COMPETENCE-AWARE ORCHESTRATION FOR LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems (MAS) have emerged as a promising paradigm for enhancing complex reasoning through role specialization and collaboration, yet a key challenge remains: *how can collaboration be adapted to a backbone's competence on the current query?* This problem matters because more collaboration is not always better—richer coordination may help on difficult cases, but it also incurs additional token cost and redundant interactions. We propose a two-stage competence-aware framework for automatic MAS orchestration. It first models the LLM backbone's task-conditioned competence for the current query, and then uses this signal to adapt role activation and the communication graph. Across six benchmarks, our method achieves an average score of 93.81%. Experiments with six LLM backbones of varying competence levels further show that our method adjusts collaboration according to the inferred competence: for higher-competence backbones, it improves average accuracy from 87.22 to 89.97 while reducing token cost by 36.26%; for lower-competence backbones, it yields a further average gain of 4.58 points over existing methods. The same trend also holds on unseen backbones.

1 INTRODUCTION

Recent advances in large language models (LLMs) have greatly expanded the use of Multi-Agent Systems (MAS) for reasoning, code generation, and complex problem solving (Hong et al., 2024; Wu et al., 2024; Qian et al., 2024; Chen et al., 2024). Related progress in prompting, search, self-refinement, ensembling, and text-based optimization has further improved LLM reasoning and adaptation (Nori et al., 2023; Shinn et al., 2023; Jiang et al., 2023; Yuksekgonul et al., 2024; Liang et al., 2024). By assigning specialized roles and coordinating their interactions, LLM-based MAS can outperform direct single-agent reasoning (Du et al., 2024; Wang et al., 2024; Liu et al., 2023). However, their effectiveness depends critically on collaboration topology and communication pattern: different tasks, and even different instances, may benefit from different forms and scales of collaboration (Zhang et al., 2025c; Yue et al., 2025; Dang et al., 2026). Moreover, more collaboration is not always better: richer role specialization and denser communication may help on difficult cases, but also introduce additional token cost, latency, and redundant interactions (Zhang et al., 2025b; Wang et al., 2025b). This motivates a competence-aware perspective on automatic MAS construction: *How can collaboration be adapted to a backbone's competence on the current query?*

Recent studies have moved from manually designed pipelines and hand-crafted role-based collaboration patterns (Hong et al., 2024; Wu et al., 2024; Qian et al., 2024; Du et al., 2024; Wang et al., 2024) to frameworks that automatically optimize or dynamically generate multi-agent structures (Liu et al., 2023; Zhuge et al., 2024; Hu et al., 2025; Zhang et al., 2025d;c;a; Yue et al., 2025; Dang et al., 2026; Wang et al., 2025a). By generating role configurations and communication topologies from task descriptions, query content, candidate roles, or runtime feedback, these methods make MAS increasingly task-aware and adaptive (Zhang et al., 2025d; Dang et al., 2026). Yet the main focus has been on organizing agent collaboration around task semantics, while relatively less attention has been paid to explicitly conditioning its scale and structure on the backbone's competence for the current query. We argue that the utility of collaboration is not fixed: it depends on the backbone's standalone capability, the difficulty of the query, and the coordination overhead introduced by the system. When the backbone's standalone competence is high on a query, extra roles and intensive communication may add cost with little benefit (Zhang et al., 2025b; Wang et al., 2025b); in contrast, weaker back-

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

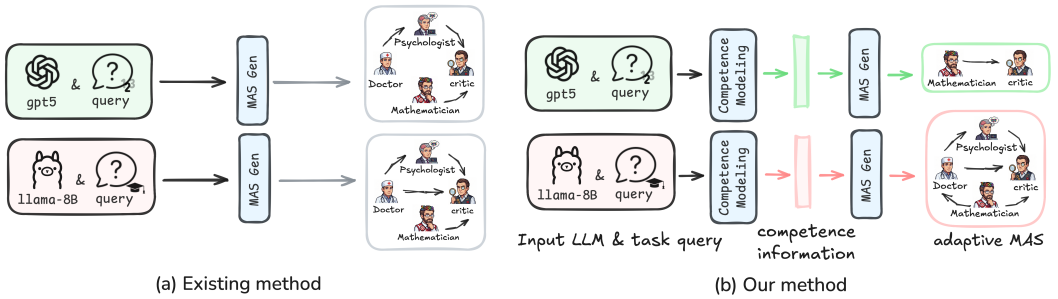


Figure 1: Comparison between existing MAS construction methods and our method.

bones or harder queries may benefit substantially from richer coordination and verification. This pattern also appears in broader agentic benchmarks (Qian et al., 2025; Kim et al., 2025). Motivated by these observations, we aim to adapt both **role participation** and **communication structure** to the backbone's **task-conditioned competence**.

To this end, we propose a competence-aware method for automatic MAS construction. Instead of mapping a query directly to a collaboration structure, our method first constructs a task-conditioned competence representation for the given backbone, and then uses it as a control signal for role activation and communication graph construction.

Our contributions can be summarized as follows:

- We design a competence modeling mechanism that combines a reusable backbone capability fingerprint with query-specific task information to derive a backbone-aware, query-conditioned competence representation.
- We develop a competence-aware orchestration method that uses this representation to configure the multi-agent structure at two levels—role activation and communication graph generation—adapting team composition and coordination topology to both the backbone and the current query.
- Experiments across multiple benchmarks and backbones show that our method improves overall performance, reduces collaboration token usage on strong backbones, and brings larger gains on weaker backbones, with the same trend also observed on unseen backbones.

2 Related Work

Static multi-agent collaboration and design. Early LLM-based multi-agent systems mainly relied on manually designed collaboration templates, including sequential pipelines and debate-style communication Du et al. (2024). Meanwhile, reasoning can also be improved within a single model through methods such as chain-of-thought Wei et al. (2022) and self-consistency Wang et al. (2022). To reduce manual engineering in multi-agent systems, subsequent work began to automate structure design, including workflow optimization methods such as AFlow Zhang et al. (2025d) and DSPy Khattab et al. (2024), as well as graph-based approaches such as GPTSwarm Zhuge et al. (2024), MacNet Qian et al. (2025), AgentPrune Zhang et al. (2025b), and AgentDropout Wang et al. (2025b). These methods improve system design by searching, optimizing, or pruning collaboration structures before deployment. However, whether hand-crafted or automatically optimized, they remain largely input-independent and follow a *learn once, deploy many* paradigm.

Dynamic orchestration. More recent studies move beyond static structures and dynamically organize collaboration at inference time. Rather than reusing one fixed design, these methods instantiate or route agent interaction according to the current query or runtime setting. Representative examples include G-Designer, which generates query-specific communication graphs Zhang et al. (2025c); MasRouter, which routes topology, roles, and models Yue et al. (2025); and MAAS, Puppeteer, CARD, and MAAS2, which dynamically orchestrate agent workflows under different settings Zhang et al. (2025a); Dang et al. (2026); Wu et al. (2026); Wang et al. (2025a). This line

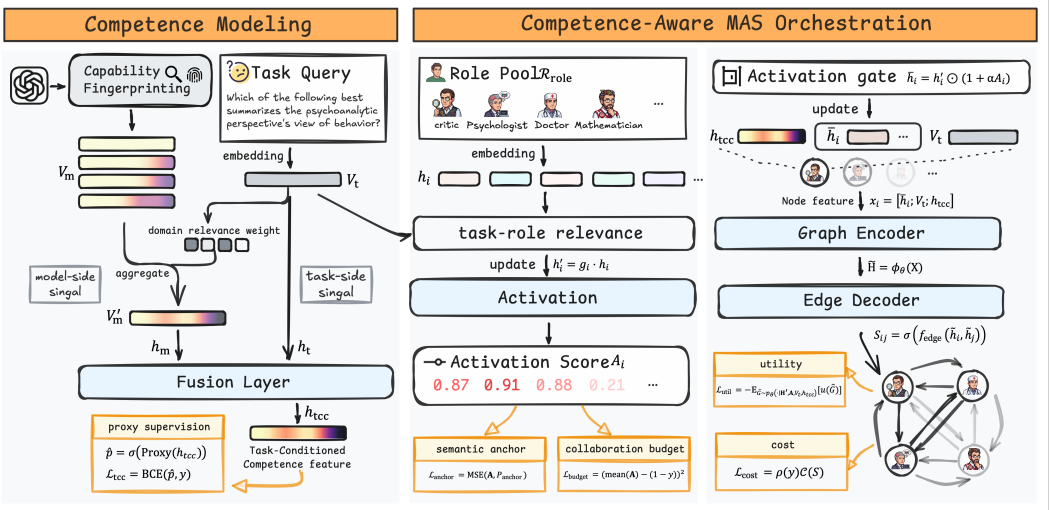


Figure 2: Overview of the proposed competence-aware orchestration framework. Stage I learns a competence representation from the backbone's capability fingerprint and the current query under success-rate supervision. Stage II uses this representation to guide role activation and communication graph generation.

of work makes multi-agent systems more responsive to instance-level variation. However, existing methods mainly adapt collaboration to task semantics or runtime context, with limited explicit modeling of the backbone's query-specific competence. Our work uses task-conditioned competence as a signal for adapting role participation and communication.

3 Method

3.1 Overview

Our method follows a two-stage competence-aware orchestration pipeline. In Stage I, we learn a task-conditioned competence representation from a reusable backbone capability fingerprint and query information. In Stage II, we use this representation to adapt the multi-agent structure at two levels: role activation and communication graph construction.

3.2 Stage I: Competence Modeling

3.2.1 Capability Fingerprinting

To support downstream query-level competence modeling, we precompute a reusable capability fingerprint for each LLM backbone using a fixed probe set $\mathcal{P}$. The probes are shared across backbones, organized by capability domain, and ordered within each domain by anchor-calibrated difficulty, yielding a common easy-to-hard scale for direct comparison. Note that this fingerprinting step is performed only once for each backbone: after a backbone M is first evaluated on $\mathcal{P}$, its fingerprint V_m can be reused in all subsequent queries.

We represent the fingerprint as a binary matrix $V_m \in \{0,1\}^{4\times 12}$, where each row corresponds to one capability domain and the columns within each row are ordered from easy to hard. Let $\mathcal{D} = \{1,2,3,4\}$ index the domains, and let $\{x_{d,j}\}_{j=1}^{12}$ denote the fixed probes in domain d. Running M on each probe $x_{d,j}$ yields a binary correctness signal $r(x_{d,j}; M) \in \{0,1\}$, from which we construct

$$V_m[d,j] \;=\; r(x_{d,j}; M), \quad d \in \mathcal{D}. \tag{1}$$

Because probe identities and their within-domain difficulty ordering are shared across backbones, V_m serves as a directly comparable capability prior for downstream competence modeling: each row summarizes performance in one capability domain, while the column position reflects how far

competence extends along the shared difficulty scale. Full probe-set construction details are provided in Appendix A.1.

3.2.2 TASK-CONDITIONED COMPETENCE MODELING

To condition the capability prior V_{m} on the current query, we combine it with the task representation to learn a task-conditioned competence representation h_{tcc}. Query-level success rates provide proxy supervision for learning this representation, which then conditions downstream role activation and communication graph construction.

Given a query q, we first encode it into a task representation $V_{\mathrm{t}} \in \mathbb{R}^d$. To capture the domain demands implied by q, we introduce a set of learnable domain prototypes $V_{\mathrm{d}} \in \mathbb{R}^{4\times d}$, with one prototype for each capability domain in V_{m}, initialized from the embeddings of short GPT5-generated domain descriptions. We then compute the similarity between V_{t} and each domain prototype, and normalize the resulting scores with a softmax to obtain domain relevance weights $w \in \mathbb{R}^4$. Using w, we aggregate the domain-wise fingerprint into task-conditioned capability evidence:

$$V'_{\mathrm{m}} = \sum_{i=1}^{4} w_i \cdot V_{\mathrm{m}}[i,:]. \tag{2}$$

Since the columns of V_{m} are ordered by increasing difficulty, V'_{m} provides a query-relevant and difficulty-aware summary of the backbone's capability fingerprint, which is then fused with task semantics to form the final competence representation.

We then project V_{t} and V'_{m} into a shared latent space, denoted by h_{t} and h_{m}, and fuse them through an interaction-aware fusion layer:

$$h_{\mathrm{tcc}} = \phi_{\mathrm{f}}\Big(\big[\, h_{\mathrm{t}};\ h_{\mathrm{m}};\ (h_{\mathrm{t}} - h_{\mathrm{m}});\ (h_{\mathrm{t}} \odot h_{\mathrm{m}}) \,\big]\Big), \tag{3}$$

where $\odot$ denotes element-wise multiplication, $[\cdot;\cdot]$ denotes concatenation, and $\phi_{\mathrm{f}}(\cdot)$ is a learnable fusion network.

We use query-level success rates as proxy supervision for h_{tcc}. An auxiliary linear head followed by a sigmoid maps this representation to a predicted success rate:

$$\hat{p} = \sigma\big(\mathrm{Proxy}(h_{\mathrm{tcc}})\big). \tag{4}$$

The supervision target $y \in [0,1]$ is the empirical success rate of the corresponding LLM backbone on the query, estimated from repeated trials. Details of the training data construction are provided in Appendix A.2. We optimize the model by minimizing the binary cross-entropy (BCE) between the predicted success rate $\hat{p}$ and the supervision target y. The prediction head supplies auxiliary supervision for learning competence-related information; downstream role activation and communication graph generation are conditioned on the representation h_{tcc}.

3.3 STAGE II: COMPETENCE-AWARE MAS ORCHESTRATION

Given the task-conditioned competence representation produced in Stage I, Stage II adapts the multi-agent structure for the current query through two mechanisms: role activation and communication graph construction. The former assesses how suitable each role is for participation, while the latter determines how the activated roles should communicate.

3.3.1 ROLE ACTIVATION

Given the query representation V_{t} and the task-conditioned competence representation h_{tcc}, role activation predicts a query-specific activation vector $A \in (0,1)^N$ over the candidate role pool. It combines task–role relevance with competence-conditioned modulation, and the resulting A serves as a prior for downstream communication graph construction.

Let the candidate role pool be $\mathcal{R}_{\mathrm{role}} = \{r_i\}_{i=1}^{N}$, where each role r_i is associated with a textual description d_i. We encode each description into a role representation $h_i \in \mathbb{R}^{d_{\mathrm{r}}}$, and compute a relevance gate based on projected task–role similarity to modulate each role representation:

$$g_i = \sigma\big((W_{\mathrm{t}}V_{\mathrm{t}})^{\top}(W_{\mathrm{r}}h_i)\big), \quad h'_i = g_i \cdot h_i. \tag{5}$$

We then predict the final activation score of role i by conditioning on both the gated role representation and the task-conditioned competence signal:

$$A_i = \sigma\Big(\mathrm{MLP}\big([h'_i;\, h_{\mathrm{tcc}}]\big)\Big), \quad i = 1, \ldots, N, \tag{6}$$

yielding $\mathbf{A} = [A_1, \ldots, A_N] \in (0,1)^N$. Here, task–role relevance determines which roles are plausible for the current query, while h_{tcc} modulates how strongly they should be activated given the backbone's estimated competence.

Since direct supervision for role activation is unavailable, we train the activator with two weak supervision signals. The first is a semantic anchor derived from task–role similarity, which provides role-level relevance supervision. The second reuses the query-level success-rate supervision from Section 3.2.2 to form a soft budget prior that regulates the overall activation scale.

Specifically, we compute the cosine similarity between the projected task representation and each projected role representation:

$$\tilde{V}_{\mathrm{t}} = W_{\mathrm{t}} V_{\mathrm{t}},\ \tilde{h}_i = W_{\mathrm{r}} h_i,\ s_i = \cos(\tilde{V}_{\mathrm{t}}, \tilde{h}_i). \tag{7}$$

We then apply min–max normalization over the role pool to obtain a query-local semantic anchor target P_{anchor}. Let $P_{\text{anchor}} = [P_{\text{anchor},1}, \ldots, P_{\text{anchor},N}] \in [0,1]^N$. We align the predicted activations A with this semantic anchor using

$$\mathcal{L}_{\text{anchor}} = \mathrm{MSE}\big(\mathbf{A}, P_{\text{anchor}}\big). \tag{8}$$

This objective provides relative supervision within each query, encouraging higher activation for roles that are more semantically compatible with the task.

To regulate the overall activation scale, we again reuse the success-rate supervision target $y \in [0,1]$ from Section 3.2.2. We use $1 - y$ as a query-level soft budget prior and regularize the mean role activation toward this target:

$$\mathcal{L}_{\text{budget}} = \Big(\,\mathrm{mean}(\mathbf{A}) - (1-y)\Big)^2. \tag{9}$$

This term encourages lower mean activation for queries with higher standalone success rates and higher mean activation for those with lower success rates. Together with the semantic anchor, it provides weak supervision for competence-conditioned role selection.

The overall objective is

$$\mathcal{L} = \lambda_{\text{anchor}} \mathcal{L}_{\text{anchor}} + \lambda_{\text{budget}} \mathcal{L}_{\text{budget}}. \tag{10}$$

The resulting activation vector A determines role participation for the current query and serves as input to downstream communication topology generation.

3.3.2 Graph-based MAS Generation

Building on the competence-aware role activation above, we generate a directed communication graph for the current query. Specifically, given the task representation V_{t}, the task-conditioned competence representation h_{tcc}, the competence-aware role representations $\mathbf{H}' = \{h'_i\}_{i=1}^N$, and the corresponding activation scores $\mathbf{A} = \{A_i\}_{i=1}^N$, we predict a directed communication graph $G_{\mathrm{com}} = (\mathcal{V}, \mathcal{E}_{\mathrm{com}})$ over the role set $\mathcal{V} = \{1, \ldots, N\}$. Each directed edge represents a potential information-flow path used in multi-round multi-agent interaction. Similar to Zhang et al. (2025c), we initialize the graph encoder with an anchor topology, which serves only as a structural prior for message passing.

Task-conditioned graph encoder. To predict the query-specific communication topology, we encode each role node using its competence-aware role representation and activation score. Specifically, we use the activation score A_i to modulate the corresponding role representation:

$$\bar{h}_i = h'_i \odot (1 + \alpha A_i), \tag{11}$$

where α is a scaling coefficient. We then construct the input feature of each role node by concatenating the modulated role representation with the task representation V_{t} and the task-conditioned competence context h_{tcc}:

$$x_i = [\,\bar{h}_i;\, V_{\mathrm{t}};\, h_{\mathrm{tcc}}\,]. \tag{12}$$

Let $\mathbf{X} = \{x_i\}_{i=1}^{N}$ denote the resulting node features. A graph encoder ϕ_θ then performs message passing over the initialized role graph and produces contextualized node representations:

$$\tilde{\mathbf{H}} = \phi_\theta(\mathbf{X}), \tag{13}$$

where $\tilde{\mathbf{H}} = \{\tilde{h}_i\}_{i=1}^{N}$. In this way, topology prediction is jointly conditioned on role semantics, task information, task-conditioned competence context, and role activation strength.

Edge decoding and graph sampling. Based on the encoded node representations, we compute an edge score for each allowable directed edge $(i, j) \in \mathcal{E}_{\text{allow}}$:

$$S_{ij} = \sigma\big(f_{\text{edge}}(\tilde{h}_i, \tilde{h}_j)\big), \tag{14}$$

where f_{edge} is the edge decoder and $S_{ij} \in (0, 1)$ is interpreted as the activation probability of edge (i, j). The query-specific communication graph G_{com} is obtained by selecting edges according to these scores over the allowable edge set. During training, we sample a graph instance $\hat{G}$ from the predicted edge distribution for rollout and optimization. We optimize the expected utility–cost trade-off. Specifically, the utility term is defined as

$$\mathcal{L}_{\text{util}} = -\mathbb{E}_{\hat{G}\sim p_\theta(\cdot|\mathbf{H}',\mathbf{A},V_t,h_{tcc})}\big[u(\hat{G})\big], \tag{15}$$

where $u(\hat{G})$ denotes the utility obtained from the sampled topology $\hat{G}$. The communication cost term is defined as

$$\mathcal{L}_{\text{cost}} = \rho(y)\,\mathcal{C}(S), \tag{16}$$

where $y \in [0, 1]$ is the instance-wise success-rate signal introduced in Section 3.2.2, namely, the empirical success rate of the base model on the current query estimated from repeated trials. We use this signal to modulate the communication penalty and define the communication cost as

$$\rho(y) = \max(y, y_\epsilon),\ \mathcal{C}(S) = \sum_{(i,j)\in\mathcal{E}_{\text{allow}}} S_{ij}. \tag{17}$$

Here $y_\epsilon > 0$ is a small floor constant, and $\mathcal{C}(S)$ corresponds to the expected number of active edges under the predicted edge probabilities. This competence-conditioned weight reduces the relative communication penalty on queries with lower standalone success rates, while the utility term rewards task performance. The overall objective is

$$\mathcal{L}_{\text{graph}}(\theta) = \mathcal{L}_{\text{util}} + \lambda\mathcal{L}_{\text{cost}}, \tag{18}$$

where λ balances task utility and communication cost. The utility term is optimized with REINFORCE, while the cost term backpropagates directly through S.

4 Experiments

4.1 Experimental Setting

Datasets and Metrics. Following prior work on automatic multi-agent system design and orchestration (Zhang et al., 2025c), we evaluate our method on six widely used benchmark datasets: MMLU (Hendrycks et al., 2020), GSM8K (Cobbe et al., 2021), MultiArith (Roy & Roth, 2015), SVAMP (Patel et al., 2021), AQuA (Ling et al., 2017), and HumanEval (Chen et al., 2021). We report the standard evaluation metric for each dataset, using accuracy for MMLU, GSM8K, MultiArith, SVAMP, and AQuA, and Pass@1 for HumanEval. Dataset statistics are reported in Table 6.

Baselines. Following prior work (Zhang et al., 2025c), we compare our method with three groups of baselines. The first group consists of *single-agent methods*, including Chain-of-Thought (CoT) (Wei et al., 2022) and Self-Consistency (Wang et al., 2022). The second group includes *multi-agent systems with fixed collaboration patterns*, such as Chain, Tree, Complete Graph, and Random Graph (Qian et al., 2025), as well as debate-based interaction instantiated by LLM-Debate (Du et al., 2024). The third group covers *multi-agent methods with dynamic or optimized structures*, including AgentPrune (Zhang et al., 2025b), AgentDropout (Wang et al., 2025b), MasRouter (Yue et al., 2025), CARD (Wu et al., 2026), and G-Designer (Zhang et al., 2025c).

Implementation Details We access GPT models via the OpenAI API, primarily using `gpt-4o-2024-08-06` (GPT-4o) as the target backbone. Unless otherwise specified, all LLM calls in evaluation use a decoding temperature of 0. For multi-agent methods, we adopt explicit agent profiling with a predefined pool of textual role descriptions, following classical configurations in prior LLM-based MAS work (Liu et al., 2023; Zhuge et al., 2024; Yin et al., 2023). We additionally include a summarizer agent to aggregate the dialogue history and produce the final answer. Both task queries and role descriptions are encoded using `bge-small-en-v1.5` (Xiao et al., 2023), yielding 384-dimensional embeddings. We train Task-Conditioned Competence Modeling, Role Activation, and Graph-based MAS Generation sequentially offline, keeping the other modules frozen at each stage. Deployment reuses the trained model without further parameter updates. For training the graph-generation module, following Zhang et al. (2025c), we use 40 queries from each dataset for training. We use dataset-specific training and evaluation splits; their definitions and the construction of capability fingerprints and supervision targets are detailed in Appendix A.

4.2 Performance Comparison

The comparison results are presented in Table 1. Under the same backbone setting as all baselines, our method achieves the best average performance of 93.81% across the six benchmarks, establishing state-of-the-art overall results. In Section 4.4, we further examine whether this advantage generalizes across different target backbones, including unseen models.

Table 1: Performance comparison (%) on six benchmarks.

Method	MMLU	GSM8K	AQuA	MultiArith	SVAMP	HumanEval	Average
Vanilla	80.30	82.43	71.16	92.77	86.32	71.56	80.76
CoT	81.64 ↑1.34	86.81 ↑4.38	73.21 ↑2.05	93.12 ↑0.35	87.33 ↑1.01	74.75 ↑3.19	82.81 ↑2.05
SC (CoT)	83.30 ↑3.00	81.86 ↓0.57	75.79 ↑4.63	94.30 ↑1.53	88.44 ↑2.12	79.87 ↑8.31	83.93 ↑3.17
Chain	82.66 ↑2.36	88.28 ↑5.85	74.38 ↑3.22	93.42 ↑0.65	86.95 ↑0.63	81.57 ↑10.01	84.54 ↑3.78
Tree	80.93 ↑0.63	85.28 ↑2.85	71.59 ↑0.43	93.54 ↑0.77	89.09 ↑2.77	80.74 ↑9.18	83.53 ↑2.77
Complete	82.59 ↑2.29	79.76 ↓2.67	72.86 ↑1.70	94.82 ↑2.05	84.28 ↓2.04	79.07 ↑7.51	82.23 ↑1.47
Random	84.12 ↑3.82	87.09 ↑4.66	76.89 ↑5.73	94.14 ↑1.37	87.56 ↑1.24	82.67 ↑11.11	85.41 ↑4.65
LLM-Debate	84.67 ↑4.37	91.47 ↑9.04	77.79 ↑6.63	96.33 ↑3.56	90.12 ↑3.80	85.04 ↑13.48	87.57 ↑6.81
AgentPrune	85.40 ↑5.10	91.32 ↑8.89	80.52 ↑9.36	94.91 ↑2.14	90.17 ↑3.85	86.94 ↑15.38	88.21 ↑7.45
AgentDropout	85.71 ↑5.41	91.47 ↑9.04	81.12 ↑9.96	95.18 ↑2.41	91.41 ↑5.09	86.24 ↑14.68	88.52 ↑7.76
MasRouter	89.54 ↑9.24	94.37 ↑11.94	87.14 ↑15.98	92.14 ↓0.63	85.94 ↓0.38	90.08 ↑18.52	89.87 ↑9.11
CARD	86.93 ↑6.63	94.53 ↑12.10	89.05 ↑17.89	89.29 ↓3.48	86.67 ↑0.35	87.60 ↑16.04	89.01 ↑8.25
G-designer	87.14 ↑6.84	93.76 ↑11.33	81.61 ↑10.45	96.61 ↑3.84	93.18 ↑6.86	88.32 ↑16.76	90.10 ↑9.34
Ours	92.81 ↑12.51	95.00 ↑12.57	87.14 ↑15.98	99.11 ↑6.34	96.25 ↑9.93	92.56 ↑21.00	93.81 ↑13.05

4.3 Ablation Study

Our full method learns its competence representation from two inputs: the task representation V_{t} and the capability fingerprint V_{m}. We evaluate their downstream contribution through three ablated variants. **w/o task** removes V_{t} from the competence module, so its representation is no longer query-conditioned. **w/o model** removes V_{m}, so the module no longer has access to a model-specific capability prior. **w/ desc.** replaces V_{m} with free-form model descriptions (e.g., textual summaries of parameter scale and characteristics).

Table 2: Downstream ablation of competence modeling.

Setting	HumanEval	AQuA	MMLU	Avg.	Tokens / Task
Full	92.56	87.14	92.81	90.84	2360.45
w/o task	90.91	85.71	88.24	88.29 ↓2.55	2758.02 ↑16.84%
w/o model	87.60	87.62	88.24	87.82 ↓3.02	6010.64 ↑154.64%
w/ desc.	88.43	85.24	89.54	87.74 ↓3.10	4736.68 ↑100.67%

Task conditioning improves the performance–cost trade-off. Removing the task representation reduces the average score by 2.55 points and increases token cost by 16.84%. These results support conditioning the competence representation on the current query for downstream orchestration in this setting.

The capability prior helps control collaboration cost. Removing V_{m} causes the largest token increase (+154.64%) and also lowers the average score by 3.02 points. Under our orchestration design, the backbone-specific capability prior thus supports lower token usage together with higher average performance.

Structured fingerprints outperform the evaluated textual descriptions. Replacing V_{m} with textual descriptions drops the average score by 3.10 points and increases token cost by 100.67%. This comparison favors the structured fingerprint over these descriptions as a conditioning signal for orchestration.

4.4 Cross-Backbone Performance and Generalization to Unseen Models

We evaluate our competence-aware collaboration on MMLU with performance-oriented and efficiency-oriented backbones. We mark as unseen (*) backbones used neither as probe-construction anchors nor for competence-supervision data collection. Tables 3 and 4 report accuracy, with Avg. computed across the three backbones in each table. Figure 3 shows agent counts across backbones and MMLU queries.

Table 3: Results on MMLU with performance-oriented target backbones.

Method	GLM 5.1*	Gemini 3.1 Pro*	GPT-4o	Avg.
Vanilla	83.01	81.70	80.30	81.67
G-Designer	88.89	85.62	87.14	87.22
Ours	**89.53**	**87.58**	**92.81**	**89.97** ↓36.26%

On performance-oriented backbones, our method maintains strong performance with substantially lower collaboration cost. As shown in Table 3, our method achieves the best average score on this group, improving over G-Designer from 87.22 to 89.97, while using ↓36.26% fewer tokens per query after deployment. Capability fingerprinting adds a small one-time cost per backbone, which these savings recover after only about 28 queries on average across this group (Appendix A.3). This result shows that performance-oriented backbones can still benefit from our collaboration strategy, while incurring much lower collaboration overhead.

On efficiency-oriented backbones, our method brings larger performance gains. In Table 4, our method improves the average score from 74.07 to 78.65, with the largest gain on Gemma-3-12B. Figure 3 further shows that the number of agents in the generated MAS varies substantially across target models and tasks.

These improvements also hold on unseen LLM backbones. The gains also hold on unseen backbones GLM 5.1* and Gemini 3.1 Pro*. These results support transferring competence-conditioned

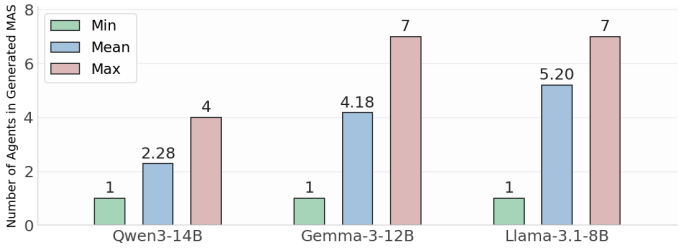


Figure 3: Number of agents in the generated MAS on efficiency-oriented target backbones.

Table 4: Results on MMLU with efficiency-oriented target backbones.

Method	Qwen3-14B	Llama3.1-8B	Gemma-3-12B	Avg.
Vanilla	77.79	63.40	66.67	69.29
G-Designer	84.31	67.32	70.58	74.07
Ours	**87.58**	**70.59**	**77.78**	**78.65**

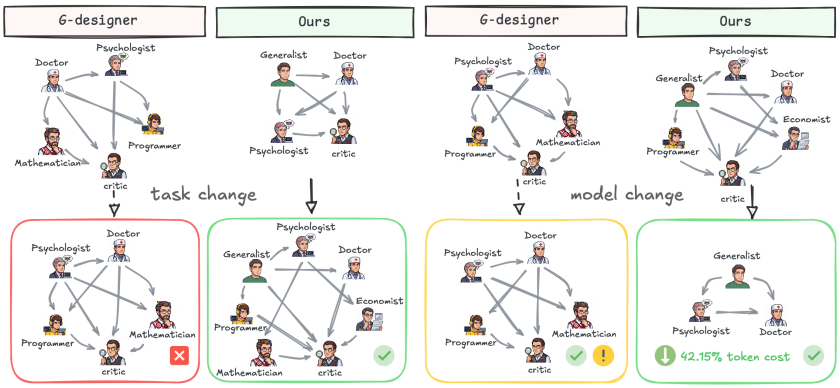


Figure 4: Case studies on MMLU. Left: changing the task while fixing the backbone (Llama-3.1-8B). Right: changing the backbone from Llama-3.1-8B to GPT-4o while fixing the task. check/cross marks indicate answer correctness.

orchestration beyond the backbones used for probe construction and supervision. We further evaluate the trained model on unseen datasets with an additional unseen backbone in Appendix A.4.

4.5 Case Study

We present two MMLU case studies in Figure 4 to show how competence-aware orchestration adapts MAS to different backbone–task settings.

Changing the task while fixing the backbone. We fix the backbone to Llama-3.1-8B and compare two psychology questions from `high_school_psychology` and `professional_psychology`. The latter requires more specialized expertise. Without explicit competence-aware adaptation, G-Designer makes only limited structural changes and fails on the harder question. In this example, our method activates additional roles and returns a correct answer, illustrating query-dependent adaptation of the collaboration structure.

Changing the backbone while fixing the task. We then fix the question and change the backbone from Llama-3.1-8B to GPT-4o. G-Designer continues to instantiate a relatively large collaboration structure. In contrast, our method activates fewer roles with GPT-4o while preserving the correct answer and reducing token cost by 42.2%. This example illustrates backbone-dependent adaptation with lower collaboration cost.

5 Conclusion

In this paper, we study *how to adapt multi-agent collaboration to a backbone's competence on the current query*. We propose a competence-aware framework that learns a task-conditioned representation under success-rate supervision and uses it to guide role activation and communication topology. Experiments across six benchmarks and multiple LLM backbones show that our method improves performance, reduces collaboration cost on stronger backbones, and yields larger gains on weaker ones, with the same trend also observed on unseen backbones.

AI USE STATEMENT

Generative AI tools supported this work, including language polishing and code implementation and debugging. The authors reviewed the AI-assisted materials and take full responsibility for the final manuscript and results.

REFERENCES

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

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

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

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059, 2026.

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

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, 2023.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26473–26501, 2025. doi: 10.18653/v1/2025.acl-long.1285.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, et al. Dspy: compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 17889–17904, 2024.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 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)*, pp. 158–167, 2017.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 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*, pp. 2080–2094, 2021.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pp. 41488–41505, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. Mas^2: Self-generative, self-configuring, self-rectifying multi-agent systems. *arXiv preprint arXiv:2509.24323*, 2025a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 257–279, 2024.

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

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

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

Tongtong Wu, Yanming Li, Ziye Tang, Chen Jiang, Linhao Luo, Guilin Qi, Shirui Pan, and Gholamreza Haffari. Card: Towards conditional design of multi-agent topological structures. In *The Fourteenth International Conference on Learning Representations*, 2026.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

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

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *International Conference on Machine Learning*, pp. 75834–75852. PMLR, 2025a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 75389–75428, 2025b.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *International Conference on Machine Learning*, pp. 76678–76692. PMLR, 2025c.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025d.

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

A Appendix

A.1 Probe Set Construction for Capability Fingerprinting

Training-query pool. Following prior work on automatic MAS design and orchestration (Zhang et al., 2025c;b), which uses $B' \in \{40, 80\}$ training queries per dataset, we adopt the 40-query setting. We randomly sample 40 queries from each of the six benchmark datasets, forming a pool of 240 training-side queries. Probe-set construction and competence-supervision data collection both draw from this pool, without introducing additional distinct task queries. This matches the distinct-query budget, not the offline LLM execution budget.

Table 5: Domain assignment used in probe-set construction.

Domain	Source datasets
`code_logic`	HumanEval
`quantitative_reasoning`	AQuA
`basic_math`	GSM8K, MultiArith, SVAMP
`general_knowledge`	MMLU

Dataset-specific splits. The dataset loader uses the MMLU development split for training queries and the validation split for evaluation queries. For the other five benchmarks, including HumanEval, fixed dataset-specific index files define the training and evaluation subsets. The training and evaluation index sets in these files are disjoint, and the evaluation loader selects only the designated evaluation indices.

Design goals. We construct the probe set to satisfy three requirements. First, it should provide *clear difficulty resolution*, so that backbones of different competence levels can be meaningfully separated rather than collapsing to similar outcomes on the same probe set. Second, it should provide *multi-domain coverage*, so that the resulting capability fingerprint reflects multiple capability dimensions instead of only one narrow skill type. Third, it should remain *compact*, so that one-time fingerprint registration incurs only limited overhead.

Anchor-based difficulty calibration. To obtain a stable and interpretable difficulty scale, we evaluate four anchor backbones with progressively stronger standalone capability—`Llama-3.2-3B-Instruct`, `Llama-3.1-8B-Instruct`, `GPT-4o`, and `GPT-5`—on all 240 sampled queries. For each item x, we record the binary pass vector

$$\mathbf{s}(x) = (s_1, s_2, s_3, s_4), \qquad s_k \in \{0, 1\}, \tag{19}$$

where $s_k = 1$ indicates that the k-th anchor solves x correctly, and $s_k = 0$ otherwise.

We order the four anchors from weaker to stronger and retain only items whose pass patterns are monotone under this ordering. Intuitively, if a weaker anchor succeeds while a stronger anchor fails, the item is not suitable for reliable difficulty calibration and is therefore discarded as ambiguous. We additionally remove all-fail items $(0, 0, 0, 0)$, since such items do not help locate the capability range of the anchors and may be confounded by ambiguity, annotation noise, or excessively difficult instances.

The remaining monotone pass patterns define four relative difficulty levels, ranging from easiest to hardest: items solved by all four anchors $(1, 1, 1, 1)$, items solved by all but the weakest anchor $(0, 1, 1, 1)$, items solved only by the two strongest anchors $(0, 0, 1, 1)$, and items solved only by the strongest anchor $(0, 0, 0, 1)$. This yields an empirically grounded difficulty scale shared across candidate backbones.

Domain stratification. To ensure that the probe set captures multiple capability dimensions, we further partition the candidate items into four coarse domains according to their source datasets:

Balanced sampling and final probe set. Combining the four domains with the four difficulty levels yields 16 domain–difficulty cells. We sample 3 items from each cell, resulting in a final probe set of 48 items. For each domain, the selected probes are ordered from easy to hard as three items from each difficulty level, yielding 12 probes per domain in total. This gives the 4×12 layout used in capability fingerprinting: the four rows correspond to capability domains, and the 12 columns within each row form a shared easy-to-hard scale.

Overall, this construction provides a compact but structured probe set that simultaneously offers clear difficulty resolution, multi-domain coverage, and low one-time registration cost.

Table 6: Dataset descriptions and statistics.

Dataset	Answer Type	Metric	#Test	License
MMLU	Multi-choice	Acc.	153	MIT License
GSM8K	Number	Acc.	1,319	MIT License
MultiArith	Number	Acc.	600	Unspecified
SVAMP	Number	Acc.	1,000	MIT License
AQuA	Multi-choice	Acc.	254	Apache-2.0
HumanEval	Code	Pass@1	164	MIT License

A.2 Proxy Supervision Data Construction for Task-Conditioned Competence Modeling

The goal of task-conditioned competence modeling is to make the fused representation capture how competent a given LLM backbone is expected to be on the current query. Since such query-level competence is not directly annotated, we construct a proxy supervision signal from empirical task performance. The intuition is straightforward: if a backbone succeeds more frequently on a query across repeated trials, it should be regarded as more competent on that query; conversely, a lower success rate indicates lower competence.

We collect this supervision data offline from the same pool of 240 training-side samples used in Appendix A.1. After excluding the 48 samples selected as the probe set, we use the remaining 192 samples to build the proxy supervision data. Thus, the probe set is used only for Capability Fingerprint registration, while the non-probe samples are used to estimate query-level competence targets.

For each backbone m and each non-probe query q, we repeatedly run the backbone $R = 10$ times with decoding temperature set to 0.2, and define the supervision target as the empirical success rate:

$$y_{m,q} = \frac{1}{R} \sum_{r=1}^{R} z_{m,q}^{(r)}, \tag{20}$$

where $z_{m,q}^{(r)} \in \{0, 1\}$ denotes the outcome of the r-th trial. We use a small non-zero temperature to preserve limited stochastic variation across repeated runs while avoiding excessive randomness in the estimated competence targets. For HumanEval, $z_{m,q}^{(r)}$ is the pass/fail result under the benchmark unit tests, so $y_{m,q}$ is an empirical estimate of `pass@1`. For all other benchmarks, $z_{m,q}^{(r)}$ is the standard correctness indicator, and $y_{m,q}$ therefore corresponds to the empirical accuracy on query q.

To ensure that the supervision data covers diverse competence patterns, we construct it using 14 LLM backbones with varying scales, capability levels, and specialization characteristics, including general-purpose, code-specialized, math-specialized, and reasoning-oriented models (Table 7). GPT-4o serves as a probe-construction anchor (Appendix A.1), but is not among these supervision backbones.

For each backbone, we first obtain its Capability Fingerprint from the 48-sample probe set. We then pair this fingerprint with each non-probe query and its corresponding target $y_{m,q}$, yielding supervised instances of the form

$$\big(\mathrm{Fingerprint}(m),\ q,\ y_{m,q}\big). \tag{21}$$

To avoid an overly skewed target distribution and improve the relative coverage of different competence levels, we perform distribution-aware sampling over these candidate instances with respect to $y_{m,q}$. The resulting training set for task-conditioned competence modeling contains 1060 instances.

These proxy supervision data are used to train task-conditioned competence modeling (§3.2.2) and provide the supervision basis for downstream role activation (§3.3.1). By supervising the fused representation against empirical query-level success outcomes, the model is encouraged to encode competence-relevant information that can directly support role selection and communication graph construction.

Evaluation of the auxiliary prediction head. We evaluate the auxiliary success-rate prediction head used to supervise task-conditioned competence representation learning. We use query-disjoint

Table 7: LLM backbones used for proxy-supervision data construction.

Family	Backbones
Phi	`Phi-3.5-mini-instruct`
Qwen2.5	`Qwen2.5-Coder-7B-Instruct`, `Qwen2.5-Math-7B-Instruct`
DeepSeek	`deepseek-v3.2`
Gemma	`gemma-3-12b-it`
GPT	`gpt-4o-mini`, `gpt-5`
Llama	`llama-3.1-8b-instruct`, `llama-3.2-1b-instruct`, `llama-3.2-3b-instruct`
Qwen3	`qwen3-1.7b`, `qwen3-14b`, `qwen3-32b`
QwQ	`qwq-32b`

training and evaluation partitions of the existing success-rate supervision data. All backbone–query pairs sharing the same query are assigned to a single partition, so no evaluation query appears in training or validation. We train the Stage-I module with BCE supervision and select checkpoints using validation MAE. The evaluation covers 153 backbone–query pairs spanning 28 held-out queries from backbones and domains represented during training. Across three training seeds, the predicted success rates achieve a pooled Spearman correlation of 0.629 ± 0.022 with empirical success rates and an MAE of 0.234 ± 0.010 on the $[0, 1]$ scale (mean $\pm$ standard deviation across training seeds). The prediction task provides auxiliary supervision for representation learning, while downstream orchestration is conditioned on the learned representation rather than the scalar success-probability estimate.

A.3 Amortized Cost of Capability Fingerprinting

One-time registration cost. Our efficiency comparison measures deployment-time LLM token usage separately from offline supervision collection and model training. The trained orchestration model is reused without further parameter updates. A new backbone requires a one-time fingerprint registration: evaluating it on the fixed 48-item probe set to obtain V_m, without repeating supervision collection or model training. We measure this registration cost for the three performance-oriented backbones in Section 4.4: `Gemini 3.1 Pro`, `GLM 5.1`, and `GPT-4o` (Table 8). The amortization below concerns this registration cost, not offline training.

Average cost recovery. In Section 4.4, the average online token usage is 3953.94 tokens/query for G-Designer and 2520.26 tokens/query for our method, yielding an average saving of

$$\Delta_{\text{save}} = 3953.94 - 2520.26 = 1433.68 \tag{22}$$

tokens per query. Since this saving is averaged over the three performance-oriented backbones, we use it only to compute an average amortization point. Specifically, the average one-time fingerprinting cost across the three backbones is 39,840 tokens, which corresponds to 830.0 tokens per probe over the 48-item probe set. Therefore, the average cost-recovery point is

$$N_{\text{break-even}} = \frac{39840}{1433.68} = 27.79. \tag{23}$$

That is, the one-time cost of obtaining V_m is recovered after only 27.79 queries on average. These results suggest that the probe set remains compact in practice and that the additional registration cost of capability fingerprinting can be amortized quickly in realistic multi-query deployment settings.

A.4 Generalization to Harder Unseen Benchmarks with Gemma4

We further evaluate our trained model on two unseen datasets, GPQA Diamond (Rein et al., 2023) and BBEH-mini (Kazemi et al., 2025), with an unseen LLM backbone, Gemma4. Table 9 reports task performance and collaboration cost.

Table 8: One-time token cost of Capability Fingerprinting for the three performance-oriented backbones in Section 4.4.

Backbone	Total Tokens	Avg. Tokens / Probe
`Gemini 3.1 Pro`	28,576	595.3
`GLM 5.1`	78,716	1640.0
`GPT-4o`	12,228	254.7
Average	39,840	830.0

Table 9: Results on harder unseen benchmarks with Gemma4. Accuracy is reported in percent. Green annotations denote token savings relative to G-Designer for the same benchmark and backbone.

Benchmark	Backbone	Method	Acc.
GPQA	Gemma4	Vanilla	59.60
		G-Designer	72.73
		Ours	**75.76** ↓33.6%
BBEH-mini	Gemma4	Vanilla	30.22
		G-Designer	45.43
		Ours	**47.61** ↓33.6%

Results. In the comparisons reported in Table 9, Ours improves accuracy over G-Designer on Gemma4 by 3.03 points on GPQA and 2.18 points on BBEH-mini, while reducing tokens by 33.6% on both benchmarks. These results support the transfer of competence-conditioned orchestration to these two unseen benchmarks with Gemma4 without additional training data.

A.5 Anonymous Code Repository

An anonymized code repository is available at Competence-Aware Orchestration for LLM-based Multi-Agent Systems.