SOME-FLOW: UTILITY-GUIDED STRUCTURAL AGENTIC INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

As agentic reasoning shifts from static workflows toward query-adaptive topologies, reasoning structures can now adapt dynamically to individual queries. Yet structural adaptivity alone does not ensure that the resulting computation achieves favorable accuracy–cost utility, nor does it determine whether further computation remains worthwhile after an initial execution. Efficient structural inference therefore requires two coupled decisions: *how to allocate high-utility structural computation to each query initially*, and *whether additional structural computation is worth its cost after the initial execution*. To address these decisions, we introduce **SOME-Flow** (**S**tructural **O**ptimization, **M**atching, and **E**volution; hereafter **SOME**), which constructs a reusable, utility-shaped structural action space through offline topology discovery and amortizes per-query structural optimization into lightweight cost-aware matching. After initial execution, SOME selectively evolves the active structure according to predicted marginal utility, allocating additional computation only when it is worthwhile. Across five reasoning and code-generation benchmarks, SOME achieves the highest macro-average performance of 82.57%, outperforming the best automated structural baseline by 2.76 percentage points. Notably, initial structural matching alone reaches 81.41%, only 1.16 points below the full method, while reducing relative inference cost by 43.2%; selective post-execution evolution then allocates additional computation to obtain the remaining performance gains. The code is available at https://anonymous.4open.science/r/SOME-EC4B.

1 INTRODUCTION

As the reasoning capabilities of Large Language Models (LLMs) continue to advance, structured workflows and multi-agent collaboration have become important paradigms for complex reasoning (Wei et al., 2022; Zhuge et al., 2024; Wu et al., 2024; Hong et al., 2024). Early approaches primarily relied on manually designed workflows or automated search to obtain a unified reasoning structure for a task (Hu et al., 2025; Zhang et al., 2025c; Saad-Falcon et al., 2025). More recent studies instead dynamically select, compose, or generate structures according to the query, task condition, or runtime environment (Yue et al., 2025; Zhang et al., 2025b; Wu et al., 2026; Feng et al., 2026; Jiang et al., 2026). Agentic reasoning is thus shifting from task-level static workflow optimization toward query-adaptive structural inference, turning reasoning topology from a fixed system configuration into an active test-time decision.

However, structural adaptivity alone does not guarantee favorable accuracy–cost trade-offs. Recent query-adaptive methods have increasingly incorporated efficiency considerations into structural routing or generation, for example by balancing performance against cost or optimizing multi-objective rewards (Yue et al., 2025; Zhang et al., 2025b; Wu et al., 2026; Jiang et al., 2026; Niu et al., 2025). These advances motivate a complementary question: even when performance–cost trade-offs are considered during selection, can efficient inference be ensured if the candidate structural action space itself has not been shaped toward high-utility computation? If the available structures contain costly, redundant, or weakly complementary computation patterns, query-adaptive selection may still incur unnecessary execution cost (Zhang et al., 2025a; Wang et al., 2025).

This utility also changes after execution. Once an initial topology has run, its cost has been incurred and its outputs provide a new inference context (Niu et al., 2025; Feng et al., 2026). The relevant

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.

decision is then no longer which topology best matches the query, but whether additional structural computation offers sufficient incremental value over stopping with the current result. Always stopping leaves residual failures unresolved, whereas uniformly adding collaboration can erase the efficiency gained from adaptation. Efficient structural inference must therefore manage utility both before and after execution.

Our first observation is that useful accuracy–cost characteristics can be identified within a utility-shaped structural action space before online adaptation. By sufficiently exploring the topology space through offline execution, one can uncover executable structures with diverse reasoning patterns, cost levels, and regions of competence (Hu et al., 2025; Zhang et al., 2025c; Saad-Falcon et al., 2025). By jointly considering correctness, cost, and complementary coverage during discovery and bank construction, this process yields a reusable, utility-shaped structural action space rather than an arbitrary pool of candidate graphs.

Our second observation is that structural adaptation can continue after the initial execution. Once initial topology has executed, a candidate topology should be evaluated by its *marginal* contribution relative to the current inference state rather than by its standalone quality. Additional computation may correct a residual error (Madaan et al., 2023; Shinn et al., 2023), but may also duplicate sufficient reasoning or introduce harmful conflicts (Huang et al., 2024). It should therefore be activated only when its expected improvement justifies its additional cost.

These observations lead to two coupled questions:

Scientific Questions

Which structural action provides the best utility for the current query?

After the initial computation, is additional structural computation still worthwhile?

Together, they recast topology adaptation as utility-aware structural allocation across inference stages rather than a one-shot routing decision.

Based on this view, we introduce **SOME-Flow** (**S**tructural **O**ptimization, **M**atching, and **E**volution; hereafter **SOME**), a utility-guided framework for query-adaptive structural inference. Offline, SOME discovers and filters executable topologies to construct a reusable utility-shaped topology bank. At inference time, it amortizes per-query structural optimization into lightweight, cost-aware query–topology matching to select the initial structure G_1. After G_1 executes, SOME estimates the marginal value of complementary structures relative to retaining the initial answer. It selectively activates additional computation when such computation is expected to yield positive utility gains. SOME therefore unifies utility-aware structural discovery, efficient initial matching, and selective post-execution evolution within a common accuracy–cost framework. Figure 1 illustrates this progression, with SOME extending structural adaptation to utility-aware decisions before and after execution.

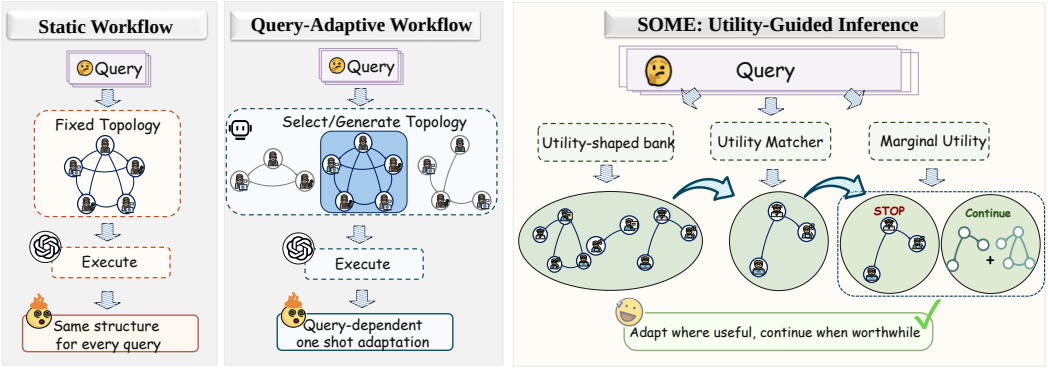


Figure 1: Paradigm workflow comparison between static, query-adaptive, and utility-guided structural inference.

We evaluate SOME on five reasoning and code-generation benchmarks using Qwen3-8B as the common execution LLM. SOME achieves the best performance on four of the five benchmarks and the highest macro-average score of 82.57%, exceeding the best automated structural baseline by 2.76 percentage points. Across all five benchmarks, SOME attains empirical Pareto-efficient performance–cost operating points, indicating that its performance gains are not simply obtained through increased inference expenditure. Notably, SOME-G_1 already reaches a macro-average score of 81.41%, only 1.16 points below the full method, while reducing average relative inference cost by 43.2%. Selective post-execution evolution then allocates additional computation when beneficial, yielding the remaining performance gains.

We make three contributions. **(1) Problem Formulation.** We cast query-adaptive agentic reasoning as a *structural utility* problem, distinguishing initial structural allocation from post-execution continuation under a shared accuracy–cost objective. This perspective reframes topology adaptation from one-shot structural routing into stage-wise allocation of reasoning computation. **(2) SOME.** We develop SOME, a unified framework that operationalizes this formulation by coupling reusable structural optimization with stage-wise inference decisions. It provides a principled mechanism for allocating structural computation according to its expected utility rather than structural adaptivity alone. **(3) Extensive Evaluation.** Experiments on MMLU, MATH, GSM8K, MBPP, and HumanEval demonstrate that SOME consistently achieves strong predictive performance and favorable accuracy–cost trade-offs, validating both the effectiveness and efficiency of utility-guided structural inference.

2 Related Work

Agentic Workflow Optimization. Agentic workflows organize LLM calls and information flow into structured reasoning procedures (Yao et al., 2023; Besta et al., 2024), extending manually designed strategies such as chain-of-thought prompting (Wei et al., 2022). Recent work automates workflow design through diverse representations and search strategies (Yuan et al., 2025). GPTSwarm (Zhuge et al., 2024) optimizes agent graphs and prompts, ADAS (Hu et al., 2025) iteratively searches agent programs with a meta agent, and AFlow (Zhang et al., 2025c) combines reusable operators, Monte Carlo Tree Search, and execution feedback. Archon (Saad-Falcon et al., 2025) further searches compositions of inference-time techniques and LLMs under compute constraints. These methods show that reasoning structures can be optimized beyond individual prompts, but primarily focus on discovering effective task-level workflows rather than deciding which structure to execute for each query or whether further computation remains worthwhile after execution.

Adaptive Agentic Inference. Adaptive methods select or construct reasoning structures according to queries, resources, or execution context (Liu et al., 2024). MasRouter (Yue et al., 2025) performs cost-aware routing over collaboration modes, agent roles, and LLMs; G-Designer (Zhang et al., 2025b), CARD (Wu et al., 2026), and Guided Topology Diffusion (Jiang et al., 2026) generate query- or context-dependent communication topologies, with the latter explicitly incorporating multi-objective rewards. Adaptation can also occur during execution: Flow (Niu et al., 2025) dynamically reallocates subtasks, while GraphPlanner (Feng et al., 2026) performs sequential routing with graph memory and reinforcement learning. While these approaches enable query-adaptive or utility-aware inference, efficient structural allocation also depends on the quality and cost profiles of the candidate structures and on whether additional computation remains beneficial after an initial execution.

SOME addresses both aspects by jointly considering utility across structural optimization, initial matching, and post-execution evolution. It constructs a reusable topology bank based on correctness, cost, and complementary coverage, performs execution-supervised cost-aware matching for the initial structure, and activates complementary structures only when their predicted marginal utility over stopping is positive.

3 Problem Setup

To formalize utility-aware structural inference, we define the following policy-level objective.

SOME Problem Formulation

Given a query $x \sim \mathcal{D}$, an executable topology $g = (V, E, K, A)$ specifies role-conditioned nodes, communication edges, interaction rounds, and an aggregation rule. Rather than optimizing an isolated topology, we consider a feasible structural policy $\pi \in \Pi$ that selects structural actions and may condition subsequent computation on the resulting inference state. The objective is

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{x \sim \mathcal{D}} \left[Y_\pi(x) - \lambda C_\pi(x) \right], \tag{1}$$

where $Y_\pi(x)$ denotes final task correctness, $C_\pi(x)$ is the total trajectory cost, including output resolution when invoked, and $\lambda \geq 0$ controls the accuracy–cost trade-off.

SOME instantiates this objective as a two-stage structural policy: it first selects and executes an initial topology G_1, yielding a post-execution state s_1. Conditioned on s_1, the policy either stops and retains G_1's output or activates one complementary topology G_2.

4 The Design of SOME-Flow

SOME manages structural computation at three levels: *which structures are worth retaining*, *which should execute first*, and *whether additional computation is worthwhile*. Offline optimization shapes a reusable, utility-aware structural action space, while matching amortizes query-specific structural search into lightweight initial selection. Once the initial structure has been executed, however, the inference state and decision baseline change: additional computation must be evaluated relative to stopping with the result already obtained. Evolution therefore activates complementary structures only when their state-conditional marginal utility justifies further computation, as illustrated in Fig. 2.

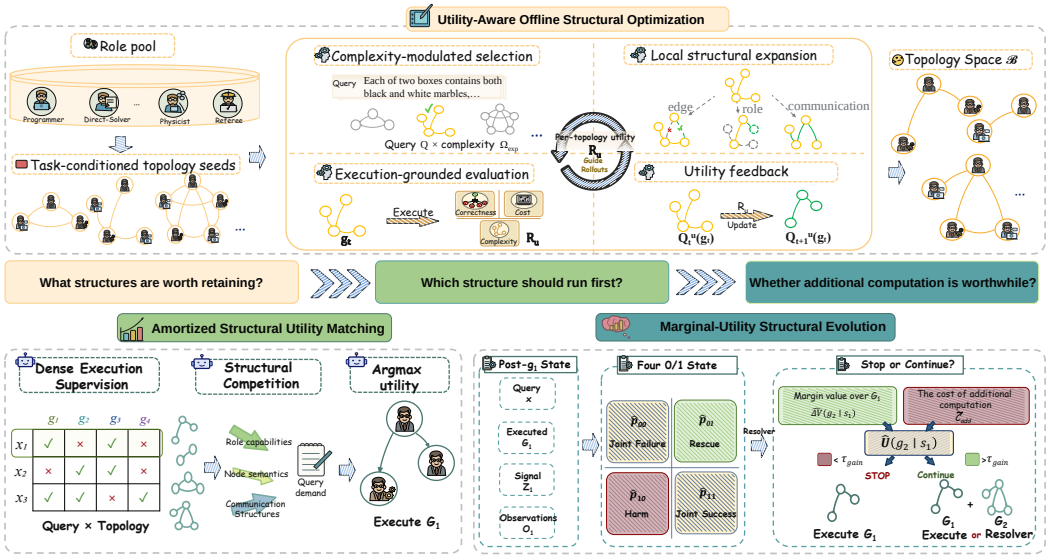


Figure 2: **Overview of SOME.** SOME manages structural computation through three stages. *Utility-Aware Offline Structural Optimization* starts from task-conditioned topology seeds and performs complexity-modulated selection, execution-grounded evaluation, utility feedback, and local structural expansion to construct a reusable topology bank $\mathcal{B}$. *Amortized Structural Utility Matching* uses dense query–topology execution supervision to select the initial structure G_1 from this bank. After G_1 executes, *Marginal-Utility Structural Evolution* estimates the state-conditional rescue and harm of complementary structures and continues computation only when the predicted marginal value is sufficient and conditional harm remains controlled; otherwise, SOME stops and retains G_1's output. Agreement is retained directly, whereas conflicting outputs are passed to a resolver.

Utility-Aware Offline Structural Optimization. A high-utility structural action space is essential for subsequent query-adaptive inference. Starting from task-conditioned seeds, SOME discovers topologies through real executions and evaluates each executed structure by jointly accounting for task correctness, computational cost, and structural complexity through the utility return

$$R_U(x,g) = w_y Y(x,g) + w_c\big(1 - \widetilde{C}(x,g)\big) - w_s \Omega_{\mathrm{reg}}(g), \tag{2}$$

Here, $Y(x,g)$ is the observed correctness of an actual execution, $\widetilde{C}(x,g)$ is a normalized execution-cost proxy, and $\Omega_{\mathrm{reg}}(g)$ regularizes structural complexity in the utility assigned to an evaluated topology. The weights prioritize correctness while favoring computationally efficient structures.

To allocate a finite exploration budget before execution, SOME modulates query–topology compatibility by structural complexity:

$$P_\Omega(g \mid x) = P_0(g \mid x)\exp\big(-\xi \Omega_{\mathrm{exp}}(g)\big), \tag{3}$$

Here, $P_0(g \mid x)$ denotes query–topology compatibility, $\Omega_{\mathrm{exp}}(g)$ measures search-time structural complexity, and ξ controls its strength. A PUCT-style (Silver et al., 2017) rule combines P_Ω with execution-grounded utility and visitation statistics, while periodically expanding the candidate set through local mutations of promising evaluated structures. The PUCT selection rule and expansion procedure are detailed in Appendix B.2.

Across offline queries, discovered topologies and their execution records are pooled and screened by correctness, cost, and complementary coverage to form a frozen bank $\mathcal{B}$. The resulting bank preserves favorable accuracy–cost operating points while retaining structures that succeed on queries missed by other candidates. Thus, offline discovery yields a *utility-shaped structural action space*, rather than a single optimized workflow, which is shared by initial matching and subsequent marginal-utility expansion. Algorithm 1 summarizes offline discovery and policy fitting; bank construction and screening details are provided in Appendix B.3.

Amortized Structural Utility Matching. Given the frozen bank $\mathcal{B}$, SOME replaces per-query structural search with learned query–topology matching. Executing every bank topology on each training query yields dense success labels y_{ij}, providing execution-grounded supervision over alternative structures for the same query. A matcher predicts execution-success scores $s_\theta(x,g)$ from query features and structural representations of topology semantics, roles, communication, and execution characteristics.

$$\mathcal{L}_{\mathrm{match}} = \alpha_{\mathrm{set}}\mathcal{L}_{\mathrm{set}} + \alpha_{\mathrm{rank}}\mathcal{L}_{\mathrm{rank}} + \alpha_{\mathrm{bce}}\mathcal{L}_{\mathrm{bce}}. \tag{4}$$

The set loss accommodates multiple successful structures, the ranking loss contrasts successful and unsuccessful alternatives within each query, and BCE retains pair-level success supervision. Detailed loss definitions and model features are provided in the Appendix.

Learning execution success is only part of the initial utility decision: inexpensive selection does not ensure inexpensive execution. SOME therefore combines learned success preference with an explicit deployment-time cost term.

$$\begin{aligned} \widehat{U}_1(x,g) &= z_{\mathcal{B}}(s_\theta(x,g)) - \lambda_1 \widetilde{C}_{\mathcal{B}}(g), \\ G_1(x) &= \arg\max_{g\in\mathcal{B}} \widehat{U}_1(x,g). \end{aligned} \tag{5}$$

Here, $z_{\mathcal{B}}$ standardizes scores within the query, and $\widetilde{C}_{\mathcal{B}}$ is a precomputed bank-normalized cost proxy. Adjusting λ_1 changes the deployment cost preference without retraining the matcher. The resulting utility is a cost-adjusted ranking score rather than a calibrated expected reward, and only G_1 is executed at this stage.

Marginal-Utility Structural Evolution. After executing G_1, SOME shifts from selecting a suitable standalone structure to deciding whether additional structural computation is worthwhile. The natural reference is *STOP*: retain the output of G_1 and incur no further computation. Continuation is therefore useful only when a complementary structure is expected to improve upon this baseline.

Let s_1 denote the deployment-available state after executing G_1. For each candidate $g_2 \in \mathcal{B}\setminus\{G_1\}$, SOME predicts the joint correctness of the initial and candidate structures:

$$\widehat{p}_{ij}(g_2 \mid s_1) \coloneqq \Pr_\theta(Z_1 = i, Z_2 = j \mid s_1, g_2), \qquad i,j \in \{0,1\}. \tag{6}$$

where Z_1 and Z_2 indicate the correctness of G_1 and g_2, respectively. The four outcomes distinguish rescue (01), harm (10), joint failure (00), and joint success (11). Correctness labels are used only for offline supervision; at inference time, predictions depend solely on deployment-available features.

Joint correctness alone, however, does not determine the value of continuation because the two outputs must ultimately be consolidated. Agreement retains the shared answer, whereas conflict invokes a resolver that may either recover a correct candidate answer or overturn a correct initial one. We estimate offline, for each joint state ij, the probability that the answer-resolution procedure yields a correct final answer, denoted by $\widehat{v}_{ij}$. The expected improvement over STOP is then:

$$\widehat{\Delta V}(g_2 \mid s_1) = \sum_{i,j\in\{0,1\}} \widehat{p}_{ij}(g_2 \mid s_1)\,(\widehat{v}_{ij} - i)\,. \tag{7}$$

Subtracting i anchors the comparison to retaining G_1: state 01 captures potential rescue, whereas state 10 captures possible degradation of an initially correct answer. States 00 and 11 account for any change introduced by resolution under joint failure or joint success.

SOME then discounts this expected gain by the cost of additional computation:

$$\widehat{U}_2(g_2 \mid s_1) = \widehat{\Delta V}(g_2 \mid s_1) - \lambda_2 \widetilde{C}_{add}(g_2), \tag{8}$$

where $\widetilde{C}_{add}$ is a normalized pre-execution cost proxy and $\lambda_2 \geq 0$ controls the accuracy–cost trade-off.

Let $\mathcal{E}(s_1)$ contain candidates whose predicted conditional-harm risk does not exceed τ_{harm} (Appendix B.5). When this set is nonempty, SOME selects the candidate with the highest predicted continuation utility:

$$G_2^* = \arg\max_{g_2\in\mathcal{E}(s_1)} \widehat{U}_2(g_2 \mid s_1). \tag{9}$$

The selected structure is activated only when $\widehat{U}_2(G_2^* \mid s_1) \geq \tau_{\text{gain}}$; otherwise, SOME stops and retains G_1's output. Thus, candidate selection determines which complementary structure offers the highest utility, while the gain threshold determines whether further structural computation is worthwhile. When activated, G_2^* expands the active structure set from $\{G_1\}$ to $\{G_1, G_2^*\}$ without modifying the frozen topology bank. Algorithm 2 summarizes initial matching and selective continuation. Gate definitions and cost normalizations are provided in Appendices B.5 and D.3.

5 Experiments

5.1 Experimental Setup

Benchmarks and baselines. We evaluate on five public benchmarks from their official releases: MMLU (Hendrycks et al., 2021a) for multiple-choice reasoning, MATH (Hendrycks et al., 2021b) and GSM8K (Cobbe et al., 2021) for mathematical reasoning, and MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) for code generation. For MBPP and HumanEval, we construct disjoint subsets for offline optimization and final evaluation from the official collections. For MATH, we use 3,000 training problems to construct offline query–topology supervision and, following AFlow (Zhang et al., 2025c), evaluate on difficulty-level-5 problems from Counting & Probability, Number Theory, Prealgebra, and Precalculus. The MATH frozen bank and matcher are transferred to GSM8K without additional structural search or retraining. For MMLU, we reserve 400 questions for evaluation following MasRouter (Yue et al., 2025).

Baselines include *Vanilla LLM* (Qwen3-8B (Yang et al., 2025)) with direct generation; *manually designed reasoning and agentic methods*, including CoT (Wei et al., 2022), SC-CoT@3 (Wang et al., 2023), and LLM-Debate (Du et al., 2024); and *automatically optimized or selected topologies*, including GPT-Swarm (Zhuge et al., 2024), AFlow (Zhang et al., 2025c), and MasRouter (Yue et al., 2025).

Execution model and reproducibility. We use Qwen3-8B as the common execution LLM for all main experiments to ensure reproducibility and controlled comparison. Its publicly released weights

allow the execution model to remain fixed across methods and runs. We additionally evaluate SOME with other execution models, including hosted API models, in the model-agnostic transfer experiments described in Section 5.2.

Evaluation criteria. For mathematical reasoning, we report *Solve Rate* (%), defined as the percentage of problems whose final answers are judged equivalent to the reference answers. For knowledge-based question answering, we report *Accuracy* (%), defined as the percentage of questions answered correctly. For code generation, we report *pass@1* (Chen et al., 2021), defined as the percentage of problems for which the single final program produced by the workflow passes all evaluation tests. A workflow may involve multiple internal LLM calls before producing this final program. Across all tasks, we measure inference cost by the average total number of input and output tokens consumed per problem, summed over all LLM calls in the executed workflow.

5.2 Main Results: Overall Accuracy–Cost Performance

Utility-guided inference consistently delivers strong overall performance. SOME achieves the best performance on four of the five benchmarks in Table 1, reaching 81.50% on MMLU, 71.40% on MATH, 82.70% on MBPP, and 84.73% on HumanEval. On GSM8K, where competitive methods already achieve similarly high accuracy, the performance gap is naturally smaller. Overall, SOME attains the highest macro-average score of 82.57%, outperforming MasRouter and AFlow by 2.76 and 3.73 percentage points, respectively, and demonstrating consistent gains across knowledge reasoning, mathematical reasoning, and code generation.

Table 1: Results with Qwen3-8B (Yang et al., 2025). Mul., Auto., and Util. denote multi-agent collaboration, automated structural optimization, and explicit accuracy–cost guidance. † marks utility guidance applied only at initial routing (Yue et al., 2025), whereas SOME extends it to the whole process. On GSM8K and HumanEval, the continuation policy does not activate post-execution evolution for any evaluated query, so SOME coincides with SOME-G_1 on these two benchmarks. Avg. Acc. is the unweighted mean over five benchmarks. Rel. Cost averages per-benchmark token costs normalized to Vanilla ($1.00\times$); accounting details are provided in Appendix D.3. Bold marks the best available accuracy, including ties.

Method	**Mul.**	**Auto.**	**Util.**	**Benchmarks**					**Avg. Acc.** ↑	**Rel. Cost** ↓
				MMLU	MATH	GSM8K	MBPP	HumanEval		
Vanilla	×	×	×	67.75	58.44	86.92	65.10	79.39	71.52	1.00×
CoT	×	×	×	70.70	53.50	92.23	68.03	80.91	73.07	1.79×
SC-CoT@3	×	×	×	76.00	54.94	92.42	69.21	80.15	74.54	5.37×
LLM-Debate	✓	×	×	81.00	70.37	**92.98**	80.94	83.97	81.85	38.93×
GPT-Swarm	✓	✓	×	80.50	61.72	92.16	71.85	81.68	77.58	11.59×
AFlow	✓	✓	×	80.75	63.17	**92.98**	73.31	83.97	78.84	10.72×
MasRouter	✓	✓	✓†	80.25	61.93	92.32	82.11	82.44	79.81	37.52×
SOME-G_1	✓	✓	✓†	79.25	68.72	92.51	81.82	**84.73**	81.41	11.04×
SOME	✓	✓	✓	**81.50**	**71.40**	92.51	**82.70**	**84.73**	**82.57**	19.43×

SOME provides Pareto-efficient performance–cost operating points across benchmarks. Figure 3 visualizes the accuracy–cost trade-offs on MATH, MMLU, and MBPP, with GSM8K and HumanEval reported in Appendix D.3. Across all five benchmarks, SOME attains empirical Pareto-efficient operating points, showing that its performance gains are not simply obtained by increasing inference expenditure. Notably, SOME-G_1 already provides a strong low-cost operating point, achieving a macro-average accuracy of 81.41% at only $11.04\times$ the Vanilla cost. Compared with full SOME, it sacrifices only 1.16 percentage points of accuracy while reducing relative cost by 43.2%. Full SOME then selectively invests additional computation to raise the macro-average accuracy to 82.57%, providing a higher-accuracy operating point at $19.43\times$ cost. Together, these results show that utility-aware initial matching captures most of the achievable performance efficiently, while post-execution evolution uses additional computation to obtain the marginal gains.

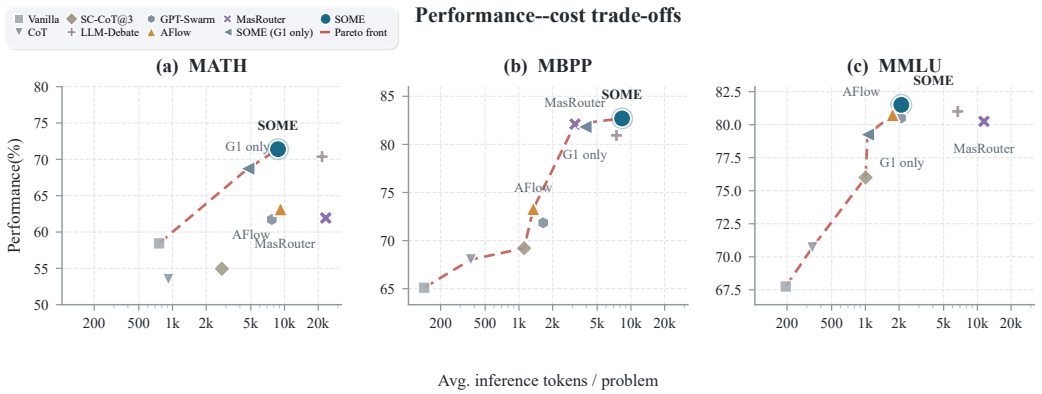


Figure 3: Performance–cost trade-offs on MATH, MBPP, and MMLU with Qwen3-8B. Performance is measured by Solve Rate, pass@1, and Accuracy, respectively. The x-axis shows the average total input and output tokens per problem across all LLM calls on a logarithmic scale. Dashed lines denote the empirical Pareto frontier formed by nondominated operating points. SOME (G_1 only) executes only the initially matched structure, without post-execution evolution. Results on GSM8K and HumanEval and exact numerical values are provided in Appendix D.3.

SOME generalizes effectively across execution LLMs without model-specific search or retraining. To examine whether SOME generalizes beyond Qwen3-8B, we directly transfer the structures and matching policy learned on Qwen3-8B to GPT-4o-mini (OpenAI, 2024) and DeepSeek-4.1-Flash (DeepSeek-AI, 2026), using them as execution LLMs on the same MATH test set without additional search or retraining (Table 2). SOME consistently outperforms the compared topology-design baselines across different execution models, demonstrating that its utility-guided structural inference generalizes beyond the backbone LLM used for training.

Table 2: Comparison of manually designed reasoning workflows and automatically optimized topology methods across three test-time executor LLMs on MATH: Qwen3-8B, GPT-4o-mini, and DeepSeek-V4.1-flash. Vanilla, CoT, and SC-CoT@3 are manually designed, while AFlow and MasRouter perform automated topology optimization. Results are reported as accuracy (%). The results demonstrate the strong transferability and model-agnostic applicability of SOME across different executor LLMs.

Model	**Methods**					
	Vanilla	CoT	SC-CoT@3	AFlow	MasRouter	**SOME**
Qwen3-8B	58.44	53.50	54.94	63.17	61.93	**71.40**
GPT-4o-mini	48.56	48.77	50.41	56.17	58.02	**59.47**
DeepSeek-V4.1-flash	69.55	72.22	72.22	85.19	87.86	**90.33**

5.3 Ablation and Mechanism Analysis

We organize our ablations around three structural decisions made by SOME: which structures enter the reusable action space, which structure is selected for initial execution, and whether post-execution structural evolution is worthwhile. Additional discovery results on MMLU and MBPP, complete five-benchmark initial-matching ablations, and continuation analyses for all benchmarks that activate post-execution evolution are provided in Appendix D.5.

Does utility-shaped discovery construct a better structural action space? We examine whether offline discovery introduces structural alternatives with higher attainable utility than the initial seeds, rather than merely enlarging the candidate set. Figure 4(a) compares two best-so-far statistics computed from the same 280 MATH search trajectories, each with 16 rollouts. Full discovery considers all structures executed so far, including dynamically generated topologies, whereas the Seed reference restricts the same statistic to seed structures encountered along those trajectories. This paired

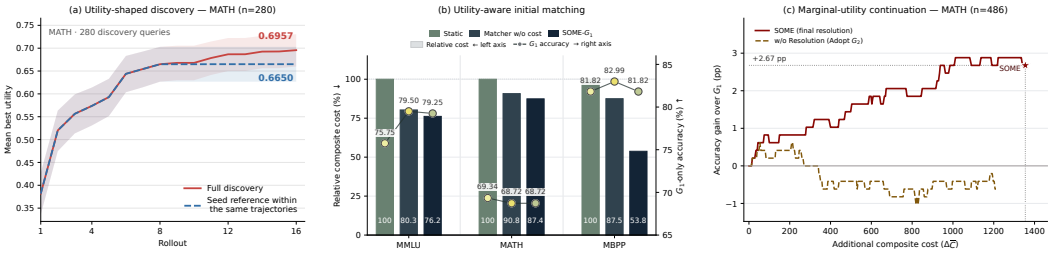


Figure 4: Ablation and mechanism analysis of SOME. **(a)** Utility-shaped discovery on 280 MATH queries over 16 rollouts; curves show mean best-so-far utility for full discovery and the seed reference within the same trajectories. **(b)** Utility-aware initial matching on MMLU, MATH, and MBPP; bars show composite cost normalized to Static, and dots show G_1-only accuracy. **(c)** Utility-guided continuation on 486 MATH queries; curves show cumulative accuracy gain against average additional composite cost, with and without final-answer resolution.

construction keeps the queries and search histories fixed, providing a controlled within-trajectory comparison of generated structures against the observed seeds. The seed reference plateaus at 0.6650, while Full discovery continues to improve and reaches 0.6957 by the final rollout, yielding a 0.0307 increase in mean best-so-far utility. Thus, on MATH, utility-shaped discovery does more than expand the candidate pool: generated structures extend the attainable action space beyond the observed seeds with higher-utility operating points under the accuracy–cost objective.

Does utility-aware initial matching realize query-adaptive structural allocation? Figure 4(b) compares three G_1-only policies on MMLU, MATH, and MBPP: *Static*, which applies one development-selected topology to all queries; *Matcher w/o cost*, which performs query-adaptive selection with $\lambda_1 = 0$; and the deployed *SOME-*G_1, which further incorporates cost awareness. Dots show G_1-only accuracy, while bars show composite cost normalized to Static within each dataset. Compared with Static, SOME-G_1 reduces cost by 23.8%, 12.6%, and 46.2% on MMLU, MATH, and MBPP, respectively, while changing accuracy by only $+3.50$, -0.62, and 0.00 percentage points. Removing the cost term yields similar accuracy but consistently higher cost. Overall, query-adaptive matching enables per-query structural allocation, while cost awareness shifts this allocation toward cheaper operating points with little loss in accuracy.

When is post-execution structural evolution worthwhile? Figure 4(c) examines this decision on 486 MATH queries by ordering candidate continuations according to their predicted marginal utility. SOME executes G_2 only when the expected benefit justifies its additional cost, selecting 303 queries under the deployed stopping criterion while retaining G_1 for the remainder. Starting from 334/486 correct predictions (68.72%), selective continuation with final-answer resolution reaches 347/486 (71.40%), yielding a net gain of 2.67 percentage points at an average additional composite cost of 1354.79. Directly adopting the same selected G_2 outputs instead yields 68.11%, below the G_1 baseline. Thus, complementary computation is worthwhile when it is selectively activated under the marginal-utility criterion and its output is resolved against the initial prediction, rather than treated as an unconditional replacement.

6 Conclusion

We introduced SOME-Flow, a utility-guided framework for structural agentic inference that unifies offline structural discovery, cost-aware initial matching, and selective post-execution evolution under a shared accuracy–cost objective. Across five reasoning and code-generation benchmarks, SOME achieves the highest macro-average performance of 82.57%, outperforming the best automated structural baseline by 2.76 percentage points. SOME-G_1 reaches 81.41%, only 1.16 points below the full method, while reducing average relative inference cost by 43.2%. These results show that coordinating reusable structural optimization, amortized matching, and marginal-utility evolution can improve reasoning performance while controlling inference cost.

AI USE STATEMENT

We used generative AI tools to aid and polish writing and to assist with research execution, including code development and debugging and providing feedback on experimental design and the interpretation of results. The tools were used only in an assistive capacity. All methodological decisions, experimental procedures, quantitative results, citations, and scientific claims were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

ETHICS STATEMENT

This work uses publicly available benchmark datasets and does not involve human-subject studies or the collection of personally identifiable or sensitive personal data. We are not aware of specific ethical concerns arising from the proposed methodology or experimental protocol beyond the general risks associated with automated language-model reasoning. All experiments and results are reported in accordance with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We have made extensive efforts to facilitate the reproducibility of SOME. The overall formulation and the three stages of the method are described in Sections 3 and 4. Appendix A provides the complete algorithms for offline discovery, policy fitting, and online structural allocation. Appendix B details the executable topology representation, utility-guided search, topology-bank construction, matcher representations and training objectives, and marginal-utility continuation policy. Appendix C specifies the task-family seeds and frozen topology banks, while Appendix D provides data partitions, training and inference configurations, token-cost accounting, transfer settings, and additional ablation results. Appendix E includes the essential task-family, communication, and resolution prompt templates, and Appendix F provides the out-of-fold construction, continuation configuration, and representative case studies illustrating structural selection, continuation, and stopping decisions. Anonymous source code and experimental configurations are available at https://anonymous.4open.science/r/SOME-EC4B.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024. doi: 10.1609/aaai.v38i16.29720. URL https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph memory-augmented agentic routing for multi-agent LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZdGB7MNQDT.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Eric Hanchen Jiang, Levina Li, Frank Wan, Xiao Liang, Sophia Yin, Yuchen Wu, Xinfeng Li, Yizhou Sun, Wei Wang, Kai-Wei Chang, and Ying Nian Wu. Dynamic generation of multi LLM agents communication topologies with graph diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 38042–38060. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1764. URL https://aclanthology.org/2026.acl-long.1764/.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ba84da6921f3040b74ee163aa7451f53-Abstract-Conference.html.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Kumar Guha, E. Kelly Buchanan, Mayee F. Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. An architecture search framework for inference-time techniques. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 52475–52507. PMLR, 2025. URL https://proceedings.mlr.press/v267/saad-falcon25a.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1170. URL https://aclanthology.org/2025.acl-long.1170/.

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*, volume 35, 2022. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

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. URL https://openreview.net/forum?id=JgvJdICc6P.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang

Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.315. URL https://aclanthology.org/2025.naacl-long.315/.

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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

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 *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=LkzuPorQ5L.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692. PMLR, 2025b. URL https://proceedings.mlr.press/v267/zhang25cu.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui 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. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=z5uVAKwmjf.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

TABLE OF CONTENTS

A Algorithms 15

A.1 Offline discovery and policy fitting 15

A.2 Online structural allocation 16

B Additional Method Details 17

B.1 Executable structures and communication 17

B.2 Utility return and complexity-modulated PUCT 17

B.3 Topology-bank construction 18

B.4 Matcher representations and training losses 18

B.5 Joint outcomes, gates, and marginal value 19

B.6 Calibration and information boundaries 20

C Task-Family Seeds and Frozen Banks 21

C.1 Mathematical reasoning: MATH and GSM8K 21

C.2 Multiple-choice reasoning: MMLU 21

C.3 Code generation: MBPP and HumanEval 22

D Experimental Details and Additional Analyses 23

D.1 Data partitions and evaluation protocol 23

D.2 Training and inference configuration 23

D.3 Token accounting and complete cost frontiers 25

D.4 Transfer across tasks and execution models 25

D.5 Ablations by structural decision 27

D.5.1 Discovery on source benchmarks 27

D.5.2 Initial matching across evaluation benchmarks 28

D.5.3 Continuation on evolution-active benchmarks 30

D.6 Operating points 31

E Essential Prompt Templates 32

E.1 Task-family role instructions 32

E.2 Communication and resolution 32

F Case Studies 34

F.1 Five-fold OOF construction 34

F.2 MATH Continuation Configuration 34

F.3 Correcting a counting error 34

F.4 Expanding a discovered singleton 35

F.5 Retaining a sufficient initial answer 35

A ALGORITHMS

We separate offline optimization from online inference. A topology is a complete executable communication structure, not a single LLM call. Offline labels are used for fitting and calibration only; reference answers and unexecuted candidate outputs are unavailable to the online policy.

A.1 OFFLINE DISCOVERY AND POLICY FITTING

Algorithm 1 Offline Discovery and Policy Fitting

Require: Offline data $\mathcal{D}$, seeds $\mathcal{S}$, rollout budget B
Ensure: Frozen bank $\mathcal{B}$, matcher s_θ, continuation policy Π_{cont}

Partition $\mathcal{D}$ into discovery, screening, training, and development subsets
$\mathcal{H} \leftarrow \varnothing$
for each $x \in \mathcal{D}_{\text{search}}$ **do**
  $\mathcal{A} \leftarrow \mathcal{S}$; initialize $P_\Omega(\cdot \mid x)$, $N(\cdot)$, and $Q(\cdot)$
  for $t = 1, \ldots, B$ **do**
    $g \leftarrow \text{SelectPUCT}(\mathcal{A}, P_\Omega, N, Q)$
    $(o, y, c) \leftarrow \text{ExecuteAndEvaluate}(x, g)$
    $r \leftarrow R_{\text{search}}(x, g)$ ▷ Appendix B.2
    $(N(g), Q(g)) \leftarrow \text{Update}(N(g), Q(g), r)$
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x, g, o, y, c, r)\}$
    if $\text{ExpansionStep}(t)$ **then**
      $g_p \leftarrow \text{SelectParent}(\mathcal{A}, \mathcal{H}_x)$
      $\mathcal{A} \leftarrow \mathcal{A} \cup \text{ValidUnique}(\text{Mutate}(g_p))$; initialize new arms
    end if
  end for
end for
$\mathcal{B} \leftarrow \text{ScreenBank}(\mathcal{H}, \mathcal{D}_{\text{screen}})$
$\mathcal{T} \leftarrow \text{DenseExecute}(\mathcal{D}_{\text{train}}, \mathcal{B})$
$\theta \leftarrow \text{FitMatcher}(\mathcal{T}, \mathcal{L}_{\text{match}})$; select λ_1 on development data
$\mathcal{P} \leftarrow \text{BuildOOFPairs}(\mathcal{T}, \mathcal{B})$
$\Pi_{\text{cont}} \leftarrow \text{FitAndCalibrate}(\mathcal{P})$
return $\text{Freeze}(\mathcal{B}, s_\theta, \lambda_1, \Pi_{\text{cont}})$

Here, $\mathcal{H}_x$ denotes execution records for query x. Π_{cont} includes the joint-outcome predictors, reference resolver profile, and frozen continuation settings.

A.2 Online Structural Allocation

Algorithm 2 Initial Matching and Selective Continuation

Require: Query x, frozen bank $\mathcal{B}$, matcher s_θ, continuation policy Π_{cont}
Ensure: Final answer o

$G_1 \leftarrow \arg\max_{g \in \mathcal{B}} \big[z_{\mathcal{B}}(s_\theta(x, g)) - \lambda_1 \widetilde{C}_{\mathcal{B}}(g) \big]$
$o_1 \leftarrow \mathrm{Execute}(x, G_1)$
$s_1 \leftarrow \mathrm{BuildState}(x, G_1, o_1)$
if $\neg\,\mathrm{Enabled}(\Pi_{\mathrm{cont}})$ or $\neg\,\mathrm{Admit}(s_1)$ **then return** o_1
$\mathcal{E} \leftarrow \varnothing$
for each $g \in \mathcal{B} \setminus \{G_1\}$ **do**
  $\{\widehat{p}_{ij}(g \mid s_1)\}_{i,j \in \{0,1\}} \leftarrow \mathrm{PredictJoint}(s_1, g)$
  $\widehat{h}(g \mid s_1) \leftarrow \mathrm{ConditionalHarm}(\widehat{p}_{10}, \widehat{p}_{11})$
  if $\widehat{h}(g \mid s_1) \le \tau_{\mathrm{harm}}$ **then**
    $\widehat{\Delta V}(g \mid s_1) \leftarrow \sum_{i,j} \widehat{p}_{ij}(g \mid s_1)(\widehat{v}_{ij} - i)$
    $\widehat{U}_2(g \mid s_1) \leftarrow \widehat{\Delta V}(g \mid s_1) - \lambda_2 \widetilde{C}_2(g)$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{g\}$
  end if
end for
if $\mathcal{E} = \varnothing$ **then return** o_1
$G_2^* \leftarrow \arg\max_{g \in \mathcal{E}} \widehat{U}_2(g \mid s_1)$
if $\widehat{U}_2(G_2^* \mid s_1) < \tau_{\mathrm{gain}}$ **then return** o_1
$o_2 \leftarrow \mathrm{Execute}(x, G_2^*)$
if $\mathrm{Agree}(o_1, o_2)$ **then return** o_1
return $\mathrm{Resolve}(x, o_1, o_2)$

$\mathrm{BuildState}$ retains only features used by the deployed policy: MATH/MMLU exclude generated-answer features, whereas Code's admission model additionally uses o_1. $\mathrm{ConditionalHarm}$ computes $\widehat{p}_{10}/(\widehat{p}_{10} + \widehat{p}_{11})$ with the numerical conventions in Appendix B.5. Utility ties favor lower structural cost. $\mathrm{Resolve}$ follows the task-specific output and retry contracts in Appendix E.2.

B Additional Method Details

B.1 Executable Structures and Communication

A topology $g = (V, E, K, A)$ contains role-conditioned nodes, undirected edges, K interaction rounds, and aggregation. Round zero generates an independent output at each executable node. Later updates use preceding-round messages:

$$o_v^{(0)} = f_v(x; I_v), \qquad o_v^{(k)} = f_v\Big(x, o_v^{(k-1)}, \{m_u^{(k-1)} : u \in \mathcal{N}(v)\}; I_v\Big). \tag{10}$$

Calls may be scheduled sequentially, but message passing is synchronous: a node never consumes another node's partially updated state from the same round. A generator–reviewer–refiner loop generally requires $K = 2$. A chain denotes undirected connectivity, not sequential program execution.

Messages contain compact previous-round outputs alongside the original problem. Mathematical and multiple-choice tasks aggregate final node answers by majority vote; code generation uses a task-specific final-program selector.

B.2 Utility Return and Complexity-Modulated PUCT

Let $n = |V|$, $e = |E|$, and $L(g)$ be the static call estimate. With ordinary one-call nodes and rule aggregation, $L(g) = n(1 + K)$; retries can increase actual calls. The discovery proxy and normalization are

$$S(g) = 256L(g)(1 + 0.15e_{\text{eff}}), \quad e_{\text{eff}} = e\mathbb{I}[K > 0], \quad \widetilde{C}(g) = \min\left\{1, \frac{\log(1 + S(g))}{\log(1 + C_{\max})}\right\}. \tag{11}$$

The utility implementation regularizes normalized workload:

$$\Omega_{\text{reg}}(g) = \text{clip}_{[0,1]}\left(0.35\frac{e}{e_{\text{ref}}} + 0.25\frac{K}{K_{\text{ref}}} + 0.20\frac{n}{n_{\text{ref}}} + 0.20\frac{e\max(1,K)}{e_{\text{ref}}K_{\text{ref}}}\right). \tag{12}$$

where $(n_{\text{ref}}, e_{\text{ref}}, K_{\text{ref}}) = (3, 3, 1)$ for MATH/MMLU and $(4, 6, 2)$ for MBPP. These reference values normalize structural complexity in the discovery utility; they are not hard limits on the executable topology space.

Topology utility R_U is computed as in Equation 2, with task-specific weights reported in Table 7 and $C_{\max} = 4096$. Search updates use:

$$R_{\text{search}}(x, g) = \begin{cases} R_U(x, g), & \text{MATH/MBPP}, \\ Y(x, g) + 0.1\, R_U(x, g), & \text{MMLU}. \end{cases} \tag{13}$$

where Y denotes execution correctness.

Prior. The library computes a role-prior average plus task-specific synergy bonuses; MATH/GSM8K programmer structures also receive a query-keyword adjustment. Denote this score by $P_0(g \mid x)$. Dynamic arms use

$$\Omega_{\text{exp}} = 0.40n + 0.30e + 0.30K, \qquad P_\Omega(g \mid x) = \max\{10^{-4}, P_0(g \mid x)e^{-0.15\Omega_{\text{exp}}(g)}\}. \tag{14}$$

These scores need not sum to one. P_0 is a heuristic discovery prior, not the later-trained matcher. Search complexity and utility regularization differ.

Selection. For arm visits $N_t(g)$, total visits N_t, and mean search reward $Q_t(g)$, the root-level PUCT rule is

$$g_t = \arg\max_{g \in \mathcal{A}_t} \left[Q_t(g) + c_{\text{puct}} P_\Omega(g \mid x)\frac{\sqrt{\max(N_t, 1)}}{1 + N_t(g)} - \beta_{\text{sel}}(0.02n + 0.02e + 0.03K)\right]. \tag{15}$$

Unvisited arms use $Q_{\text{init}} = 0$. Warm-up tries highest-prior unvisited arms; each execution updates the selected arm's visit count and mean search reward. Local expansion modifies roles or edges, or changes connectivity to a chain or complete graph. Only valid, distinct executable structures are retained; code-generation structures must contain a generator.

B.3 Topology-Bank Construction

For an offline query set $\mathcal{Q}$, define

$$\mathcal{S}_g = \{x \in \mathcal{Q} : Y(x,g) = 1\}, \quad \mathrm{Cov}(B) = \left|\bigcup_{g \in B} \mathcal{S}_g\right|, \quad c_B^*(x) = \min_{g \in B : Y(x,g)=1} c(x,g). \tag{16}$$

Coverage is an offline oracle diagnostic, not an online policy score. A lower-accuracy topology can solve otherwise missed queries or provide cheaper successful executions.

Code. Screening evaluates 19 candidates on 50 MBPP queries. With direct generation retained as a low-cost anchor, subset selection preserves the solved-query set and per-query cheapest-correct profile while minimizing bank size. Screening uses $c = S + 0.1P$, as defined in Appendix D.3; non-dominated near-cheapest or complementary candidates may also be retained. The resulting eight-structure bank preserves the full pool's 44/50 screening coverage.

MATH. The final 11-structure bank excludes the programmer–referee pair: dense training executions show that its aggregation systematically favors the weaker programmer output when the two nodes disagree. This pruning uses training evidence only.

MMLU. The frozen bank contains eight seeds and six discovered alternatives. Both initial matching and continuation use the same 14 structures and their fixed cost descriptors. All task-family banks are listed in Appendix C.

B.4 Matcher Representations and Training Losses

Query features concatenate a frozen embedding of task-prefixed query text, standardized general query features, and a task-profile vector. Topology features concatenate mean/max pooled role-text embeddings, degree-weighted embeddings, mean absolute endpoint-embedding differences, capability means and presence, operator-type frequencies, and structural features. These include counts, density, rounds, components, degree statistics, graph-shape indicators, cost/call descriptors, and aggregator indicators. Query normalization statistics are fitted on training data and reused at inference.

Two projection networks produce $h_x, h_g \in \mathbb{R}^{128}$. Explicit compatibility features are

$$\phi = [h_x; h_g; h_x \odot h_g; |h_x - h_g|; \cos(h_x, h_g)], \qquad s_\theta = f_{\mathrm{shared}}(\phi) + 0.25 f_{\mathrm{refine}}(\phi). \tag{17}$$

Static descriptors are cached. Execution-dependent cost features use frozen training representatives, never an unexecuted candidate's held-out outputs. Training settings appear in Appendix D.2.

Groupwise objective. Let $\mathcal{P}_i$ and $\mathcal{N}_i$ be successful and unsuccessful candidates for query i, and write $s_{ig} = s_\theta(x_i, g)$. Set $\mathcal{P}_i^{\mathrm{rob}} = \mathcal{P}_i$ when at least two candidates succeed, and $\mathcal{P}_i^{\mathrm{rob}} = \varnothing$ otherwise. For mixed groups with sufficient positive support,

$$\ell_{\mathrm{set},i} = \log \sum_{g \in \mathcal{B}} e^{s_{ig}} - \log \sum_{g \in \mathcal{P}_i^{\mathrm{rob}}} e^{s_{ig}}, \tag{18}$$

$$\ell_{\mathrm{rank},i} = \frac{1}{|\mathcal{P}_i^{\mathrm{rob}}||\mathcal{N}_i|} \sum_{p \in \mathcal{P}_i^{\mathrm{rob}}} \sum_{n \in \mathcal{N}_i} \mathrm{softplus}(m - s_{ip} + s_{in}). \tag{19}$$

Both terms are zero if their required sets are empty. With label smoothing ϵ, $\widetilde{y} = \epsilon + (1 - 2\epsilon)y$, and pair weight $w = 0.25$ for an isolated positive and 1 otherwise,

$$\ell_{\mathrm{bce},i} = \frac{\sum_g w_{ig}\, \mathrm{BCEWithLogits}(s_{ig}, \widetilde{y}_{ig})}{\max(\sum_g w_{ig}, 10^{-6})}, \tag{20}$$

$$\mathcal{L} = \frac{\sum_i \omega_i(\alpha_{\mathrm{set}}\ell_{\mathrm{set},i} + \alpha_{\mathrm{rank}}\ell_{\mathrm{rank},i} + \alpha_{\mathrm{bce}}\ell_{\mathrm{bce},i})}{\max(\sum_i \omega_i, 10^{-8})}. \tag{21}$$

For mixed groups, $\omega_i = t_i[0.5 + \min(n_i^+, n_i^-)/(n_i^+ + n_i^-)]$. The support factor is 0.75 for direct-only groups, 0.5 for isolated interaction-rescue/dynamic-new successes, and 1 otherwise. Uniformly successful/failed groups receive 0.1.

Deployment ranking. For $M = |\mathcal{B}|$, score normalization uses sample standard deviation and cost uses min–max normalization:

$$z_g = \frac{s_g - \bar{s}}{\sqrt{\sum_j (s_j - \bar{s})^2/(M-1)}}, \qquad \widetilde{C}_{\mathcal{B}}(g) = \frac{S(g) - S_{\min}}{S_{\max} - S_{\min}}. \tag{22}$$

A numerically constant vector maps to zeros (score standard deviation at most 10^{-6}); cost normalization also guards against a zero span. Selection uses the first maximum in frozen candidate order. This ranking is not a calibrated estimate of correctness minus monetary cost.

B.5 Joint Outcomes, Gates, and Marginal Value

For the factorized predictor, let $a = \widehat{\Pr}(Z_1 = 1 \mid s_1)$, $b_0 = \widehat{\Pr}(Z_2 = 1 \mid Z_1 = 0, s_1, g_2)$, and $b_1 = \widehat{\Pr}(Z_2 = 1 \mid Z_1 = 1, s_1, g_2)$. Suppressing the conditioning arguments, the joint predictions are

$$(\widehat{p}_{00}, \widehat{p}_{01}, \widehat{p}_{10}, \widehat{p}_{11}) = ((1-a)(1-b_0), (1-a)b_0, a(1-b_1), ab_1). \tag{23}$$

This factorization allows dependent correctness. Expanding the main-text prediction gain gives

$$\widehat{\Delta V} = \widehat{p}_{00}\widehat{v}_{00} + \widehat{p}_{01}\widehat{v}_{01} - \widehat{p}_{10}(1 - \widehat{v}_{10}) - \widehat{p}_{11}(1 - \widehat{v}_{11}). \tag{24}$$

For MATH, the approximation $\widehat{v}_{00} = 0$, $\widehat{v}_{11} = 1$ reduces this to

$$\widehat{\Delta V}_{\text{two}} = r_{01}(1-a)b_0 - h_{10}a(1-b_1), \tag{25}$$

where $r_{01} = \widehat{v}_{01}$ and $h_{10} = 1 - \widehat{v}_{10}$. These are reference-resolution estimates, not correctness guarantees. Code uses a direct four-state predictor and a separate answer-aware admission model. Its gain retains all four terms in Equation 24; only the admission model reads the generated G_1 program.

Utility-based selection. When post-execution evolution is enabled, SOME ranks eligible complementary structures by a resolver-aware continuation utility:

$$\widehat{U}_2(g_2 \mid s_1) = \widehat{\Delta V}(g_2 \mid s_1) - \lambda_2 \widetilde{C}_{add}(g_2), \tag{26}$$

where $\widetilde{C}_{add}(g_2)$ is a pre-execution structural-cost proxy and $\lambda_2 \geq 0$ is dataset-specific. The candidate with the largest $\widehat{U}_2$ is selected, and continuation is activated only if its utility meets τ_{gain}. Thus, the third stage uses utility-based selection for all datasets, while the cost normalization and λ_2 value may differ across task families.

Eligibility and stopping. Conditional error risk defines the eligible set:

$$\widehat{h}(g_2 \mid s_1) = \frac{\widehat{p}_{10}}{\widehat{p}_{10} + \widehat{p}_{11}}, \qquad \mathcal{E}(s_1) = \{g_2 \in \mathcal{B} \setminus \{G_1\} : \widehat{h}(g_2 \mid s_1) \leq \tau_{\text{harm}}\}. \tag{27}$$

For the factorized predictor, this risk is $1 - b_1$, so $\tau_{\text{harm}} = 1 - \tau_{\text{safe}}$. The conditional form $1 - b_1$ avoids dividing by a numerically vanishing initial-success probability. For a nonempty eligible set,

$$G_2^* = \arg\max_{g_2 \in \mathcal{E}(s_1)} \widehat{U}_2(g_2 \mid s_1). \tag{28}$$

Let $\Gamma(s_1)$ denote query admission. Conditional on dataset-level activation, the final execution rule is

$$\text{Continue}(s_1) = \mathbb{I}\left[\Gamma(s_1) = 1 \ \wedge \ \mathcal{E}(s_1) \neq \varnothing \ \wedge \ \max_{g_2 \in \mathcal{E}(s_1)} \widehat{U}_2(g_2 \mid s_1) \geq \tau_{\text{gain}}\right], \tag{29}$$

with STOP when the eligible set is empty. Factorized admission tests $1 - a \geq \tau_{\text{risk}}$; Code uses its actionable-rescue admission model. Threshold comparisons are inclusive. Utility ties favor lower structural cost, followed by frozen candidate order.

MATH configuration. MATH uses the two-state gain above and

$$\widetilde{C}_2(g_2) = S(g_2)/1000, \qquad \lambda_2 = 0.005, \tag{30}$$

with $\tau_{\text{risk}} = 0.3$, $\tau_{\text{safe}} = 0.7$, and $\tau_{\text{gain}} = 0$. The fixed scaling of S differs from the bank-wise min–max normalization used in initial matching. This selection proxy excludes resolver cost; evaluation counts both executed topologies and resolution.

Dataset-level activation. An offline gate enables the query policy when expected gain, raw net gain, and expected gain per structural cost are positive, at least five directional correctness changes are observed, and at least 90% of fold gains are nonnegative. It additionally requires

$$\Pr(\theta > 1/2 \mid \theta \sim \mathrm{Beta}(1 + F_+, 1 + F_-)) \geq 0.95, \tag{31}$$

where F_+ and F_- count strictly positive and negative fold gains. This is an empirical activation rule, not a formal risk guarantee.

B.6 Calibration and Information Boundaries

Out-of-fold supervision splits at query level, keeping all topology alternatives for a query together. Joint-outcome predictors are calibrated using offline predictions, and reference resolver rates use offline executions with $\mathrm{Beta}(1, 1)$ smoothing. Split sizes and training settings are reported in Appendix D.

Reference correctness and hidden tests are supervision signals, not online features. MATH/MMLU continuation uses query, matcher, and structural features; Code's admission model additionally uses the executed first program. Neither policy accesses an unexecuted G_2 output. Resolver inputs contain the problem, executed candidates, and permitted evidence, but no correctness labels or hidden tests.

Offline resolution outcomes provide the state-wise correctness estimates used to compute continuation gain. Final inference uses task-specific resolution prompts and output contracts, with realized rescue and harm evaluated on the resulting trajectories.

C Task-Family Seeds and Frozen Banks

The seed library has three task families: MATH/GSM8K, MMLU, and MBPP/HumanEval. Below, $u - v - w$ denotes undirected chain edges; $u + (v - w)$ an isolated node plus one edge. Role abbreviations are family-local and map to Section E.1.

C.1 Mathematical Reasoning: MATH and GSM8K

D, S, V, C, R, P denote direct solver, structured solver, verifier, challenger, referee, and programmer. The **six seeds** are D ($K = 0$), C–R, P–R, D–C–R, V–C–R, and D–P–R (all $K = 1$). MATH/GSM8K share these seeds but use different answer-format adapters. GSM8K transfer reuses the MATH-trained bank. The initial seed library and retained bank are distinct: the P–R seed is excluded from Bank11 following the training-based pruning described in Appendix B.3. The table specifies the frozen executable bank used for both inference stages.

Table 3: MATH Bank11. Dynamic suffixes identify exact saved structures. S is a structural proxy, not tokens.

Candidate / suffix	Connectivity	K	$\lvert E\rvert$	S
Direct	D	0	0	256.0
Challenger–referee	C–R	1	1	1177.6
Direct–challenger–referee	D–C–R	1	2	1996.8
Direct–programmer–referee	D–P–R	1	2	1996.8
Verifier–challenger–referee	V–C–R	1	2	1996.8
Dynamic verifier / `f3b8a413`	V	0	0	256.0
Dynamic challenger / `f4f5f2d9`	C	0	0	256.0
Dynamic referee / `a3f7b20f`	R	0	0	256.0
Dynamic add-role / `114479bb`	C–P–R	1	2	1996.8
Dynamic add-role / `9f78f02f`	P–R–C	1	2	1996.8
Dynamic remove-edge / `4239891c`	(V–C)+R	1	1	1766.4

C.2 Multiple-Choice Reasoning: MMLU

A, T, O, K, S, C, V, R denote direct answer, direct CoT, option eliminator, knowledge solver, subject expert, skeptical challenger, consistency verifier, and final referee. The **eight seeds** are A and T ($K = 0$), O–R, K–R, S–R, T–C–R, O–V–R, and K–V–R ($K = 1$). All eight are retained.

Table 4: MMLU Bank14 in manifest order. Role K is knowledge solver; numeric K is the interaction-round count.

Candidate / suffix	Connectivity	K	S
Option–referee	O–R	1	1177.6
Knowledge–referee	K–R	1	1177.6
Knowledge–verifier–referee	K–V–R	1	1996.8
CoT–skeptic–referee	T–C–R	1	1996.8
Direct answer	A	0	256.0
Direct CoT	T	0	256.0
Option–verifier–referee	O–V–R	1	1996.8
Subject–referee	S–R	1	1177.6
Dynamic knowledge / `c779804d`	K	0	256.0
Dynamic option / `800f60b5`	O	0	256.0
Dynamic add-role / `74ede6b7`	T–K	1	1177.6
Dynamic add-role / `91902dda`	T–O	1	1177.6
Dynamic replace-role / `1ed2bb27`	O–K–R	1	1996.8
Dynamic replace-role / `fc9f0ff6`	T–O–R	1	1996.8

C.3 CODE GENERATION: MBPP AND HUMANEVAL

G, A, R, D, T, F denote generator, algorithm designer, reviewer, debugger, unit tester, and refiner. The **seven seeds** are G ($K = 0$), A–G ($K = 1$), G–R, G–D, G–R–F, G–D–F, and G–T–F ($K = 2$). The unit tester supplies specification-derived feedback without access to hidden tests; its name does not imply execution of the evaluation suite.

Table 5: Code8 bank shared by MBPP/HumanEval. The final structure intentionally leaves G isolated while D and F exchange messages.

Candidate / suffix	Connectivity	K	S
Direct generation	G	0	256.0
Design–generate	A–G	1	1177.6
Generate–review	G–R	2	1766.4
Generate–test–refine	G–T–F	2	2995.2
Dynamic add-role / `bdba6ab8`	G–F	1	1177.6
Dynamic add-role / `248b9cf9`	G–T–F–D	2	4454.4
Dynamic add-role / `69260726`	A–G–D	2	2995.2
Dynamic remove-edge / `e811300e`	G+(D–F)	2	2649.6

D Experimental Details and Additional Analyses

This section complements the main results with SOME's experimental settings, complete cost frontiers, transfer results, and stage-wise analyses.

D.1 Data Partitions and Evaluation Protocol

Table 6: SOME's offline supervision and evaluation sizes, counted in queries. Shared denotes transfer of an existing task-family bank and learned models.

Dataset	Matcher train	Continuation target	Dev/support	Final	Bank
MATH	824	1138	200	486	11
MMLU	824	2535	200	400	14
MBPP	289	569	160	341	8
HumanEval	shared	shared	33	131	8
GSM8K	shared	shared	264	1055	11

Mathematical reasoning. MATH starts from dense executions on 3,000 training problems; filtering produces the 824-query matcher-fitting set and the 1,138-query continuation-supervision set. The 486-query MATH evaluation set contains level-5 problems in Prealgebra, Number Theory, Precalculus, and Counting & Probability. The five-fold OOF construction is detailed in Appendix F.1. GSM8K reuses the frozen MATH topology bank and matcher. Its split contains 264 support and 1,055 evaluation questions. The support set is used only for the task-level decision of whether to enable post-execution evolution. Under this activation check, GSM8K retains G_1 without additional evolution. No additional topology discovery or matcher retraining is performed on GSM8K.

Multiple-choice and code tasks. MMLU uses 200 development and 400 evaluation questions, with subjects taken from canonical metadata. Its continuation target contains 2,535 queries and 32,955 ordered first-structure/alternative pairs. The Code support collection contains 629 MBPP and 33 HumanEval problems. MBPP uses 289 no-all-correct query groups for matcher fitting and 569 groups for continuation supervision. The MBPP development and final partitions are completely disjoint: final-341 examples do not enter development, checkpoint selection, or development-based configuration selection. HumanEval transfers the Code bank and matcher to a separate 131-query evaluation partition.

Grading. MATH uses normalized mathematical-answer equivalence, GSM8K numerical-answer grading, and MMLU final-option accuracy. MBPP and HumanEval report pass@1 for one submitted program per workflow. Visible examples and assertions may enter the code task contract; hidden tests are used only for evaluation.

D.2 Training and Inference Configuration

The deployed initial-matching cost coefficients are $\lambda_1 = 0.25$ for MATH/GSM8K, 0.15 for MMLU, and 2 for MBPP/HumanEval.

Dataset-specific continuation parameters are summarized in Table 9. The MATH cost scaling and worked decision examples are provided in Appendix F.2.

Execution and resolution. Qwen3-8B topology execution uses temperature 0.7, top-$p = 1$, a 1,024-token response limit, and disabled thinking. Final-answer resolution uses deterministic decoding with temperature 0 and top-$p = 1$. For mathematical and multiple-choice tasks, the resolver verifies the executed candidate outputs against the original problem and returns a single final answer, with a bounded output budget. For code generation, resolution selects one of the executed candidate programs without rewriting it.

Randomness and implementation. The split/OOF seed is 2027. MATH and Code discovery use seed 20260805; selected matcher seeds are 2 for MATH and 3 for MMLU and Code. Factorized

Table 7: Discovery Configuration. Task-specific settings are followed by shared search settings. Discovery curves report utility for all three sources.

Setting	MATH	MMLU	MBPP
Discovery queries	280	1,300	90
Rollouts per query	16	16	16
Warm-up arms	6	8	7
Expansion start	rollout 8	rollout 8	rollout 8
Expansion interval	every 2 rollouts	every 2 rollouts	every 2 rollouts
Search update signal	Utility	Correctness $+0.1\times$ utility	Utility
Utility weights (w_y, w_c, w_s)	$(0.8, 0.2, 0.05)$	$(0.85, 0.15, 0.05)$	$(0.8, 0.2, 0.05)$

Shared search setting	Value
PUCT coefficient	$c_{\text{puct}} = 1.2$
Initial value	$Q_{\text{init}} = 0$
Expansion limit	At most 4 children per expansion
Structure limit	At most 4 nodes and 4 edges
Cost normalization bound	$C_{\max} = 4096$

Table 8: Shared Matcher Configuration.

Setting	Value
Text encoder	Frozen `all-MiniLM-L6-v2`
Encoder batch size	64
Maximum text length	256 tokens
Projection dimension	128
Dropout	0.25
Refinement scale	0.25
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	0.01
Query-group batch size	24
Maximum epochs	100
Early-stopping patience	20
Loss weights	$(\alpha_{\text{set}}, \alpha_{\text{rank}}, \alpha_{\text{bce}}) = (1, 0.25, 0.5)$
Ranking margin	0.5
Label smoothing	0.05
Isolated-positive BCE weight	0.25

Table 9: Continuation settings. Code uses answer-aware actionable admission; MATH and MMLU use factorized risk admission.

Setting	MATH	MMLU	Code
τ_{risk}	0.30	0.45	–
τ_{safe}	0.70	0.60	0.80
τ_{gain}	0	0	0.005
$\tau_{\text{actionable}}$	–	–	0.40
λ_2	0.005	0	0
Reference r_{01}	0.477733	0.470199	four-state profile
Reference h_{10}	0.230769	0.241722	four-state profile
Refit epochs (A, B_0, B_1)	$(36, 8, 26)$	$(20, 14, 11)$	OOF-native fold 0

refit seeds for (A, B_0, B_1) are $(2001, 1, 1101)$. GSM8K inherits MATH's learned parameters, and HumanEval inherits Code's. Results describe selected checkpoints and saved execution realizations. The implementation uses PyTorch, sentence transformers, an OpenAI-compatible executor endpoint, and task-specific answer/program graders.

Baseline context. Baselines follow the main-text grouping: Vanilla LLM provides direct generation; CoT, SC-CoT@3, and LLM-Debate are manually designed reasoning and agentic methods; GPT-Swarm, AFlow, and MasRouter use automatically optimized or selected topologies. AFlow optimizes a task-level workflow, whereas the reproduced MasRouter configuration permits up to six agents. Their main-table operating points contextualize SOME's accuracy–cost allocation. The detailed ablations below focus on SOME's own components.

D.3 Token Accounting and Complete Cost Frontiers

For query i, inference tokens sum input and output usage over all included topology and resolver attempts:

$$T_i = \sum_{a \in \mathcal{A}_i} (T_{ia}^{\text{input}} + T_{ia}^{\text{output}}), \qquad \overline{T} = N^{-1} \sum_i T_i. \tag{32}$$

Offline discovery, dense execution, fitting, and calibration are separate expenditures. The main table averages benchmark-normalized costs:

$$\text{RelCost}(m) = \frac{1}{5} \sum_{d \in \mathcal{D}} \frac{\overline{T}_{m,d}}{\overline{T}_{\text{Vanilla},d}}, \quad \mathcal{D} = \{\text{MMLU}, \text{MATH}, \text{GSM8K}, \text{MBPP}, \text{HumanEval}\}. \tag{33}$$

Table 10: Mean input-plus-output tokens per problem underlying the main-table relative costs. Ratios are computed before rounding.

Method	MMLU	MATH	GSM8K	MBPP	HumanEval	Rel. Cost
Vanilla	194.61	758.61	246.45	142.46	250.54	1.00
CoT	335.89	919.28	399.43	372.02	446.73	1.79
SC-CoT@3	1005.00	2758.15	1194.68	1113.49	1346.21	5.37
LLM-Debate	6689.09	21647.44	9885.83	7430.04	9892.67	38.93
GPT-Swarm	2084.35	7710.19	4364.56	1647.18	1954.35	11.59
AFlow	1755.24	9223.04	4100.04	1343.75	1594.34	10.72
MasRouter	11449.49	23346.76	10766.84	3169.40	8034.39	37.52
SOME-G_1	1045.05	4700.88	2323.68	3887.56	1731.13	11.04
SOME	2094.29	8750.59	2323.68	8336.77	1731.13	19.43

Structural proxy for within-SOME ablations. The ablations additionally report the saved composite proxy

$$P_i = \max(1, |x_i|/4) A_i + O_i, \tag{34}$$

$$S_i = S(G_{1,i}) + \mathbb{I}[G_2 \text{ executed}] S(G_{2,i}) + 256 R_i, \tag{35}$$

$$C_i^{\text{proxy}} = S_i + 0.1 P_i. \tag{36}$$

Here A_i counts calls, O_i recorded output tokens, and R_i resolver attempts. This combines structural workload with approximate text workload; it is not an additional token bill. Selection penalties use pre-execution structural descriptors, whereas reported token costs use recorded executions.

D.4 Transfer Across Tasks and Execution Models

GSM8K and HumanEval. SOME transfers its MATH structural action space to GSM8K and its Code action space to HumanEval. The evaluated stopping configurations retain G_1 for every query. Table 11 completes the numerical panels omitted from the main cost figure. GSM8K uses

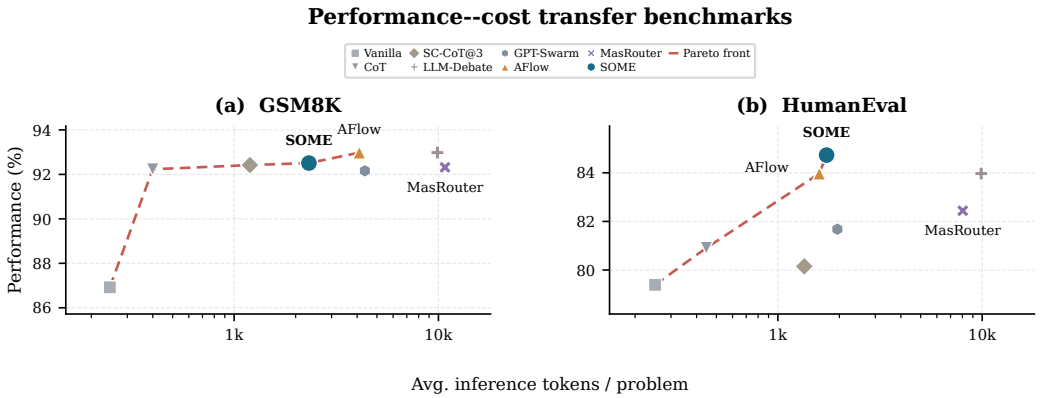


Figure 5: Additional accuracy–cost panels for GSM8K and HumanEval, using the same colors, markers, logarithmic token axis, and frontier definition as Figure 3. SOME and SOME-G_1 coincide because neither evaluation activates continuation.

Table 11: Task-transfer operating points. Accuracy and mean input-plus-output tokens follow the main-table evaluation configuration.

Method	GSM8K (1055) Solve rate (%)	GSM8K (1055) Tokens	HumanEval (131) pass@1 (%)	HumanEval (131) Tokens
Vanilla	86.92	246.45	79.39	250.54
CoT	92.23	399.43	80.91	446.73
SC-CoT@3	92.42	1194.68	80.15	1346.21
LLM-Debate	92.98	9885.83	83.97	9892.67
GPT-Swarm	92.16	4364.56	81.68	1954.35
AFlow	92.98	4100.04	83.97	1594.34
MasRouter	92.32	10766.84	82.44	8034.39
SOME-G_1 / SOME	92.51	2323.68	**84.73**	1731.13

6,186 model calls and 2,451,487 input-plus-output tokens; HumanEval uses 337 calls and 226,778 tokens. SOME reaches 92.51% on GSM8K at 43.33% less token cost than AFlow, with a 0.47-point accuracy difference. On HumanEval it improves on AFlow by 0.76 points for 8.58% more tokens. Both are empirical accuracy–cost trade-offs rather than additional continuation gains.

API executor transfer on MATH. The completed API runs keep the Qwen3-trained topology bank, matcher, and utility-guided structural decisions, while re-executing the selected structures with each target model. All executors are evaluated on the exact same 486 MATH instances, enabling a matched comparison across execution models. The fixed SOME policy uses $\lambda_1 = 0.25$ and $\lambda_2 = 0.005$, activates continuation on 303 queries, and stops after G_1 on the remaining 183, matching the Qwen3 configuration used in the main experiments. No target-model topology search, matcher retraining, or route re-optimization is performed. Both target models use temperature 0.7, top-$p = 1$, and a 1,024-token topology-output limit; final-answer resolution uses deterministic decoding with temperature 0 and the same resolution procedure across executors.

Table 12: Completed SOME transfer results on the same 486 MATH queries. Costs include topology execution and recorded resolution attempts.

Executor	G_1 correct	Final correct	G_1 (%)	Full (%)	G_1 tokens	Full tokens
GPT-4o-mini	286	289	58.85	59.47	4662.95	9580.85
DeepSeek-V4.1-flash	428	439	88.07	90.33	3801.81	7725.72

GPT-4o-mini has 152 conflicts requiring resolution, with 21 rescues and 18 harms; DeepSeek has 61, with 17 rescues and 6 harms. The resulting net improvements over G_1 are 0.62 and 2.26 percentage points, respectively. Full SOME uses 5,307 model calls for GPT-4o-mini and 4,973 for DeepSeek, including 409 and 76 resolver calls, respectively. These results show that the fixed structural decisions retain positive post-execution gains across different executors, without target-model search or matcher retraining.

D.5 Ablations by Structural Decision

We evaluate discovery on three source benchmarks, initial matching on all five benchmarks, and continuation on the three benchmarks where post-execution evolution is activated (Table 13). Discovery is performed independently on MMLU, MATH, and MBPP. The resulting MATH bank is transferred to GSM8K, and the MBPP bank to HumanEval, without additional structural search. All five benchmarks use initial matching; GSM8K and HumanEval use initial matching only and do not activate post-execution structural evolution.

Table 13: Experimental scope of each structural decision. Transfer reuses the source bank rather than repeating discovery.

Evaluation	Discovery source	Initial matching	Continuation
MMLU	MMLU	Yes	Yes
MATH	MATH	Yes	Yes
GSM8K	MATH (transfer)	Yes	No
MBPP	MBPP	Yes	Yes
HumanEval	MBPP (transfer)	Yes	No

D.5.1 Discovery on Source Benchmarks

We complement the detailed MATH discovery trajectory in the main text with source-specific diagnostics for MBPP and MMLU. By rollout 16, full discovery reaches a mean best-so-far utility of 0.6957, compared with 0.6650 for the observed-seed reference, and generated structures provide a strictly better final best-so-far utility on 23 of 280 queries. Figure 6 adds the analogous MBPP and MMLU discovery trajectories.

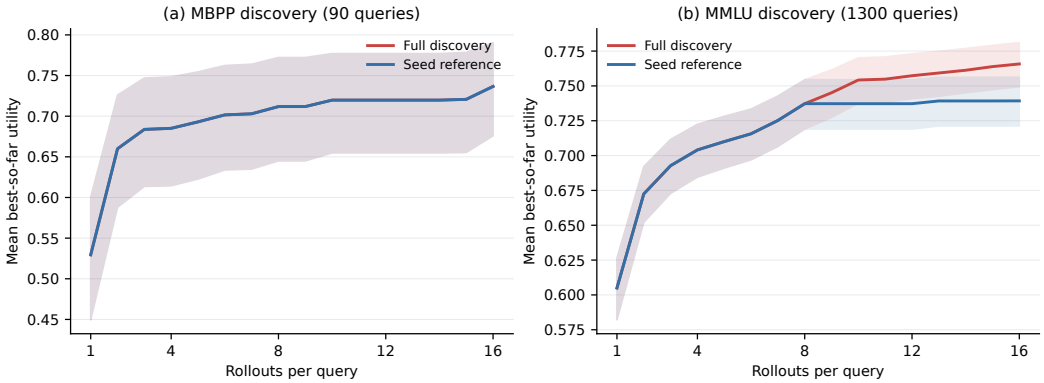


Figure 6: Discovery trajectories on the additional source benchmarks. Left: MBPP best-so-far utility over 16 rollouts on 90 queries; shading gives pointwise 95% intervals from 5,000 query-bootstrap resamples. Full and observed-seed curves coincide. Right: MMLU best-so-far utility over 16 rollouts on the fixed 1,300-query discovery split, with the same bootstrap convention.

MBPP. Using the same within-trajectory reference as MATH, the full-discovery and observed-seed curves both reach 0.7366 at rollout 16 on the 90-query discovery set. Generated structures therefore do not improve the final best-so-far utility beyond the observed seeds in this run. A separate 50-query screening experiment nevertheless shows that the candidate pool can be substantially compressed: the full 19-candidate pool solves 44 queries, and an eight-structure subset preserves all 44 solved queries together with the lowest-cost correct execution available for each query.

MMLU. On MMLU, full discovery reaches a mean best-so-far utility of 0.7658 at rollout 16, compared with 0.7393 for the observed-seed reference. Generated structures provide a strictly higher final best-so-far utility on 42 of 1,300 queries. The resulting 14-member bank is subsequently evaluated on a separate 1,433-query dense table used for router supervision and bank screening.

Table 14: Source-domain discovery and screening diagnostics. Statistics are computed on their stated offline query sets.

Source	Diagnostic	Frozen bank size
MATH	280 queries, 16 rollouts: 0.6650 seed / 0.6957 full utility	11
MMLU	1,300 queries, 16 rollouts: 0.7393 seed / 0.7658 full utility	14
MBPP	90 queries, 16 rollouts: 0.7366 seed / 0.7366 full utility	8

D.5.2 Initial Matching Across Evaluation Benchmarks

Static applies one development-selected topology for every query. Matcher w/o cost keeps the same frozen bank and matcher but sets $\lambda_1 = 0$. SOME-G_1 uses the task-specific deployed λ_1 and stops after initial execution. For a policy with mean composite cost $\overline{C}$, we report $C_{\mathrm{rel}} = 100\overline{C}/\overline{C}_{\mathrm{Static}}$, the corresponding cost saving $100 - C_{\mathrm{rel}}$, and the performance change relative to Static.

Transfer evaluations. Initial matching is also evaluated on GSM8K and HumanEval using the frozen source-task banks: GSM8K reuses the MATH bank, while HumanEval reuses the MBPP bank. No additional structural discovery is performed on either transfer benchmark.

Additional cost sensitivity. Figure 8 extends the comparison between the deployed and zero-cost settings to a broader frozen-matcher sweep. Increasing λ_1 progressively favors lower-cost structures without rerunning discovery or retraining the matcher. The sweep characterizes the resulting accuracy–cost sensitivity; the deployed coefficients remain unchanged.

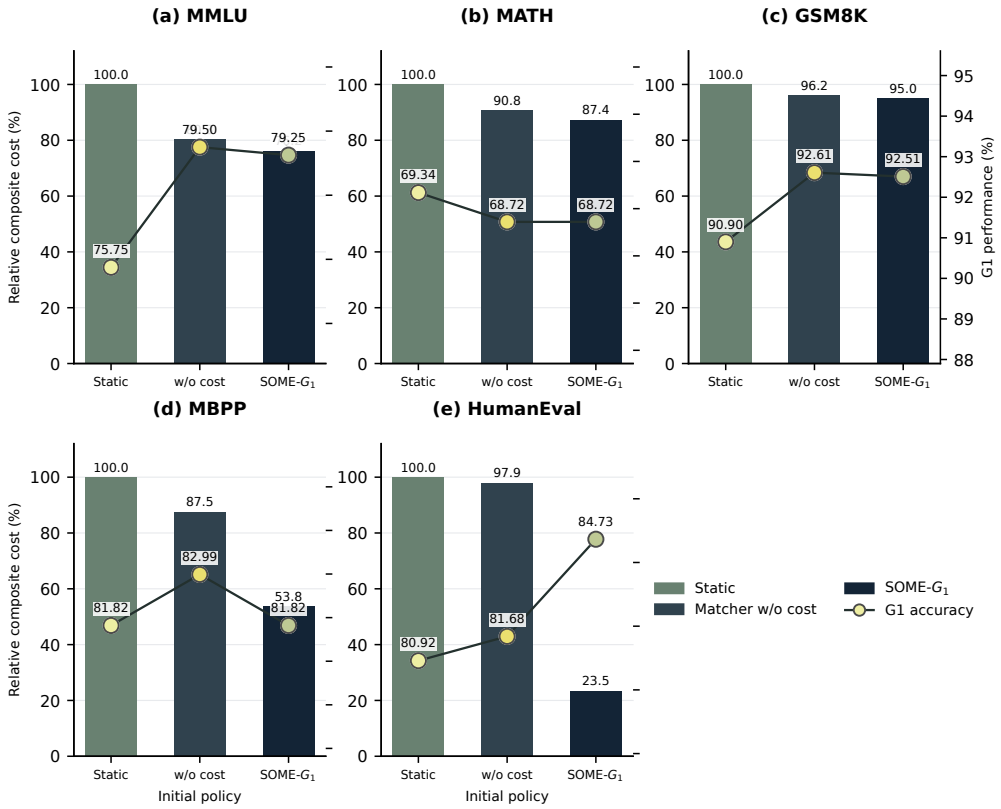


Figure 7: Complete initial-matching controls on all five evaluation benchmarks. Bars show composite cost relative to Static; dots show initial-execution performance.

Table 15: Complete five-benchmark initial-matching controls. Performance is accuracy for MMLU, MATH, and GSM8K, and pass@1 for MBPP and HumanEval. Accuracy change is in percentage points; normalized cost and saving are percentages.

Dataset	Policy	Perf.	$\overline{C}$	C_{rel}	Cost-Saving	ΔPerf.
MMLU	Static	75.75	1245.81	100.00	0.00	0.00
	Matcher w/o cost	79.50	1000.34	80.30	19.70	+3.75
	SOME-G_1	79.25	948.90	76.17	23.83	+3.50
MATH	Static	69.34	2275.51	100.00	0.00	0.00
	Matcher w/o cost	68.72	2065.45	90.77	9.23	-0.62
	SOME-G_1	68.72	1989.30	87.42	12.58	-0.62
MBPP	Static	81.82	3422.02	100.00	0.00	0.00
	Matcher w/o cost	82.99	2994.73	87.51	12.49	+1.17
	SOME-G_1	81.82	1840.17	53.77	46.23	0.00
GSM8K	Static	90.90	2104.98	100.00	0.00	0.00
	Matcher w/o cost	92.61	2024.28	96.17	3.83	+1.71
	SOME-G_1	92.51	1999.21	94.97	5.03	+1.61
HumanEval	Static	80.92	3491.01	100.00	0.00	0.00
	Matcher w/o cost	81.68	3417.45	97.89	2.11	+0.76
	SOME-G_1	84.73	819.37	23.47	76.53	+3.81

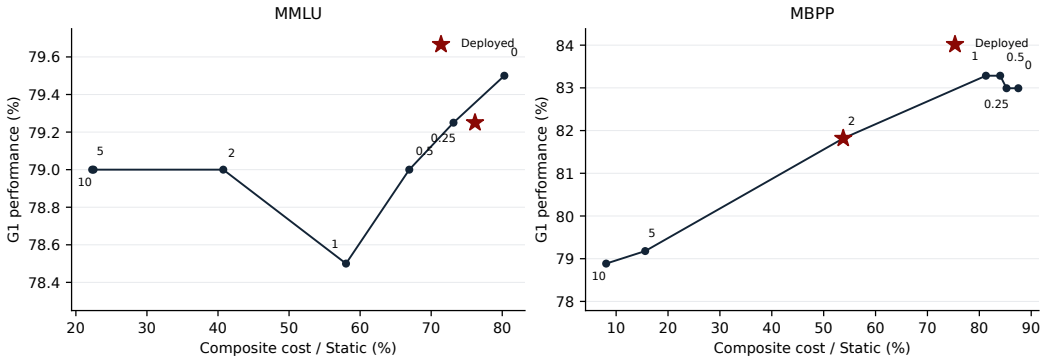


Figure 8: Initial-matching cost sensitivity on MMLU and MBPP. The sweep evaluates $\lambda_1 \in \{0, 0.25, 0.5, 1, 2, 5, 10\}$; stars denote the deployed coefficients (0.15 for MMLU and 2 for MBPP). All points are measured from saved runs.

D.5.3 Continuation on Evolution-Active Benchmarks

Under the deployed configuration, post-execution evolution is activated only on MMLU, MATH, and MBPP. For each benchmark, we hold fixed the deployed G_1 outputs, activation decisions, and selected G_2 structures, and vary only how the executed outputs are used. We compare retaining G_1, directly adopting G_2 on activated queries, and applying SOME's final resolution. This matched replay isolates the contribution of final-answer resolution from structural selection and activation.

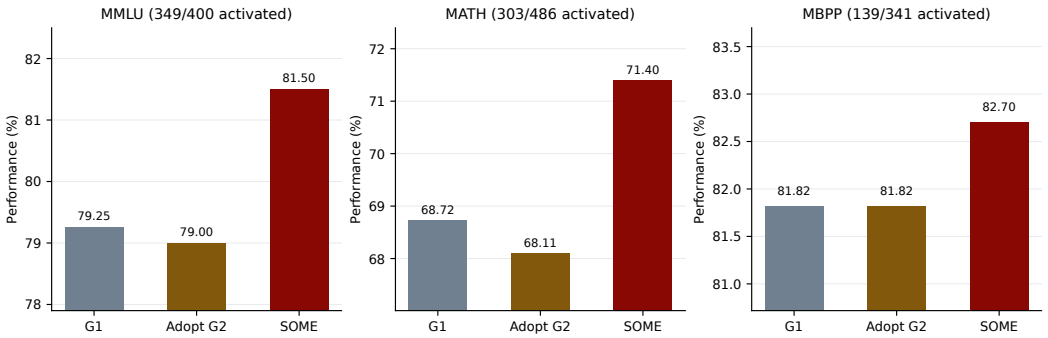


Figure 9: Matched continuation analysis on the three evolution-active benchmarks. Direct adoption uses the same activated queries and selected complementary outputs as SOME. Titles report the number of activated queries.

Table 16: Continuation outcomes on the three evolution-active benchmarks. Additional composite cost is averaged over all evaluation queries, with stopped queries contributing zero additional cost.

Dataset	G_1 (%)	Adopt G_2 (%)	SOME (%)	Gain (pp)	Rescue	Harm
MMLU	79.25	79.00	81.50	+2.25	11	2
MATH	68.72	68.11	71.40	+2.67	22	9
MBPP	81.82	81.82	82.70	+0.88	8	5

Dataset	Activated / total	Activation (%)	G_1 cost	Additional cost
MMLU	349/400	87.25	948.90	1037.56
MATH	303/486	62.35	1989.30	1354.79
MBPP	139/341	40.76	1840.17	1545.13

Because the activation decisions and selected G_2 outputs are held fixed, the difference between direct adoption and SOME isolates the effect of final resolution. Resolution adds 10, 16, and 3 net

correct answers on MMLU, MATH, and MBPP, respectively. Direct G_2 adoption alone is worse than retaining G_1 on MMLU and MATH and unchanged on MBPP, showing that complementary execution benefits from explicit final-answer resolution.

Table 17: Continuation-cost sensitivity on the 486-query MATH evaluation set. The always-continue control executes the complementary proposals selected under $\lambda_2 = 0$ for all queries, bypassing gain-based stopping. Changing λ_2 may also change which complementary structure is selected.

Policy	Activated	Accuracy (%)	Additional composite cost
SOME, $\lambda_2 = 0.005$	303	71.40	1354.79
SOME, $\lambda_2 = 0$	390	72.84	1935.50
Always continue, original proposals	486	74.07	2396.57

Continuation-cost sensitivity. Relative to $\lambda_2 = 0$, additional utility-guided continuation with $\lambda_2 = 0.005$ reduces additional composite cost by 30.00%; relative to always continuing, the reduction is 43.47%. These controls expose the accuracy–cost trade-off induced by the continuation decision around the deployed operating point.

D.6 Operating Points

Table 18: SOME operating points used in the main accuracy–cost analysis. Performance denotes accuracy for MMLU, MATH, and GSM8K, and pass@1 for MBPP and HumanEval. λ_1 and λ_2 are the deployed cost coefficients for initial matching and continuation, respectively. A dash indicates that post-execution evolution is not activated.

Dataset	λ_1	λ_2	G_1 / full performance (%)	G_1 / full tokens	Continuation
MMLU	0.15	0.00	79.25 / 81.50	1045.05 / 2094.29	Active
MATH	0.25	0.005	68.72 / 71.40	4700.88 / 8750.59	Active
MBPP	2	0.00	81.82 / 82.70	3887.56 / 8336.77	Active
HumanEval	2	–	84.73 / 84.73	1731.13 / 1731.13	Inactive
GSM8K	0.25	–	92.51 / 92.51	2323.68 / 2323.68	Inactive

E Essential Prompt Templates

E.1 Task-Family Role Instructions

SOME composes the task-family roles listed in Appendix C. Table 19 condenses their responsibilities; the task-profile registry contains the exact strings. Structural adaptation changes which roles communicate, not the task's reference answer or tests. Output contracts request a mathematical final answer for MATH, a numeric answer for GSM8K, one of A/B/C/D for MMLU, and the required Python program for MBPP/HumanEval. Programmer nodes return an executable `solve()` function whose textual result can be passed to other roles.

Table 19: Role instructions by task family (condensed).

Family	Role responsibilities
MATH / GSM8K	Direct solver solves the problem; structured solver decomposes it; verifier checks the proposed result; challenger examines assumptions; referee reconciles reasoning; programmer supplies a Python calculation.
MMLU	Direct solver selects an option; complementary knowledge and analytical roles provide task-specific reasoning; challenger checks alternatives; referee reconciles the final choice.
MBPP / HumanEval	Algorithm designer plans the solution; generator writes code; reviewer examines correctness; tester checks visible examples; refiner improves the candidate; debugger diagnoses concrete defects.

E.2 Communication and Resolution

Shared communication wrapper

```
Problem:
{problem}

Your previous output:
{current_output}

Other operator outputs:
[{neighbor_node_id}]
{compact_neighbor_output}
```

Empty context blocks are omitted. Neighbor messages always come from the preceding interaction round, matching Equation 10.

Mathematical resolution: condensed verifier template

```
Act as a skeptical mathematical verifier.

First read the ORIGINAL problem carefully.
Original problem: {question}

Candidate A answer: {proposal_1}
Candidate B answer: {proposal_2}

Neither candidate is preferred.
Audit Candidate A and Candidate B separately against the ORIGINAL
problem.
Recompute or verify the critical step independently.
If both candidates fail or remain unsupported, discard both and solve
the original problem independently.

End with exactly one line:
Final Answer: \boxed{answer}
```

Executed reasoning accompanies each candidate in the full template. The verifier checks the requested quantity, constraints, and decisive calculation. The format-v2 suffix requests concise verification and one complete final box. MMLU uses the corresponding option-verification contract. The Code resolver compares two existing programs against the explicit task and visible assertions, then copies the selected program verbatim. It does not synthesize a third program. These shared templates capture SOME's communication and resolution mechanisms without repeating every wrapper.

F CASE STUDIES

All cases below are drawn from the same 486-query MATH evaluation set under the deployed continuation configuration ($\lambda_2 = 0.005$) used in the main accuracy–cost and continuation analyses. Each decision is matched by query ID to its saved execution and resolution records. Short mathematical checks are provided to verify the reported outcomes; they are not intended as verbatim model reasoning.

F.1 FIVE-FOLD OOF CONSTRUCTION

Before final evaluation, SOME constructs execution-supervised continuation examples entirely from the training data as follows.

1. Partition the 1,138-query continuation-supervision set into five outer folds at the query level using seed 2027. Their sizes are 228, 228, 228, 227, and 227; all topology alternatives for a query remain in its fold.
2. For each outer fold, remove any overlapping queries from the 824-query matcher-training source. The remaining matcher source is then split into fitting and validation subsets, with 10% reserved for checkpoint selection. The five fitting/validation sizes are 593/66, 598/66, 589/65, 581/64, and 607/67.
3. Score each outer-fold query with its held-out matcher using frozen query and topology descriptors. Apply the same score normalization and $\lambda_1 = 0.25$ selection rule to obtain its OOF G_1. The 314 continuation-supervision queries that are absent from the matcher-training source are scored by their assigned fold model and never used for matcher fitting.
4. Pair each OOF-selected G_1 with each of the other ten members of the 11-structure bank. Dense execution outcomes supply (Z_1, Z_2) labels for 11,380 ordered pairs. These OOF-derived pairs provide supervision for the initial-success, rescue, and conditional-preservation predictors. Offline resolver executions estimate r_{01} and h_{10} with Beta$(1, 1)$ smoothing.
5. Refit the deployment continuation predictors on the resulting OOF supervision using the fixed training configuration, and apply them to the final evaluation queries. Final reference answers and unexecuted alternative outputs do not enter online selection features.

This aligns continuation supervision with G_1 choices produced by a matcher that was not trained on the scored query. The 486 final evaluation queries are never used to fit either the matcher or the continuation predictors.

F.2 MATH CONTINUATION CONFIGURATION

Initial matching uses $\lambda_1 = 0.25$. The continuation stage uses

$$\widetilde{C}_2(g) = S(g)/1000, \qquad \lambda_2 = 0.005, \quad \tau_{\text{risk}} = 0.30, \quad \tau_{\text{safe}} = 0.70, \quad \tau_{\text{gain}} = 0. \tag{37}$$

Its resolver-aware gain is Equation 25, with $r_{01} = 0.477733$ and $h_{10} = 0.230769$. An eligible three-node, two-edge, one-round structure has $S = 256 \cdot 6(1 + 0.15 \cdot 2) = 1996.8$, so its cost penalty is 0.009984. Continuation requires predicted gain minus this penalty to be nonnegative. The pre-selection proxy excludes resolver cost; reported evaluation cost includes resolution when invoked.

F.3 CORRECTING A COUNTING ERROR

Case 1: Complementary reasoning resolves an initial error

Problem. Bekah has four house-number digits, 2, 3, 5, and 7, one of each. How many distinct numbers can she form using one or more digits?

Initial structure. D–P–R, $K = 1$, returns 15.

Case 1: Complementary reasoning resolves an initial error (continued)

Utility-guided continuation. SOME selects D–C–R, $K = 1$. The predicted marginal gain is 0.016352; subtracting 0.009984 gives utility $0.006368 > 0$. The complementary structure returns 64, and final resolution retains 64.

Mathematical check. Order matters. Summing permutations over lengths one through four gives $4 + 4 \cdot 3 + 4 \cdot 3 \cdot 2 + 4! = \boxed{64}$. The initial answer 15 instead equals the number of nonempty subsets.

Mechanism. The positive continuation utility activates the complementary D–C–R structure, after which resolution retains the corrected answer.

Trace ID: `hendrycks_math_test_prealgebra_b4758b0db31f8960`.

F.4 Expanding a Discovered Singleton

Case 2: Low-cost initialization followed by useful collaboration

Problem. A positive integer a satisfies $\mathrm{lcm}(a + 1, a - 5) = 10508$. Find $a^2 - 4a + 1$.

Initial structure. The discovered singleton referee `a3f7b20f` returns 6.

Utility-guided continuation. SOME selects V–C–R, $K = 1$, with a predicted marginal gain of 0.015398 and utility 0.005414 after the 0.009984 penalty. It returns 21022; final resolution also returns 21022.

Mathematical check. Let $d = \gcd(a + 1, a - 5)$, so d divides 6 and $(a + 1)(a - 5) = 10508d$. With $t = a - 2$, $t^2 = 10508d + 9$. Among $d \in \{1, 2, 3, 6\}$, only $d = 2$ gives a square, $145^2 = 21025$. Hence $a^2 - 4a + 1 = t^2 - 3 = \boxed{21022}$.

Mechanism. SOME initially selects a discovered single-role structure, then activates complementary collaboration when its predicted marginal gain justifies the additional cost. Both structures belong to the frozen bank.

Trace ID: `hendrycks_math_test_number_theory_96b49a8d07419dee`.

F.5 Retaining a Sufficient Initial Answer

Case 3: Stopping avoids unnecessary additional execution

Problem. Express $1.\overline{234}$ as an improper fraction.

Initial structure. D–C–R, $K = 1$, returns $137/111$.

Utility-guided stopping. For the proposed V–C–R alternative, the predicted marginal gain is -0.002264. The cost-adjusted utility is -0.012248, below zero. SOME retains the first answer with no G_2 or resolver call.

Mathematical check. Writing $x = 1.\overline{234}$ gives $1000x - x = 1233$ and $x = 1233/999 = \boxed{137/111}$.

Mechanism. Candidate eligibility does not imply execution: SOME stops when the best available continuation falls below the continuation-utility threshold.

Trace ID: `hendrycks_math_test_prealgebra_de9f3f0ae637909d`.

Together, these examples illustrate initial matching and Marginal-Utility Structural Evolution through successful continuation, resolution, and stopping decisions under the same MATH evaluation configuration. They complement the full-set rescue/harm and cost measurements in Tables 16 and 17.