PAGO: PROMPT AUGMENTED GRAPH ORGANIZATION FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph-based multi-agent language-model systems usually represent agents as nodes and communication channels as edges. This abstraction captures who communicates, but leaves the system's behavioral organization implicit: which operations are required, who should perform them, how they depend on one another, and how intermediate messages should be handled. We introduce PAGO (Prompt Augmented Graph Organization), a framework that extends multi-agent organization from topology design to typed, executable graph composition. PAGO treats prompts as first-class graph primitives, jointly modeling agent and instruction nodes, their bindings and dependencies, communication edges, and message protocols. For each query, PAGO retrieves reusable primitives, selects a task-specific graph under legality and resource constraints, and materializes it into executable agents, prompts, communication channels, and schedules while keeping the backbone language model frozen. Across seven benchmarks, PAGO with DeepSeek-v4-pro achieves a seven-task macro-average of 81.58% versus 77.64% for the strongest non-PAGO baseline. Further analyses show task-dependent graph structures, robustness to candidate noise, and cross-backbone transfer, demonstrating that explicit prompt-level organization enables adaptive and executable multi-agent systems rather than optimizing agent connectivity alone.

1 INTRODUCTION

What constitutes an executable organization for a multi-agent language-model system? Multi-agent systems (MAS) increasingly solve tasks by coordinating specialized agents through role-based interaction, tools, and structured workflows (Li et al., 2023; Wu et al., 2023; Hong et al., 2023; Chen et al., 2023; Qian et al., 2024). A common abstraction represents such a system as a graph: agents form the nodes, and communication channels form the edges. This view has motivated methods that optimize computational graphs, prune redundant communication, remove unnecessary agents, and generate task-aware topologies (Zhuge et al., 2024; Zhang et al., 2025a; Wang et al., 2025; Zhang et al., 2025b; Li et al., 2026; Yun et al., 2026; Wang et al., 2026). These methods establish agent participation and connectivity as important design variables, but an executable organization must specify more than participants and communication for each query.

An interaction graph does not uniquely determine the computation it executes. Consider two systems with the same solver, verifier, and directed edge. One asks the solver to derive an answer and the verifier to check the completed derivation; the other asks the verifier to diagnose an execution failure and return instructions for revision. Their topologies are identical, yet their operations, dependencies, and message semantics define different procedures. When these choices remain inside role prompts or hand-written orchestration, they cannot be selected jointly with the graph. The resulting topology is a communication scaffold whose executable behavior still depends on external configuration. We refer to this mismatch between the represented graph and the realized computation as the *organization gap* in current MAS.

Closing this gap requires four coupled capabilities. An organization must select the participants needed for the query, compose the atomic operations that form its solution procedure, assign and order those operations, and specify how intermediate messages are interpreted. These capabilities cannot be safely separated: weak participant selection wastes calls, missing dependencies produce invalid execution orders, and an underspecified message can change verification into unstructured

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.

discussion. The required structure also varies across queries. Arithmetic may need derivation and checking; proofs may require decomposition, independent reasoning, and synthesis; code generation may require execution feedback and repair. A single fixed workflow cannot express these variations without either omitting necessary operations or activating unnecessary ones.

Existing research addresses parts of this problem from two directions. Graph-based organization optimizes agents and communication structure, but typically leaves local behaviors implicit. Prompting and language-model program optimization improve reasoning trajectories, instructions, or modular pipelines (Wei et al., 2022; Wang et al., 2023; Yao et al., 2023; Zhou et al., 2023; Yang et al., 2023; Khattab et al., 2023), but do not by themselves assign operations to agents or guarantee a valid multi-agent runtime. Connecting the two directions raises a central question: *how can reusable prompt-level behaviors be composed with agent-level structures for each query while preserving executability and resource control?* Answering it requires a typed representation, task-conditioned composition, structural constraints during selection, and a graph-to-runtime mapping that remains valid without post hoc workflow repair.

We introduce **PAGO** (**Prompt Augmented Graph Organization**), which treats prompts as first-class organizational primitives. *Agent Prompt Nodes* specify roles, and *Instruction Prompt Nodes* specify independently selectable operations. *Binding*, *Dependency*, and *Communication Edges* encode assignment, execution order, and protocol-governed information transfer. For each query, PAGO retrieves a high-recall candidate graph from a fixed Prompt Primitive Bank and selects nodes, edges, and protocols using a query-conditioned policy. Legality masks enforce endpoint, assignment, dependency, reachability, protocol, output-path, and resource constraints. A deterministic materializer compiles the selection into agents, system prompts, channels, and schedules. Only the organization policy learns; the backbone remains frozen. PAGO thus jointly composes structure and behavior without generating unconstrained orchestration code at inference. The learned decisions have explicit runtime meanings: instructions specify what to do, bindings determine who does it, and dependencies and protocols determine when and how outputs are consumed. The fixed bank preserves these meanings across queries while the policy adapts their composition, making adaptation observable in both performance and organization. This separation also makes an organization's execution requirements explicit: an instruction requires a capable owner, and a cross-agent dependency requires a path for its output. These conditions can be checked before invoking the backbone. They establish structural executability rather than answer correctness; task feedback is still needed to learn which valid compositions are useful under the available resource budget.

Across seven benchmarks, PAGO with DeepSeek-v4-pro achieves a seven-task macro-average of 81.58%, compared with 77.64% for the strongest non-PAGO baseline. Further experiments examine the effect of individual design components, task-dependent graph structure, robustness to candidate noise, and transfer to Qwen3-14B. We make three contributions:

- **A typed, executable organization abstraction.** PAGO elevates prompts from hidden agent attributes to **independently selectable graph primitives** that jointly represent agents, instructions, bindings, dependencies, communication links, and protocols. This separation exposes participation, behavior, and execution structure within one typed graph. Consequently, identical agent topologies can express different computations while remaining structurally checkable before runtime.
- **Legality-aware, query-conditioned organization learning.** PAGO retrieves reusable primitives and jointly selects nodes, edges, and protocols through a query-conditioned policy. **Legality masks and deterministic materialization** enforce assignment, dependency, reachability, protocol, and resource constraints, then compile each selected graph into executable prompts, communication channels, and schedules without updating the backbone language model, preserving the selected organization specification.
- **Evaluation beyond aggregate task accuracy.** Across seven benchmarks, PAGO achieves **a seven-task macro-average of 81.58% and a 3.94-point gain** over the strongest reported non-PAGO baseline. Component ablations and controlled studies additionally examine task-dependent graph structure, candidate-noise robustness, and transfer to Qwen3-14B, testing both effectiveness and organizational adaptation across the reported conditions.

2 PRELIMINARIES

2.1 PROBLEM FORMULATION AND NOTATION

We consider query-conditioned organization of a multi-agent language-model system. Let q denote an input query and y its reference answer, used to evaluate the system's prediction. An organization is represented as

$$\mathcal{G}(q) = \big(\mathcal{V}(q), \mathcal{E}(q), \mathcal{A}(q)\big), \tag{1}$$

where $\mathcal{V}(q)$ contains organizational units, $\mathcal{E}(q)$ specifies their assignment, dependency, and communication relations, and $\mathcal{A}(q)$ collects attributes such as roles, instructions, tools, and message protocols. Thus, the decision concerns both the participating units and the behavior and coordination assigned to them. Organizations with the same participants and communication links may therefore implement different computations through their attributes and execution relations.

Let $\pi_\Theta(\mathcal{G} \mid q)$ be an organization policy with trainable parameters Θ. Its output must satisfy $\mathrm{Valid}(\mathcal{G}) = 1$, meaning that the organization is executable and respects the prescribed structural and resource constraints. Validity concerns whether operations have appropriate owners, required inputs can reach their consumers, and execution stays within the available budget; it does not guarantee a correct answer. Let $\hat{y}$ denote the answer predicted by executing the admissible organization $\mathcal{G}(q)$ on query q with backbone parameters θ_{LLM}, and let C_{run} denote the incurred runtime cost. The net task utility is

$$\widehat{R}(q, \mathcal{G}) = R(y, \hat{y}) - C_{\mathrm{run}}(\mathcal{G}), \tag{2}$$

where $R(y, \hat{y})$ measures answer quality and C_{run} combines weighted token, latency, and model-call costs incurred by executing $\mathcal{G}$.

The learning problem is to maximize expected net utility over queries and policy-selected valid organizations. The policy conditions on q, while y supplies evaluation feedback rather than an input to organization selection. The objective balances answer quality against the resources used to obtain it. Throughout learning and evaluation, θ_{LLM} remains frozen; only the organization policy is learned. The next section specifies the representation, selection procedure, and optimization used to solve this problem.

2.2 RELATED WORK

Prompt optimization and multi-agent systems. Chain-of-thought and self-consistency improve reasoning (Wei et al., 2022; Wang et al., 2023); APE and OPRO optimize instructions (Zhou et al., 2023; Yang et al., 2023), and DSPy compiles modular pipelines (Khattab et al., 2023). These approaches optimize instructions or modular calls but do not determine which agent owns an operation, when it runs, or how its output is communicated. CAMEL studies role interaction (Li et al., 2023), AutoGen supports event-driven conversations (Wu et al., 2023), AgentVerse provides a collaboration framework (Chen et al., 2023), and MetaGPT and ChatDev coordinate specialized software roles (Hong et al., 2023; Qian et al., 2024). They establish role-based division of labor, yet roles and interaction rules are generally chosen before inference. PAGO unifies these concerns by making operations, assignments, dependencies, and protocols jointly selectable for each query.

Adaptive graphs and automated organization design. DyLAN selects teams (Liu et al., 2024), GPTSwarm optimizes agent graphs (Zhuge et al., 2024), and AgentPrune, AgentDropout, and G-Designer adapt messages, agents, or topologies (Zhang et al., 2025a; Wang et al., 2025; Zhang et al., 2025b). Together, these methods show that team membership and connectivity need not remain fixed. ADAS searches code-defined designs (Hu et al., 2024), AFlow searches workflows (Zhang et al., 2024), and ARG-Designer, Graph-of-Agents, and MAS^2 generate organizations (Li et al., 2026; Yun et al., 2026; Wang et al., 2026). These systems automate larger portions of organization design, although operations and message conventions can remain embedded in generated code or role text. PAGO instead exposes reusable agents, instructions, bindings, dependencies, links, and protocols as typed variables under shared legality constraints, then materializes each valid selection deterministically. This makes the organization checkable before it consumes model calls or modifies downstream task state. Fixing the added variables recovers the conventional agent-node and communication-edge view.

3 METHODOLOGY

Overview. PAGO addresses Section 2.1 by making the behaviors a query needs explicit, assigning them to agents, and turning their coordination into an executable system. Reusable primitives supply the building blocks; query-conditioned selection composes them under legality and resource constraints. Execution then supplies feedback for learning better compositions (Figure 1).

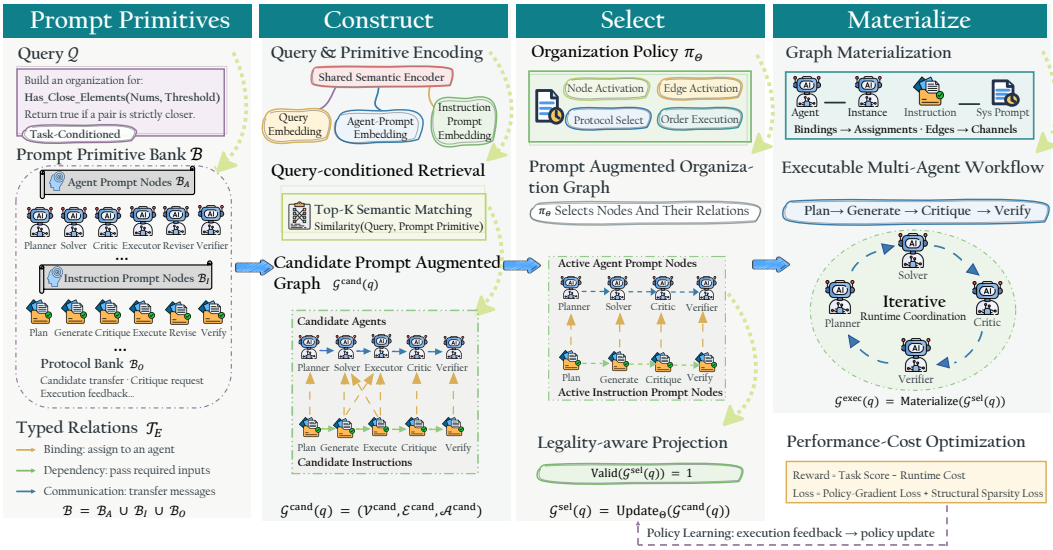


Figure 1: Overview of Prompt Augmented Graph Organization. PAGO retrieves typed prompt primitives, selects a valid query-conditioned organization, materializes it into an executable multi-agent system, and learns the organization policy from task utility and execution cost.

3.1 PROMPT PRIMITIVE BANK AND ORGANIZATION GRAPH

Reusable primitive bank. Query-specific organization requires reusable building blocks whose roles and runtime effects are defined before selection. PAGO stores these blocks in the Prompt Primitive Bank, a fixed catalog $\mathcal{B} = \mathcal{B}_A \cup \mathcal{B}_I \cup \mathcal{B}_O$, whose three disjoint collections contain agent, instruction, and communication-protocol primitives, respectively. Agent and instruction primitives supply candidate nodes; protocol primitives specify message handling on communication edges. Each stored primitive has an identifier, a type, prompt content, input and output interfaces, required capabilities, and a deterministic materialization rule. The bank's contents remain fixed during policy training and evaluation: learning changes their selection and composition. Appendix A details the primitive schema, bank construction, and compatibility rules.

Typed prompt primitives. Composing these building blocks requires explicit ownership, ordering, and communication. PAGO captures these relations alongside the components they connect. A Prompt Node v^P carries declarative content and a materialization rule; a Prompt Edge e^P carries a typed relation and, when required, a relation-specific protocol:

$$\begin{aligned} v^P &= (\tau_v, z_v, \phi_v), \qquad \tau_v \in \mathcal{T}_V = \{\texttt{agent}, \texttt{instruction}\}, \\ e^P &= (u, v, \tau_e, o_e), \qquad \tau_e \in \mathcal{T}_E = \{\texttt{bind}, \texttt{depend}, \texttt{communicate}\}. \end{aligned} \tag{3}$$

Here, τ_v and τ_e are node and edge types drawn from the type sets $\mathcal{T}_V$ and $\mathcal{T}_E$. The node prompt is z_v, and ϕ_v specifies its deterministic runtime realization; u and v are edge endpoints, and o_e is an optional edge prompt or message protocol. Agent nodes define executable roles and capabilities, while instruction nodes define independently selectable local operations. Binding edges assign operations, dependency edges order them, and communication edges carry their outputs between agents under the selected protocol.

Unified organization graph. These node and edge types bring participants, operations, and their coordination into one representation. The resulting heterogeneous declarative graph instantiates the

generic organization $\mathcal{G}$ as

$$\mathcal{G} = (\mathcal{V}_A \cup \mathcal{V}_I, \mathcal{E}_B \cup \mathcal{E}_D \cup \mathcal{E}_C, \mathcal{A}), \tag{4}$$

The subscripts A, I, B, D, and C denote agent nodes, instruction nodes, binding edges, dependency edges, and communication edges, respectively. The attribute collection $\mathcal{A}$ stores node prompts, materialization rules, and edge protocols. Query dependence is suppressed here for readability. Each active element of this declarative organization has a prescribed runtime effect. If instruction nodes, binding edges, and dependency edges are removed and all communication edges share one protocol, Eq. (4) reduces to the conventional agent-node and communication-edge representation.

3.2 Prompt Augmented Graph Organization Learning

The representation above specifies what an organization can express. We now choose which composition should answer each query, progressing from available candidates to a valid selection and then to execution:

1. **Construct the candidate graph $\mathcal{G}^{\mathrm{cand}}(q)$.** Retrieve query-relevant roles and operations, then enumerate compatible relations. This supplies alternatives without committing to activation. Protocol primitives remain available for later communication decisions.
2. **Select the organization $\mathcal{G}^{\mathrm{sel}}(q)$.** Choose participants, operations, relations, and protocols in semantic dependency order. Legality masks constrain each choice so the completed graph specifies a valid organization within the resource budget.
3. **Materialize the runtime system $\mathcal{G}^{\mathrm{exec}}(q)$.** Compile the selected organization into agents, prompts, schedules, and message channels. Execution with the frozen backbone θ_{LLM} produces $\hat{y}$ while preserving the declared runtime behavior.

Execution feedback trains selection, while a structural penalty encourages compact organizations. Below, superscripts cand, sel, and exec track these stages for graphs and element sets; subscripts identify types. Unmarked $\mathcal{G}$ denotes a generic organization.

Query-conditioned candidate construction. Construction first narrows the search to relevant roles and operations, retrieving each type separately before adding compatible relations. A frozen encoder Enc maps q and the prompt content z_p of each node primitive $p \in \mathcal{B}_A \cup \mathcal{B}_I$ into embeddings h_q and h_p. For each type $t \in \{A, I\}$, PAGO retrieves up to K_t primitives by cosine similarity:

$$\begin{aligned} \big(h_q, h_p\big) &= \big(\mathrm{Enc}(q), \mathrm{Enc}(z_p)\big), \\ \mathcal{V}_t^{\mathrm{cand}}(q) &= \mathrm{TopK}_{K_t}\left\{ \frac{h_q^\top h_p}{\|h_q\|_2 \|h_p\|_2} : p \in \mathcal{B}_t \right\}. \end{aligned} \tag{5}$$

Here, TopK_{K_t} returns the candidate nodes corresponding to the K_t highest-scoring primitives, giving $\mathcal{V}_t^{\mathrm{cand}}(q)$; $\mathcal{V}^{\mathrm{cand}} = \mathcal{V}_A^{\mathrm{cand}} \cup \mathcal{V}_I^{\mathrm{cand}}$ is the combined candidate node set. Type and capability compatibility determine candidate binding, dependency, and communication edges $\mathcal{E}^{\mathrm{cand}}$. Together with their inherited attributes $\mathcal{A}^{\mathrm{cand}}$, they form $\mathcal{G}^{\mathrm{cand}}(q) = (\mathcal{V}^{\mathrm{cand}}(q), \mathcal{E}^{\mathrm{cand}}(q), \mathcal{A}^{\mathrm{cand}}(q))$. Protocols from $\mathcal{B}_O$ are selected after communication-edge activation. This graph defines the choices available to the policy; selection determines which components proceed to execution.

Typed organization decisions. Retrieval supplies alternatives; the organization policy must choose a useful combination by activating nodes and edges and selecting communication protocols. For each candidate node v and edge $e = (u, v)$, $g_v, g_e \in \{0, 1\}$ indicate activation. For an active communication edge, $o_e \in \mathcal{B}_O$ denotes its protocol. PAGO parameterizes these decisions as

$$\begin{aligned} p_\Theta(g_v = 1 \mid q) &= \sigma(f_{\Theta_V}(h_q \parallel h_v)), \\ p_\Theta(g_e = 1 \mid q) &= \sigma(f_{\Theta_E}(h_q \parallel h_u \parallel h_v \parallel h_{\tau_e})), \\ p_\Theta(o_e \mid e, q) &= \mathrm{Softmax}(W_o[h_q \parallel h_u \parallel h_v]). \end{aligned} \tag{6}$$

Here, h_u and h_v are the endpoint node embeddings, h_{τ_e} is an edge-type embedding, $\parallel$ denotes concatenation, and σ is the sigmoid function. The node and edge scoring functions f_{Θ_V} and f_{Θ_E} and protocol projection W_o are parameterized by the organization-policy parameters Θ. Softmax gives a distribution over protocol choices. The choices follow the semantic dependency order

$\mathcal{S} = (V^A, V^I, E^B, E^D, E^C, O)$, denoting the stages for agent, instruction, binding, dependency, communication, and protocol decisions. This order lets each later choice be checked against the structure already established, which is essential for the constrained selection step below.

Legality-aware selection. A high-scoring combination can still be invalid: an instruction may lack an owner, or its dependencies may form a cycle. PAGO therefore filters each choice before sampling, using the partial organization built by earlier decisions. Write the kth decision as $a_k \in \mathcal{A}_k$, where $\mathcal{A}_k$ is its action set, $a_{<k}$ denotes previous decisions, and $\mathcal{G}^{\mathrm{sel}}_{k-1}$ is the organization after $k-1$ decisions. The operator $\oplus$ applies an action to that organization. A binary legality mask m_k, expressed using the indicator $\mathbb{I}$, removes invalid choices before sampling and renormalizes the remaining probability mass:

$$
\begin{aligned}
m_k(a; \mathcal{G}^{\mathrm{sel}}_{k-1}) &= \mathbb{I}\big[\mathrm{Valid}(\mathcal{G}^{\mathrm{sel}}_{k-1} \oplus a) = 1\big], \\
\widetilde{p}_\Theta(a_k \mid q, a_{<k}) &= \frac{m_k(a_k; \mathcal{G}^{\mathrm{sel}}_{k-1}) p_\Theta(a_k \mid q, a_{<k})}{\sum_{a' \in \mathcal{A}_k} m_k(a'; \mathcal{G}^{\mathrm{sel}}_{k-1}) p_\Theta(a' \mid q, a_{<k})}.
\end{aligned} \tag{7}
$$

For a partial graph, validity checks whether the update can still form an executable organization; for a completed graph, all constraints must hold. The masked distribution is denoted by $\widetilde{p}_\Theta$. Validity requires active edge endpoints, an executable owner for each active instruction, acyclic instruction dependencies, a reachable path for cross-agent inputs, protocol–capability compatibility, an output path, and compliance with the resource budget. The sequence defines $\mathcal{G}^{\mathrm{sel}}(q) = \mathrm{Update}_\Theta(\mathcal{G}^{\mathrm{cand}}(q))$ and satisfies $\mathrm{Valid}(\mathcal{G}^{\mathrm{sel}}(q)) = 1$; the checks are specified in Appendix B.

Deterministic materialization. Selection establishes who performs each operation and how information flows. Materialization turns that specification into concrete prompts, agents, and execution order through a type-directed compiler. For each active agent v_a, its assigned instruction nodes are topologically ordered, serialized with the role prompt and incoming protocols, and used to instantiate the runtime agent:

$$
\begin{aligned}
\mathrm{Seq}(v_a) &= \mathrm{TopoSort}\big(\mathcal{V}^{\mathrm{sel}}_{I,a}, \mathcal{E}^{\mathrm{sel}}_{D,a}\big), \\
s_a &= \mathrm{Serialize}\big(z_a, \mathrm{Seq}(v_a), \mathcal{C}^{\mathrm{in}}(v_a)\big), \\
v_a^{\mathrm{exec}} &= \mathrm{Instantiate}(M_a, T_a, s_a).
\end{aligned} \tag{8}
$$

Here, $\mathcal{V}^{\mathrm{sel}}_{I,a}$ contains instructions assigned to agent v_a, $\mathcal{E}^{\mathrm{sel}}_{D,a}$ contains their local dependency edges, and $\mathrm{Seq}(v_a)$ is their topological order. The agent's role prompt z_a and incoming protocol information $\mathcal{C}^{\mathrm{in}}(v_a)$ are serialized with that sequence to produce its system prompt s_a. M_a is the agent's backbone model, T_a its available tool set, and v_a^{exec} the instantiated runtime agent. Cross-agent dependencies impose agent-level precedence. The resulting system is $\mathcal{G}^{\mathrm{exec}}(q) = \mathrm{Materialize}(\mathcal{G}^{\mathrm{sel}}(q))$; serialization and iterative execution are detailed in Appendix C.

Performance-cost optimization. Executability alone does not establish usefulness. To improve selection, PAGO evaluates the answer $\hat{y}$ from $\mathcal{G}^{\mathrm{exec}}(q)$ against the resources consumed, closing the loop from execution to learning. We evaluate the utility in Eq. (2) with token, latency, and model-call costs denoted by C_{tok}, C_{lat}, and C_{call}, and corresponding weights λ_{tok}, λ_{lat}, and λ_{call}:

$$
\begin{aligned}
C_{\mathrm{run}}(\mathcal{G}^{\mathrm{exec}}) &= \lambda_{\mathrm{tok}} C_{\mathrm{tok}} + \lambda_{\mathrm{lat}} C_{\mathrm{lat}} + \lambda_{\mathrm{call}} C_{\mathrm{call}}, \\
\widehat{R}(q, \mathcal{G}^{\mathrm{sel}}) &= R(y, \hat{y}) - C_{\mathrm{run}}(\mathcal{G}^{\mathrm{exec}}).
\end{aligned} \tag{9}
$$

This net reward evaluates the complete organization and trains the decisions that produced it. For the full sequence $a_{1:K}$ of K decisions, the masked organization policy $\widetilde{\pi}_\Theta$ satisfies $\log \widetilde{\pi}_\Theta(a_{1:K} \mid q) = \sum_{k=1}^{K} \log \widetilde{p}_\Theta(a_k \mid q, a_{<k})$. PAGO combines the corresponding policy-gradient objective with a typed structural penalty:

$$
\begin{aligned}
\mathcal{L}_{\mathrm{PG}} &= -\mathbb{E}\big[(\widehat{R} - b(q)) \log \widetilde{\pi}_\Theta(a_{1:K} \mid q)\big], \\
\mathcal{L}_{\mathrm{struct}} &= \sum\nolimits_{t \in \mathcal{T}} \beta_t \sum\nolimits_{x \in \mathcal{X}_t^{\mathrm{cand}}} \widetilde{g}_x, \\
\mathcal{L}_{\mathrm{PAGO}} &= \mathcal{L}_{\mathrm{PG}} + \mathcal{L}_{\mathrm{struct}}.
\end{aligned} \tag{10}
$$

Here, the expectation is over training queries and sampled decisions, and $b(q)$ is a query-conditioned reward baseline. For structural type $t \in \mathcal{T} = \{A, I, B, D, C\}$, $\mathcal{X}_t^{\mathrm{cand}}$ is its candidate node or

Table 1: Overall results (%) on seven benchmarks; higher is better. ΔVan. is the macro-average gain over Direct Prompt. Green marks the best and second-best shared-backbone scores.

	Reasoning			Code		Knowledge/Web			
Method	MMLU	GSM8K	MATH	HumanEval	MBPP	HotpotQA	BrowseComp+	Avg.	ΔVan.
Direct Prompt	83.87	95.26	63.42	85.31	73.42	68.57	12.98	68.98	–
CoT	84.39	96.19	66.13	86.79	74.98	70.05	15.43	70.57	↑ 1.59
Self-Consistency	86.12	96.84	67.81	89.63	77.29	72.18	13.85	71.96	↑ 2.98
AgentDropout	87.21	95.82	64.15	88.19	75.04	71.49	13.12	70.72	↑ 1.74
AgentPrune	87.95	96.03	65.28	88.74	75.89	71.92	13.34	71.31	↑ 2.33
GPTSwarm	87.38	96.47	66.02	90.15	77.36	75.28	14.89	72.51	↑ 3.53
G-Designer	88.41	96.91	68.84	89.68	78.52	76.81	14.26	73.35	↑ 4.37
ARG-Designer	88.92	97.15	70.19	91.24	79.16	79.03	15.61	74.47	↑ 5.49
GoA (TopK=3)	89.03	97.28	71.95	91.82	79.63	79.84	15.82	75.05	↑ 6.07
MAS^2	88.76	97.39	71.22	95.31	83.92	88.15	18.74	77.64	↑ 8.66
PAGO-DeepSeek	**91.07**	**98.45**	**76.79**	**96.91**	**86.91**	**92.13**	**28.78**	**81.58**	↑ 12.60
PAGO-QWEN	89.61	97.78	73.25	95.89	85.01	89.61	22.41	79.08	–
PAGO-GPT5.5	91.84	98.97	77.98	97.21	87.19	95.21	35.12	83.36	–

edge set, $\widetilde{g}_x$ is the relaxed activation of candidate x, and β_t controls the corresponding sparsity penalty. The type labels A, I, B, D, C have the meanings defined in Section 3.1. The objective rewards useful executable graphs while penalizing unnecessary runtime and structural activation. Appendix D specifies the discrete relaxation and policy-update procedure used to optimize Eq. (10) and train the task-conditioned organization policy before deployment.

4 Experiments

We organize the evaluation around five questions. RQ1 measures overall effectiveness across reasoning, code, knowledge, and web tasks; RQ2 separates the contribution of prompt augmented representation from query-conditioned organization; RQ3 examines how the selected graph changes across task families; RQ4 tests robustness to irrelevant candidate primitives; and RQ5 studies one direction of cross-backbone transfer. This progression moves from overall performance to representation, structure, robustness, and transfer without repeating the same comparison in multiple forms.

4.1 Experimental Setup

Tasks and metrics. We evaluate MMLU, GSM8K, MATH, HumanEval, MBPP, HotpotQA, and BrowseComp+, spanning knowledge, mathematical reasoning, code generation, multi-hop question answering, and web research. All task scores are reported as percentages, and the primary summary is the equally weighted macro average

$$\bar{S}_m = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} S_{m,d}, \qquad |\mathcal{D}| = 7, \tag{11}$$

where $S_{m,d}$ denotes the recorded score of method m on task d. Recomputed averages match the reported values up to displayed precision.

Baselines and controls. Single-model baselines are Direct Prompt, chain-of-thought (CoT), and Self-Consistency. Organization baselines are AgentDropout, AgentPrune, GPTSwarm, G-Designer, ARG-Designer, GoA (TopK=3), and MAS^2. The shared-backbone comparison uses DeepSeek-v4-pro; additional PAGO results use Qwen3-14B and GPT-5.5. RQ2 uses five restricted PAGO variants, RQ4 tests nine methods under candidate noise, and RQ5 evaluates policy transfer to Qwen3-14B.

4.2 RQ1: Overall Effectiveness

Table 1 reports a macro-average of 81.58% for PAGO across all seven evaluated tasks in the reported comparison: 12.60 points above Direct Prompt and 3.94 above MAS^2, the strongest reported base-

line in this shared-backbone setting. MAS^2 margins are 2.31 on MMLU, 1.06 on GSM8K, 5.57 on MATH, 1.60 on HumanEval, 2.99 on MBPP, 3.98 on HotpotQA, and 10.04 on BrowseComp+.

Figure 2 summarizes Vanilla gains. PAGO's largest observed gains are on HotpotQA (+23.56), BrowseComp+ (+15.80), MBPP (+13.49), and MATH (+13.37), versus +3.19 on GSM8K. The larger gains on multi-hop question answering, web research, and code generation suggest that organizing intermediate operations and their coordination can matter beyond selecting participants. The smaller GSM8K gain also occurs against a high Direct Prompt score of 95.26. This pattern is consistent with the role of assignments, dependencies, and protocols, but aggregate results cannot tie gains to one graph decision without query-level traces and controlled component interventions.

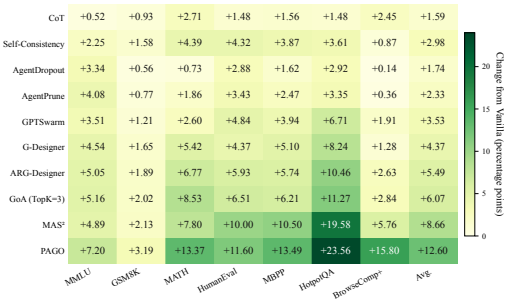


Figure 2: Performance gains over Direct Prompt (Vanilla) across seven benchmarks.

4.3 RQ2: Representation and Organization Learning

RQ2 expands the organization space incrementally through six ordered stages. In this progression, Agent-Only scores 69.03; a fixed prompt instruction raises it to 70.87; instruction nodes with binding and dependency edges reach 71.71; and a fixed protocol gives 72.05. At the next stage, Static-PAGO retains the typed representation but fixes the organization, scoring 77.62, whereas full PAGO reaches 81.58. Its gains are 12.55 over Agent-Only, 9.53 over Uniform-Protocol, and 3.96 over Static-PAGO. Thus, typed primitives expand the representational space, while the PAGO-Static-PAGO gap supports query-conditioned selection. Because adjacent variants change multiple choices, Table 2 is a staged comparison, not an isolated component estimate under the reported configurations.

Table 2: Ablation study of different prompt augmented organization configurations.

Variant	Agent Node	Instr. Node	Bind. Edge	Depend. Edge	Protocol	Avg.↑
Agent-Only	✓	×	×	×	Default	69.03
Prompt-as-Attribute	✓	Attr.	×	×	Default	70.87
Node-Augmented	✓	✓	✓	✓	Default	71.71
Uniform-Protocol	✓	✓	✓	✓	Fixed	72.05
Static-PAGO	✓	✓	✓	✓	✓	77.62
PAGO	✓	✓	✓	✓	✓	**81.58**

4.4 RQ3: Task-Dependent Organization Structure

Figure 3 shows task-specific structures. GSM8K is most compact, averaging 1.80 agents, 2.42 bindings, and 1.20 communication edges. HumanEval is densest (3.54, 5.80, and 4.50), whereas MATH has the greatest depth (2.90). MBPP is intermediate; MMLU uses fewer bindings and communication edges than HumanEval. Instruction and dependency counts follow the same broad ordering: adaptation changes selected operations and execution relations, beyond the number of agents. Even the two coding benchmarks select different organizations. These averages establish between-task variation; within-task variability and causal links to difficulty require query-level graphs and repeated runs.

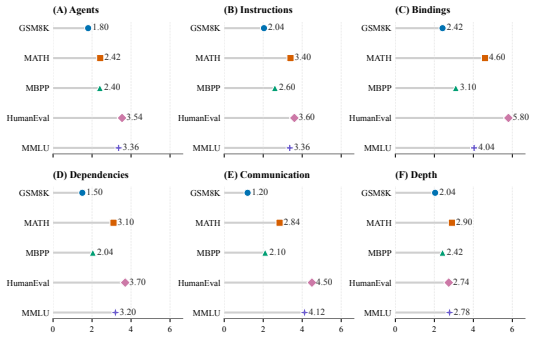


Figure 3: Task-dependent graph structure.

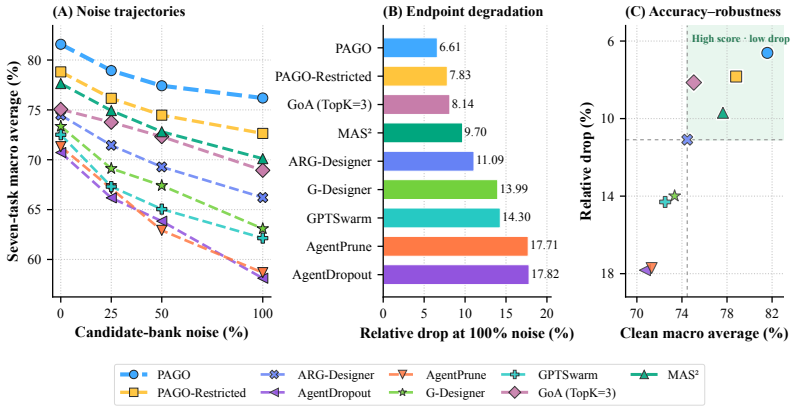


Figure 4: Robustness of PAGO under increasing candidate-bank noise. PAGO reaches the highest scores at all tested noise levels and the lowest relative drop at 100% noise.

4.5 RQ4: Robustness to Candidate Noise

Figure 4 measures percentage loss from the clean score at 0%, 25%, 50%, and 100% noise; additional ticks are not observations. PAGO leads throughout (81.58, 78.94, 77.42, 76.19), with relative drops of 3.24%, 5.10%, and 6.61%. At maximum noise, it exceeds PAGO-Restricted by 3.55 points and MAS^2 by 6.08, with the smallest relative drop. GoA drops less at 25% and 50% (1.72%, 3.64%), but scores 5.18 and 5.10 points below PAGO. Thus, absolute performance and relative retention give complementary comparisons: PAGO preserves its clean-score advantage and has the strongest retention at maximum noise, although it does not have the smallest relative drop at every level.

4.6 RQ5: Cross-Backbone Adaptation

RQ5 transfers a policy learned with DeepSeek-v4-pro to Qwen3-14B. The Qwen3-14B single-agent and target-trained PAGO reference scores, denoted S_{single} and S_{target}, are 66.42 and 79.08. For transferred score S_{tr}, RTE is the share of the single-agent-to-target-trained gap recovered. Zero-shot transfer reaches 73.42 (RTE 0.55), showing that the source policy retains part of its organizational benefit on the target backbone before adaptation.

Table 3 shows monotonic gains with 8, 16, and 32 target examples: scores rise to 75.18, 76.24, and 77.05, and RTE to 0.69, 0.78, and 0.84. At 32 shots, 84% of the target-trained improvement is recovered, leaving a 2.03-point gap. Limited target examples can therefore progressively adapt the organization policy in this setting. The evaluation covers one source–target pair; additional backbones and repeated runs are needed to establish generality across model families.

Table 3: Few-shot cross-backbone adaptation of PAGO organization policy.

Method	Source	Adapt.	Avg.↑	Gap↓	RTE↑
Single Agent	–	0	66.42	12.66	0.00
Zero-shot PAGO	DS-v4-pro	0	73.42	5.66	0.55
8-shot PAGO	DS-v4-pro	8	75.18	3.90	0.69
16-shot PAGO	DS-v4-pro	16	76.24	2.84	0.78
32-shot PAGO	DS-v4-pro	32	**77.05**	**2.03**	**0.84**
Target-trained PAGO	Qwen3-14B	Full	**79.08**	**0.00**	**1.00**

Appendix E.6 illustrates how selected primitives, instruction assignments, and feedback protocols compose an executable program-repair workflow.

5 Conclusion

We presented PAGO, a Prompt Augmented Graph Organization framework that represents multi-agent systems as typed executable graphs. By treating prompts as first-class organizational primitives, PAGO jointly represents agent roles, atomic instructions, assignments, execution dependencies, and communication protocols. Its query-conditioned policy selects valid organizations under structural and resource constraints, and deterministic materialization maps these decisions to runtime behaviors with a frozen backbone. Experiments across seven benchmarks support the effec-

tiveness of this representation and organization-learning procedure. The selected structures vary across tasks, and PAGO maintains strong performance under candidate-bank noise. The evaluated cross-backbone transfer further shows that limited target examples can adapt a learned organization policy. Together, these results support exposing prompt-level behaviors as graph components with explicit execution semantics.

Reproducibility Statement

We provide the implementation details of PAGO in the main text and appendix, including the prompt primitive bank, candidate construction, organization-policy optimization, legality constraints, graph materialization, baseline settings, and evaluation metrics. All experiments use the MMLU, GSM8K, MATH, HumanEval, MBPP, HotpotQA, and BrowseComp+ benchmarks. The code, configurations, and evaluation scripts of PAGO will be released to support reproduction of the reported results.

AI USE STATEMENT

Generative AI tools were used to polish the writing and improve the clarity of this manuscript, but were not used in the development of the method. PAGO itself uses generative models as experimental backbones and runtime agents; this use is part of the evaluated methodology.

References

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023. URL https://arxiv.org/abs/2308.10848.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. URL https://arxiv.org/abs/2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023. URL https://arxiv.org/abs/2303.17760.

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

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2310.02170.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2307.07924.

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. In *International Conference on Learning Representations*, 2026.

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

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. *arXiv preprint arXiv:2503.18891*, 2025.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Zou, Guohao Li, and Tianlong Chen. Graph-of-agents: A graph-based framework for multi-agent LLM collaboration. In *International Conference on Learning Representations*, 2026.

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

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024. URL https://arxiv.org/abs/2410.10762.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

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

A Prompt Primitive Bank and Compatibility Rules

This section specifies the reusable objects from which PAGO constructs a query-specific organization. The bank is fixed during policy training and evaluation: learning changes the selected subgraph, not the text or interface of a stored primitive. This separation keeps graph decisions identifiable and ensures that a selected object has the same runtime meaning across queries.

A.1 PRIMITIVE SCHEMA AND BANK CONSTRUCTION

The complete catalog contains agent, instruction, and communication-protocol primitives,

$$\mathcal{B} = \mathcal{B}_A \cup \mathcal{B}_I \cup \mathcal{B}_O, \qquad \mathcal{B}_A \cap \mathcal{B}_I = \mathcal{B}_A \cap \mathcal{B}_O = \mathcal{B}_I \cap \mathcal{B}_O = \varnothing. \tag{12}$$

Here, $\mathcal{B}_A \cup \mathcal{B}_I$ supplies candidate nodes, whereas $\mathcal{B}_O$ supplies protocols for candidate communication edges. Node retrieval in Eq. (5) operates on $\mathcal{B}_A \cup \mathcal{B}_I \subseteq \mathcal{B}$; the complete bank $\mathcal{B}$ includes all three primitive types. Protocols are selected only after the corresponding communication edges are activated. The bank used in all reported PAGO configurations contains $|\mathcal{B}_A| = 12$ agent primitives, $|\mathcal{B}_I| = 18$ instruction primitives, and $|\mathcal{B}_O| = 16$ communication protocols. Candidate construction retrieves at most $K_A = 6$ agents and $K_I = 12$ instructions for each query before compatible binding, dependency, and communication edges are enumerated.

Each stored primitive has an identifier, a type, prompt content, an input interface, an output interface, required capabilities, and a deterministic materialization rule. For a primitive p, we write

$$p = \big(\mathrm{id}(p), \tau_p, z_p, \mathrm{In}(p), \mathrm{Out}(p), \mathrm{Req}(p), \phi_p\big). \tag{13}$$

The bank-construction process follows collection, normalization, atomization, deduplication, and capability annotation. Normalization gives semantically equivalent fields a common interface name; atomization splits a compound instruction into independently assignable operations; deduplication removes redundant templates; and annotation records the capabilities and interfaces used by the compatibility checks below. The bank stores reusable units rather than complete task workflows, so task-specific ordering and ownership remain decisions of the organization policy.

A.2 PRIMITIVE FAMILIES

Agent primitives specify executable roles and capabilities but do not contain a task-specific workflow. Instruction primitives specify atomic operations that may be assigned to compatible agents. Protocol primitives specify how a receiver should interpret a message on a selected communication edge. Tables 4–6 summarize representative families stated in the framework specification and specify their corresponding runtime semantics.

Table 4: Representative agent-prompt families and their runtime roles.

Primitive	Required capability	Runtime role
Planner	Planning	Decompose and order operations
General Solver	Reasoning	Produce a candidate solution
Mathematical Solver	Mathematical reasoning	Derive and calculate
Retriever	Retrieval	Locate task-relevant information
Code Generator	Code generation	Synthesize a program
Executor	Tool execution	Run a program and return output
Debugger	Debugging	Locate and repair failures
Critic	Critique	Identify defects in a candidate
Verifier	Verification	Check a candidate result
Aggregator	Aggregation	Combine compatible outputs
Finalizer	Response generation	Produce the final response

Table 5: Representative instruction-prompt families. Each instruction is independently selectable and must be bound to a compatible active agent.

Instruction	Required capability	Output interface
Analyze Task	Reasoning	Task interpretation
Decompose Task	Planning	Ordered subtask set
Generate Candidate	Reasoning	Candidate solution
Derive Solution	Mathematical reasoning	Derivation and answer
Generate Code	Code generation	Executable program
Execute Program	Tool execution	Output and error status
Critique Candidate	Critique	Defect report
Verify Candidate	Verification	Validation status
Aggregate Results	Aggregation	Integrated result
Generate Response	Response generation	Final answer

Table 6: Representative communication-protocol families.

Protocol	Receiver-side interpretation
Candidate Transfer	Treat the message as a candidate to consume or extend.
Critique Request	Diagnose defects in the attached candidate.
Verification Request	Check the candidate under the stated criterion.
Execution Feedback	Consume tool output, status, and available error text.
Revision Feedback	Revise the candidate using the reported defect and guidance.
Aggregation Input	Merge the message with other compatible intermediate outputs.
Finalization Transfer	Convert the validated result to the required answer format.

A.3 Local compatibility

Compatibility rules remove locally invalid pairs before the policy searches over complete organizations. An instruction i may be bound to agent a only if the agent satisfies its capability requirement; a dependency $i \to j$ is admissible only if the producer and consumer interfaces overlap; and protocol o is admissible on $a \to b$ only if both endpoints satisfy its sender and receiver requirements:

$$C^{AI}_{ia} = \mathbb{I}[\mathrm{Req}(i) \subseteq \mathrm{Cap}(a)], \tag{14}$$

$$C^{II}_{ij} = \mathbb{I}[\mathrm{Out}(i) \cap \mathrm{In}(j) \neq \varnothing], \tag{15}$$

$$C^{O}_{ab,o} = \mathbb{I}[\mathrm{Req}_s(o) \subseteq \mathrm{Cap}(a)]\mathbb{I}[\mathrm{Req}_r(o) \subseteq \mathrm{Cap}(b)]. \tag{16}$$

These indicators test one proposed relation at a time. They do not guarantee that all selected instructions are owned, that the dependency graph is acyclic, or that a final output is reachable; those are global conditions.

B Legality-Aware Organization Constraints

Let $\mathcal{G}^{\mathrm{sel}}_k$ be the partial organization after the first k decisions and let a be a candidate update. PAGO admits the update only if $\mathcal{G}^{\mathrm{sel}}_k \oplus a$ can still form an executable organization. The validity predicate used by the mask in Eq. (7) is

$$\begin{aligned}\mathrm{Valid}(\mathcal{G}) = & C_{\mathrm{type}} \wedge C_{\mathrm{end}} \wedge C_{\mathrm{bind}} \wedge C_{\mathrm{dag}} \wedge C_{\mathrm{iface}} \wedge C_{\mathrm{cross}} \\ & \wedge C_{\mathrm{reach}} \wedge C_{\mathrm{output}} \wedge C_{\mathrm{protocol}} \wedge C_{\mathrm{budget}}.\end{aligned} \tag{17}$$

The constraints are defined as follows.

- C_{type} permits only binding, dependency, and communication relations with the endpoint types defined in Eq. (3).
- C_{end} requires both endpoints of every selected edge to be active.
- C_{bind} requires each active instruction to have at least one active, locally compatible owner.

- C_{dag} requires the selected instruction-dependency subgraph to be acyclic before temporal unrolling.
- C_{iface} requires every selected dependency to satisfy Eq. (15).
- C_{cross} requires a communication path whenever a producer and consumer are bound to different agents.
- C_{reach} removes active nodes disconnected from the query-conditioned computation.
- C_{output} requires at least one path from an active instruction to a final-response operation.
- C_{protocol} requires each active communication edge to carry one compatible protocol from $\mathcal{B}_O$.
- C_{budget} enforces the configured graph, call, token, latency, and interaction-round limits.

For an active instruction i, the ownership condition can be written

$$\sum_{a \in \mathcal{V}_A^{\mathrm{cand}}} g_a g_i g_{ia}^B C_{ia}^{AI} \geq g_i, \tag{18}$$

and a cross-agent dependency $i \to j$, with owners $a \neq b$, requires

$$g_i g_j g_{ij}^D g_{ia}^B g_{jb}^B \leq \mathbb{I}[a \rightsquigarrow b \mid \mathcal{E}_C^{\mathrm{sel}}], \tag{19}$$

where $a \rightsquigarrow b$ denotes reachability in the selected communication subgraph. The budget condition is

$$\mathbf{C}(\mathcal{G}) = (C_{\mathrm{graph}}, C_{\mathrm{call}}, C_{\mathrm{tok}}, C_{\mathrm{lat}}, C_{\mathrm{round}}) \preceq \mathbf{B}. \tag{20}$$

The implementation checks a proposed update in three stages: local type and interface compatibility, endpoint and ownership consistency, and global DAG, reachability, output-path, and budget conditions. Local checks are cached for the fixed candidate graph. DAG and reachability checks use adjacency lists and are linear in the active graph size, $O(|\mathcal{V}^{\mathrm{sel}}| + |\mathcal{E}^{\mathrm{sel}}|)$. The mask proves structural executability under the declared interfaces; it does not guarantee that the backbone will produce a correct task answer.

C Deterministic Graph Materialization

Materialization maps every selected declarative element to a runtime object. For active agent a, let $\mathcal{V}_{I,a}^{\mathrm{sel}}$ be the instructions connected to a by active binding edges. Their local order is

$$\mathrm{Seq}(a) = \mathrm{TopoSort}\big(\mathcal{V}_{I,a}^{\mathrm{sel}}, \mathcal{E}_D^{\mathrm{sel}}|_{\mathcal{V}_{I,a}^{\mathrm{sel}}}\big), \tag{21}$$

with deterministic tie breaking by primitive identifier when several valid topological orders exist. The system prompt is then a serialization of the role, ordered instructions, input contract, inbound and outbound protocols, and output format:

$$s_a = \mathrm{Serialize}\big(z_a, \mathrm{Seq}(a), X_a, \mathcal{C}_a^{\mathrm{in}}, \mathcal{C}_a^{\mathrm{out}}, F_a\big). \tag{22}$$

This expands Eq. (8): binding edges determine ownership, dependency edges determine precedence, and communication edges instantiate typed channels. Because serialization is a deterministic function of the selected graph, two runs with the same graph and primitive versions receive the same organizational prompts, apart from any separately configured query or model-sampling variation.

Each transmitted message uses a protocol-aware envelope

$$m_{a \to b}^{(r)} = \big(\mathrm{id}(a), \mathrm{id}(b), r, o_{ab}, \mathrm{payload}, \mathrm{status}\big), \tag{23}$$

so the receiver knows whether the payload is a candidate, a verification request, tool feedback, a revision request, an aggregation input, or a final result. Cross-agent instruction dependencies contribute both an agent-level scheduling relation and the communication channel needed to transport the producer output.

Feedback does not introduce a cycle into a single execution layer. An edge from a back to b is unrolled across bounded interaction rounds,

$$(a^{(r)}, b^{(r+1)}) \in \mathcal{E}^{\mathrm{exec}}, \qquad 0 \leq r < R_{\max}, \tag{24}$$

which preserves an acyclic schedule over the expanded temporal graph. Execution terminates when the selected verification condition succeeds, the candidate no longer changes, no executable successor remains, or a declared round, call, token, or latency limit is reached. The terminal active output path is then converted to the task-specific answer format.

D ORGANIZATION-POLICY OPTIMIZATION

The policy follows the semantic decision order $\mathcal{S} = (V^A, V^I, E^B, E^D, E^C, O)$. Its masked probability factorizes as

$$\widetilde{\pi}_\Theta(\mathcal{G}^{\mathrm{sel}} \mid q) = \prod_{k=1}^{K} \widetilde{p}_\Theta(a_k \mid q, a_{<k}), \qquad \log \widetilde{\pi}_\Theta = \sum_{k=1}^{K} \log \widetilde{p}_\Theta(a_k \mid q, a_{<k}). \tag{25}$$

The mask is part of the sampling distribution, so rejected updates contribute no probability mass and later decisions are conditioned on a legal partial organization.

For a binary node or edge decision with logit α_x, training uses a Binary Concrete relaxation

$$\widetilde{g}_x = \sigma\left(\frac{\alpha_x + \log u - \log(1-u)}{\tau}\right), \qquad u \sim \mathrm{Uniform}(0,1), \tag{26}$$

and the forward pass uses

$$g_x^{\mathrm{hard}} = \mathbb{I}[\widetilde{g}_x > \eta], \qquad g_x = g_x^{\mathrm{hard}} - \mathrm{sg}(\widetilde{g}_x) + \widetilde{g}_x, \tag{27}$$

where sg stops the gradient. The framework specification uses the threshold $\eta = 0.5$. At training step t, the Binary Concrete temperature follows the exponential schedule

$$\tau_t = \max\bigl(0.5,\ 1.2 \times 0.98^t\bigr), \tag{28}$$

which gradually sharpens relaxed activations while retaining a positive temperature floor. At inference, activation is deterministic and an active communication edge uses the maximum-probability compatible protocol.

Let $\widehat{R}$ be the task reward after runtime costs, as defined in Eq. (9). Subtracting any query-dependent baseline $b(q)$ that is independent of the sampled graph leaves the expected policy gradient unbiased:

$$\nabla_\Theta J = \mathbb{E}\left[(\widehat{R} - b(q)) \sum_{k=1}^{K} \nabla_\Theta \log \widetilde{p}_\Theta(a_k \mid q, a_{<k})\right]. \tag{29}$$

Optimization hyperparameters. The organization policy and query-conditioned value baseline are optimized jointly with Adam at learning rate 10^{-3}. Training uses one query and one sampled organization per update (batch size 1), gradient clipping at Euclidean norm 5.0, entropy coefficient 10^{-3}, and value-loss coefficient 0.5. Task-specific policy-training and validation budgets are summarized in Table 7.

The structural coefficients are

$$(\beta_A, \beta_I, \beta_B, \beta_D, \beta_C) = (0.01, 0.01, 0.005, 0.005, 0.005), \tag{30}$$

and the runtime-cost coefficients are

$$(\lambda_{\mathrm{tok}}, \lambda_{\mathrm{lat}}, \lambda_{\mathrm{call}}) = (0.1, 0, 0.05), \tag{31}$$

where C_{tok} is measured in thousands of tokens, C_{lat} in seconds, and C_{call} in model calls. Thus, one thousand tokens and one model call incur penalties of 0.1 and 0.05, respectively; latency is recorded but receives zero weight in the reported configuration.

Table 7: Candidate-construction and policy-training configurations. Panel A reports shared settings and task-specific token limits; Panel B reports training and validation budgets.

A. Candidate construction and optimization

Setting	Value
Primitive bank	12 agents; 18 instructions; 16 protocols
Candidate retrieval	$K_A = 6$; $K_I = 12$
Optimizer / learning rate	Adam / 10^{-3}
Update batch size	1 query; 1 sampled graph
Binary Concrete temperature	$\tau_t = \max(0.5, 1.2 \times 0.98^t)$
Hard threshold / gradient clip	0.5 / 5.0
Structural weights	$(0.01, 0.01, 0.005, 0.005, 0.005)$
Runtime-cost weights	$(\lambda_{\text{tok}}, \lambda_{\text{lat}}, \lambda_{\text{call}}) = (0.1, 0, 0.05)$
Entropy / value-loss coefficients	10^{-3} / 0.5
Graph/runtime limits	4 agents; 6 communication edges; 4 calls; 1 round
Total token limit	8,192 per query; 10,000 for MBPP

B. Task-specific policy training

Task / Setting	Training source	Train queries	Update steps	Valid. queries
MMLU	Task-specific subset	30	30	10
GSM8K	GSM8K train	50	50	20
MATH	MATH train	50	50	20
HumanEval	Task-specific subset	30	30	10
MBPP	MBPP train	30	30	10
HotpotQA	HotpotQA train	40	40	20
BrowseComp+	Task-specific subset	50	50	20
RQ5 adaptation	Qwen3-14B target examples	8/16/32	20	10

Combining the negative estimator with the typed structural penalty yields the objective stated in Eq. (10). A training iteration therefore constructs the candidate graph, samples a legality-masked organization, materializes and executes it with the frozen backbone, computes task utility and measured runtime cost, and updates only the organization-policy parameters.

E Additional Experimental Details

This section specifies the experimental protocol, task interfaces, baseline semantics, metric calculations, supplementary results, and the program-repair case. The knowledge benchmark is **MMLU** throughout.

E.1 Detailed Experimental Protocol

Model and decoding configuration. The primary backbone is DeepSeek-v4-pro accessed through the official API with reasoning mode disabled. Decoding uses temperature 0 except for Self-Consistency, whose sampling settings are specified in Appendix E.2. Each method receives the same task input and task-specific output budget. PAGO uses the fixed bank of 12 agent, 18 instruction, and 16 protocol primitives described in Appendix A, with $K_A = 6$ and $K_I = 12$. The complete organization-policy hyperparameters, including the optimizer, temperature schedule, structural coefficients, and runtime-cost weights, are given in Appendix D.

Benchmarks and score interfaces. The evaluation contains seven benchmarks: MMLU, GSM8K, MATH, HumanEval, MBPP, HotpotQA, and BrowseComp+. MMLU, GSM8K, and MATH use correctness-based task scores; HumanEval and MBPP use program correctness under

the benchmark tests, reported as Pass@1. HotpotQA and BrowseComp+ use their task-level scores on the same percentage scale.

Table 8: Task interfaces used by the reported evaluation.

Benchmark	Task family	Reported score	Evaluation detail
MMLU	Knowledge reasoning	Accuracy (%)	Correct selected answer.
GSM8K	Mathematical reasoning	Accuracy (%)	Correct final answer after benchmark answer extraction.
MATH	Mathematical reasoning	Accuracy (%)	Correct final answer after task-specific answer extraction.
HumanEval	Code generation	Pass@1 (%)	One generated solution is evaluated against the task tests.
MBPP	Code generation	Pass@1 (%)	One generated solution is evaluated against the task tests.
HotpotQA	Multi-hop QA	Task score (%)	Task-level score for the multi-hop answer.
BrowseComp+	Web-oriented QA	Task score (%)	Task-level score for the web-oriented response.

Query-to-execution procedure. PAGO retrieves candidate nodes from $\mathcal{B}_A \cup \mathcal{B}_I \subseteq \mathcal{B}$, where $\mathcal{B}_A$ contains agent-role templates and $\mathcal{B}_I$ contains atomic instruction templates. The complete frozen bank also includes communication protocols $\mathcal{B}_O$. For query q, a frozen encoder retrieves a high-recall candidate set:

$$\mathcal{V}_A^{\text{cand}}(q) = \text{TopK}_{K_A}\left(\cos(\text{Enc}(q), \text{Enc}(z_p)) : p \in \mathcal{B}_A\right), \tag{32}$$

$$\mathcal{V}_I^{\text{cand}}(q) = \text{TopK}_{K_I}\left(\cos(\text{Enc}(q), \text{Enc}(z_p)) : p \in \mathcal{B}_I\right), \tag{33}$$

with $K_A = 6$ and $K_I = 12$. Retrieved node identifiers and similarity scores are retained with the candidate-graph record before policy selection.

Candidate binding, dependency, and communication relations are constructed only when local capability and interface checks pass. For instruction i, agent a, instructions i, j, and protocol o, the local compatibility indicators are

$$C_{ia}^{AI} = \mathbb{I}[\text{Req}(i) \subseteq \text{Cap}(a)], \tag{34}$$

$$C_{ij}^{II} = \mathbb{I}[\text{Out}(i) \cap \text{In}(j) \neq \varnothing], \tag{35}$$

$$C_{ab,o}^{O} = \mathbb{I}[\text{Req}_s(o) \subseteq \text{Cap}(a)]\mathbb{I}[\text{Req}_r(o) \subseteq \text{Cap}(b)]. \tag{36}$$

Compatibility is local: it determines whether one candidate relation is well typed and executable in isolation. It does not establish that the assembled organization is globally executable.

Legality-aware selection. The organization policy processes decisions in the order $[V^A, V^I, E^B, E^D, E^C, O]$. Before sampling candidate update a_k, let $\mathcal{G}_{k-1}^{\text{sel}}$ denote the partial organization after the preceding $k-1$ decisions. PAGO evaluates

$$m(a_k \mid \mathcal{G}_{k-1}^{\text{sel}}) = \mathbb{I}[\text{Valid}(\mathcal{G}_{k-1}^{\text{sel}} \oplus a_k) = 1] \tag{37}$$

and renormalizes the action distribution over legal updates:

$$\widetilde{p}(a_k \mid \mathcal{G}_{k-1}^{\text{sel}}, q) = \frac{m(a_k \mid \mathcal{G}_{k-1}^{\text{sel}}) p(a_k \mid \mathcal{G}_{k-1}^{\text{sel}}, q)}{\sum_{a'} m(a' \mid \mathcal{G}_{k-1}^{\text{sel}}) p(a' \mid \mathcal{G}_{k-1}^{\text{sel}}, q)}. \tag{38}$$

The validity predicate follows Appendix B:

$$\begin{aligned}\text{Valid}(\mathcal{G}) = C_{\text{type}} \wedge C_{\text{end}} \wedge C_{\text{bind}} \wedge C_{\text{dag}} \wedge C_{\text{iface}} \wedge C_{\text{cross}} \\ \wedge C_{\text{reach}} \wedge C_{\text{output}} \wedge C_{\text{protocol}} \wedge C_{\text{budget}}.\end{aligned} \tag{39}$$

These conditions enforce edge typing, active endpoints, instruction ownership, acyclic dependencies, compatible interfaces, cross-agent transport, query reachability, an output path, protocol compatibility, and resource limits. Legality guarantees structural executability, not semantic correctness.

Materialization and bounded feedback. For each active agent a, assigned instructions are ordered by

$$I_a = \mathrm{TopoSort}(\mathcal{V}^{\mathrm{sel}}_{I,a}, \mathcal{E}^{\mathrm{sel}}_{D,a}). \tag{40}$$

The runtime prompt is serialized as

$$P_a = \mathrm{Serialize}(R_a, I_a, X_a, O^{\mathrm{in}}_a, O^{\mathrm{out}}_a, F_a), \tag{41}$$

where the fields contain the role, ordered instructions, query and upstream context, inbound and outbound protocol descriptions, and output format. Cross-agent dependencies become both scheduling constraints and message channels. A feedback edge is temporally unrolled from $u^{(k)}$ to $v^{(k+1)}$, so a bounded revise–verify loop remains acyclic at runtime. Execution stops when verification succeeds, revision no longer changes the candidate, or the round, call, token, or time budget is reached.

Organization-policy optimization. The frozen backbone produces task outcomes, while the organization policy is optimized using task reward net of measured runtime cost:

$$\widehat{R} = R_{\mathrm{task}} - \big(\lambda_{\mathrm{tok}} C_{\mathrm{tok}} + \lambda_{\mathrm{lat}} C_{\mathrm{lat}} + \lambda_{\mathrm{call}} C_{\mathrm{call}}\big). \tag{42}$$

For selected graph $\mathcal{G}^{\mathrm{sel}}$ and query-conditioned baseline $b(q)$, the policy loss is

$$\mathcal{L}_{\mathrm{PG}} = -\mathbb{E}[(\widehat{R} - b(q)) \log \pi_\Theta(\mathcal{G}^{\mathrm{sel}} \mid q)]. \tag{43}$$

The graph probability is factorized into decisions over selected nodes $\mathcal{V}^{\mathrm{sel}}$, edges $\mathcal{E}^{\mathrm{sel}}$, and edge-associated protocol choices $\mathcal{O}^{\mathrm{sel}}$:

$$\pi_\Theta(\mathcal{G}^{\mathrm{sel}} \mid q) = \pi^V_\Theta(\mathcal{V}^{\mathrm{sel}} \mid q) \pi^E_\Theta(\mathcal{E}^{\mathrm{sel}} \mid \mathcal{V}^{\mathrm{sel}}, q) \pi^O_\Theta(\mathcal{O}^{\mathrm{sel}} \mid \mathcal{E}^{\mathrm{sel}}, q). \tag{44}$$

The relaxed structural penalty is

$$\mathcal{L}_{\mathrm{struct}} = \sum_{t \in \mathcal{T}} \beta_t \sum_{x \in \mathcal{X}^{\mathrm{cand}}_t} \widetilde{g}_x, \qquad \mathcal{T} = \{A, I, B, D, C\}, \tag{45}$$

where $\widetilde{g}_x$ is the relaxed activation of candidate primitive x. The reported method objective is therefore

$$\mathcal{L}_{\mathrm{PAGO}} = \mathcal{L}_{\mathrm{PG}} + \mathcal{L}_{\mathrm{struct}}, \tag{46}$$

which is identical to Eq. (10). The reported configuration uses Adam with learning rate 10^{-3}, batch size 1, the temperature schedule in Eq. (28), structural coefficients in Eq. (30), and cost coefficients in Eq. (31). Legality-masked Binary Concrete selection, straight-through discretization, and deterministic inference are specified in Appendix D.

E.2 Complete Baseline Settings

Direct Prompt requests one answer, while CoT elicits a step-by-step reasoning trajectory (Wei et al., 2022). Both use temperature 0. Self-Consistency instead generates three independently sampled reasoning trajectories using temperature 0.7 and top-p 1.0 (Wang et al., 2023). This sampling configuration is an exception to the default temperature-zero setting. Each trajectory receives the same task input, and final answers are extracted using the same task-specific parser before majority voting. Cases without a strict majority are resolved deterministically by selecting the answer from the first sampled trajectory.

The agent-organization group covers optimized computational graphs, communication pruning, agent elimination, task-conditioned topology design, autoregressive role/edge generation, graph-based agent selection, and self-generating multi-agent systems (Zhuge et al., 2024; Zhang et al., 2025a; Wang et al., 2025; Zhang et al., 2025b; Li et al., 2026; Yun et al., 2026; Wang et al., 2026). Table 9 summarizes the operational semantics of the compared configurations.

Table 9: Operational semantics of the RQ1 baseline configurations. Self-Consistency uses stochastic sampling rather than the default temperature-zero decoding.

Method	Organized object	Setting represented by the comparison row
Direct Prompt	One model response	A direct-answer prompt decoded at temperature 0 without an explicit multi-agent organization.
CoT	One reasoning trajectory	A fixed step-by-step reasoning instruction decoded at temperature 0.
Self-Consistency	Three sampled trajectories	Three independently sampled reasoning trajectories whose extracted answers are aggregated by majority vote.
AgentDropout	Agent participation	Less useful agents are dynamically removed from a multi-agent organization.
AgentPrune	Communication structure	Redundant communication is pruned from an agent graph.
GPTSwarm	Computational agent graph	Agent components and connections form an optimizable graph.
G-Designer	Task-conditioned topology	A graph model produces a task-dependent communication topology.
ARG-Designer	Generated roles and edges	Agent roles and their communication relations are generated autoregressively.
GoA (TopK=3)	Selected agent graph	Three selected agents are connected into the reported graph-of-agents configuration.
MAS^2	Generated multi-agent system	The system is generated, configured, and rectified under the reported method setting.

RQ2 uses staged internal configurations that progressively add representation and update capabilities. Table 10 makes query conditioning explicit as the defining difference between Static-PAGO and full PAGO, while both retain all typed primitive families.

Table 10: Complete semantics of the internal RQ2 configurations. "Attribute" means that instruction text is fixed inside an agent prompt instead of represented as independently selectable graph nodes.

Variant	Graph elements				Query-conditioned	Interpretation
	Agent nodes	Instruction nodes	Bind/ depend	Edge protocol		
Agent-Only	✓	×	×	Default	✓	Agent-level topology only
Prompt-as-Attribute	✓	Attribute	×	Default	✓	Fixed instruction text inside roles
Node-Augmented	✓	✓	✓	Default	✓	Typed nodes and ownership/order
Uniform-Protocol	✓	✓	✓	Fixed	✓	One shared communication behavior
Static-PAGO	✓	✓	✓	Typed	×	One fixed full organization
PAGO	✓	✓	✓	Typed	✓	Full query-conditioned organization

For RQ4, a noise level ρ adds irrelevant or misleading primitives in proportion to the clean candidate-bank size. Agent-centric methods receive noise expressible in their agent pool; PAGO additionally admits instruction noise because its search space contains instruction primitives. The comparison is thus representation-aware. For RQ5, the reported source backbone is DeepSeek-v4-pro and the runtime target is Qwen3-14B. The transfer settings use 0, 8, 16, or 32 target examples; Target-trained PAGO uses the full target-training condition and defines the reference used by the transfer efficiency metric.

E.3 Metric Definitions and Calculation

Correctness-based task scores. For a correctness-scored benchmark with N_d examples, the percentage score is

$$S_{m,d} = \frac{100}{N_d} \sum_{i=1}^{N_d} \mathbb{I}[\mathrm{Score}_d(\hat{y}_{m,d,i}, y_{d,i}) = 1]. \tag{47}$$

For HumanEval and MBPP, Score_d executes one generated program against the benchmark tests, so Eq. (47) is the reported Pass@1 percentage. For HotpotQA and BrowseComp+, Score_d denotes the corresponding benchmark evaluator.

Seven-task macro average. Every benchmark receives equal weight regardless of dataset size:

$$\bar{S}_m = \frac{1}{7} \sum_{d \in \mathcal{D}_7} S_{m,d}, \tag{48}$$

$$\mathcal{D}_7 = \{\text{MMLU, GSM8K, MATH, HumanEval, MBPP, HotpotQA, BrowseComp+}\}.$$

The seven displayed task scores are averaged before rounding to obtain each RQ1 macro average. An absolute comparison is expressed in percentage points:

$$\Delta^{\mathrm{pp}}_{m_1,m_2} = \bar{S}_{m_1} - \bar{S}_{m_2}. \tag{49}$$

This quantity is distinct from a relative percentage improvement.

Average graph statistics. For structural feature k and the N_d evaluated queries of task d, RQ3 reports

$$\bar{x}_{d,k} = \frac{1}{N_d} \sum_{i=1}^{N_d} x_{d,i,k}, \tag{50}$$

where k ranges over active agents, active instructions, binding edges, dependency edges, communication edges, and graph depth. These per-query statistics are averaged independently for each task and structural feature.

Candidate-noise metrics. Let $|\mathcal{X}_0^{\mathrm{cand}}|$ be the number of clean candidates and $|\mathcal{X}_{\mathrm{noise}}^{\mathrm{cand}}|$ the number added by corruption. The declared noise level is

$$\rho = 100 \times \frac{|\mathcal{X}_{\mathrm{noise}}^{\mathrm{cand}}|}{|\mathcal{X}_0^{\mathrm{cand}}|}\%, \tag{51}$$

and the relative performance drop for method m is

$$\mathrm{Drop}_m(\rho) = 100 \times \frac{S_m(0) - S_m(\rho)}{S_m(0)}\%. \tag{52}$$

The stored drop values agree with Eq. (52) after two-decimal rounding.

Cross-backbone transfer metrics. Let S_{single} be the target-backbone single-agent score, $S^{(n)}_{\mathrm{transfer}}$ the transferred PAGO score after n target examples, and S_{target} the fully target-trained PAGO reference. The remaining transfer gap, recovered organization gain, and relative transfer efficiency are

$$\Delta^{(n)}_{\mathrm{transfer}} = S_{\mathrm{target}} - S^{(n)}_{\mathrm{transfer}}, \tag{53}$$

$$G^{(n)}_{\mathrm{transfer}} = S^{(n)}_{\mathrm{transfer}} - S_{\mathrm{single}}, \tag{54}$$

$$\mathrm{RTE}^{(n)} = \frac{G^{(n)}_{\mathrm{transfer}}}{S_{\mathrm{target}} - S_{\mathrm{single}}}. \tag{55}$$

RTE normalizes the transferred gain by the improvement achieved through full target-backbone training; values of 0 and 1 correspond to the single-agent and target-trained references, respectively.

Numerical precision. Scores and derived quantities are computed before display rounding and are reported to two decimal places. Percentage-point differences use unrounded task scores and are rounded only after aggregation.

E.4 Supplementary Result Tables

This section reports the complete aggregate values underlying the compressed figures in the main text, including task-level graph statistics, noise-level scores, degradation rates, and cross-backbone transfer quantities.

Table 11: Complete task-level mean organization statistics used by the RQ3 figure. Each cell is a per-query mean for the corresponding task.

Task	Agents	Instructions	Bindings	Dependencies	Communications	Depth
GSM8K	1.80	2.04	2.42	1.50	1.20	2.04
MATH	2.42	3.40	4.60	3.10	2.84	2.90
MBPP	2.40	2.60	3.10	2.04	2.10	2.42
HumanEval	3.54	3.60	5.80	3.70	4.50	2.74
MMLU	3.36	3.36	4.04	3.20	4.12	2.78

Table 12: Complete RQ4 candidate-noise results. Scores are seven-task macro-average percentages; parenthesized values are relative drops from the clean score computed with Eq. (52).

Method	Clean	25% noise	50% noise	100% noise
AgentDropout	70.72	66.18 (6.42%)	63.84 (9.73%)	58.12 (17.82%)
AgentPrune	71.31	67.11 (5.89%)	62.92 (11.77%)	58.68 (17.71%)
GPTSwarm	72.51	67.31 (7.17%)	65.05 (10.29%)	62.14 (14.30%)
G-Designer	73.35	69.12 (5.77%)	67.41 (8.10%)	63.09 (13.99%)
ARG-Designer	74.47	71.45 (4.06%)	69.28 (6.97%)	66.21 (11.09%)
GoA (TopK=3)	75.05	73.76 (1.72%)	72.32 (3.64%)	68.94 (8.14%)
MAS^2	77.64	74.93 (3.49%)	72.81 (6.22%)	70.11 (9.70%)
PAGO-Restricted	78.81	76.16 (3.36%)	74.47 (5.51%)	72.64 (7.83%)
PAGO	81.58	78.94 (3.24%)	77.42 (5.10%)	76.19 (6.61%)

Table 13: Complete RQ5 transfer record. The gap is measured to the target-trained PAGO score of 79.08; RTE follows Eq. (55).

Setting	Target examples	Score	Gap	RTE
Single Agent	0	66.42	12.66	0.00
Zero-shot PAGO	0	73.42	5.66	0.55
8-shot PAGO	8	75.18	3.90	0.69
16-shot PAGO	16	76.24	2.84	0.78
32-shot PAGO	32	77.05	2.03	0.84
Target-trained PAGO	Full	79.08	0.00	1.00

E.5 Run Manifest and Query-Level Records

Each execution is represented by a run manifest and query-level record. Table 14 defines the fields used to bind a reported score to its data, model, bank, selected organization, runtime trace, resource usage, scorer, and policy-training configuration.

Table 14: Run-manifest fields for exact PAGO reproduction.

Record	Required fields
Data	Benchmark version, split, sample ID, input hash, target hash
Model	Provider, model identifier, decoding parameters, tool access
Bank	Primitive-bank version and primitive-content hashes
Retrieval	Candidate IDs, retrieval scores, and candidate limits
Selection	Selected IDs, decision order, mask decisions, termination state
Runtime	Serialized prompts, ordered messages, tool results, stopping reason
Usage	Model calls, input/output tokens, latency, and monetary cost
Scoring	Parser version, task scorer, final prediction, and task score
Training	Data IDs, optimizer settings, coefficients, steps, and random seeds

E.6 Case Study: From Prompt Primitives to Executable Program Repair

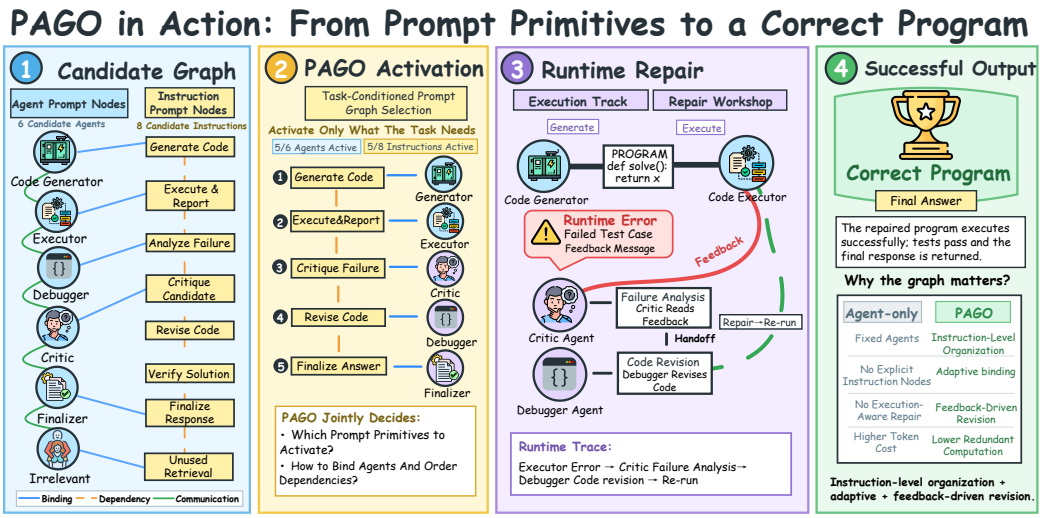


Figure 5: Illustrative program-repair organization in PAGO, showing primitive selection, instruction assignment, and feedback-driven execution.

Figure 5 illustrates program repair through selected and bound primitives. After failure, the Critic diagnoses, the Debugger revises, and the Executor retests; dependencies and protocols coordinate the loop until verification or budget exhaustion. This five-agent illustration differs from the four-agent, four-call evaluation in Table 7.

The code-generation case expands a complete execution sequence: initial generation, failed execution, critique, revision, successful re-execution, and finalization. It makes the selected operations, bindings, temporal dependencies, and communication protocols explicit.

Candidate primitives. For this task type, a high-recall candidate graph may contain agent roles capable of generation, execution, critique, debugging, and finalization, together with instruction primitives for producing a candidate program, running tests, diagnosing an error, revising code, and emitting the final answer. The instruction path is

$$i_{\text{gen}} \to i_{\text{exec}}^{(1)} \to i_{\text{crit}} \to i_{\text{rev}} \to i_{\text{exec}}^{(2)} \to i_{\text{final}}. \tag{56}$$

In a valid materialization, each instruction would be bound to an active, compatible agent. The first and second execution nodes are temporal instances of the same execution operation, which prevents the revise–execute feedback loop from creating a cycle in the runtime graph.

Message protocols. The graph topology alone does not specify how an intermediate artifact is consumed. The generator sends the initial program to the executor through Candidate Transfer.

After a failed test, the executor sends the program, test outcome, and error log to the critic through Execution Feedback. The critic then sends the diagnosed error location and modification guidance to the debugger through Revision Feedback. The debugger returns the revised program to the executor through Candidate Transfer. After successful re-execution, the executor sends the verified program and terminal status to the finalizer through Finalization Transfer. A Critique Request would instead be appropriate when a sender provides a candidate solution and explicitly requests defect analysis; it is not used as the critic-to-reviser message in this flow.

Legality checks in the case. Before execution, any graph implementing this flow must satisfy four case-specific conditions in addition to the generic predicate in Eq. (39): the execution instruction is bound to a tool-capable agent; error feedback has a communication path to the critic; the revision consumes the critic output; and the finalizer is reachable only after an execution status is available. If any condition fails, the corresponding candidate update has zero probability under Eq. (38).

Mechanism illustrated by the case. The case shows how prompt primitives form an executable repair program: instructions are explicit nodes, binding edges assign ownership, dependency edges encode the repair order, and protocol-labeled communication edges carry the program, execution status, diagnosis, and revision between agents. The bounded feedback loop reuses the execution operation in a later temporal layer and preserves the acyclic runtime schedule.