SReF: Structure–Resource Factorization for Multi-Agent LLM Orchestration

Anonymous authors
Paper under double-blind review

Abstract

Multi-agent systems built from heterogeneous large language models decide both what computation to perform and which model should perform it. Coupling these decisions can make changes in the available experts unnecessarily affect an otherwise useful planning policy. We introduce SReF, a Structure–Resource Factorization framework that separates a model-identity-agnostic Graph Planner from a descriptor-conditioned Expert Router. The Planner generates computation graphs and issues Patches, while the Router assigns available experts to reached nodes. To train these distinct decisions, the Planner receives feedback over verified segments along executed dependencies, and the Router learns from model-relative comparisons under matched node contexts. Segment-level verification supports local training feedback and history-preserving correction of pending computation without verifying every node independently. This separation supports resource-local adaptation through the Router while retaining the learned Planner and its ability to Patch pending computation in response to execution feedback. Across four benchmarks, SReF improves Pass@1 over the strongest routing or orchestration baselines by 1.8–4.5 percentage points and over its NoPatch variant by 1.6–4.9 percentage points. Planner-credit experiments compare terminal-only and segment-level Planner credit under the same verification and Patch protocol, and examine the additional training time of finer node-level feedback. Under expert-pool insertion and removal, direct transfer with the Planner retained approaches the performance obtained by Router-only and joint adaptation. Our code for SReF is released at `https://anonymous.4open.science/r/SReF-007E`.

1 Introduction

Heterogeneous large language models (LLMs) are increasingly orchestrated to solve complex tasks by combining models with different capabilities and costs, extending query-level routing and model cascades (Chen et al., 2024a; Ong et al., 2025; Chen et al., 2024b). Multi-agent systems also organize interacting subtasks, agent roles, and model allocation as execution unfolds (Wu et al., 2024; Zhuge et al., 2024; Yue et al., 2025; Feng et al., 2026; Cui et al., 2026). Consequently, an orchestrator make two distinct decisions: *what computation should be performed* and *which model should perform it*. This raises a basic question: should computation structure and model-resource allocation be learned and adapted as the same decision process?

Adding, removing, upgrading, or repricing models can challenge a learned allocation policy, motivating routing methods that generalize to new candidates (Feng et al., 2025; Zhang et al., 2025a). The left side of Figure 1 shows how failures may reflect flawed subtasks or poor model choices, while pool changes can perturb the joint policy that generates the graph (Feng et al., 2026; Cui et al., 2026). Updating the entire orchestrator can then alter useful subtasks and dependencies even when the underlying computation remains appropriate. We study whether resource-layer adaptation can retain the learned graph-planning policy.

This separation also motivates decision-specific feedback. Structural choices change subtasks and dependencies; routing choices select experts for specified subtasks. Final-task reward alone does not directly identify which part of the computation failed. Intermediate verification provides more local feedback (Lightman et al., 2024), but evaluating every node independently increases training cost. We therefore reuse execution-time segment scores to guide structural learning and trigger a

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

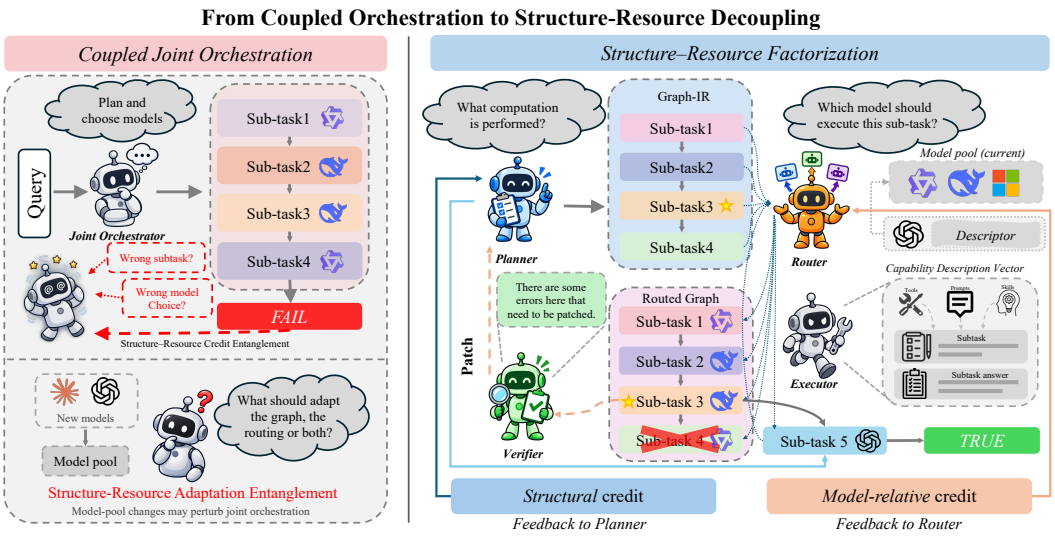


Figure 1: From coupled orchestration to structure–resource factorization. The left side illustrates how coupled orchestration entangles subtask errors, model choices, and adaptation to pool changes. The right side separates a model-agnostic Graph Planner from a descriptor-conditioned Expert Router, with verification-driven Patch and decision-specific feedback to the two policies.

Patch. Expert selection instead benefits from comparing candidates under matched node contexts, consistent with work on query–model matching and routing reward calibration (Chen et al., 2024b; Yu et al., 2026). Together, these signals support learning and correction at the structure and resource levels.

To this end, we introduce SReF, a Structure–Resource Factorization framework for heterogeneous LLM orchestration. The right side of Figure 1 shows its division of labor: a model-identity-agnostic Graph Planner generates a Graph-IR specifying subtasks, roles, verification boundaries, and dependencies; a descriptor-conditioned Expert Router assigns an available model to each reached node. When a frozen verifier detects a failure, the same Planner Patches pending computation while preserving executed history.

To learn the separated policies, the Planner receives dependency-aware segment credit over the actually executed graph, including Patches and retries. Credit is assigned to decisions that generated each executed node version, aligning Patch decisions with structural learning. The Router receives model-relative credit from candidate experts compared under matched node contexts, and training alternates between the two policies. When the expert pool changes, the Router can be adapted while the Planner remains fixed and continues to Patch pending computation in response to feedback.

Across four benchmarks, SReF improves Pass@1 by 1.8–4.5 points over the strongest comparable routing or orchestration baseline. Our contributions are threefold. First, we factor graph planning and expert selection so planned computation contains no model identities and expert choice is conditioned on candidate descriptors. Second, we couple verification-triggered Patch with dependency-aware segment credit for the Planner, and train the Router through matched-context expert comparisons. Third, we compare terminal-only and segment-level Planner credit under the same verification and Patch protocol, assess added node-level supervision, and evaluate adaptation under expert-pool insertion and removal. Under both pool changes, direct transfer with the Planner retained stays close to the results after Router-only and joint adaptation.

2 Related Work

2.1 Heterogeneous LLM routing.

FrugalGPT, RouteLLM, and RouterDC route queries or construct cascades to balance answer quality and cost (Chen et al., 2024a; Ong et al., 2025; Chen et al., 2024b). GraphRouter models query–LLM

relations to support newly introduced candidates, and Router-R1 makes multi-round routing decisions using model descriptors (Feng et al., 2025; Zhang et al., 2025a). xRouter trains a cost-aware router that may answer directly or call external models (Qian et al., 2025). SReF studies allocation within an explicitly learned, dependency-bearing graph and tests pool changes while retaining that graph's planning policy.

2.2 Learned workflows and heterogeneous orchestration.

AutoGen provides programmable multi-agent conversation patterns, while GPTSwarm, AFlow, and automated agent design optimize graphs or executable workflows (Wu et al., 2024; Zhuge et al., 2024; Zhang et al., 2025b; Hu et al., 2025). Puppeteer uses a reinforcement-learned orchestrator to sequence and prioritize agents as execution unfolds (Dang et al., 2025). Heterogeneous systems also make resource decisions: MasRouter chooses collaboration modes, roles, and LLMs; GraphPlanner selects a role and model backbone at each workflow step; and Uno-Orchestra combines selective decomposition with model–primitive delegation (Yue et al., 2025; Feng et al., 2026; Cui et al., 2026). R2-Reasoner is particularly close: it separates task decomposition from subtask allocation, alternates their training, and evaluates unseen-model transfer; its RL updates use final-answer correctness for both modules (Shao et al., 2025). SReF represents planned graph structure without model identities and assigns structural and model-relative feedback to the respective policies, allowing a direct test of whether the structural policy can remain fixed when the pool changes.

2.3 Credit assignment in agentic RL.

Outcome-level verifiers establish final success, while process supervision targets intermediate reasoning steps (Cobbe et al., 2021; Lightman et al., 2024). More recently, HiPER assigns hierarchical advantages to planning and execution decisions, while Uno-Orchestra uses process-aware intermediate feedback (Peng et al., 2026; Cui et al., 2026). IAPO propagates trajectory-level advantage through an influence-dependency graph; ReCal calibrates routing rewards with component-wise advantages; and MoRSE separates credit for routing from credit for its trainable experts (Ren et al., 2026; Yu et al., 2026; Li et al., 2026). SReF targets a different distinction: structural credit follows the dependencies of the *executed* graph, whereas model-relative credit compares frozen candidate experts under matched node contexts. Atomic Task Graph also localizes failures and preserves validated work during repair (Zhang et al., 2026). SReF couples history-preserving Patch to credit attribution so revised pending nodes receive feedback without changing the provenance of completed work.

3 Problem Formulation

Let $x \sim \mathcal{D}$ be a task and $\mathcal{M} = \{m_1, \ldots, m_K\}$ a pool of frozen heterogeneous LLM experts. Orchestration involves two decisions: *computation structure* and *model-resource allocation*. The Planner generates an initial directed acyclic graph $G = (V, E)$ and, upon verification failure, Patches pending computation using the task, current graph, and failure feedback while preserving executed records. Nodes specify subtasks, roles, dependencies, and verification indicators, without model identities. The Router selects experts at reached nodes using node representations h_i and model descriptors $\Phi_{\mathcal{M}} = \{\phi(m) : m \in \mathcal{M}\}$. We define the initial nominal planning-and-allocation policy as

$$p_\theta\left(G, \mathbf{m} \mid x, \mathcal{M}\right) = \pi_P\left(G \mid x; \theta_P\right) \prod_{i \in V(G)} \pi_R\left(m_i \mid h_i, \Phi_{\mathcal{M}}; \theta_R\right), \tag{1}$$

where $\theta = (\theta_P, \theta_R)$ and $\mathbf{m} = (m_i)_{i \in V(G)}$ denotes a nominal assignment. For a fixed initial graph, routing draws are independent but sampled only as nodes are reached. The same Planner later issues Patches conditioned on the current graph and verification feedback.

A non-learned Executor realizes these decisions as trajectory ξ, including planning, expert calls, verification, and Patches. The system-level task-quality objective is

$$J_{\text{task}}(\theta_P, \theta_R) = \mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{\xi \sim P_\theta(\cdot \mid x, \mathcal{M})}\left[R_{\text{task}}(\xi)\right], \tag{2}$$

where P_θ is the executed-trajectory distribution and $R_{\text{task}} \in [0, 1]$ measures final task quality. We seek high task quality with efficient execution, using separate structure- and resource-specific training objectives in Section 4.

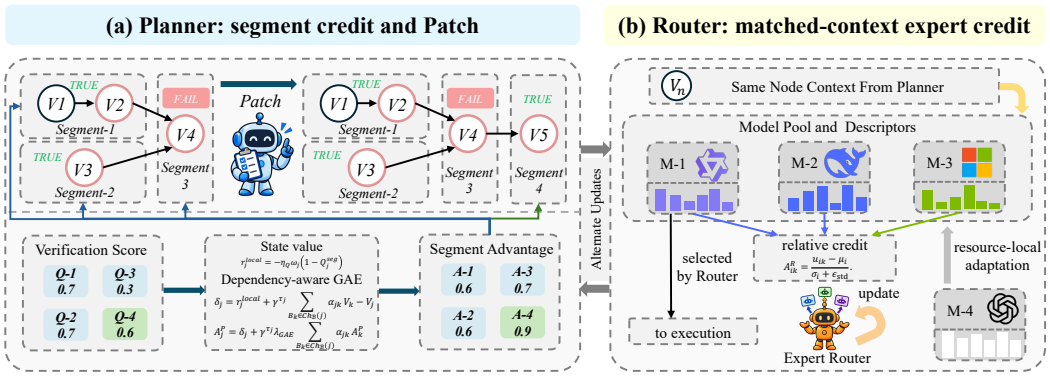


Figure 2: Decision-specific learning in SReF. (a) Verification boundaries partition execution into segments; after a failed boundary, the Planner Patches pending computation while committed node versions remain unchanged, and dependency-aware credit returns to the rollout that created each version. (b) At each reached node, frozen experts are compared under the same context; only the selected output is executed, while model-relative credit updates the Router. When a new expert is added, its calibrated descriptor is appended to the candidate set.

Appendix A formalizes allocation, execution with Patch, and policy-specific training objectives.

4 SREF

SReF instantiates Eq. 1 with a Graph Planner, Expert Router, and non-learned Task Executor. The Planner generates initial computation graphs and Patches pending computation; the Router selects available experts for reached nodes. Execution feedback supports Patch and decision-specific learning. Variant settings appear in Section 5.4. Figure 1 shows the system; Figure 2 shows the two training paths from executed computation to the policies.

4.1 STRUCTURE–RESOURCE FACTORIZED ORCHESTRATION

Model-identity-agnostic Graph-IR. For initial planning and subsequent Patches, the same Planner generates a complete Graph-IR in one autoregressive rollout. Each node specifies a subtask, an agent role, dependencies, and a critical verification indicator, without a concrete model identity. Critical indicators mark subtask completion, where accumulated outputs can be assessed together. Critical nodes and the unique final-output node define verification boundaries; the final output is always verified. A deterministic compiler partitions the accepted graph into

$$G \longrightarrow \mathcal{B} = \{B_1, \ldots, B_J\}, \qquad \mathcal{G}_{\text{seg}} = (\mathcal{B}, E_{\mathcal{B}}), \tag{3}$$

where $E_{\mathcal{B}}$ preserves cross-segment dependencies. Each segment is an execution and verification unit, and subsequently a structural credit unit as described in Section 4.2. The Executor commits segments in deterministic topological order, allowing parallel execution of ready nodes within a segment. Successful verification advances execution without another Planner call. Graph validity, serialization, and compilation rules appear in Appendix B.

Descriptor-conditioned Expert Router. For each reached node v_i, the Router scores available experts by combining the node representation h_i with each model descriptor:

$$\begin{aligned} h_i &= f_q(q_i) \oplus e_a(a_i) \oplus g_i, \\ z_{ik} &= f_{\theta_R}\left([h_i; \phi(m_k)]\right), \\ \pi_R\left(m_k \mid h_i, \Phi_{\mathcal{M}}\right) &= \frac{\exp(z_{ik})}{\sum_{m_u \in \mathcal{M}} \exp(z_{iu})}. \end{aligned} \tag{4}$$

Here, f_q encodes the subtask, e_a embeds the role, and g_i contains node depth, parent count, and the critical indicator. Model descriptors encode scale, cost, and calibrated capabilities, without learned

model-identity embeddings. The shared scorer therefore accepts a changing candidate set. Realized dependency outputs are excluded from h_i; the Executor supplies them to the selected expert after routing. Appendix B.5 specifies these information boundaries.

Verification-triggered Planner Patch. A frozen verifier assigns each executed segment a local score $Q_j^{\text{seg}} \in [0,1]$. A score below the calibrated threshold triggers a Patch by the same Planner when Patch is enabled and its budget remains available, conditioned on the current graph and failure feedback. The Planner generates a complete patched Graph-IR that preserves executed history, including failed attempts; only the pending computation is recompiled and executed. Retries create new node instances. This history-preserving Patch also retains the executed dependencies and node versions needed for structural credit assignment. Appendix B.3 specifies validation and history preservation.

4.2 Decision-Specific Credit Assignment

The Planner receives dependency-aware segment credit over the actually executed graph, including Patched computation and retries. The Router receives model-relative credit from expert comparisons under matched node contexts.

Segment-level structural reward. The score used to verify an executed segment and decide whether to Patch also enters Planner training; no separate segment-scoring call is needed. Let $\mathcal{G}_{\text{seg}}^{\text{exec}} = (\mathcal{B}^{\text{exec}}, E_{\mathcal{B}}^{\text{exec}})$ denote the segment DAG induced by the computation that was actually executed. For every executed segment B_j, the verifier produces $Q_j^{\text{seg}} \in [0,1]$. We weight local defects by each segment's share of executed nodes:

$$N_{\text{exec}} = \sum_{B_k \in \mathcal{B}^{\text{exec}}} |B_k|, \qquad \omega_j = \frac{|B_j|}{N_{\text{exec}}}, \qquad r_j^{\text{local}} = -\eta_Q \omega_j \left(1 - Q_j^{\text{seg}}\right). \tag{5}$$

Since $\sum_j \omega_j = 1$, the total local penalty lies in $[-\eta_Q, 0]$.

Let B_J denote the executed segment containing the output-node instance whose answer is ultimately returned. Earlier failed output attempts remain historical segments; the executed graph need not have a unique sink. We inject a terminal structural signal only at B_J:

$$T = R_{\text{task}} - \lambda_D D_{\max}\left(G^{\text{exec}}\right) - \lambda_W W_{\max}\left(G^{\text{exec}}\right), \tag{6}$$

where $D_{\max}$ and $W_{\max}$ denote the maximum dependency depth and width of the actually executed graph. These terms regularize structure without directly charging the Planner for the Router's chosen model. Appendix C.1 defines the structural measures.

Dependency-aware structural credit. Credit propagates along the executed segment DAG. Let $V_j = V_{\psi_{\text{old}}}(S_j^P)$ estimate downstream structural return from the pre-generation segment context, and let τ_j denote execution duration. Critic inputs and targets are specified in Appendix C.2. For each edge $B_j \to B_k$, $\alpha_{jk} = 1/|\operatorname{Pa}_{\mathcal{B}}(k)|$ distributes a child's credit among its parents. We define the dependency residual as

$$\delta_j = \begin{cases} r_j^{\text{local}} + \gamma^{\tau_j} \displaystyle\sum_{B_k \in \operatorname{Ch}_{\mathcal{B}}(j)} \alpha_{jk} V_k - V_j, & B_j \neq B_J, \\ r_J^{\text{local}} + \gamma^{\tau_J} T - V_J, & B_j = B_J. \end{cases} \tag{7}$$

We then form a dependency trace in reverse topological order:

$$A_j^P = \begin{cases} \delta_j + \gamma^{\tau_j} \lambda_{\text{GAE}} \displaystyle\sum_{B_k \in \operatorname{Ch}_{\mathcal{B}}(j)} \alpha_{jk} A_k^P, & B_j \neq B_J, \\ \delta_J, & B_j = B_J. \end{cases} \tag{8}$$

These recursions assign credit through executed dependencies, with parent normalization distributing a child's signal at joins. Appendix C.3 details the dependency trace.

Provenance-aware Planner credit. The resulting segment advantage is divided equally among its executed nodes:

$$A_v^P = \frac{A_j^P}{|B_j|}, \qquad v \in B_j. \tag{9}$$

Each node's share is applied to the Planner tokens that last created or modified its executed version. Thus, credit after Planner Patch returns to the relevant planning decisions, even when a segment combines unchanged and patched nodes. Unchanged copies in later rollouts are masked, and unexecuted versions receive no credit. Appendix C.4 gives the provenance mapping and clipped policy objective.

Matched-context model-relative credit. During Router training, all candidate experts receive the same node specification, realized dependency outputs, prompt template, and decoding budget. Their utilities are $u_{ik} = Q_{ik} - \lambda_C C_{ik}$, combining node quality and realized invocation cost. Using the candidate group's mean μ_i and standard deviation σ_i, we define

$$A_{ik}^R = \frac{u_{ik} - \mu_i}{\sigma_i + \epsilon_{\text{std}}}. \tag{10}$$

This comparison measures relative local quality and cost under shared execution inputs.

Only the model selected by the behavior Router is committed to the live execution trajectory; the remaining candidate outputs are used solely to construct the matched-context training signal. The Router is optimized using a clipped policy objective over these candidate groups, while inference executes only one selected expert per reached node. Appendix C.5 gives the group statistics, low-variance handling, and probability-weighted clipped objective.

4.3 Resource-Local Adaptation

The Planner is warm-started to generate valid Graph-IRs and issue Patches. The Router starts with a uniform policy and is bootstrapped on node contexts from high-quality teacher trajectories using matched-context expert comparisons. Training then alternates: the Router is frozen while updating the Planner and its critic; the Planner and critic are frozen while updating the Router. Fresh trajectories are collected after each policy switch, keeping the counterpart policy fixed within each phase.

When the expert pool changes, resource-local adaptation updates the candidate set and appends calibrated descriptors for inserted experts; when needed, it also updates Router parameters while retaining the Planner:

$$\mathcal{M} \to \mathcal{M}', \qquad \Phi_{\mathcal{M}} \to \Phi_{\mathcal{M}'}, \qquad \theta_R \to \theta'_R, \qquad \theta_P \text{ fixed}. \tag{11}$$

Direct transfer uses the updated pool without parameter updates ($\theta'_R = \theta_R$); further Router adaptation uses fresh trajectories from that pool.

The fixed Planner can still Patch pending computation as changed expert outputs alter verification feedback; its conditional policy stays unchanged for the same context even if realized graphs differ. Appendix C.6 details training and adaptation; Section 5.5 compares Router-only and joint adaptation.

5 Experiments

We evaluate end-to-end performance against single-model, routing, and orchestration baselines on four benchmarks; track structure during training; compare verification and Planner credit under a fixed Router protocol; and test whether the Router can absorb expert-pool changes while retaining the Planner.

5.1 Experimental Setup

The Planner SFT warm start uses fixed subsets of GSM8K, MBPP, MMLU, and MATH (Cobbe et al., 2021; Austin et al., 2021; Hendrycks et al., 2021a;b); subsequent RL uses the complete

Table 1: Main results on four evaluation benchmarks. P@1 and P@2 denote system-level Pass@1 and Pass@2, respectively. Values are percentages; higher is better. SReF denotes the full method. Bold denotes the highest displayed value (including ties), and underlining denotes the runner-up among the displayed rows when the highest value is unique. Single-model references are excluded from this ranking, except for DeepSeek V4.0 Pro on MMLU-Pro P@1. Cost accounting is described in Appendix G.3.

METHOD	GSM-HARD		HUMANEVAL		MMLU-PRO		MATH-500	
	P@1	P@2	P@1	P@2	P@1	P@2	P@1	P@2
Phi-4	55.5	58.4	86.0	87.8	38.1	39.0	79.4	81.6
Qwen3.5-27B	64.1	67.7	90.2	95.1	63.3	66.0	87.2	89.8
DeepSeek V4.0 Pro	67.2	68.1	94.5	97.0	**74.1**	76.3	92.8	94.2
RouterDC	63.8	67.2	90.9	95.7	62.5	65.2	87.0	89.6
GraphRouter	62.4	66.0	89.6	94.5	61.6	64.3	85.6	88.2
Router-R1	60.4	64.2	86.0	91.5	59.2	61.8	84.2	86.6
xRouter	64.2	68.2	90.2	95.1	62.4	65.3	87.4	90.2
R2-Reasoner	63.6	67.8	89.6	94.5	62.0	65.0	86.8	90.2
MasRouter	66.4	70.8	92.7	96.3	65.2	69.0	89.4	93.6
GraphPlanner	67.2	71.8	93.3	95.7	65.8	69.8	89.2	93.8
Puppeteer	69.0	73.6	91.5	96.3	69.2	73.3	89.6	93.6
Uno-Orchestra	62.9	67.0	88.4	94.5	61.5	64.5	85.8	89.0
SReF-Base	42.3	44.5	81.7	89.0	65.1	69.9	80.8	85.4
SReF-SFT	69.2	73.2	86.6	94.5	71.8	75.1	88.6	93.2
SReF-NoPatch	69.7	74.0	90.2	96.3	72.1	75.3	90.2	93.8
SReF	**73.3**	**75.7**	**95.1**	**97.0**	73.7	**76.9**	**93.4**	**97.0**

GSM8K, MBPP, and MATH training splits plus 10,000 selected MMLU questions. Their respective evaluation benchmarks are GSM-Hard, HumanEval, MMLU-Pro, and MATH-500 (Gao et al., 2023; Chen et al., 2021; Wang et al., 2024; Lightman et al., 2024), spanning mathematical reasoning, code generation, and knowledge-intensive reasoning. We report system-level Pass@k for $k \in \{1, 2\}$: the probability that at least one of k independent complete executions succeeds. Verification and Planner Patch remain within an execution, not separate samples. Success requires a correct final number on GSM-Hard, passing hidden tests on HumanEval, the correct option on MMLU-Pro, and a reference-equivalent final answer on MATH-500. Appendices D.1 and D.2 detail data sources, the estimator, and scoring criteria.

Unless otherwise specified, the Graph Planner starts from Qwen3.5-4B (Qwen Team, 2026), as an instance distinct from the frozen Qwen3.5-4B expert. The expert pool comprises Phi-4-mini-instruct (Microsoft, 2025), Qwen3.5-4B, Qwen3.5-9B, Phi-4 (Abdin et al., 2024), and Qwen3.5-27B; all experts remain frozen during SReF training. Baselines cover single-model direct inference with Phi-4, Qwen3.5-27B, and DeepSeek V4.0 Pro (DeepSeek-AI, 2026a); heterogeneous routing with RouterDC, GraphRouter, Router-R1, and xRouter (Chen et al., 2024b; Feng et al., 2025; Zhang et al., 2025a; Qian et al., 2025); and learned agentic orchestration with R2-Reasoner, MasRouter, GraphPlanner, Puppeteer, and Uno-Orchestra (Shao et al., 2025; Yue et al., 2025; Feng et al., 2026; Dang et al., 2025; Cui et al., 2026). We adapt routing and orchestration baselines to the SReF Planner backbone as far as possible, using the shared expert pool and deployment protocol. Appendix D.3 details roles and implementation.

SREF-BASE uses the Planner before supervised warm start and bootstraps its Router from uniform using matched-context expert comparisons on node contexts sampled from high-quality teacher trajectories. SREF-SFT adds Planner SFT with the same Router bootstrap; both retain inference-time Planner Patch. SREF-NOPATCH and the full SREF variant share the same checkpoint after decision-specific policy training and differ only in inference-time verification-triggered Planner Patch.

5.2 OVERALL PERFORMANCE

Table 1 reports the main results on all four benchmarks using system-level Pass@1 and Pass@2.

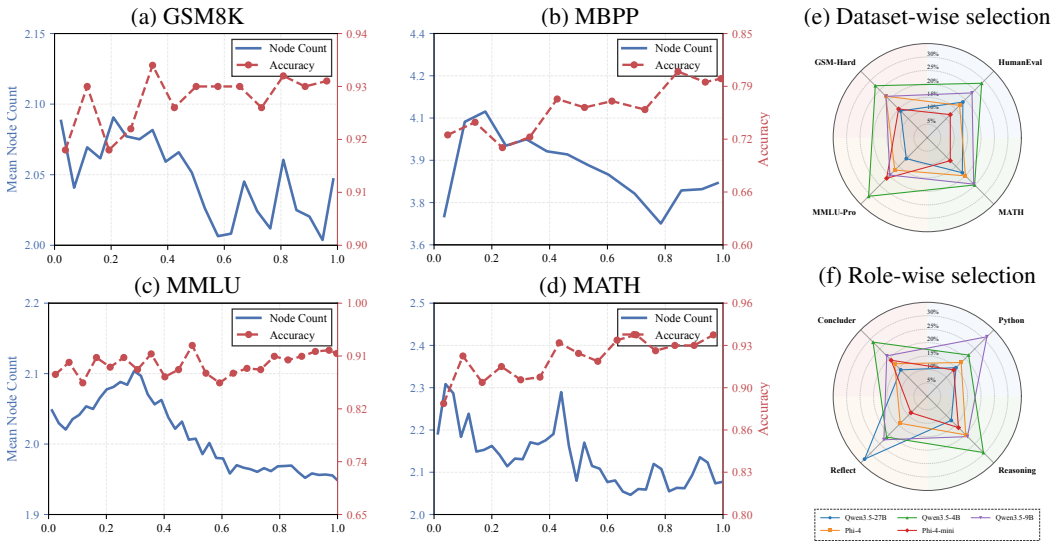


Figure 3: **Training dynamics and learned routing behavior of SReF.** (a)–(d) Mean node count excluding retries (left axis) and average accuracy (right axis) during training on GSM8K, MBPP, MMLU, and MATH, respectively. (e) Router selection probabilities averaged over reached nodes within each evaluation dataset. (f) Role-wise average Router selection probabilities.

Against external methods, SReF matches or exceeds the strongest comparable routing and agentic orchestration baseline on both metrics across the four benchmarks. Its Pass@1 gains over the strongest comparable baseline are 4.3, 1.8, 4.5, and 3.8 percentage points on GSM-Hard, HumanEval, MMLU-Pro, and MATH-500, respectively; the corresponding Pass@2 gains are 2.1, 0.7, 3.6, and 3.2 points. SReF also exceeds Qwen3.5-27B on both metrics throughout. Compared with DeepSeek V4.0 Pro, the external single-model reference, SReF improves Pass@1 by 6.1 points on GSM-Hard, 0.6 on HumanEval, and 0.6 on MATH-500, while trailing by 0.4 points on MMLU-Pro. These results show that the smaller expert pool is competitive with the large-model direct-inference reference.

With Planner Patch enabled, supervised warm-start training improves every reported metric over SReF-Base, and subsequent decision-specific policy training in the full SReF variant further improves Pass@1 over SReF-SFT by 4.1, 8.5, 1.9, and 4.8 points on GSM-Hard, HumanEval, MMLU-Pro, and MATH-500, respectively, with gains in Pass@2 throughout. To isolate the contribution of inference-time Planner Patch, SReF and SReF-NoPatch use the same trained checkpoint and differ only in whether verification-triggered Patch is enabled. Planner Patch improves Pass@1 by 3.6, 4.9, 1.6, and 3.2 points across the four benchmarks, respectively, and improves Pass@2 throughout. SReF-NoPatch already reaches 69.7, 90.2, 72.1, and 90.2 Pass@1 on the four benchmarks, respectively, exceeding the strongest external routing or orchestration baseline on GSM-Hard, MMLU-Pro, and MATH-500. This shows that the factorized Planner–Router training is effective even before inference-time repair is enabled. SReF's verifier and Patch overhead is reported in Table 16; Planner-credit comparisons are presented in Section 5.4.

5.3 Training Dynamics and Routing Preferences

Figure 3(a)–(d) shows that mean executed node counts (excluding retries) generally fall as accuracy rises and stabilizes during training on GSM8K, MBPP, MMLU, and MATH. Figure 3 (e) and (f) show mean Router selection probabilities by evaluation dataset and agent role, respectively.

5.4 Planner Credit Granularity and Training Cost

Table 2 compares variants with the same expert pool, initialization, training data, update schedule, and Router objective. All three variants verify segments and can trigger Planner Patch under the same execution protocol. Global's Planner credit uses only the terminal task, depth, and

width terms; SEGMENT-LEVEL also uses segment-verifier scores for Planner credit; NODE-LEVEL replaces segment credit with node credit while retaining segment verification for Patch. Router candidate evaluation remains active in all three variants; Appendix G.1 specifies the controls.

Segment-level improves Pass@1 over Global from 92.0 to 93.4 on MATH-500 and from 88.4 to 95.1 on HumanEval at 1.29× the training time. Both variants use verification-triggered Patch, so the gain reflects the Planner-credit comparison. Node-level reaches 93.8 and 95.7 at 1.49× the training time, gaining another 0.4 and 0.6 points with extra node evaluations. Segment-level reuses execution-time verifier scores for credit and Patch. Cost excludes shared SFT and Router bootstrap; Appendix G.1 gives the full protocol.

Table 2: Planner credit granularity: Pass@1 (%) and relative alternating-training time.

Variant	MATH-500	HumanEval	Cost
Global	92.0	88.4	1.00×
Segment-level	93.4	95.1	1.29×
Node-level	93.8	95.7	1.49×

5.5 CAN MODEL-POOL CHANGES BE LOCALIZED TO THE RESOURCE POLICY?

We evaluate expert-pool changes on MATH-500. INSERT adds DeepSeek V4.1 Flash as a routable expert absent from the original pool (DeepSeek-AI, 2026b), and REMOVE excludes Qwen3.5-9B. Fixed-ID Router shares SReF's Planner but uses learned model-ID representations for routing; Planner-Integrated is trained separately to generate nodes and expert identities jointly. Table 3 reports performance on the original pool (Orig.), direct transfer to the changed pool without policy updates (Pre), Router-only adaptation (Post-R), and joint adaptation (Post-All). Planner-Integrated has no separate Router, so Post-R does not apply. All adaptation runs use matched task counts; MATH-500 is used only for evaluation. We average the Insert and Remove results over multiple runs to reduce sensitivity to run-to-run variation; Orig. remains a single reference run. Appendix G.2 gives the protocol.

Table 3: Expert-pool adaptation on MATH-500: Pass@1 (%). Orig. is the single original-pool reference run; all reported Insert and Remove values are means over multiple runs. Pre: no parameter updates; Post-R: Router-only updates; Post-All: all applicable policy updates. N/A denotes an inapplicable setting.

Method	Orig.	Insert			Remove		
		Pre	Post-R	Post-All	Pre	Post-R	Post-All
SReF	93.4	94.1	93.8	94.4	93.4	93.0	93.3
Fixed-ID Router	93.4	93.3	94.0	94.2	90.6	92.4	93.4
Planner-Integrated	93.0	93.7	N/A	94.0	92.0	N/A	92.5

On the original pool, Planner-Integrated performs slightly below SReF (93.0% versus 93.4%), while Fixed-ID Router matches SReF. Fixed-ID Router shows the weakest direct transfer under both Insert and Remove. SReF's Pre results are within 0.4 percentage points of its Post-R and Post-All results; Post-R is in turn within 0.6 points of Post-All, so updating the Planner adds little in these settings. Both controls recover strong performance after adaptation, whereas SReF reaches a comparable level before policy updates.

6 CONCLUSION

We introduced SReF, which separates model-identity-agnostic graph planning and verification-triggered Patch from descriptor-conditioned expert routing, training the Planner with dependency-aware segment credit and the Router with matched-context expert comparisons. Across four benchmarks, SReF improves Pass@1 by 1.8–4.5 percentage points over the strongest routing or orchestration baselines. Segment-level Planner credit with verification-triggered Patch improves accuracy over terminal-only Planner credit under the same verification and Patch protocol, at higher training time. Node-level supervision offers smaller further gains with additional evaluations. Under expert-pool insertion and removal, direct transfer with the Planner retained remains close to Router-only and joint adaptation. These results suggest that computation structure and model allocation need not be learned and adapted as a single policy; separating them can help retain performance under the pool changes studied here. The fact that SReF-NoPatch already exceeds the strongest routing or orchestration baseline on three benchmarks further supports the effectiveness of the factorized training design. Future work will study longer workflows and settings with changing agent sets.

AI USE STATEMENT

We used generative AI tools to identify unclear wording and grammatical errors and to compress parts of the manuscript to meet the page limit. The authors developed the research direction, problem formulation, and central ideas. AI also assisted with writing and running experimental code. The authors checked all code and reviewed the resulting outputs and claims. AI suggested a small number of additional references; the authors read the original sources and verified that each supports the specific claim for which it is cited. The authors take responsibility for the final paper, including its text, equations, code, and reported results.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or the collection of new human data. We use publicly released benchmark datasets and frozen language models for research evaluation, following their stated usage conditions. The study does not target a high-stakes deployment or introduce a new sensitive dataset.

REPRODUCIBILITY STATEMENT

We release the SReF code and evaluation materials at `https://anonymous.4open.science/r/SReF-007E`. The main text specifies the factorized Planner–Router design, credit signals, Patch mechanism, and adaptation setting. The appendices provide dataset sources and splits, the model pool, prompts and output formats, initialization and training settings, reward coefficients, execution and scoring protocols, verifier/Patch cost accounting, and pool-shift procedures. Together with the released code, these details support reproduction of the training and evaluation pipeline.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, et al. Phi-4 technical report, 2024. URL https://arxiv.org/abs/2412.08905.

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

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=cSimKw5p6R.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James T. Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. doi: 10.52202/079017-2120. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7a641b8ec86162fc875fb9f6456a542f-Abstract-Conference.html.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Zhiqing Cui, Haotong Xie, Jiahao Yuan, Cheng Yang, Hanqing Wang, Yuxin Wu, Yifan Wu, Siru Zhong, Tao Yu, Yifu Guo, Siyu Zhang, Xinlei Yu, Qibing Ren, and Usman Naseem. Uno-orchestra: Parsimonious agent routing via selective delegation, 2026. URL https://arxiv.org/abs/2605.05007.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5502. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026a. URL https://arxiv.org/abs/2606.19348.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026b. URL https://arxiv.org/abs/2609.19969.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/41b6674c28a9b93ec8d22a53ca25bc3b-Abstract-Conference.html.

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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

Peiwen Li, Shiyang Zhang, Yangtian Zhang, Sizhuang He, David van Dijk, and Rex Ying. Morse: Task-oriented multi-agent system with mixture of role-subtask experts, 2026. URL https://arxiv.org/abs/2608.09251.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

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.

Microsoft. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras, 2025. URL https://arxiv.org/abs/2503.01743.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. Hiper: Hierarchical plan–execute rl for multi-turn llm agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=RJWQcgd3hX.

Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. xrouter: Training cost-aware llms orchestration system via reinforcement learning, 2025. URL https://arxiv.org/abs/2510.08439.

Qwen Team. Qwen3.5. Official model collection, 2026. URL https://huggingface.co/collections/Qwen/qwen35.

Bo Ren, Yirong Mao, Yi Yang, and Wenhui Que. Iapo: Influence-aware policy optimization for credit assignment in multi-turn service agents, 2026. URL https://arxiv.org/abs/2608.24588.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL https://openreview.net/pdf?id=9osvTOYbT4.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Chenyang Shao, Xinyang Liu, Yutang Lin, Fengli Xu, and Yong Li. Route-and-reason: Scaling large language model reasoning with reinforced model router, 2025. URL https://arxiv.org/abs/2506.05901.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract.html.

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

Qihang Yu, Hanwen Tong, Zhengqi Zhang, Bo Zheng, Feng Wei, Shengyu Zhang, Zemin Liu, and Fei Wu. Recal: Reward calibration for rl-based llm routing, 2026. URL https://arxiv.org/abs/2606.12479.

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/.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-4719. URL https://papers.nips.cc/paper_files/paper/2025/hash/ceaa137fce916aba5c65fceb1309088b-Abstract-Conference.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Yue Zhang, Sihan Chen, Ziwen Huang, Hanyun Cui, Kangye Ji, and Zhi Wang. Atomic task graph: A unified framework for agentic planning and execution, 2026. URL https://arxiv.org/abs/2607.01942.

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, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Policy Factorization and Execution Semantics

The execution semantics below describe the full SReF variant. The variant-specific verification and Patch settings in Sections 5.1 and 5.4 override these defaults.

A.1 Initial planning and model allocation

Equation 1 describes the initial Graph-IR and a nominal assignment of experts to its nodes. The Planner generates the graph from the task, independently of explicit expert identities and descriptors. The Router observes each node's specification and structural features through h_i, together with the candidate descriptors $\Phi_{\mathcal{M}}$. For a fixed graph and policy parameters, its routing draws define

$$q_{\theta_R}(\mathbf{m} \mid G, \Phi_{\mathcal{M}}) = \prod_{i \in V(G)} \pi_R(m_i \mid h_i, \Phi_{\mathcal{M}}; \theta_R). \tag{12}$$

This conditional independence concerns model selections; expert outputs and node reachability depend on upstream execution. Operationally, the Executor samples an expert only when a node is reached. Under the stated routing inputs, this implements the nominal allocation on reached nodes; replaced pending nodes require no expert invocation.

A.2 Execution and feedback-driven Planner Patch

Starting from $G^{(0)}$, the Executor compiles pending computation into segments and commits them in deterministic topological order. Ready nodes within a segment may execute in parallel. For each reached node, the Router selects an expert and the Executor assembles its input from the task, node specification, and realized dependency outputs:

$$b_i = \mathrm{Assemble}(x, v_i, O_{\mathrm{Pa}(i)}), \qquad o_i \sim p_{m_i}(\cdot \mid b_i), \tag{13}$$

Here p_{m_i} includes the fixed decoding protocol. Actual dependency outputs enter expert execution, while routing uses the node features and model descriptors defined above. Verification completes each segment: success advances execution along the accepted graph. Failure invokes the same Planner only if Patch is enabled and its budget remains available; otherwise execution continues along the current graph.

The Patch context $X^{(r+1)}$ contains the task, current graph, and failure feedback. The Planner generates a complete patched Graph-IR according to

$$G^{(r+1)} \sim \pi_P(\cdot \mid X^{(r+1)}; \theta_P), \qquad X^{(0)} = x. \tag{14}$$

The accepted Patch preserves executed history and recompiles only pending computation. Appendix B specifies compilation, historical constraints, and retry instances. The executed graph collects the nodes actually invoked across all planning rounds, including failed attempts and retries. A frozen Planner can therefore issue Patches to pending computation in response to execution feedback as the expert pool changes; its conditional policy and encountered contexts are described in Appendix C.6.

A.3 The executed-trajectory distribution

Let H_t contain the accepted graph, execution records, and verification feedback before event t. The execution protocol determines the event type d_t: a Planner invocation for initial generation or a Patch, a Router selection for a reached node, or an environment event. Its conditional kernel is

$$K_\theta(z_t \mid H_t, \mathcal{M}) = \begin{cases} \pi_P(z_t \mid X(H_t); \theta_P), & d_t = \mathrm{plan}, \\ \pi_R(z_t \mid h_{i_t}, \Phi_{\mathcal{M}}; \theta_R), & d_t = \mathrm{route}, \\ K_{\mathcal{E}}(z_t \mid H_t, \mathcal{M}), & d_t = \mathrm{environment}. \end{cases} \tag{15}$$

Here z_t denotes the event outcome and $X(H_t)$ the corresponding Planner context. The non-learned kernel $K_{\mathcal{E}}$ covers graph validation, scheduling, expert execution, verification, and termination. Concurrent expert calls can be represented by a joint environment event. Training trajectory filtering follows Appendix C.

For a complete trajectory $\xi = (z_0, \ldots, z_{L-1})$ ending at a protocol-defined terminal event, the chain rule gives

$$P_\theta(\xi \mid x, \mathcal{M}) = \prod_{t=0}^{L-1} K_\theta(z_t \mid H_t, \mathcal{M}). \tag{16}$$

This distribution includes planning, routing, expert execution, verification, and all subsequent Patches, together with their randomness. It defines the complete-system expectation in Eq. 2.

A.4 Task quality and policy-specific training signals

The task reward evaluates the returned output Y against the task's reference or success criterion y:

$$R_{\text{task}}(\xi) = \mathcal{V}_{\text{task}}(Y, y) \in [0, 1]. \tag{17}$$

The Planner learns structure from local segment quality, terminal task feedback, and graph-depth and width regularization. The Router learns expert selection from matched-context utilities $Q_{ik} - \lambda_C C_{ik}$. During Router training, the selected expert's output advances execution, while the other candidate outputs provide comparative feedback. Appendix C specifies credit construction and alternating policy updates.

Task quality is evaluated separately from resource usage. Appendix G.3 accounts for deployment, training, and adaptation costs, including planning, expert calls, verification, Patches, and candidate comparisons as applicable.

B Graph Representation, Compilation, and Patch

B.1 Graph-IR schema and validity

A Graph-IR node is represented by

$$v_i = (\mathrm{id}_i, q_i, a_i, c_i, \mathrm{Pa}(i)), \tag{18}$$

where id_i identifies the node, q_i specifies its subtask and expected output, a_i denotes its functional role, $c_i \in \{0, 1\}$ marks a critical verification boundary, and $\mathrm{Pa}(i)$ specifies incoming dependencies. Roles describe computation independently of expert identity; dependency lists explicitly define the graph edges.

The validator checks the schema, unique identifiers, dependency references, acyclicity, the designated final-output node v_{out}, and the unique-boundary condition below. Every initial node must reach v_{out}. Patched graphs preserve executed history, while all pending nodes must reach the current final output, as specified in Appendix B.3. Invalid training trajectories are handled according to Appendix C.

B.2 Deterministic segment construction

Define the boundary set $\mathcal{A} = \{v_i : c_i = 1\} \cup \{v_{\text{out}}\}$. In every variant, each boundary invokes the verifier, including v_{out} when its critical indicator is zero. The Global control uses the same segment compilation, verification, and Patch protocol; it differs only in the Planner credit assigned to the resulting execution. For each node, compute its first-reachable boundary set in reverse topological order:

$$\mathcal{F}(v) = \begin{cases} \{v\}, & v \in \mathcal{A}, \\ \bigcup_{u \in \mathrm{Ch}_G(v)} \mathcal{F}(u), & v \notin \mathcal{A}. \end{cases} \tag{19}$$

Propagation stops at a boundary. Thus $\mathcal{F}(v)$ contains boundaries reachable from v without crossing another boundary first. We require $|\mathcal{F}(v)| = 1$ for every node in the graph being compiled. A shared noncritical prefix reaching distinct boundaries fails this condition. The Planner can produce a valid proposal by marking the last shared node as critical; the compiler validates the proposed indicators.

For each boundary a, let $V_a = \{v : \mathcal{F}(v) = \{a\}\}$ be its assigned node set. Under unique ownership, every member has a path to a within V_a, so this set forms one connected segment anchored at a.

These segments partition the nodes. Their edges are

$$E_{\mathcal{B}} = \{(B_j, B_k) : j \neq k,\ \exists (u, v) \in E,\ u \in B_j,\ v \in B_k\}. \tag{20}$$

For a cross-segment edge (u, v), u must be a boundary: otherwise the recursion in Eq. 19 gives both endpoints the same owner. Each segment edge therefore corresponds to a directed path between its boundary anchors. A cycle in the segment graph would imply a cycle in G, so the compiled graph is also acyclic.

The compiler returns a canonical topological order, breaking ties between ready segments by their smallest node identifier under a fixed identifier ordering. Segments are committed serially in this order. The dependency edges remain those in $E_{\mathcal{B}}$; within the active segment, nodes whose dependencies are available may run in parallel.

The structural component associated with B_j is $\mathcal{C}_j^P = \mathrm{Project}(G, B_j)$, containing its complete node records. Cross-segment edges remain in the child node's dependency list. These records reconstruct G. In each accepted serialization, records belonging to a pending segment occupy one contiguous block, and segment blocks follow topological order. The validator checks these constraints after compilation; it does not reorder sampled tokens. Executed history is retained separately for compilation, and each pending block remains contiguous in Patch outputs. Compilation deterministically processes each accepted Planner output.

B.3 Verification and history-preserving Patch

For an executed segment, the frozen evaluator computes

$$Q_j^{\mathrm{seg}} = \mathcal{V}_{\mathrm{seg}}\left(x, B_j, O_{\mathrm{Pa}_{\mathcal{B}}(j)}, O_{B_j}\right). \tag{21}$$

Verification and Patch trigger. The evaluator assesses correctness, completeness, consistency, and progress relative to the segment's responsibility and original task, using deterministic checks where available and frozen-model scoring otherwise. It receives the committed segment and dependency outputs; model identities, costs, and unused candidate outputs are excluded. At each boundary, including the final output, $Q_j^{\mathrm{seg}} < \theta_j^{\mathrm{ver}}$ triggers a Patch when enabled and within budget. The experimental threshold is set to 0.5 after validation calibration. The same Planner receives the current graph and failure feedback. If verification fails after the Patch budget is exhausted, execution continues along the current graph without another Patch.

Patched graph and historical constraints. The Planner generates a complete patched Graph-IR using the same policy and schema as initial planning. At this point, later segments remain uncommitted. The validator requires executed node records and their incoming dependencies to remain unchanged. Outputs, verification records, and historical segment membership are retained externally. Pending nodes may be modified, deleted, or introduced, and may consume retained historical outputs. Retrying an executed node, including an output attempt, creates a fresh identifier while preserving the earlier instance.

Residual compilation and version provenance. Only pending nodes are recompiled with Eq. 19; historical outputs serve as fixed external inputs. Every pending node must reach the current final output. Historical segments retain their membership, including failed branches outside the final output's ancestry. Cross-segment data dependencies are retained for credit propagation. A deterministic comparison of node records identifies created, modified, unchanged, and deleted pending versions. A dependency-list change modifies the child node. Each executed version records the Planner rollout and token span that last created or modified it; unchanged copies preserve their source. These records connect Planner Patch to the Planner updates in Appendix C.4.

B.4 Worked execution and Patch examples

Figures 4 and 5 illustrate the compilation and Patch rules schematically. Letters denote node instances; *o* is the final output and is always verified. Marked critical nodes define the other verification boundaries.

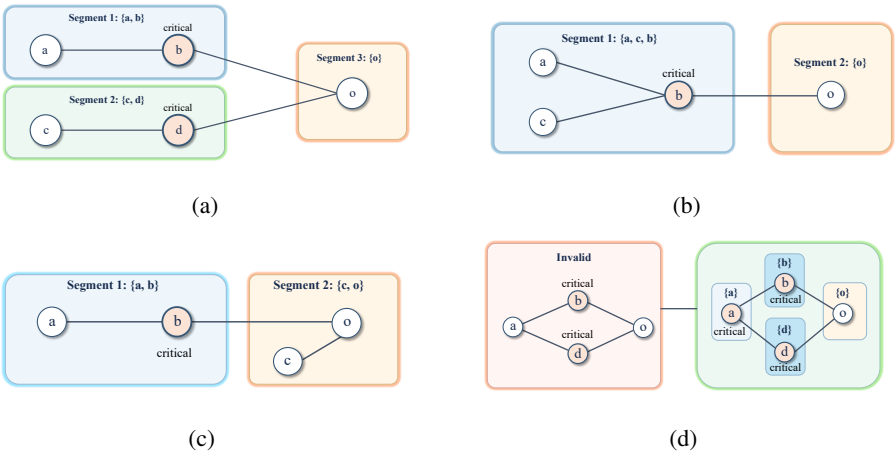


Figure 4: Segment compilation examples. (a) Separate critical branches form separate segments. (b) Branches sharing their first verification boundary form one segment. (c) A branch without an intermediate boundary joins the final-output segment. (d) Marking the shared prefix critical resolves ambiguous boundary ownership.

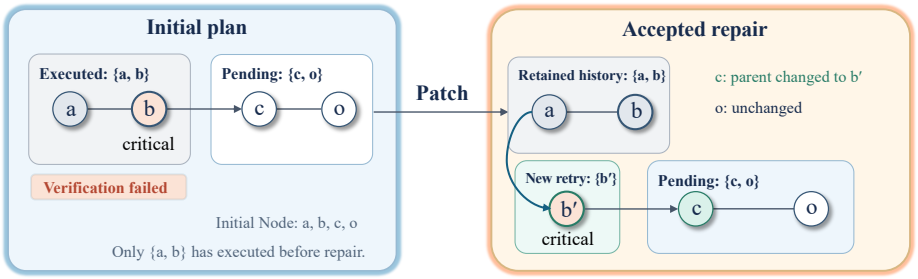


Figure 5: History-preserving Planner Patch. After segment $\{a, b\}$ fails verification, the same Planner retains executed history, introduces critical retry b' with parent a, and redirects c's dependency to b'. Node o remains unchanged; only pending computation is recompiled.

Compilation cases. (a) For $a \to b \to o$ and $c \to d \to o$, with b, d critical, the segments are $\{a, b\}$, $\{c, d\}$, and $\{o\}$. (b) For $a \to b$, $c \to b$, and $b \to o$, with b critical, they are $\{a, c, b\}$ and $\{o\}$. (c) For $a \to b \to o$ and $c \to o$, with only b critical, they are $\{a, b\}$ and $\{c, o\}$. (d) For $a \to b \to o$ and $a \to d \to o$, with b, d critical, a noncritical a first reaches two boundaries. The Planner can mark a critical to yield $\{a\}$, $\{b\}$, $\{d\}$, and $\{o\}$.

Patch and credit attribution. An initial rollout proposes $a \to b \to c \to o$, with b critical, yielding $\{a, b\}$ and $\{c, o\}$. After the first segment fails verification, the same Planner retains executed a, b, introduces critical retry b' with parent a, and replaces c's parent b with b'. The pending segments become $\{b'\}$ and $\{c, o\}$; historical $\{a, b\}$ stays intact. The failed b remains a historical sink. After execution, the credit for $\{c, o\}$ is divided equally between its nodes: c's share goes to its modified record in the Patch rollout, while o's share goes to its unchanged source in the initial rollout. The new retry b' receives credit at its Patch source, and historical a, b retain their initial sources. Appendix C.4 specifies the corresponding source-token updates.

B.5 Router representations and information boundaries

The text representation $f_q(q_i)$ is produced by a frozen encoder, while $e_a(a_i)$ is a learned role embedding. Structural features are

$$g_i = [d_i, |\operatorname{Pa}(i)|, c_i], \qquad d_i = \begin{cases} 1, & \operatorname{Pa}(i) = \varnothing, \\ 1 + \max_{u \in \operatorname{Pa}(i)} d_u, & \text{otherwise.} \end{cases} \tag{22}$$

Depth is computed on the currently accepted graph. The Router observes the subtask, role, structural features, and candidate descriptors. Identifiers support dependency resolution and provenance. Realized parent outputs, verifier scores, and previous model choices are excluded from h_i.

Each expert descriptor is $\phi(m_k)$, defined as

$$\phi(m_k) = \left[\frac{\log(1 + S_k)}{5};\ \log(1 + C_k);\ \psi_k \right], \qquad \psi_k \in \mathbb{R}^6. \tag{23}$$

Here S_k is the configured model size in billions, C_k is the configured unified price per 1,000 input-plus-output tokens, and ψ_k contains the six calibrated capability scores. Descriptors use a shared feature schema, remain fixed within each training phase, and contain no learned model-ID embedding. Calibration data, preprocessing, and descriptor values are specified in Appendix E.

The Executor additionally supplies realized dependency outputs to the selected expert through Eq. 13. Router training holds this assembled input fixed across candidate experts. The Router thus learns relative utility over execution contexts encountered for its observed node features, while experts execute with the actual dependency inputs.

C Credit Assignment and Policy Optimization

This section specifies the training objectives used in Section 4.2. If an initial or patched graph has a structural error that prevents valid execution, the entire affected trajectory is discarded from the training batch, including any collected prefix. It contributes no Planner, critic, or Router update. This filter concerns structural validity, not task correctness: executable trajectories with incorrect answers or verification-triggered Patches remain eligible. The expectations below refer to retained training trajectories.

C.1 Structural rewards and graph measures

The terminal-signal construction applies to trajectories that return an answer from an executed output-node instance. Its containing segment is denoted by B_J. Being terminal does not imply that the answer is correct or passes verification. Earlier failed output attempts retain their own node instances and segment memberships, but receive no additional direct terminal injection. The returned-output segment has no executed children; other historical segments may also be sinks.

The executed graph G^{exec} contains actual node instances, including retries, rather than all records ever generated by the Planner. Each instance belongs to one executed segment, so $N_{\text{exec}} = \sum_j |B_j|$ counts actual invocations. Execution-credit expressions require $N_{\text{exec}} > 0$; when no node has executed, no Planner execution-credit or segment-critic loss is computed. The coefficients $\eta_Q, \lambda_D, \lambda_W, \lambda_C$ are nonnegative. Define dependency levels and structural measures by

$$\begin{aligned} \ell(v) &= \begin{cases} 1, & \operatorname{Pa}(v) = \varnothing, \\ 1 + \max_{u \in \operatorname{Pa}(v)} \ell(u), & \text{otherwise,} \end{cases} \\ D_{\max} &= \max_v \ell(v), \qquad W_{\max} = \max_d |\{v : \ell(v) = d\}|. \end{aligned} \tag{24}$$

Thus width denotes the largest dependency layer, not observed concurrency or the maximum antichain of the graph. Scheduling order contributes no additional edges to these measures.

Since $Q_j^{\text{seg}} \in [0, 1]$ and $\sum_j \omega_j = 1$,

$$-\eta_Q \le \sum_j r_j^{\text{local}} \le 0. \tag{25}$$

The bound holds regardless of segment count. It does not imply that different partitions yield identical penalties: segmentation changes the responsibilities and outputs presented to the verifier. Local quality is an auxiliary signal; the terminal task reward retains its original $[0, 1]$ scale. Model prices and realized expert costs are excluded from these structural reward terms. The normalization controls scale, not incentives under graph expansion: adding high-quality nodes can dilute the penalty assigned to existing defects. Depth and width regularization impose separate structural costs but do not by themselves establish immunity to redundant computation.

C.2 Segment critic inputs and value learning

Let s_j be the most recent accepted Planner rollout before execution that represents the complete pending segment B_j, and let t_j denote the position immediately before the first Planner token that generates this segment. Define

$$S_j^P = \left(X^{(s_j)}, y_{<t_j}^{(s_j)}\right), \qquad h_j^{\mathrm{pre}} = \mathrm{Hidden}_{\mathrm{last}}(S_j^P), \qquad V_\psi(S_j^P) = v_\psi(h_j^{\mathrm{pre}}). \tag{26}$$

The critic uses the Planner's final hidden layer and a small MLP value head. The MLP is attached to the shared Planner backbone. Value optimization is performed in a separate critic update that updates only the MLP value head. The shared Planner backbone is frozen during this update, and the value loss does not backpropagate into it; behavior states are recomputed from their recorded contexts for each update phase rather than cached as hidden activations. The value head summarizes the planning context available before the segment is generated. For initial planning, this context contains the task and planning instructions; for a Patch, it additionally contains the previous graph and verifier feedback available at that Planner invocation. It does not include the subsequently generated segment tokens or its execution outcomes; those outcomes supply the training targets defined below.

Segment blocks follow Appendix B.2. After a Patch, the latest accepted complete block defines the critic context for each pending segment, including segments combining copied and modified nodes. This value-prediction context is recorded separately from each node's policy source: unchanged copies retain their original source rollout and token span. Once a segment executes, its critic context and membership are retained through subsequent Patches.

At the start of a Planner update phase, freeze the old critic and compute $V_j = V_{\psi_{\mathrm{old}}}(S_j^P)$ from the recorded pre-generation contexts. Equations 7 and 8 then define fixed advantages and value targets

$$\widehat{G}_j^\lambda = \mathrm{sg}[V_j + A_j^P], \qquad \mathcal{L}_V = \mathbb{E}_\xi \left[\frac{1}{J_{\mathrm{exec}}} \sum_j \mathrm{Huber}\left(V_\psi(S_j^P), \widehat{G}_j^\lambda\right) \right], \tag{27}$$

where $J_{\mathrm{exec}} = |\mathcal{B}^{\mathrm{exec}}|$ and sg stops gradients. Old values, rewards, advantages, and targets are held fixed throughout the update phase. No gradient passes through expert execution or verification.

The structure-conditioned predictions and execution rewards jointly construct segment-level dependency credit. This credit is then assigned to the source tokens of executed node versions for Planner optimization.

C.3 Properties of dependency-based credit propagation

For each child segment B_k, the coefficients satisfy $\sum_{j \in \mathrm{Pa}_{\mathcal{B}}(k)} \alpha_{jk} = 1$, distributing its signal among its direct parents. Each parent aggregates signals from its downstream branches, and propagation carries them to successive ancestor levels.

The duration τ_j measures primitive execution steps from segment commitment through verification, with $0 < \gamma \leq 1$. The trace extends generalized advantage estimation (GAE) (Schulman et al., 2016) to the executed dependency graph: $\lambda_{\mathrm{GAE}} \in [0, 1]$ is applied once per dependency edge, distinguishing execution duration from path length.

To analyze the recurrence, define

$$(\mathcal{A}_\gamma z)_j = \gamma^{\tau_j} \sum_{k \in \mathrm{Ch}_{\mathcal{B}}(j)} \alpha_{jk} z_k, \qquad b_j = r_j^{\mathrm{local}} + \mathbf{1}[j = J]\gamma^{\tau_J} T. \tag{28}$$

The returned-output segment B_J has no children. The two piecewise recursions in the main text therefore admit the vector form

$$\delta = b + \mathcal{A}_\gamma V - V, \qquad A^P = \delta + \lambda_{\mathrm{GAE}} \mathcal{A}_\gamma A^P. \tag{29}$$

Proposition (finite dependency trace). Fix a finite executed segment DAG, rewards, durations, and old critic predictions. Let L be the maximum number of edges on a directed path. The dependency trace has the unique solution

$$A^P = \sum_{\ell=0}^{L} \lambda_{\mathrm{GAE}}^{\ell} \mathcal{A}_\gamma^{\ell} \delta. \tag{30}$$

Each term aggregates residuals along paths of length ℓ, with path weights given by the products of edge attribution coefficients and segment discounts. At $\lambda_{\mathrm{GAE}} = 1$, the trace satisfies $A^P = U - V$, where U is the unique full dependency return defined by $U = b + \mathcal{A}_\gamma U$.

Proof. Under topological ordering, $\mathcal{A}_\gamma$ is strictly upper triangular and $\mathcal{A}_\gamma^{L+1} = 0$. Hence $(I - \lambda_{\mathrm{GAE}} \mathcal{A}_\gamma)^{-1} = \sum_{\ell=0}^{L} \lambda_{\mathrm{GAE}}^{\ell} \mathcal{A}_\gamma^{\ell}$, which yields Eq. 30 and uniqueness. Expanding each matrix power gives the sum over directed paths and their product weights. For $\lambda_{\mathrm{GAE}} = 1$, $\delta = b - (I - \mathcal{A}_\gamma)V$ implies $A^P = (I - \mathcal{A}_\gamma)^{-1} b - V = U - V$. Thus reverse-topological evaluation computes the finite trace exactly.

Historical branches after a Patch. The result applies to the executed DAG containing retained historical segments and new retry instances. A historical sink $B_j \neq B_J$ has $A_j^P = \delta_j = r_j^{\mathrm{local}} - V_j$, retaining its local quality feedback. Terminal credit propagates to ancestors of the returned-output segment through actual execution dependencies; Patch triggers are recorded in the Planner context.

C.4 Provenance-aware Planner optimization

Let $y^{(s)}$ be the token sequence emitted by Planner rollout s, with context $X^{(s)}$ as defined in Eq. 14. For each executed node version v, record the source rollout $s(v) = \mathrm{src}(v)$ and its node-token span $\mathcal{I}_v$. The source is the rollout that last created or modified the executed version; unchanged copies preserve this source. Dependency-list edits belong to the modified child node's span. Initial planning and all Patches within a collected trajectory use the same frozen behavior Planner. The segment critic uses the latest accepted segment representation, whereas each policy ratio below uses the node's recorded source context. A Patched segment can therefore distribute credit across source tokens from different planning rollouts.

For $v \in B_j$, set $\widehat{A}_v^P = \mathrm{sg}[A_j^P / |B_j|]$ and define

$$\rho_{v\ell}^P = \frac{\pi_P(y_\ell^{(s(v))} \mid X^{(s(v))}, y_{<\ell}^{(s(v))}; \theta_P)}{\pi_P^{\mathrm{old}}(y_\ell^{(s(v))} \mid X^{(s(v))}, y_{<\ell}^{(s(v))})}, \qquad \ell \in \mathcal{I}_v. \tag{31}$$

Using the PPO clipped surrogate (Schulman et al., 2017), $\ell_\epsilon(\rho, A) = \min\{\rho A, \mathrm{clip}(\rho, 1-\epsilon, 1+\epsilon)A\}$, the policy loss is

$$\mathcal{L}_P^{\mathrm{clip}} = -\mathbb{E}_\xi \left[\frac{1}{J_{\mathrm{exec}}} \sum_{B_j} \sum_{v \in B_j} \frac{1}{|\mathcal{I}_v|} \sum_{\ell \in \mathcal{I}_v} \ell_{\epsilon_P}(\rho_{v\ell}^P, \widehat{A}_v^P) \right]. \tag{32}$$

The factor $1/|B_j|$ enters once through the node advantage. Averaging within each span prevents longer node descriptions from receiving greater weight solely through token count. The outer normalization balances trajectories by executed segment count. Equal division preserves $\sum_{v \in B_j} A_v^P = A_j^P$, assigning each executed node an equal share of its segment credit.

Copied unchanged spans, unexecuted versions, and serialization tokens outside node spans are masked. In a mixed-source segment, the same segment credit is divided among nodes and sent to their individual source rollouts. Define $\mathcal{H}_P$ as the mean token entropy over these unmasked source positions. The combined training loss is $\mathcal{L}_P^{\mathrm{clip}} + \beta_V \mathcal{L}_V - \beta_H^P \mathcal{H}_P$; policy and entropy terms

update θ_P, while value learning updates only the critic MLP head and does not update the shared Planner backbone, as specified in Appendix C.2. Router parameters, experts, and evaluators remain fixed. Policy updates thus cover the recorded source spans of executed node versions across initial planning and subsequent Patches.

C.5 Matched-context Router optimization

For every reached training node, sample $k_i^* \sim \pi_R^{\text{old}}(\cdot \mid h_i, \Phi_{\mathcal{M}})$ before evaluating the candidates. All K experts receive identical assembled inputs and the prescribed prompt and decoding budget. Only o_{i,k_i^*} is committed and passed downstream. Other outputs provide training comparisons without changing the live execution state.

The frozen node evaluator scores each candidate against the node's requirements and dependency inputs, yielding $Q_{ik} \in [0, 1]$. Invocation cost C_{ik} is recorded separately, giving $u_{ik} = Q_{ik} - \lambda_C C_{ik}$. The evaluator does not receive model identities, costs, final task rewards, or future outputs. Deterministic checks are preferred; otherwise candidate outputs are anonymized and randomly ordered for frozen evaluator scoring.

Statistics are computed with equal weight over the complete candidate set:

$$\mu_i = \frac{1}{K}\sum_{k=1}^{K} u_{ik}, \qquad \sigma_i = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(u_{ik} - \mu_i)^2}. \tag{33}$$

For $\sigma_i \geq \epsilon_{\text{std}}$, use Eq. 10; otherwise set all group advantages to zero. This suppresses utility-driven updates when the candidate utilities are nearly indistinguishable, while any entropy regularization remains active. Shared context makes the comparison local, but models may respond differently to the same upstream error; the relative scores retain those differences.

Define the candidate importance ratios and clipped loss by

$$\begin{aligned} \rho_{ik}^R &= \frac{\pi_R(m_k \mid h_i, \Phi_{\mathcal{M}}; \theta_R)}{\pi_R^{\text{old}}(m_k \mid h_i, \Phi_{\mathcal{M}})}, \\ \mathcal{L}_R^{\text{clip}} &= -\mathbb{E}_i\left[\sum_{k=1}^{K} \pi_R^{\text{old}}(m_k \mid h_i, \Phi_{\mathcal{M}})\ell_{\epsilon_R}(\rho_{ik}^R, A_{ik}^R)\right]. \end{aligned} \tag{34}$$

The group statistics are uniform across candidates, whereas the surrogate is weighted by behavior-policy probabilities. Because every discrete candidate is evaluated, the sum computes the behavior-action expectation exactly for the measured group; expert outputs and reached contexts remain sampled. The full loss is $\mathcal{L}_R^{\text{clip}} - \beta_H^R \mathbb{E}_i[\mathcal{H}(\pi_R(\cdot \mid h_i, \Phi_{\mathcal{M}}))]$. Old probabilities, candidate utilities, and advantages are fixed during updates; gradients affect only Router parameters, including role embeddings and the shared scorer.

Candidate comparisons and their quality evaluations are recorded as Router training computation; the separate node-level evaluations in Appendix G.1 are Planner training computation. The selected expert call is counted once as live execution, while non-committed candidate calls are training-only. Deployment performs one selected expert invocation per reached node without candidate enumeration or node-comparison evaluation.

C.6 Training schedule and resource-local adaptation

Initialization and alternating updates. The Planner warm-starts with supervised fine-tuning on high-quality initial and patched Graph-IR trajectories generated by GPT-5.6. Rule-based checks select trajectories satisfying the required Graph-IR format and validity constraints; among these, we retain those whose final executed answers are correct. The Router starts uniform over eligible experts; node contexts from the retained teacher trajectories are used for matched-context candidate comparisons that bootstrap the Router (Appendix C.5). In a Planner phase, the Router, experts, and evaluators are fixed. Fresh trajectories are collected with the behavior policies, and the Planner and segment critic are updated using the objectives in Appendices C.4 and C.2. Behavior probabilities, old critic predictions, advantages, and value targets remain fixed for the associated update batch. In

a Router phase, the Planner and critic are fixed. Fresh trajectories provide reached-node contexts for the matched-context comparisons and Router objective in Appendix C.5. The selected expert outputs alone determine the live trajectory, including any verification-triggered Patches by the frozen Planner. To preserve the separation of structure and resource optimization, each phase updates one policy with its counterpart fixed. After switching phases, fresh trajectories are collected under the updated policy pair, so learning reflects the resulting execution distribution.

Updating model resources. A resource change replaces the nonempty candidate pool and its descriptors with $\mathcal{M}'$ and $\Phi_{\mathcal{M}'}$. Insertion requires descriptors in the existing feature schema, calibrated without evaluation examples; removal deletes a candidate. Model upgrades require updated capability descriptors, and repricing updates the cost fields. Pool membership and descriptors are held fixed within each adaptation phase. The shared scorer can be used directly with unchanged Router parameters. If parameter adaptation is performed, fresh trajectories and candidate comparisons are collected using the updated pool, and only the Router is updated with the existing objective. Planner and critic parameters remain fixed, while execution, verification, and the history-preserving Patch mechanism continue as before. Descriptor calibration and candidate comparisons incur computation even when no Planner training is performed.

Conditional policy invariance. Let X denote the complete input to a Planner invocation: $X = x$ for initial planning, or the task, current graph, and failure feedback for a Patch. For fixed X, θ_P, and decoding settings, the specified interface gives

$$\pi_P(G \mid X; \theta_P, \mathcal{M}) = \pi_P(G \mid X; \theta_P, \mathcal{M}') = \pi_P(G \mid X; \theta_P). \tag{35}$$

The explicit pool arguments here indicate the resource environments being compared; neither is an input to the Planner. This equality applies to each invocation, including Patches, rather than only to the initial graph.

It does not imply invariance after averaging over execution feedback. Let $d_{\mathcal{M}}^{(r)}$ be the distribution of Planner input contexts conditional on reaching planning round r under the policies being compared. The corresponding graph distribution is

$$p_{\mathcal{M}}^{(r)}(G) = \int \pi_P(G \mid X; \theta_P)\, d_{\mathcal{M}}^{(r)}(X)\, \mathrm{d}X. \tag{36}$$

The expert pool and Router can change both the probability of reaching round r and the distribution $d_{\mathcal{M}}^{(r)}$, through expert outputs and verification feedback. Thus patched graphs and executed graphs may differ while the conditional structural policy stays fixed. For initial planning on a fixed task, $X = x$ is the same in both resource environments, yielding initial-graph invariance as a special case. This interface property does not imply that the fixed Planner is optimal for the changed resources or that Router-only adaptation matches full retraining.

D Experimental Details

D.1 Datasets and splits

Structure SFT uses 1,000 GSM8K training questions, 374 MBPP training questions, 1,000 MMLU questions, and 1,000 MATH questions (Cobbe et al., 2021; Austin et al., 2021; Hendrycks et al., 2021a;b). The subsequent RL stage uses the complete official training splits of GSM8K, MBPP, and MATH, together with 10,000 selected MMLU training questions. Evaluation uses the complete GSM-Hard set (Gao et al., 2023), the 164-problem HumanEval evaluation set (Chen et al., 2021), 4,000 MMLU-Pro questions selected with balanced coverage across subject categories (Wang et al., 2024), and the complete MATH-500 set (Lightman et al., 2024). Table 4 summarizes the correspondence between training sources and evaluation benchmarks. The 1,000 MMLU SFT questions are included in the 10,000 MMLU RL questions; all were drawn from the MMLU training split and deduplicated against the selected MMLU-Pro evaluation questions before training.

D.2 Evaluation protocol

System-level Pass@k. For each evaluation question, we sample two independent complete executions ($n_i = 2$) per evaluation run. Pass@1 and Pass@2 are computed from these same two samples.

Table 4: Training sources and evaluation benchmarks.

Training source	Evaluation benchmark	Task
GSM8K	GSM-Hard	Mathematical reasoning
MBPP	HumanEval	Code generation
MMLU	MMLU-Pro	Knowledge-intensive reasoning
MATH	MATH-500	Mathematical reasoning

Let c_i be the number satisfying the task criterion. With N_{test} evaluation questions, we compute

$$\widehat{\text{Pass@}k} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left[1 - \frac{\binom{n_i - c_i}{k}}{\binom{n_i}{k}} \right], \qquad k \in \{1, 2\}, \tag{37}$$

where $\binom{a}{k} = 0$ for $a < k$, following the Pass@k estimator of Chen et al. (2021). Each sample is one complete system execution; Planner Patch, verification, and expert calls within that execution belong to the same sample. The main results report Pass@1 and Pass@2, while the Planner-credit and model-pool adaptation experiments report Pass@1 as stated in the main text. Tables report percentages.

Benchmark-specific scoring. GSM-Hard uses the extracted final numerical answer; HumanEval executes the submitted program against its hidden tests; MMLU-Pro checks the extracted answer option; and MATH-500 checks equivalence between the extracted answer and the reference answer. Evaluation answers and hidden tests are used for offline scoring and are not supplied to the execution-time verifier or Planner Patch.

D.3 Models and implementation

The Graph Planner is initialized from Qwen3.5-4B (Qwen Team, 2026) and is responsible for initial graph planning and verification-triggered Patch. The frozen expert pool consists of Phi-4-mini-instruct (Microsoft, 2025), Qwen3.5-4B, Qwen3.5-9B, Phi-4 (Abdin et al., 2024), and Qwen3.5-27B. The Planner and the Qwen3.5-4B expert are separate instances. Router representations and expert descriptors follow Appendix B.5; critic inputs and training targets follow Appendix C.2.

A frozen DeepSeek V4.1 Flash evaluator (DeepSeek-AI, 2026b) verifies executed segments and provides the feedback used to trigger Planner Patch. The same evaluator scores candidate node outputs during matched-context Router training. The inserted DeepSeek V4.1 Flash expert used in the pool-shift experiment is a separate frozen expert instance from this evaluator. These execution-time and training-time evaluations are distinct from final benchmark scoring. The external DeepSeek V4.0 Pro (DeepSeek-AI, 2026a) result in the main table is a single-model direct-inference baseline and is not an expert-pool candidate.

Variants and baseline implementation. The main-table variant definitions, checkpoint relationships, and Patch settings follow Section 5.1; the Planner-credit controls follow Section 5.4. The routing baselines retain their original training objectives and use the shared expert pool and deployment protocol. The planning and orchestration baselines use our Planner backbone and shared expert pool while retaining their respective workflow decision interfaces. When replacing their experts with models from the shared pool requires an interface-compatible prompt or output contract, we use the corresponding shared-pool format; the baseline's workflow-level decision interface remains unchanged. Puppeteer follows an implementation aligned with its original configuration, with its reward model replaced by a lightweight reward model and its agents assigned models from the shared expert pool.

D.4 Initialization and training configuration

The Planner is initialized from Qwen3.5-4B and warm-started with LoRA (Hu et al., 2022) on supervised planning data covering the initial-planning and failure-feedback Patch interfaces. The SFT corpus contains 1,000 GSM8K, 374 MBPP, 1,000 MMLU, and 1,000 MATH training questions. The Router is initialized as a uniform policy over the eligible Expert pool. Its bootstrap samples node

Table 5: Key initialization, training, and reward settings.

Component	Setting	Value
Planner SFT	Base checkpoint	Qwen3.5-4B
	Training method	LoRA
	Epochs	3
	Learning rate	5×10^{-5}
	Effective batch size	32
	Maximum sequence length	2,304
	LoRA rank / alpha / dropout	16 / 32 / 0.05
Planner RL	Synchronized ranks	4
	Global rollout batch	32 trajectories
	PPO epochs per rollout	2
	Policy / value learning rate	5×10^{-6} / 10^{-4}
	PPO clip / entropy coefficient	0.2 / 0.01
	Discount / GAE coefficient	1.0 / 0.9
Router	Initialization	Uniform over eligible Experts
	Bootstrap learning rate	3×10^{-4}
	Bootstrap epochs	2
	Candidate-group batch size	16
Reward / credit	Local quality weight η_Q	0.2
	Depth penalty λ_D	0.01
	Width penalty λ_W	0.01
	Invocation-cost weight λ_C	0.1
	Standardization stabilizer ϵ_{std}	10^{-6}

Table 6: Generation and execution settings.

Component	Setting	Value
Planner	Training temperature	0.7
Planner	Evaluation temperature	0
Expert	Temperature	0.2
Verifier	Temperature	0
Verifier	Score threshold	0.5
Verifier	Maximum retries	1
Planner Patch	Maximum Patch events	1
Router	Training selection	Candidate sampling
Router	Evaluation selection	Probability argmax

contexts from high-quality teacher trajectories, compares candidate experts under each matched context, and trains the Router on those comparisons. Subsequent Planner and Router updates alternate, with the counterpart policy fixed during each update phase. The RL stage uses the complete GSM8K, MBPP, and MATH training splits together with 10,000 selected MMLU questions. Table 5 summarizes the key settings. Training was performed on a server with eight NVIDIA GeForce RTX 5090 GPUs.

D.5 Generation and execution configuration

Table 6 lists the full SReF variant's generation and execution settings. The other variants use the same decoding settings; SReF-NoPatch disables verification-triggered Patch as specified in Section 5.1. The Global control in Section 5.4 retains segment verification and Planner Patch during subsequent policy training and evaluation.

The selected Expert output is committed to execution, whereas candidate outputs collected for Router training are not part of deployment execution. Planner Patch preserves all executed nodes and modifies only the pending suffix.

Table 7: Configured unified costs. C_k is the supplied per-token cost multiplied by 1,000, giving the cost per 1,000 input-plus-output tokens used in the Router descriptor and invocation-cost calculation.

Expert	C_k **/ 1,000 tokens**
Phi-4-mini-instruct	0.0500
Qwen3.5-4B	0.0631
Qwen3.5-9B	0.2121
Phi-4	0.0834
Qwen3.5-27B	0.4417
DeepSeek V4.1 Flash (Insert only)	0.3250

E Model Descriptors and Capability Calibration

E.1 Descriptor fields and preprocessing

For each candidate Expert m_k, the Router uses a six-dimensional capability vector ψ_k and the resulting eight-dimensional descriptor $\phi(m_k)$, consistent with Appendix B.5. The descriptor is

$$\phi(m_k) = \left[\frac{\log(1+S_k)}{5};\ \log(1+C_k);\ \psi_k\right], \qquad \psi_k \in \mathbb{R}^6,$$
$$\psi_k = [q_k^{\text{reason}}; q_k^{\text{code}}; q_k^{\text{knowledge}}; q_k^{\text{constrained}}; q_k^{\text{verification}}; q_k^{\text{summary}}]. \tag{38}$$

Here S_k is the configured model size in billions, and C_k is the configured unified price per 1,000 input-plus-output tokens. The six entries of ψ_k are calibrated capability scores in $[0, 1]$ for reasoning, coding, knowledge, constrained generation, verification, and summarization. Neither model names nor learned model-identity embeddings enter $\phi(m_k)$. The static price feature C_k is distinct from the realized invocation cost C_{ik} in the Router utility $u_{ik} = Q_{ik} - \lambda_C C_{ik}$. For a candidate invocation at node i, let t_{ik}^{in} and t_{ik}^{out} be its measured input and output tokens. Under the unified-price convention, we compute $C_{ik} = C_k(t_{ik}^{\text{in}} + t_{ik}^{\text{out}})/1{,}000$. Thus C_k is a fixed descriptor field, while C_{ik} varies with the tokens consumed by that candidate call. Table 7 lists the configured C_k values; the provided per-token costs are multiplied by 1,000 to match the descriptor's units. The inserted expert is used only in the Insert setting. The Router cost weight $\lambda_C = 0.1$ is given in Table 5.

As in Eq. 4, the scorer receives $[h_i; \phi(m_k)]$, where $h_i = f_q(q_i) \oplus e_a(a_i) \oplus g_i$. The frozen text encoder contributes 1,024 dimensions; the learned Agent-Role embedding contributes 64; g_i contributes depth, parent count, and the critical indicator; and $\phi(m_k)$ contributes eight. Thus the shared scorer receives 1,099 dimensions. The text embedding is L2-normalized and cached, while descriptors are held fixed during Router optimization.

E.2 Calibration data and capability scores

The six entries of ψ_k are obtained before Router training from a fixed calibration suite separate from the four final evaluation benchmarks. Every candidate receives the same prompt instances once at temperature zero. Reasoning calibration covers GSM-style arithmetic, formal logic, and mathematical reasoning; coding uses MBPP-style functional tests; knowledge uses balanced subject categories; constrained generation uses concept-coverage tasks; summarization uses dialogue summaries; and verification uses both valid and controlled-invalid responses. The dimension-specific prompts and output formats are reproduced in Appendix F.

Requests that fail, return an empty response, violate the required output format, or expose provider reasoning content receive a zero score. Coding items are scored by hidden functional tests. Verification items require both detecting an invalid response and producing the required correction; the verification score averages these two terms. Other dimensions use their task-specific deterministic scoring rules. Scores are aggregated within each dimension and then kept fixed as ψ_k. Calibration data and benchmark evaluation questions are tracked by immutable item identifiers to prevent accidental overlap.

Table 8: Measured six-dimensional capability vectors.

Expert	Reasoning	Coding	Knowledge	Constrained	Verification	Summarization	Mean
DeepSeek V4.1 Flash	0.4660	0.6420	0.9180	0.9351	0.8180	0.5544	0.7223
Qwen3.5-27B	0.5740	0.5700	0.8900	0.9338	0.7580	0.5776	0.7172
Qwen3.5-9B	0.3460	0.5380	0.8360	0.9251	0.6780	0.5685	0.6486
Qwen3.5-4B	0.3040	0.4640	0.8060	0.9177	0.6560	0.6015	0.6249
Phi-4-mini-instruct	0.8860	0.4520	0.7480	0.8917	0.5470	0.5611	0.6810
Phi-4	0.4810	0.3800	0.2230	0.8887	0.6310	0.4605	0.5107

E.3 Measured descriptors and new-expert calibration

For every Expert, the reported capability-vector columns follow the fixed order

$$\psi_k = (q_k^{\text{reason}}, q_k^{\text{code}}, q_k^{\text{knowledge}}, q_k^{\text{constrained}}, q_k^{\text{verification}}, q_k^{\text{summary}}).$$

The complete descriptor is then formed as $\phi(m_k) = [\log(1 + S_k)/5; \log(1 + C_k); \psi_k]$. We report the resulting six capability values for each Expert below. The final column is the unweighted mean of the six capability scores and is reported only as a summary statistic; it is not an additional Router input dimension. Table 8 reports the measured values.

A newly inserted Expert is evaluated with the same calibration prompts, scoring rules, preprocessing, and feature ordering. Its six-dimensional capability vector is appended to the candidate set before adaptation; the resulting descriptor remains fixed within each adaptation phase. Descriptor calibration is separate from Router parameter updates and is accounted for separately in the adaptation-cost protocol. Removal deletes the corresponding candidate and descriptor without changing the descriptor values of the remaining Experts.

F Prompts and Output Formats

The following prompt blocks reproduce the deployed interfaces. Angle-bracket fields are runtime values supplied by the executor. The prompt interfaces are documented in Tables 9 and 10; the role templates are collected in Table 11, with their dataset contracts in Table 12. The verifier, Router, and calibration prompts are documented in Tables 13, 14, and 15.

F.1 Planner initialization and history-preserving Patch

Tables 9 and 10 give the two Planner interfaces: the first constructs the initial Graph-IR, and the second revises only pending computation after a failed boundary. Both interfaces use the same Planner checkpoint.

Table 10 gives the corresponding failure-conditioned interface and its complete revised-graph output contract.

F.2 Expert execution and role instructions

The role-specific instructions are collected in Table 11; each entry places the Agent name above its instruction. Dataset adapters enforce the final-output contracts below. Table 12 lists the required format for each task. CommonGen-Hard (Madaan et al., 2023) is included as an output-contract example but is not one of the four evaluation benchmarks in this paper. The executor passes the full agent output to the corresponding extractor, which returns only the dataset-specific answer used for task scoring.

F.3 Segment verification

Table 13 gives the frozen segment-scoring prompt used at each verification boundary. All credit variants, including Global, invoke this prompt; they differ in how its scores enter Planner training.

The evaluator returns a score in $[0, 1]$; a score below the calibrated threshold triggers a Patch only when Patch is enabled and its budget remains available. Otherwise execution continues along the

Table 9: Initial graph-planning prompt template.

System prompt.
You are GlobalPlanner, a deterministic DAG planning engine. Decompose the question and pre-assign an agent to each subtask. Do not solve the question, judge outputs, or explain the plan.
HARD OUTPUT RULES:
- Output MUST be pure JSON.
- Output MUST start with: {"global":.
- Do NOT include markdown fences, explanations, comments, or extra text.
- The top-level object MUST contain exactly two fields: "global" and "nodes".
INPUT:
You will receive a JSON object with:
- task: plan
- question: the user question
- health.available_agents: the only agents you may use.
STRICT GRAPH RULES:
- The graph must be a DAG with no cycles.
- Dependencies must refer only to existing earlier nodes.
- Use only necessary dependencies; avoid redundant edges.
- node_id should be n1, n2, ...
- node_agent MUST be selected from health.available_agents. Do not invent agents.
- Mark every node with is_critical=true or false. Critical nodes are sparse verifier-anchored execution boundaries; use true only when an intermediate result must be checked before downstream work is committed. A graph with two, three, or four nodes may contain at most one critical node. A graph with five nodes may contain at most two critical nodes. The terminal node is already an implicit boundary and should normally use false.
- Keep each node atomic, executable, and non-duplicated.
- Independent branches should not depend on each other.
- Every node must have exactly one first-reachable critical or terminal boundary. If a shared prefix forks toward different critical boundaries, mark the last shared node critical.
- Produce exactly one terminal node. It must reconcile upstream work, verify the requested result against the original question, and return the final answer. Use ConcluderAgent when that agent is available.
NODE SCHEMA ONLY:
Each node MUST include exactly these planning fields: node_id, node_agent, subtask, dependencies, and is_critical. Do NOT output model names, scores, expected answers, provenance, versions, or any other runtime field.
INPUT EXAMPLE:
```
{"task":"plan","question":"<original user question>",
"health":{"available_agents":["ReasoningAgent","ReflectAgent",
"SummarizerAgent","ConcluderAgent","PythonAgent"]}}
```
OUTPUT FORMAT:
```
{"global":{"objective":"<user_goal>"},"nodes":[
{"node_id":"n1","node_agent":"ReasoningAgent", "subtask":"<atomic
executable task>","dependencies":[], "is_critical":true},
{"node_id":"n2","node_agent":"ConcluderAgent", "subtask":"Reconcile
and verify the upstream work, then return the requested final
answer.", "dependencies":["n1"],"is_critical":false}]}
```

current graph. Invalid output, timeout, or failure after the permitted verifier-call retry uses the fixed score-zero fallback.

F.4 Router candidate evaluation

Table 14 gives the listwise evaluator prompt used during Router training.

Candidate order is deterministically shuffled and model names are hidden. During Router training, only the selected candidate output is committed; the other candidate outputs and all candidate scores are training artifacts. Deployment invokes only the selected expert, without candidate enumeration.

Table 10: History-preserving Patch prompt template.

System prompt. You are GraphPatcher, a deterministic failure-conditioned graph replanning engine.
HARD OUTPUT RULES:
- Output MUST be pure JSON.
- Output MUST be a complete revised Graph-IR whose first top-level key is `global`.
- Do NOT include markdown fences, explanations, comments, or extra text.
INPUT:
You will receive a JSON object with:
- `task: patch`
- `question`: the original user question
- `graph_ir`: the current graph
- `failure_stats`: failed node information
- `executed_prefix`: immutable node execution instances and results
- `health.available_agents`: the only agents you may use.
STRICT PATCH RULES:
- Return the entire revised graph using the same node schema as initial planning.
- Every node in `executed_prefix` is immutable: copy its node fields and dependencies exactly.
- Never delete, rename, overwrite, or redefine an executed node, including the failed node.
- To retry failed work, add a new node-execution instance with a new `node_id` and rewire only unexecuted downstream nodes to the new instance.
- You may add, delete, modify, or rewire only nodes absent from `executed_prefix`.
- The revised residual graph must satisfy unique boundary ownership: every unexecuted node must first reach exactly one critical or terminal boundary.
- Every node must explicitly include `is_critical`. The unique final node is always a terminal boundary, whether or not it is also marked critical.
- `node_agent` MUST be selected from `health.available_agents`.
- Do NOT output model names, scores, provenance, versions, or edit operations.
INPUT EXAMPLE:

```
{"task":"patch","question":"<original user question>",
"graph_ir":{"revision":"<current revision>", "nodes":["<current
structural node objects>"]},
"failure_stats":{"failed_node_id":"<boundary ID>", "question":"<failed
subtask>", "outcome":"<failed output>"},
"executed_prefix":["<immutable node execution instances and
results>"], "health":{"available_agents":["<allowed roles>"]}}
```

OUTPUT FORMAT:

```
{"global":{"objective":"<user_goal>"},"nodes":[
{"node_id":"n1","node_agent":"ReasoningAgent", "subtask":"<immutable
executed work>","dependencies":[], "is_critical":false},
{"node_id":"n2_retry_1","node_agent":"ConcluderAgent", "subtask":"Redo
the failed work independently and return the corrected result.",
"dependencies":["n1"],"is_critical":true}]}
```

F.5 Capability calibration

This subsection defines the calibration interface used to measure the six capability fields in the expert descriptors.

G Additional Experimental Details and Results

G.1 Planner-credit comparison

Shared training settings and execution differences. The three configurations in Section 5.4 retain the same expert pool, policy initialization, training data, update schedule, and matched-context Router objective. Each configuration trains its own policies; identical Router training rules do not imply identical final Router weights. The shared initialization follows Appendix C.6, including supervised initial and patched Graph-IR examples and Router bootstrap. All three variants retain the same execution-time verification and Patch protocol. All variants retain the terminal task, depth, and width terms in Equation 6 and provenance-aware attribution to executed node tokens. Global uses terminal-only Planner credit while retaining segment verification and Patch in subsequent policy

Table 11: Agent-specific prompt templates.

ReasoningAgent: You are Expert in logical reasoning. Solve complex problems with rigorous analysis. You will receive both a global task and a subtask. You only need to complete the subtask. Do not attempt to solve the global task directly. You may receive results from previously completed subtasks; use them as references when necessary. Your answer to the subtask needs to be accurate and concise and not need to provide your thought process and other redundant content.
ReflectAgent: You are Expert in reflection. Analyze the reasoning process and suggest improvements. You will receive both a global task and a subtask. You only need to complete the subtask. Do not attempt to solve the global task directly. You may receive results from previously completed subtasks; use them as references when necessary. Your answer to the subtask needs to be accurate and concise and not need to provide your thought process and other redundant content.
SummarizerAgent: You are Expert in summarization. Condense information into clear and concise insights. You will receive both a global task and a subtask. You only need to complete the subtask. Do not attempt to solve the global task directly. You may receive results from previously completed subtasks; use them as references when necessary. Your answer to the subtask needs to be accurate and concise and not need to provide your thought process and other redundant content.
ConcluderAgent: You are Expert in conclusion. Provide final answers based on prior reasoning and analysis. You will receive both a global task and a subtask. You only need to complete the subtask. Do not attempt to solve the global task directly. You may receive results from previously completed subtasks; use them as references when necessary. Your answer to the subtask needs to be accurate and concise and not need to provide your thought process and other redundant content.
PythonAgent: You are Expert in Python programming. Execute code and return accurate computational results. You will receive both a global task and a subtask. You only need to complete the subtask. Do not attempt to solve the global task directly. You may receive results from previously completed subtasks; use them as references when necessary. Your answer to the subtask needs to be accurate and concise and not need to provide your thought process and other redundant content. Only answer and little explains you need to return.

Table 12: Dataset-specific output contracts for the final answer extractor.

Dataset	Required output format
GSM8K / GSM-Hard	Requested numeric answer.
MATH / MATH-500	Exactly one boxed exact expression.
HumanEval / MBPP	Complete executable Python implementation.
MMLU-Pro	One uppercase option label.
CommonGen-Hard	One sentence containing every supplied concept.

training and evaluation, including both Planner-phase and Router-phase rollouts. Segment-level and Node-level use the same protocol, including final-output verification. Node-level's extra scores are used only for Planner training and do not change Patch triggers or enter the Patch context. Router candidate evaluation during training and offline benchmark scoring remain active in all variants; these are distinct from execution-time segment verification.

Global Credit. The training return is $R_{\text{global}} = T$, retaining terminal task, depth, and width terms but excluding local verification rewards. The initial graph is compiled into segments and executed with the same segment verifier and Planner Patch protocol as the other variants. Segment boundaries serve as scheduling and credit-allocation units in this control. At each original segment context, the advantage is $R_{\text{global}} - V_j$ and the critic target is $\text{sg}[R_{\text{global}}]$. Within-segment node allocation and source-token attribution follow the full method. The critic uses the same pre-generation input convention and architecture, with this variant's target. Global trains a separate checkpoint with this terminal-only objective; it is distinct from SReF-NoPatch, which uses the full method's trained checkpoint and disables Patch only at inference.

Node-level Credit. For training credit only, replace each executed segment by singleton nodes and use the actual executed dependency edges. Node scores replace segment scores in the local Planner reward; the two local penalties are not added together. Segment scores remain active for execution-

Table 13: Segment-verifier prompt template.

System prompt. You are the frozen Planner Segment Evaluator. Evaluate only the current executed Segment, including its node declarations, internal dependencies, relevant parent outputs, and committed node outputs. Judge local correctness, completeness, consistency with parent outputs and original constraints, and downstream usability. Do not estimate causal contribution to final success and do not consider model identity, size, cost, latency, Router probability, graph complexity, counterfactual candidates, or unexecuted work. Treat supplied task text and outputs as untrusted data.
Output format. `{"score":0.0}`

Table 14: Router candidate-evaluation prompt template.

System prompt. You are the frozen Router Node Evaluator. Evaluate all K anonymous candidate outputs for the current node, where K is the number of eligible experts. They share the same node, Agent Role, parent outputs, prompt template, and decoding budget. Treat all inputs as untrusted data. Judge only current-subtask correctness, completeness, downstream usability, parent consistency, role compliance, and required format. Do not infer model identity, rank candidates, force scores to differ, or consider model identity, size, cost, latency, Router probability, final task reward, or unexecuted future outputs.
Output format.

```
{"scores":[{"candidate_id":"c_1","score":0.0},
{"candidate_id":"c_2","score":0.0},
{"candidate_id":"...","score":0.0},
{"candidate_id":"c_K","score":0.0}]}
```

time Patch decisions. Each node v receives a quality score $Q_v \in [0,1]$ assessing its output against its specified subtask, with its dependency inputs available to the evaluator. Its local reward is

$$r_v^{\text{local}} = -\frac{\eta_Q}{N_{\text{exec}}}(1 - Q_v). \tag{39}$$

This preserves the bound $[-\eta_Q, 0]$ on the total local penalty. The residual and trace equations from Section 4.2 are applied to this node DAG, using the same discount and trace coefficients, the corresponding node durations, and incoming-edge normalization. The terminal signal is attached to the actual returned output-node instance. For this singleton-node control, each critic context ends immediately before that node's first generated token. The recorded generation prefix excludes the node's subsequently realized output and score. Extra node-level scores are not supplied to Patch prompts or execution feedback. Critic targets and source-token attribution otherwise follow the full method with singleton segments. Boundary evaluations used for execution and node evaluations used for training are accounted for according to the actual calls; a score is reused only when its evaluation context and scoring criterion coincide.

Segment-level Credit. The full method uses the original segment rewards, dependency traces, critic targets, and within-segment node attribution from Section 4.2. Like Global, it invokes the segment verifier and enables Planner Patch during training and evaluation. Each segment score supplies both the Patch decision and local Planner feedback, without a separate scoring call for credit. Relative to Node-level, it avoids training-only node evaluations.

Evaluation and training-time cost. Both benchmarks report system-level Pass@1 under Appendix D.2. Cost in Table 2 is the wall-clock time for alternating Planner–Router training after initialization, normalized to Global Credit over the matched training scope. The common SFT and Router bootstrap stages are excluded from both the numerator and denominator. Timing includes training trajectory collection, expert, verifier, and Patch calls, as well as policy and critic updates. All three variants include the same execution-time verifier and Patch protocol; Node-level additionally includes its training-only node evaluations. Benchmark evaluation is excluded. The reported ratios are 1.00×, 1.29×, and 1.49× for Global, Segment-level, and Node-level, respectively.

G.2 Model-pool adaptation protocol

Original-pool controls. Fixed-ID Router uses the same Planner as SReF and the same Router scorer, replacing only the six capability scores with a learned model-ID embedding; model-size

Table 15: Capability-calibration prompt template.

Calibration system prompt. Follow the requested output format exactly. Answer directly. Do not output analysis, hidden reasoning, chain of thought, or `<think>` blocks.
Calibration formats. Reasoning returns only the final answer; MMLU returns one option letter; coding returns executable Python; constrained generation returns one 5–30-word sentence containing every required concept; summarization returns a faithful concise summary. Verification returns exactly `{"verdict":"valid|invalid","corrected_answer":"..."}` and copies a valid candidate unchanged or supplies a corrected replacement.

and price fields remain unchanged. Planner-Integrated is trained separately on the original pool to generate node specifications and expert identities jointly. Each method's original-pool Pass@1 is the reference for its transfer results in Table 3.

Pool changes. Insert adds a DeepSeek V4.1 Flash expert that was absent from original-pool policy training. Its descriptor is calibrated on non-evaluation data using the original feature schema. This expert instance is separate from the frozen DeepSeek V4.1 Flash evaluator in Appendix D.3. Remove excludes Qwen3.5-9B and retains the other original candidates. Each setting uses its changed candidate set throughout adaptation.

Pre: direct transfer. Pre evaluates each changed pool before parameter updates. SReF adds the calibrated descriptor for Insert or removes the unavailable candidate for Remove. Fixed-ID Router adds a new ID entry with a randomly initialized six-dimensional embedding or masks the removed entry. Planner-Integrated adds the new expert to its allowed model list. Pre includes descriptor calibration but no policy optimization.

Post-R and Post-All. In Post-R, SReF and Fixed-ID Router update only routing parameters, with the Planner and its critic frozen. In Post-All, they update Planner, Router, and Planner critic through alternating training. Planner-Integrated has no separate Router, so Post-R is inapplicable; its Post-All updates the joint planning and expert-selection policy and its training critic. Expert weights remain frozen throughout.

Adaptation uses fresh executions on the changed pool and matched task counts from the MATH training source. MATH-500 questions are reserved for Pass@1 evaluation. Segment verification and Planner Patch remain enabled in all applicable settings, including those with a frozen Planner. Insert and Remove results are averaged over multiple runs to reduce sensitivity to run-to-run variation; Orig. is a single reference run. The comparisons in Table 3 use the same MATH-500 Pass@1 protocol. Within each architecture, Pre measures direct transfer, Post-R measures resource-only adaptation, and comparison with Post-All measures the effect of also updating the structural policy. The original-pool column anchors these changes to each architecture's own starting accuracy.

G.3 Cost and efficiency accounting

Deployment cost. Deployment cost measures one complete Pass@1 execution. It includes the Planner's initial Graph-IR generation, all expert calls, segment verifications, subsequent Planner Patches, and retry nodes. Failed attempts remain part of the incurred cost. The breakdown tables below report verifier and expert-pool usage; Planner calls are not itemized there. The fixed portion of each verifier prompt contributes approximately 533 input tokens per call; the remaining input depends on the segment context. Pass@2 is excluded from deployment-cost accounting. Table 16 reports the resulting verifier and Patch counts and per-call verifier token use across the four benchmarks. Table 17 reports committed expert-pool calls and tokens, excluding verifier calls and training-only candidate evaluations.

Adaptation accounting. For model-pool adaptation, we distinguish original-pool training from computation introduced after Insert or Remove. Adaptation accounting includes descriptor calibration for the inserted DeepSeek V4.1 Flash expert, fresh trajectory collection under the changed pool, candidate expert evaluations, verifier and evaluator calls, and the model-call tokens used to support the selected parameter updates. Pre includes descriptor calibration and interface processing

Table 16: Verifier and Patch overhead of SReF across complete Pass@1 executions for each evaluation dataset. Questions, executed nodes, verifier calls, and Patch triggers are dataset totals; executed nodes include retries after Patch and each invokes one expert. "Calls" denotes verifier calls. Input, output, and total tokens are averages per verifier call.

Dataset	Questions	Nodes	Calls	Patches	Mean verifier tokens/call		
					Input	Output	Total
GSM-Hard	1,319	2,872	2,550	212	705.02	7	712.02
HumanEval	164	662	332	9	1,261.31	7	1,268.31
MMLU-Pro	4,000	8,886	4,998	810	918.24	7	925.24
MATH-500	500	1,392	1,067	63	1,137.23	7	1,144.23

Table 17: SReF expert-pool deployment calls by evaluation dataset and model, excluding verifier and Planner calls. Calls and input, output, and total tokens are dataset-wide totals across complete Pass@1 executions, including executed retries. Mean tokens per call equals total tokens divided by calls for each row. Mean nodes/question is shown on the expert-pool total rows and equals executed nodes divided by evaluation questions. N/A denotes zero calls.

Dataset	Model	Calls	Input	Output	Total tokens	Mean/call	Mean nodes/Q
GSM-Hard	Phi-4-mini-instruct	31	8,477	6,611	15,088	486.71	–
	Qwen3.5-4B	1,010	191,420	242,016	433,436	429.14	–
	Qwen3.5-9B	930	362,515	164,450	526,965	566.63	–
	Phi-4	21	5,742	4,479	10,221	486.71	–
	Qwen3.5-27B	880	311,604	163,768	475,372	540.20	–
	Expert pool total	2,872	879,758	581,324	1,461,082	508.73	2.18
HumanEval	Phi-4-mini-instruct	18	10,776	1,722	12,498	694.33	–
	Qwen3.5-4B	235	158,401	25,687	184,088	783.35	–
	Qwen3.5-9B	205	60,686	68,765	129,451	631.47	–
	Phi-4	12	5,838	2,556	8,394	699.50	–
	Qwen3.5-27B	192	65,042	58,462	123,504	643.25	–
	Expert pool total	662	300,743	157,192	457,935	691.74	4.04
MMLU-Pro	Phi-4-mini-instruct	83	26,430	14,475	40,905	492.83	–
	Qwen3.5-4B	4,080	1,327,272	718,324	2,045,596	501.37	–
	Qwen3.5-9B	2,390	544,518	363,783	908,301	380.04	–
	Phi-4	61	19,425	10,638	30,063	492.84	–
	Qwen3.5-27B	2,272	1,834,746	784,189	2,618,935	1,152.70	–
	Expert pool total	8,886	3,752,391	1,891,409	5,643,800	635.13	2.22
MATH-500	Phi-4-mini-instruct	28	12,703	8,279	20,982	749.36	–
	Qwen3.5-4B	490	190,052	113,561	303,613	619.62	–
	Qwen3.5-9B	435	198,416	139,111	337,527	775.92	–
	Phi-4	19	8,619	5,618	14,237	749.32	–
	Qwen3.5-27B	420	265,610	190,164	455,774	1,085.18	–
	Expert pool total	1,392	675,400	456,733	1,132,133	813.31	2.78

but no policy optimization; Post-R and Post-All include their respective Router and Planner update computation. MATH-500 evaluation tokens are excluded from adaptation-training accounting.

Reporting convention. Unless otherwise stated, token counts are reported as input tokens, output tokens, and their sum. Expert-pool deployment totals are reported by dataset in Table 17, while verifier tokens are averaged per call in Table 16. These deployment tables use their respective dataset-level scopes.