3}
Compatibility weight	1.0
Anchor loss weight	1.0
Random seed	0
Incidence threshold	0.25
Edge prediction threshold	0.5
Temperature	0.0

For test-time skill insertion, we additionally report inserted-skill hit rate:

$$\text{Hit}_{\text{new}} = \frac{1}{|\mathcal{Q}_{\text{new}}|} \sum_{q \in \mathcal{Q}_{\text{new}}} \mathbb{I}[h_{\text{new}}(q)] \,, \tag{43}$$

where

$$h_{\text{new}}(q) = \exists m \in \mathcal{L}_{\text{new}},\ k \text{ s.t. } B_q[m,k] = 1. \tag{44}$$

F Implementation Details

F.1 Hyperparameters

Table 5 lists the main hyperparameters used by SIGMA. Values that depend on a benchmark-specific execution budget are selected on the validation split and then fixed for test evaluation.

F.2 Training Configuration

We summarize the core training configuration of SIGMA in Table 6, as well as the training dynamics of loss-related parameters in Figure 7. The incidence predictor and topology decoder are trained jointly with full-batch optimization. Unless otherwise specified, all source-library and unseen-library experiments use the same architecture, objective, and decoding thresholds.

F.3 Training Objective

The training objective combines incidence supervision, optional edge reconstruction, and sparsity regularization. The incidence generator is trained with binary cross-entropy over skill-agent assignments:

$$\mathcal{L}_{\text{inc}} = \sum_q \sum_{m=1}^{M} \sum_{k=1}^{K} \text{BCE}\left(\mathbf{B}_q^\star[m,k], p_{m,k}\right), \tag{45}$$

where $p_{m,k} = \sigma(\alpha_{m,k})$. When annotated communication graphs are available, the topology decoder is trained with

$$\mathcal{L}_{\text{edge}} = \sum_q \sum_{i \neq j} \text{BCE}(\mathbf{A}_q^\star[i,j], \sigma(\ell_{ij})). \tag{46}$$

If annotated target graphs are unavailable, this term is computed against the shared structural prior described in Appendix D.2 or omitted in the matched-decoder control. We further use a soft bundle-size cap and a graph sparsity penalty:

$$\mathcal{L}_{\text{sparse}} = \sum_q \sum_{k=1}^{K} \max\left(0, \sum_{m=1}^{M} p_{m,k} - k_s\right) + \lambda_g \sum_q \sum_{i \neq j} \sigma(\ell_{ij}). \tag{47}$$

The final objective is

$$\mathcal{L} = \mathcal{L}_{\text{inc}} + \lambda_e \mathcal{L}_{\text{edge}} + \lambda_s \mathcal{L}_{\text{sparse}}. \tag{48}$$

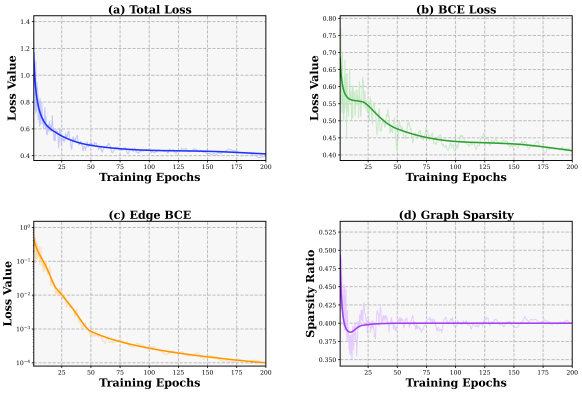


Figure 7: Training dynamic of loss.

The incidence term trains the skill-agent assignment predictor. The edge term is used only when annotated communication graphs or a shared structural prior are available. The sparsity term prevents degenerate solutions in which every slot receives too many skills or the decoded communication graph becomes overly dense.

F.4 Complexity

Let M be the number of skills and K the number of agent slots. Incidence scoring requires $O(MK)$ skill-agent pair evaluations. The skill-bundle compatibility term is not computed over all M^2 skill pairs. Instead, it is evaluated over sparse active supports. During inference, each agent has at most k_s selected skills, so pairwise skill compatibility across agent pairs costs $O(K^2 k_s^2)$. Therefore, the active compatibility computation is controlled by the bundle-size budget rather than the full skill-library size.

G Additional Results

G.1 Robustness to Unfamiliar Skill Libraries

Table 14 reports the full source-to-unseen library results across six benchmarks. SIGMA achieves the best unseen-library accuracy on every benchmark and shows the smallest performance drop in all cases. On average, SIGMA drops by only 0.95 points, compared with 2.99 for `G-Designer`, 4.06 for GPTSwarm, and 4.02 for LLM-Debate. This suggests that incidence-based skill assignment is less dependent on a fixed skill inventory and can more reliably compose newly introduced skill cards into effective agent slots.

G.2 Detailed Token and Runtime Analysis

To further quantify the computational cost of SIGMA, we report detailed token consumption and wall-clock runtime across six benchmarks and three base LLMs. As shown in Table 13, the overall cost varies across benchmarks, mainly depending on the number of evaluated questions and the difficulty of the task. GSM8K contributes the largest token usage for all three backbones due to its larger test set, while HumanEval and MMLU require substantially fewer total tokens. Despite these differences, SIGMA remains practical across model scales: all benchmark evaluations finish within a few hours, and most individual benchmark runs complete within roughly one hour.

Table 7: Skill sets used by each dataset.

Dataset	Skills
MMLU	`final_synthesis`, `problem_decomposition`, `evidence_check`, `analogy`, `counterexample`, `elimination`, `concept_recall`, `constraint_tracking`, `causal_reasoning`, `uncertainty_calibration`, `calculation`, `domain_translation`
GSM8K	`synthesize`, `verify`, `retrieve`, `reason`
MultiArith	`synthesize`, `reason`, `retrieve`, `verify`
SVAMP	`synthesize`, `retrieve`, `reason`, `verify`
AQuA	`synthesize`, `retrieve`, `reason`, `verify`
HumanEval	`edge_cases`, `implementation`, `final_code_review`, `spec_parse`, `complexity_check`

H Prompt Templates

H.1 Agent Execution Prompt

During execution, each agent receives a two-part prompt consisting of a system message and a user message. As shown in Figure 8, the system message specifies the agent identity, its assigned skill cards, and the rule that only task-relevant skills should be used. The user message provides the current task, routed mailbox summaries from predecessor agents, and the required response format. This prompt design ensures that skill assignments are not only used for node construction, but are also exposed to the LLM as executable instructions during multi-agent interaction. Here is an example: Figure 9

AGENT EXECUTION PROMPT

Message format.

```
[{"role": "system", "content": system_prompt},
 {"role": "user", "content": user_prompt}]
```

system_prompt

You are agent `{agent_id}` in a multi-agent system. Your capabilities are defined by the assigned skill cards: `{selected skill cards}`. Use only skills relevant to the current step. Read routed mailbox evidence carefully and incorporate useful constraints, tool outputs, retrieved facts, or unresolved issues.

user_prompt

task	`{query}`
mailboxes	`{skill-specific mailbox summaries}`
response	Return: (1) intermediate reasoning or action; (2) message to downstream agents; (3) tool/API/retrieval request if needed.

Figure 8: Prompt format used for agent execution.

H.2 Baseline Prompt Templates

For reproducibility, we report the concrete **Vanilla** and **CoT** prompt templates used for single-agent baselines. Each template uses the same two-message format as the agent execution prompt above. For a given dataset, `system_prompt` is constructed by concatenating the dataset-specific *System base* with either the *Vanilla suffix* or the *CoT suffix*; the `user_prompt` contains the benchmark instance. All methods use the same benchmark split, deterministic decoding with `temperature`=0.0, and the same dataset-specific answer parser.

BASELINE PROMPT TEMPLATE: MMLU

Message format.

```
[{"role": "system", "content": system_prompt},
 {"role": "user", "content": user_prompt}]
```

system_prompt

System base	I will ask you a multiple-choice question. There are 4 answer options enumerated as A, B, C, and D. Only one option is correct. Always select the best available option from A, B, C, and D, even if none seems perfect. Do not answer that none of the options is correct.
Vanilla suffix	Reply with only one letter: A, B, C, or D. Do not include any analysis.
CoT suffix	Reason step by step before choosing the answer. Use at most 5 short sentences. Do not write tables, exhaustive cases, or long derivations. Put your final answer on the last line exactly in this format: `Final answer: X` where X is one of A, B, C, or D.

user_prompt

task	`The task is:\n\n {question-with-options}`

BASELINE PROMPT TEMPLATE: AQUA

Message format.

```
[{"role": "system", "content": system_prompt},
 {"role": "user", "content": user_prompt}]
```

system_prompt

System base	I will ask you a multiple-choice math question. There are 5 answer options enumerated as A, B, C, D, and E. Only one option is correct. Always select the best available option from A, B, C, D, and E.
Vanilla suffix	Reply with only one letter: A, B, C, D, or E. Do not include any analysis.
CoT suffix	Reason step by step before choosing the answer. Use at most 5 short sentences. Do not write tables, exhaustive cases, or long derivations. Put your final answer on the last line exactly in this format: `Final answer: X` where X is one of A, B, C, D, or E.

user_prompt

task	`The task is:\n\n {question-with-A-E-options}`

BASELINE PROMPT TEMPLATE: GSM8K, MULTI-ARITH, AND SVAMP

Message format.

```
[{"role": "system", "content": system_prompt},
 {"role": "user", "content": user_prompt}]
```

system_prompt

GSM8K base	I will ask you a grade-school math word problem. Solve the problem and provide the numeric answer without units. Always put your final answer on the last line exactly in this format: `The answer is X` where X is the numeric answer.
M-Arith base	I will ask you a multi-step arithmetic word problem. Solve the problem and provide the numeric answer without units. Always put your final answer on the last line exactly in this format: `The answer is X` where X is the numeric answer.
SVAMP base	I will ask you an arithmetic word problem. Solve the problem and provide the numeric answer without units. Always put your final answer on the last line exactly in this format: `The answer is X` where X is the numeric answer.
Vanilla suffix	Reply with only the final answer line. Do not include analysis.
CoT suffix	Reason step by step before giving the final answer. Keep the reasoning concise.

user_prompt

task	`The task is:\n\n {math-word-problem}`

BASELINE PROMPT TEMPLATE: HUMANEVAL

Message format.

```
[{"role": "system", "content": system_prompt},
 {"role": "user", "content": user_prompt}]
```

system_prompt

System base	You will be given a Python function signature and docstring. Write a correct, concise Python implementation. Do not change the function name, arguments, or expected return type. The final answer must include a complete Python code block with the full function definition.
Vanilla suffix	Reply with only one Python code block. Do not include analysis outside the code block.
CoT suffix	Reason briefly before writing the implementation. Keep the reasoning concise. Put the final implementation in the last Python code block. Do not write anything after the final code block.

user_prompt

task	`The task is:\n\n {function spec}`

ANSWER EXTRACTION USED WITH THE PROMPT TEMPLATES

MMLU/AQuA	The parser first ignores any hidden reasoning before a closing `</think>` tag, then reads a leading standalone option letter, explicit final-answer markers such as `Final answer: X`, `correct answer is X`, and `answer: X`, or option-selection phrases such as `choose option X`. AQuA additionally supports matching emitted option text back to A–E.
Numeric math	The parser matches explicit answer markers including `final answer`, `the answer is`, `answer is`, `answer`, and `####`. It also supports boxed answers. If no marker is found, it uses the last numeric expression and normalizes commas, currency symbols, percent signs, braces, fractions, decimals, and surrounding punctuation.
HumanEval	We report pass@1. The evaluator extracts the last valid Python code block containing a function or class definition and scores it once with the benchmark unit tests, with no retries, no execution-feedback loop, and no access to held-out tests during generation.

I Additional Details

I.1 Dataset description

We evaluate SIGMA on six benchmarks covering general reasoning, mathematical reasoning, and code generation. MMLU is used to test broad multiple-choice reasoning ability, while GSM8K, MultiArith, SVAMP, and AQuA evaluate mathematical reasoning under both numerical-answer and multiple-choice formats. HumanEval is used to assess code generation ability with pass@1 as the evaluation metric. Table 8 summarizes the answer type, metric, test-set size, and license information for each benchmark. For MMLU, we follow the cost-controlled protocol of prior graph-based MAS designers, including `G-Designer` and ARG-Designer, and use the same fixed 153-question shuffled evaluation subset for all methods and base LLMs. This subset is fixed before method-specific tuning; optimization and pseudo-label construction use separate training/dev examples and do not access held-out answers.

We use all existing benchmark artifacts only for their intended research evaluation purposes: MMLU for multiple-choice reasoning, GSM8K, MultiArith, SVAMP, and AQuA for mathematical reasoning, and HumanEval for code-generation evaluation. We do not redistribute modified benchmark data or use these artifacts outside research

Table 8: Dataset descriptions and statistics.

Category	Dataset	Type	Metric	#Test	Lic.
General	MMLU	Multi-choice	Acc.	153	MIT
Math	GSM8K	Num.	Acc.	1,319	MIT
	M-Arith	Num.	Acc.	600	–
	SVAMP	Num.	Acc.	1,000	MIT
	AQuA	Multi-choice	Acc.	254	Apache-2.0
Code	H-Eval	Code	Pass@1	164	MIT

evaluation settings.

I.2 Skill distribution

Figure 10 and Table 7 show the normalized skill distribution across different datasets. Overall, the skill distributions reveal clear task-dependent specialization. To complement this distributional view, Table 10 reports the corresponding skill-library statistics, including the number of source and unseen skill cards, the cognitive/tool-API split, the average number of skills assigned to each agent where available, and the source–unseen library overlap. Together, these results show not only which skills are frequently selected for each dataset, but also how the underlying skill libraries are constructed and controlled across source and unseen settings.

I.3 Model Size and Computation Budget

Table 11 reports the size of the base executors and the trainable components introduced by SIGMA. The base LLMs are used only as frozen executors during multi-agent inference, while SIGMA adds a lightweight controller consisting of the incidence predictor F_θ and the topology decoder $D_{\phi,\psi}$. The controller has fewer than one million trainable parameters in total, making the additional training cost small compared with the base LLM executors.

For execution, we use three representative backbones with different serving configurations. *Qwen3-8B* and *GPT-OSS-120B* are served locally with vLLM, where *Qwen3-8B* disables the thinking template and *GPT-OSS-120B* uses reasoning effort. *GPT-4o-mini* is accessed through an OpenAI-compatible backend. All SIGMA controller training runs are conducted on $4 \times$ NVIDIA RTX A6000 GPUs 48GB.

I.4 OOD Diagnostic Analysis

In the MMLU Split-A setting, ID examples contain observed skill-bundle patterns, whereas OOD examples contain the held-out bundle {`elimination`, `evidence_check`, `final_synthesis`}, testing whether SIGMA generalizes to unseen skill compositions within the same benchmark. In the skill-library transfer setting, ID uses the source skill library, while OOD replaces it with semantically different unseen skill cards at test time, testing whether the controller can transfer across changed capability resources.

Table 12 reports ID/OOD accuracy, OOD performance change, and OOD retention. Across both diagnostics, SIGMA maintains strong OOD accuracy and non-negative OOD change. These results suggest that its gains are not merely due to memorizing in-distribution skill assignments, but also to recomposing useful capabilities under shifted skill-composition and skill-library conditions.

J Failure Analysis

We further inspect representative failure cases of SIGMA to understand when skill-incidence based multi-agent design is less effective. Overall, the failures mainly come from five sources: overly sparse topology decoding, insufficient skill diversity, biased final aggregation, benchmark-specific answer priors, and imperfect mailbox routing.

Overly sparse communication topology. A notable failure mode is that the decoded communication graph can become empty when the edge threshold is too strict or when the learned topology scores are overly sparse. In an over-sparse MMLU-153 diagnostic variant, all 153 evaluated examples were decoded as zero-edge graphs, which effectively prevents agents from exchanging information. In this case, SIGMA degenerates into isolated skill-composed agents, losing the benefit of multi-agent collaboration. This suggests that topology sparsity should be regularized but not over-constrained. A practical fix is to calibrate the edge threshold on the validation split, enforce a minimum number of edges, or add a topology loss that discourages fully disconnected graphs.

Correct minority lost during aggregation. In 9 out of 27 MMLU wrong cases, at least one agent produces the correct answer, but the final decision still follows the wrong majority. This suggests that simple majority-style aggregation can suppress useful minority evidence. The issue is not necessarily that SIGMA fails to generate the right reasoning, but that the final decision mechanism does not sufficiently distinguish high-quality

Table 9: Controlled comparison between SINGLE-AGENT+SKILLS and SIGMA using *GPT-OSS-120B*. SINGLE-AGENT+SKILLS receives the same union of selected skill cards as SIGMA, but removes multi-agent execution, topology decoding, and mailbox routing.

Method	Mul.	Topo.	Comp.	Mailbox	MMLU	GSM8K	MultiArith	SVAMP	AQuA	HumanEval	Avg.
CoT	✗	✗	✗	✗	83.39	84.81	95.63	95.19	76.02	90.87	87.65
SINGLE-AGENT+SKILLS	✗	✗	✓	✗	85.41	87.65	95.93	96.73	88.37	93.24	91.22
SIGMA	✓	✓	✓	✓	87.28	96.25	98.34	95.94	90.16	95.83	93.97

Table 10: Skill-library statistics for the source and unseen skill libraries. M_{src} and M_{unseen} denote the number of skill cards in the source and unseen libraries. Cog. and Tool/API report the number of cognitive and tool/API skills in the source/unseen libraries.

Dataset	M_{src}	M_{unseen}	Cog. S/U	Tool/API S/U	Avg. skills/agent S/U	Overlap	Examples: source → unseen
MMLU, half split	6	6	6 / 6	0 / 0	2.80 / 2.88	0	`concept_recall` → `analogy`; `calculation` → `uncertainty_calibration`
GSM8K	12	12	12 / 12	0 / 0	– / –	0	`source_extract_numbers` → `target_quantity_schema`
MultiArith	12	12	12 / 12	0 / 0	– / –	0	`source_choose_operation` → `target_operation_router`
SVAMP	12	12	12 / 12	0 / 0	– / –	0	`source_unit_tracking` → `target_answer_unit_check`
AQuA	12	12	12 / 12	0 / 0	– / –	0	`source_consistency_check` → `target_estimate_validate`
HumanEval	12	12	12 / 10	0 / 2	– / 2.42	0	`source_spec_parse` → `target_contract_reader` `source_test_design` → `target_local_test_designer`

evidence from repeated but weak agreement. A natural improvement is evidence-weighted aggregation, where final answers are weighted by skill relevance, confidence, mailbox evidence quality, or verifier judgments rather than raw vote count.

Answer-option bias. We also observe that wrong predictions sometimes overuse particular answer options, especially A or D. This points to a possible option-prior or prompt-order bias in multiple-choice tasks. Such bias may come from the base LLM, the answer-format constraint, or the distribution of predecessor messages in the mailbox. To diagnose this issue, we recommend option-order robustness tests, where the same question is evaluated under different permutations of answer choices. If predictions change substantially under permutation, the system should use option-normalized prompting or answer re-mapping during evaluation.

K Impact Statement

Positive impact. This work studies skill-incidence graphs for compositional multi-agent system design. By constructing agents from task-conditioned skill bundles rather than fixed role descriptions, SIGMA may improve the modularity, reusability, and adaptability of multi-agent systems.

Risks and mitigation. Flexible skill composition may introduce risks when unsafe, irrelevant, or conflicting skills are assigned to agents, especially in tool-augmented, code-execution, or real-world decision-making settings. Incorrect skill routing can propagate errors across agents and make system behavior harder to audit. To mitigate these risks, SIGMA should be used with bounded skill libraries, explicit execution constraints, assignment logging, and human oversight in high-stakes applications. Our experiments are limited to controlled benchmarks, and we do not recommend direct deployment in safety-critical domains without further validation and robustness evaluation.

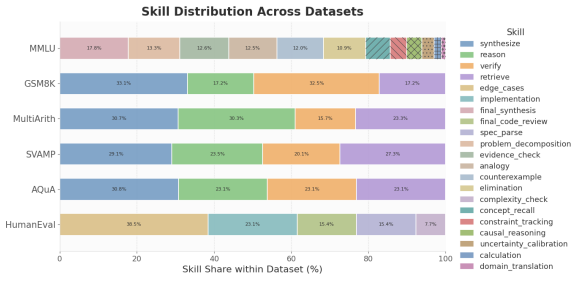


Figure 10: Skill distribution across datasets.

Table 11: Model and artifact sizes used by SIGMA.

Component	Size	Role
Base LLM executors		
Qwen3-8B	8B class	Local vLLM executor for the fixed MMLU-153 evaluation protocol.
GPT-OSS-120B	120B class	Executor for MMLU, GSM8K, MultiArith, SVAMP, AQuA, HumanEval, and ablations.
GPT-4o-mini	Undisclosed	Auxiliary executor for additional benchmark artifacts.
SIGMA trainable controller		
F_θ incidence predictor	461,953 params	Predicts the skill-agent incidence matrix B.
$D_{\phi,\psi}$ topology decoder	427,266 params	Predicts the learned communication topology.
Full SIGMA checkpoint	≈3.57 MB	Contains $F_\theta + D_{\phi,\psi}$, with 889,219 trainable parameters in total.
Incidence-only checkpoint	164,997 params / 664 KB	Early variant using only F_θ without learned topology decoding.

Table 12: OOD diagnostic results. ID denotes the in-distribution or source-library setting, while OOD denotes the held-out or unseen-library setting. Δ_{OOD} is computed as OOD accuracy minus ID accuracy, where higher values indicate better OOD retention.

Diag.	Method	ID	OOD	Δ_{OOD} ↑	Ret. ↑
MMLU Split-A	Fixed-chain	75.00	75.00	+0.00	100.00
	SIGMA	**82.50**	**85.00**	+2.50	**103.03**
Skill-Lib.	G-Designer	77.12	77.78	+0.65	100.85
	SIGMA	**77.12**	**79.08**	+1.96	**102.54**

Split details. MMLU Split-A uses {elimination, evidence_check, final_synthesis} as the held-out OOD bundle. Skill-Lib. evaluates source-to-unseen skill-card transfer at test time.

AGENT EXECUTION PROMPT EXAMPLE

Message format.

```
[{"role" : "system", "content" : system_prompt},
 {"role" : "user", "content" : user_prompt}]
```

system_prompt

skill bundle	Evidence checking: verify that the answer follows from stated evidence. Inputs: query, messages; outputs: analysis, answer; tools: none; tags: mmlu.
profile state	No accumulated interaction history before execution.
format constraint	The agent receives one question with four options A–D; exactly one option is correct. It must select the best available option, use other agents' reasoning only as critical advice, reply in fewer than 100 words, and put only one letter on the first line.

user_prompt

task	What one of the following is *not* a key management skill in planning? Option A: Conceptual skills Option B: Analytical skills Option C: IT and computing skills Option D: Communication skills
mailbox context	Mailbox[evidence_check] receives routed evidence from SIGMA slot 0 with similarity score 0.032. The predecessor selects C and argues that conceptual, analytical, and communication skills are core managerial competencies in planning, while technical/IT skills are less central.
expected output	Return the answer letter and a concise evidence-based explanation.

Figure 9: Example runtime prompt used by SIGMA. The system message injects the assigned skill card, profile state, and answer-format constraints, while the user message provides the concrete question and routed mailbox evidence.

Table 13: Token comparison across representative benchmarks and multi-agent methods.

Dataset	Method	Perf.	Tokens / Runtime
MMLU	SIGMA	**87.28**	1.18×10^5 / $18.0min$
	CARD	85.78	7.23×10^5 / $20.5min$
	ARG-Designer	85.63	5.57×10^5 / $41.2min$
	G-Designer	85.06	5.25×10^5 / $67.9min$
	GPTSwarm	83.58	2.61×10^6 / $55.1min$
	LLM-Debate	82.92	1.54×10^6 / $75.4min$
HumanEval	SIGMA	**95.83**	4.24×10^5 / $30.9min$
	CARD	93.98	7.28×10^5 / $70.2min$
	ARG-Designer	93.47	8.71×10^5 / $51.6min$
	G-Designer	92.88	7.05×10^5 / $50.8min$
	GPTSwarm	92.30	4.57×10^6 / $56.9min$
	LLM-Debate	91.43	6.90×10^6 / $68.6min$
GSM8K	SIGMA	**96.25**	5.48×10^6 / $84.1min$
	CARD	93.44	9.91×10^6 / $154.1min$
	ARG-Designer	91.08	1.06×10^7 / $149.4min$
	G-Designer	88.62	8.09×10^6 / $201.3min$
	GPTSwarm	86.88	1.44×10^7 / $241.2min$
	LLM-Debate	85.53	2.63×10^7 / $267.7min$
AQuA	SIGMA	**90.16**	8.22×10^5 / $62.9min$
	CARD	82.76	1.76×10^6 / $72.9min$
	ARG-Designer	78.62	1.09×10^6 / $67.7min$
	G-Designer	78.13	9.01×10^5 / $92.1min$
	GPTSwarm	77.41	2.07×10^6 / $109.2min$
	LLM-Debate	75.38	3.12×10^6 / $144.8min$
SVAMP	SIGMA	**95.94**	2.14×10^6 / $85.5min$
	CARD	95.48	8.01×10^6 / $117.5min$
	ARG-Designer	94.81	7.17×10^6 / $104.8min$
	G-Designer	95.21	3.07×10^6 / $101.2min$
	GPTSwarm	94.77	2.97×10^7 / $101.3min$
	LLM-Debate	94.89	3.55×10^7 / $165.9min$
MultiArith	SIGMA	98.34	8.62×10^5 / $48.1min$
	CARD	98.21	2.38×10^6 / $64.2min$
	ARG-Designer	**98.67**	1.14×10^6 / $78.7min$
	G-Designer	96.86	1.09×10^6 / $95.4min$
	GPTSwarm	96.50	2.00×10^6 / $125.5min$
	LLM-Debate	95.72	3.10×10^6 / $173.4min$

Table 14: Robustness to unfamiliar skill libraries across benchmarks using GPT-4o-mini as the base model. Δ denotes the performance drop from the source skill library to unseen skill cards, with lower values indicating stronger generalization under library changes.

Bench.	Method	Source Lib.	Unseen Lib.	$\Delta\downarrow$
MMLU	LLM-Debate	68.96	63.73	5.23
	GPTSwarm	71.37	64.84	6.53
	G-Designer	73.32	70.05	3.27
	CARD	75.11	73.68	1.43
	SIGMA	**75.68**	**75.03**	**0.65**
GSM8K	LLM-Debate	87.32	85.29	2.03
	GPTSwarm	87.83	84.47	3.36
	G-Designer	89.26	87.28	1.98
	CARD	93.21	91.44	1.77
	SIGMA	**92.96**	**91.86**	**1.10**
AQuA	LLM-Debate	72.86	66.96	5.90
	GPTSwarm	74.65	69.14	5.51
	G-Designer	76.60	70.3	6.30
	CARD	80.41	76.85	3.56
	SIGMA	**83.67**	**82.50**	**1.17**
Human.	LLM-Debate	79.4	72.58	6.82
	GPTSwarm	81.26	76.41	4.85
	G-Designer	82.48	78.45	4.03
	CARD	85.07	81.48	3.59
	SIGMA	**88.71**	**86.29**	**2.42**
SVAMP	LLM-Debate	87.43	85.46	1.97
	GPTSwarm	90.12	88.14	1.98
	G-Designer	90.32	89.17	1.15
	CARD	91.65	90.79	0.86
	SIGMA	**93.64**	**93.43**	**0.21**
Multi.	LLM-Debate	94.11	91.96	2.15
	GPTSwarm	96.07	93.93	2.14
	G-Designer	96.75	95.50	1.25
	CARD	96.94	95.95	0.99
	SIGMA	**99.64**	**99.46**	**0.18**

L Case Study

L.1 HumanEval: Communication-Linked Code Generation

Qualitative Study Setting

Evaluation Setting

Method	`SIGMA-humaneval`.
Dataset / split	`HumanEval` test.
Base LLM	`gpt-4o-mini`.
Decision method	`FinalWriteCode`.
Run summary	124 executed problems, 110 solved problems, 88.71% accuracy.
Interpretation focus	This case illustrates how communication exposes edge cases and turns them into an executable final implementation.

Learned Topology

Node ids	`5euR` (slot 0), `5Rg5` (slot 1), `7b2z` (slot 2), `7Tja` (slot 3), `4mKN` (slot 4).
Spatial chain	`5euR -> 5Rg5 -> 7b2z -> 7Tja -> 4mKN`.
Temporal edges	None.
Edge probabilities	$0\to1 = 0.9987$, $1\to2 = 0.9981$, $2\to3 = 0.9982$, $3\to4 = 0.9986$.
Graph constraint	`anchor=chain`; `acyclic=true`.

Reading Guide

❶Query	Function signature, docstring, examples, ground-truth implementation, and final prediction.
❷Trace	Selected skill bundles, chain topology, execution order, and round-level message flow.
❸Agent cards	Each card summarizes one skill-composed node and its main contribution.
❹Final	The `FinalWriteCode` decision node that writes the final Python solution.

Case: Edge-case repair for `iscube`

Query

Function specification and expected behavior

Record id: `HumanEval/77`. **Solved:** yes.

Task: implement `iscube(a)`, which returns `True` if integer `a` is a cube of some integer.

Examples: `iscube(1) -> True`, `iscube(2) -> False`, `iscube(-1) -> True`, `iscube(64) -> True`, `iscube(0) -> True`, `iscube(180) -> False`.

Ground-truth pattern: use the absolute value or signed cube-root handling, then check whether the rounded cube root cubed equals the original magnitude.

Final prediction: a negative-aware cube-root check with `root ** 3 == a`.

Trace

Skill-incidence and message flow

Execution order: `5euR -> 5Rg5 -> 7b2z -> 7Tja -> 4mKN`.
Slot 0 skills: {`implementation`, `edge_cases`, `final_code_review`}.
Slot 1 skills: {`edge_cases`, `final_code_review`}.
Slot 2 skills: {`spec_parse`, `implementation`, `edge_cases`}.
Slot 3 skills: {`implementation`, `edge_cases`}.
Slot 4 skills: {`spec_parse`, `edge_cases`, `complexity_check`}.
Interpretation: the learned chain routes the task from planning and algorithm description to code generation, testing feedback, and final bug fixing.

`5euR`: implementation + edge-case analysis + final review

Role/profile state: `SIGMA slot 0`; no accumulated history before execution.
Mailbox input: empty, because this is the first node in the chain.
Main response: proposes a single-function design and suggests checking whether the rounded cube root cubed equals the input.
Representative code pattern: `return round(a ** (1/3)) ** 3 == a`.
Diagnosis: useful high-level implementation plan, but the initial pattern is incomplete for negative inputs and may be numerically fragile.

`5Rg5`: edge-case analysis + final review

Mailbox input: receives the slot-0 design through the chain edge `5euR -> 5Rg5`.
Main response: restates the cube-root algorithm, documents expected usage, and emphasizes the equality check after rounding.
Evidence contribution: preserves the concise algorithmic structure and keeps the implementation simple.
Diagnosis: useful algorithmic clarification, but it still inherits the same cube-root risk from the initial design.

`7b2z`: specification parsing + implementation + edge-case analysis

Mailbox input: receives the slot-1 algorithm description through `5Rg5 -> 7b2z`.
Main response: writes an executable implementation using `round(a ** (1/3))`.
Internal-test signal: the generated implementation passes internal testing in the trace.
Diagnosis: partially useful because it produces runnable code, but risky because the round-then-cube method can mishandle negative inputs or large integers.

`7Tja`: implementation + edge-case analysis

Mailbox input: receives the slot-2 code through `7b2z -> 7Tja`.
Main response: warns that the implementation may fail due to floating-point rounding and highlights negative numbers and large integers as important edge cases.
Concrete checks suggested: `iscube(-8)`, `iscube(-1)`, and large cube values such as `729` or `1000000000`.
Diagnosis: this is the key corrective signal: the edge-case skill identifies the hidden weakness of the earlier implementation.

`4mKN`: specification parsing + edge-case analysis + complexity check

Mailbox input: receives the tester feedback through `7Tja -> 4mKN`.
Main response: revises the implementation to handle negative inputs by computing a signed cube-root candidate before checking `root ** 3 == a`.
Complexity: constant-time arithmetic with no auxiliary data structures.
Diagnosis: integrates the upstream warning into a bug-fixed implementation, showing how the final slot acts as a repair node rather than another independent generator.

`4sMD`: `FinalWriteCode` decision node

System role: final code writer; must output the completed Python function.
Final input pattern: task specification plus the chain-produced implementation and edge-case feedback.
Final behavior: outputs a negative-aware implementation: if `a < 0`, compute a signed cube-root candidate using `-(-a) ** (1/3)`; otherwise use `a ** (1/3)`; finally check `root ** 3 == a`.
Final result: solved.
Interpretation: the case shows a successful repair path. The early implementation nodes provide a simple candidate solution, the edge-case node exposes the negative-input risk, and the final writer incorporates that correction into the submitted code.

Figure 11: A HumanEval case where SIGMA's chain topology turns a simple but risky cube-root implementation into a negative-aware final solution.

L.2 MMLU: Skill-Conditioned Evidence Aggregation

Qualitative Study Setting

Evaluation Setting

Base LLM and split	`Qwen3-8B` on the fixed `MMLU-153` evaluation subset.
Overall run	153 executed questions, 94 solved questions, 61.44% accuracy.
Execution regime	Five `AnalyzeAgent` nodes produce skill-conditioned answers, followed by one `FinalRefer` decision node.
Interpretation focus	These cases examine whether skill-composed agents provide complementary evidence and whether the final decision can repair or amplify noisy intermediate answers.

Skill-conditioned Aggregation Regime

Node representation	Each case constructs $U_q \in \mathbb{R}^{5\times 384}$ from the selected skill bundles before graph decoding.
Observed graph state	The MMLU examples activate no pairwise spatial edge after thresholding, so the final decision node aggregates independently generated skill-conditioned evidence.
What this shows	The analysis isolates the benefit and limitation of skill composition and evidence aggregation, rather than attributing the MMLU examples to agent-to-agent message passing.

Reading Guide

❶Query	Question, options, record id, ground-truth answer, and final prediction.
❷Trace	Concrete node ids, execution order, skill-incidence bundles, activated edges, and maximum decoded edge probability.
❸Agent cards	The five executable columns selected by the skill-incidence matrix B_q. Each card lists the node id, role, injected skill profile, mailbox state, and agent output.
❹Final	The `FinalRefer` node. Its user prompt contains the task plus the five node outputs shown immediately above the final card; the final response is restricted to one letter.

Case: Answer-label repair on MMLU

Query

Question and expected answer

Record id: `mmlu-test-miscellaneous-39`. **Solved:** yes.
Question: In the film *The Talented Mr. Ripley*, who plays Mr. Ripley?
Options: A. Jude Law B. Matt Damon C. Dustin Hoffman D. Ben Affleck.
Ground truth: B.
Final prediction: B.
Case focus: the compact answer prefixes are noisy, but the rationales mostly identify Matt Damon; the final decision node repairs the answer-label mismatch.

Trace

Topology and skill-incidence trace

Nodes: `mLV2` (slot 0), `hj9X` (slot 1), `wxdu` (slot 2), `3M9o` (slot 3), `d7XF` (slot 4).
Execution order: `mLV2 -> hj9X -> wxdu -> 3M9o -> d7XF`.
Slot 0 skills: {`elimination`, `analogy`, `evidence_check`}.
Slot 1 skills: {`evidence_check`}.
Slot 2 skills: {`final_synthesis`}.
Slot 3 skills: {`counterexample`, `final_synthesis`}.
Slot 4 skills: {`counterexample`}.
Activated edges: `spatial_edges=[]`, `temporal_edges=[]`; maximum decoded edge probability $= 3.16\times10^{-3} < 0.5$.
Interpretation: the case isolates skill-conditioned evidence aggregation rather than agent-to-agent message passing.

`mLV2`: elimination + analogy + evidence check

Role/profile state: `SIGMA slot 0`; no accumulated interaction history before execution.
Injected skill profile: option elimination rules out implausible choices; analogy mapping connects the query to a known pattern; evidence checking verifies that the answer follows from stated evidence.
Mailbox input: empty because no predecessor output arrived.
Agent output: A. The rationale recalls that *The Talented Mr. Ripley* stars Matt Damon as the titular character, eliminates other options by filmography, and confirms Matt Damon as the correct answer.
Diagnosis: useful evidence, but the leading answer prefix conflicts with the rationale and the option mapping.

`hj9X`: evidence check

Role/profile state: `SIGMA slot 1`; no accumulated interaction history before execution.
Injected skill profile: evidence checking verifies that the answer follows from stated evidence.
Mailbox input: empty because no predecessor output arrived.
Agent output: A. The rationale identifies the target role, recalls that Matt Damon portrayed Mr. Ripley in the 1999 film, and selects the answer based on this information.
Diagnosis: useful evidence, again with a wrong leading letter.

`wxdu`: final synthesis

Role/profile state: `SIGMA slot 2`; no accumulated interaction history before execution.
Injected skill profile: final synthesis combines partial arguments into a concise final answer.
Mailbox input: empty because no predecessor output arrived.
Agent output: A. The response identifies the film, recalls the actor who portrayed Mr. Ripley, but incorrectly states that Jude Law played the role.
Diagnosis: wrong factual hypothesis; this is the main misleading card in the aggregation set.

`3M9o`: counterexample + final synthesis

Role/profile state: `SIGMA slot 3`; no accumulated interaction history before execution.
Injected skill profile: counterexample search looks for edge cases that falsify an answer; final synthesis combines partial arguments.
Mailbox input: empty because no predecessor output arrived.
Agent output: A. The rationale identifies Matt Damon as the actor who portrayed Mr. Ripley in *The Talented Mr. Ripley*.
Diagnosis: useful corrective evidence despite the noisy answer prefix.

`d7XF`: counterexample

Role/profile state: `SIGMA slot 4`; no accumulated interaction history before execution.
Injected skill profile: counterexample search looks for edge cases that falsify an answer.
Mailbox input: empty because no predecessor output arrived.
Agent output: A. The rationale states that Matt Damon portrayed the character in the 1999 film and explicitly ends with `Answer: B`.
Diagnosis: explicitly exposes the mismatch between the noisy prefix A and the correct textual answer B.

`UmWw`: `FinalRefer` decision node

System role: top decision-maker; must output exactly one of A, B, C, or D.
FinalRefer input: the prompt contains the task/options plus all five node outputs above: `mLV2: A`, `hj9X: A`, `wxdu: A`, `3M9o: A`, and `d7XF: A ... Answer: B`.
Final response: B.
Final result: solved.
Interpretation: the final node is not a raw majority vote over answer prefixes. It reads the rationales and repairs the answer-label noise because most cards point to Matt Damon and one card explicitly states `Answer: B`.

Figure 12: A complete case where the ffnal decision node repairs answer-label noise by reading the agent rationales.

Case B: Counterexample agents correct a noisy majority

Query

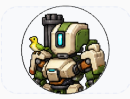
Question and expected answer

Record id: `mmlu-test-abstract_algebra-167`. **Solved:** yes.
Question: Statement 1: If a finite group has order n then the group contains a subgroup of order d for every positive divisor d of n. Statement 2: If a belongs to a finite group then $|a|$ divides $|G|$.
Options: A. True, True B. False, False C. True, False D. False, True.
Ground truth: D.
Final prediction: D.
Case focus: three agents produce noisy compact prefixes A, but most rationales identify the correct false/true mapping; counterexample agents provide the decisive correction.

Trace

Topology and skill-incidence trace

Nodes/execution order: `mLV2 -> hj9X -> wxdu -> 3M9o -> d7XF`.
Slot 0 skills: {`elimination`, `analogy`, `evidence_check`}.
Slot 1 skills: {`evidence_check`}.
Slot 2 skills: {`final_synthesis`}.
Slot 3 skills: {`counterexample`, `final_synthesis`}.
Slot 4 skills: {`counterexample`}.
Compact answers: A, A, A, D, D.
Activated edges: `spatial_edges=[]`, `temporal_edges=[]`; maximum decoded edge probability $= 1.71\times10^{-3} < 0.5$.
Interpretation: the final decision follows the mathematical evidence rather than the noisy compact-answer majority.



`mLV2`: elimination + analogy + evidence check

Mailbox input: empty.
Agent output: A. The rationale says Statement 1 is false because Lagrange's theorem only states that subgroup orders divide the group order, not that every divisor appears as a subgroup order. It also says Statement 2 is true because the order of an element divides the group order.
Diagnosis: the reasoning supports false/true, but maps this evidence to option A instead of D.

`hj9X`: evidence check

Mailbox input: empty.
Agent output: A. The rationale identifies Statement 1 as false and Statement 2 as true, and explicitly notes that A is incorrect while D is the correct false/true option.
Diagnosis: noisy prefix, but the rationale explicitly identifies D.

`wxdu`: final synthesis

Mailbox input: empty.
Agent output: A. The response again states that Statement 1 is false and Statement 2 is true, then explains that the correct choice should be D.
Diagnosis: another prefix/rationale mismatch with correct mathematical evidence.

`3M9o`: counterexample + final synthesis

Mailbox input: empty.
Agent output: D. Statement 1 is false because not all finite groups have subgroups for every divisor of their order; Statement 2 is true by Lagrange's theorem.
Diagnosis: clean counterexample-style correction that directly supports the ground-truth label.

`d7XF`: counterexample

Mailbox input: empty.
Agent output: D. The response independently confirms the same false/true mapping: subgroup orders must divide the group order, but not every divisor must occur, while element order divides group order.
Diagnosis: independently confirms the correction and reinforces the minority compact answer.

`UmWw`: `FinalRefer` decision node

System role: top decision-maker; must output exactly one of A, B, C, or D.
FinalRefer input: task/options plus five node outputs: three compact prefixes A and two compact prefixes D, with most rationales supporting false/true.
Final response: D.
Final result: solved.
Interpretation: the final node follows the consistent mathematical evidence rather than the noisy compact-answer majority, showing that skill cards are treated as evidence sources rather than plain role labels.

Figure 13: A MMLU case where counterexample skills provide the decisive corrective signal and allow the final decision node to override a noisy compact-answer majority.

Case C: Recoverable disagreement among skill-composed agents

Query

Question and expected answer

Record id: `mmlu-test-public_relations-3`. **Solved:** yes.
Question: Which of these organizations is most effective in engaging with customers online?
Options: A. Starbucks B. Coca-Cola C. Wholefoods D. Redbull.
Ground truth: A.
Final prediction: A.
Case focus: heterogeneous skill-composed agents surface both Starbucks-supporting evidence and a plausible Red Bull distractor, but the final decision selects the better-supported answer.

Trace

Topology and skill-incidence trace

Nodes/execution order: `mLV2 -> hj9X -> wxdu -> 3M9o -> d7XF`.
Slot 0 skills: {`elimination`, `analogy`, `evidence_check`}.
Slot 1 skills: {`evidence_check`}.
Slot 2 skills: {`final_synthesis`}.
Slot 3 skills: {`counterexample`, `final_synthesis`}.
Slot 4 skills: {`counterexample`}.
Compact answers: A, A, D, A, D.
Activated edges: `spatial_edges=[]`, `temporal_edges=[]`; maximum decoded edge probability $= 1.96\times10^{-3} < 0.5$.
Interpretation: the aggregation remains stable despite disagreement because the final input contains enough grounded evidence for Starbucks.

`mLV2`: elimination + analogy + evidence check

Mailbox input: empty.
Agent output: A. The response argues that Starbucks has a strong online presence, including a user-friendly website, mobile app, and social media engagement.
Evidence contribution: supports Starbucks through integrated digital strategy and customer engagement.
Diagnosis: useful Starbucks-supporting evidence.

`hj9X`: evidence check

Mailbox input: empty.
Agent output: A. The rationale emphasizes Starbucks' mobile app with rewards, active social media engagement, and integrated online customer experience.
Evidence contribution: independently reinforces the same answer with additional customer-engagement evidence.
Diagnosis: useful aligned evidence.

`wxdu`: final synthesis

Mailbox input: empty.
Agent output: D. The response argues that Red Bull has a strong online presence, interactive content, and community engagement through platforms such as YouTube and social media.
Evidence contribution: surfaces a plausible distractor.
Diagnosis: useful for stress-testing the answer, but ultimately less supported than Starbucks.

`3M9o`: counterexample + final synthesis

Mailbox input: empty.
Agent output: A. The rationale argues that Starbucks excels through its mobile app, personalized offers, and social media interaction, while Coca-Cola and Red Bull also have strong digital presences.
Evidence contribution: compares the distractors against Starbucks and supports the correct option.
Diagnosis: useful synthesis that acknowledges alternatives without switching away from A.

`d7XF`: counterexample

Mailbox input: empty.
Agent output: D. The response favors Red Bull because of interactive content, community engagement, and digital innovation.
Evidence contribution: makes the plausible Red Bull distractor visible.
Diagnosis: noisy but informative disagreement.

`UmWw`: `FinalRefer` decision node

System role: top decision-maker; must output exactly one of A, B, C, or D.
FinalRefer input: task/options plus five node outputs containing three Starbucks-supporting cards and two Red Bull distractor cards.
Final response: A.
Final result: solved.
Interpretation: disagreement is recoverable because the final node receives enough grounded evidence for Starbucks while still seeing a plausible distractor. This is the desired behavior of skill diversity: disagreement is visible but not automatically decisive.

Figure 14: A MMLU case where disagreement among skill-composed agents is recoverable because the final mailbox contains enough grounded evidence for the correct answer.

Case D: Shared misconception in a domain-specific legal question

Query

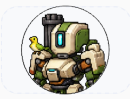
Question and expected answer

Record id: `mmlu-test-international_law-1`. **Solved:** no.
Question: Who is an "injured State" in the law of international responsibility?
Options: A. A State is "injured" in case that it has suffered a damage from the internationally wrongful conduct. B. A State is "injured" in cases that there has been a violation of a peremptory norm of international law. C. A State is "injured" should it acknowledge the existence of the internationally wrongful conduct. D. A State is "injured" if the obligation breached was owed to it individually or if it was owed to a group of States, including that State, and it was specially affected.
Ground truth: D.
Final prediction: A.
Case focus: all agents share the same false legal premise, so aggregation has no corrective evidence to follow.

Trace

Topology and skill-incidence trace

Nodes/execution order: `mLV2 -> hj9X -> wxdu -> 3M9o -> d7XF`.
Slot 0 skills: {`elimination`, `analogy`, `evidence_check`}.
Slot 1 skills: {`evidence_check`}.
Slot 2 skills: {`final_synthesis`}.
Slot 3 skills: {`counterexample`, `final_synthesis`}.
Slot 4 skills: {`problem_decomposition`, `final_synthesis`}.
Compact answers: A, A, A, A, A.
Activated edges: `spatial_edges=[]`, `temporal_edges=[]`; maximum decoded edge probability $= 1.70\times10^{-3} < 0.5$.
Interpretation: this failure reflects insufficient domain-specific evidence coverage rather than an answer-format issue.



`mLV2`: elimination + analogy + evidence check

Mailbox input: empty.
Agent output: A. The rationale defines an injured State as one that has suffered damage from an internationally wrongful act and treats option A as directly correct.
Diagnosis: shares the false premise that actual damage is the defining legal standard.

`hj9X`: evidence check

Mailbox input: empty.
Agent output: A. The response repeats that an injured State is a State that suffered damage from wrongful conduct, rejecting the other options as related but incorrect.
Diagnosis: aligned with the wrong premise; no corrective legal standard is introduced.

`wxdu`: final synthesis

Mailbox input: empty.
Agent output: A. The response synthesizes the same definition and chooses A as the most direct description.
Diagnosis: fluent synthesis amplifies the shared misconception.

`3M9o`: counterexample + final synthesis

Mailbox input: empty.
Agent output: A. The response argues that option A correctly defines the concept, while the other options misstate criteria or add unnecessary conditions.
Diagnosis: the counterexample skill does not recover the missing Article-style legal condition.

`d7XF`: problem decomposition + final synthesis

Mailbox input: empty.
Agent output: A. The response decomposes the definition, says option A captures damage from wrongful conduct, and rejects option D as overly complex.
Diagnosis: additional decomposition changes the prompt shape but still fails to retrieve the correct legal criterion.

`UmWw`: `FinalRefer` decision node

System role: top decision-maker; must output exactly one of A, B, C, or D.
FinalRefer input: task/options plus five aligned but wrong A-supporting node outputs.
Final response: A.
Final result: not solved.
Interpretation: this is a failure of evidence coverage. No card introduces the missing legal standard in option D, so the final node has no corrective signal to follow.

Figure 15: A MMLU failure case where skill diversity does not help because every card shares the same false legal premise.

Case E: Correct majority lost during final aggregation

Query

Question and expected answer

Record id: `mmlu-test-jurisprudence-37`. **Solved:** no.
Question: Which of the following criticisms of Llewellyn's distinction between the grand and formal styles of legal reasoning is the most compelling?
Options: A. There is no distinction between the two forms of legal reasoning. B. Judges are appointed to interpret the law, not to make it. C. It is misleading to pigeon-hole judges in this way. D. Judicial reasoning is always formal.
Ground truth: C.
Final prediction: A.
Case focus: three agents correctly support C, but the final node overweights a salient minority counterexample and outputs A.

Trace

Topology and skill-incidence trace

Nodes/execution order: `mLV2 -> hj9X -> wxdu -> 3M9o -> d7XF`.
Slot 0 skills: {`elimination`, `analogy`, `evidence_check`}.
Slot 1 skills: {`evidence_check`}.
Slot 2 skills: {`final_synthesis`}.
Slot 3 skills: {`counterexample`, `final_synthesis`}.
Slot 4 skills: {`counterexample`}.
Compact answers: C, C, C, A, D.
Activated edges: `spatial_edges=[]`, `temporal_edges=[]`; maximum decoded edge probability $= 2.06\times10^{-3} < 0.5$.
Interpretation: this case exposes a final-aggregation limitation: correct majority evidence can be lost when a wrong minority rationale appears forceful.

`mLV2`: elimination + analogy + evidence check

Mailbox input: empty.
Agent output: C. The rationale says option C criticizes the categorization of judicial reasoning as oversimplified, while A and D contradict each other and B is less relevant.
Diagnosis: useful correct evidence.

`hj9X`: evidence check

Mailbox input: empty.
Agent output: C. The response argues that option C challenges the validity of pigeon-holing judges into reasoning styles, while B concerns judicial role rather than the distinction itself.
Diagnosis: independently supports the correct option with aligned rationale.

`wxdu`: final synthesis

Mailbox input: empty.
Agent output: C. The response selects C as the most nuanced and relevant criticism because it argues the categorization oversimplifies judges' approaches.
Diagnosis: third aligned correct card.

`3M9o`: counterexample + final synthesis

Mailbox input: empty.
Agent output: A. The response claims option A directly challenges the core of Llewellyn's distinction by denying its existence and treats this as the strongest counterexample.
Diagnosis: salient but wrong minority rationale.

`d7XF`: counterexample

Mailbox input: empty.
Agent output: D. The response argues that option D challenges the distinction by asserting that judicial reasoning is always formal.
Diagnosis: another distractor card; it exposes an alternative challenge but does not match the best criticism.

`UmWw`: `FinalRefer` decision node

System role: top decision-maker; must output exactly one of A, B, C, or D.
FinalRefer input: task/options plus five node outputs: three C-supporting cards, one A-supporting counterexample card, and one D-supporting counterexample card.
Final response: A.
Final result: not solved.
Interpretation: this is an aggregation failure. The final node overweights a forceful but wrong minority counterexample and ignores that Agents 0–2 independently agree on C with aligned rationales.

Figure 16: A MMLU failure case where three agents identify the correct answer, but the final decision node follows a minority distractor.