FINE-GRAINED CREDIT ASSIGNMENT FOR AGENT ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

We address the problem of agent architecture search, which constructs query-specific agent architectures from a set of candidate operators. Existing approaches optimize a controller using an architecture-level reward or advantage, assigning the same learning signal to all operator decisions within a sampled architecture, regardless of their individual contributions to the final outcome. Such coarse credit assignment can reinforce ineffective or redundant operator decisions. To address this, we present fine-grained credit assignment (FCA), which augments the architecture-level advantage with operation-level advantages for individual operator decisions. To this end, we propose state-conditioned relative advantage (SRA), which assigns operation-level rewards according to intermediate correctness and computational cost, and compares them under similar reasoning states, while placing larger weights on more similar states. We further introduce operator distribution projection (ODP), which maintains the diverse operator choices required for informative relative comparisons. Extensive experiments on diverse benchmarks using Qwen3-8B and GPT-4o-mini demonstrate the effectiveness and efficiency of FCA. We will release our code upon acceptance.

1 INTRODUCTION

Multi-agent systems (Hu et al., 2025; Zhang et al., 2025b;a; Yang et al., 2026) have emerged as a promising approach to extending the capabilities of large language models (LLMs) (Chowdhery et al., 2023; Touvron et al., 2023; OpenAI, 2023). They typically employ an agent architecture that organizes multiple LLM-based agents with distinct roles and defines how they interact throughout the reasoning process. The agent architecture is thus a key factor in determining both task performance and computational cost. Early approaches (Hong et al., 2024; Chen et al., 2024) rely on manually designed architectures, while recent methods, such as ADAS (Hu et al., 2025) and AFlow (Zhang et al., 2025b), automate the design process by searching for effective architectures. These methods, however, typically optimize a single architecture for a task or dataset and apply it uniformly to all queries, without considering different reasoning requirements across individual queries.

To mitigate this, recent methods, such as MaAS (Zhang et al., 2025a) and RAAS (Yang et al., 2026), exploit an agentic supernet to construct query-specific architectures. The supernet represents a distribution over possible agent architectures, where each layer contains a set of candidate agentic operators and corresponding selection probabilities. Given a query, a controller sequentially selects operators at each layer to construct an architecture. To optimize the controller, MaAS samples a single architecture for each query and uses its reward (*i.e.*, architecture-level reward) as the learning signal. RAAS instead samples multiple architectures for the same query and optimizes the controller according to their relative performance (*i.e.*, architecture-level advantage), similar to GRPO (Shao et al., 2024). Despite this difference, both methods use a single architecture-level signal to update all operator decisions within a sampled architecture, ignoring that individual operator decisions may contribute differently to the final outcome. For example, a successful architecture may reinforce redundant operations performed after obtaining a correct answer. Conversely, a failed architecture may penalize useful operations together with detrimental ones, without identifying which operator choices are responsible for the failure. This coarse-grained credit assignment thus makes it difficult to distinguish useful operations from unnecessary or detrimental ones, potentially leading to redundant reasoning and suboptimal architectures for individual queries.

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.

To overcome this limitation, we present fine-grained credit assignment (FCA) for agent architecture search. FCA augments the architecture-level advantage with an operation-level advantage, providing distinct learning signals for individual operator decisions. To estimate the operation-level advantage, we introduce state-conditioned relative advantage (SRA), which compares the reward of each operation with those of operations taken from similar states, represented by the intermediate output before each operation. This enables the controller to identify which operator decisions are more effective, reinforcing beneficial operations while suppressing redundant ones. We also propose operator distribution projection (ODP), which encourages the controller to sample diverse operators during training. This is important for SRA, since estimating informative operation-level advantages requires comparing diverse operator choices under similar reasoning states. ODP adjusts the controller distribution only when its diversity is insufficient, preserving the operator preferences learned by the controller. Extensive experiments demonstrate that FCA improves the trade-off between task performance and computational efficiency across diverse tasks and LLMs. Our main contributions are summarized as follows:

- We identify a limitation of architecture-level credit assignment in agent architecture search. Our empirical analysis shows that sharing a single architecture-level signal across all operations can provide inaccurate supervision, leading to redundant reasoning and degraded task performance.
- We propose FCA, which augments the architecture-level advantage with the operation-level advantage for fine-grained controller optimization. To estimate informative operation-level advantages, we introduce SRA, which compares operation-level rewards under semantically similar reasoning states. We further introduce ODP, which adjusts the controller distribution to ensure sufficient diversity for relative advantage estimation.
- We conduct extensive experiments on diverse tasks and LLMs, demonstrating that FCA consistently outperforms existing agent architecture search methods in terms of a trade-off between task performance and computational efficiency.

2 Related Work

In this section, we review representative works on group-based reinforcement learning for LLMs and agent architecture search.

2.1 Group-based reinforcement learning for LLMs

Reinforcement learning (RL) (Shao et al., 2024; Liu et al., 2025; Schulman et al., 2017) improves LLMs by optimizing their responses according to reward signals. A key component in this process is advantage estimation, which provides a relative learning signal for policy updates. Conventional methods such as PPO (Schulman et al., 2017) typically rely on a value model to estimate advantages. Recent approaches instead compare multiple responses sampled for the same query to derive relative learning signals without requiring a separate value model. A representative approach is GRPO (Shao et al., 2024), which computes the advantage of each response relative to the other responses in the same group. GRPO has been widely adopted for training reasoning models, including DeepSeek-R1 (DeepSeek-AI, 2025).

Several works further improve group-based RL for LLMs. Dr. GRPO (Liu et al., 2025) analyzes the optimization bias of GRPO caused by response-length normalization and modifies its objective accordingly. DAPO (Yu et al., 2025) introduces techniques such as asymmetric clipping, dynamic sampling, and token-level policy optimization to improve training stability and efficiency. While these methods mainly consider single-turn reasoning, GiGPO (Feng et al., 2025) extends group-based policy optimization to multi-turn agents. It assigns step-level credit by grouping the steps that share an identical environment state, assuming that such states recur across trajectories. In agent architecture search, however, the intermediate outputs of LLMs are rarely identical across sampled architectures, making such grouping impractical. FCA instead estimates the expected reward under each reasoning state from the rewards of the operations performed under similar states. This requires neither identical states nor a separate value model. FCA also maintains diverse operator choices with ODP, which are required for this estimation.

2.2 AGENT ARCHITECTURE SEARCH

LLM-based multi-agent systems solve a task by assigning different roles to multiple agents and coordinating their interactions. The way these agents are organized, including their roles, communication patterns, and execution procedures, defines an agent architecture. Early methods mainly design such architectures manually. MetaGPT (Hong et al., 2024), for example, assigns specialized software-development roles to agents and coordinates them through predefined standard operating procedures. AgentVerse (Chen et al., 2024) provides a framework for organizing groups of expert agents for collaborative task solving, while ChatDev (Qian et al., 2024) uses specialized software agents that communicate through a predefined chain of dialogues for design, coding, and testing. These methods demonstrate the benefit of multi-agent collaboration, but their architectures are largely specified by human designers.

Recent methods attempt to automate agent architecture design. ADAS (Hu et al., 2025) represents an entire agent system as code and uses a meta-agent to iteratively generate and evaluate new designs. AgentSquare (Shang et al., 2025) instead defines a modular search space consisting of planning, reasoning, tool-use, and memory components, and searches for effective combinations through module evolution and recombination. AFlow (Zhang et al., 2025b) represents an agent workflow as executable code specifying LLM calls and their connections, and iteratively modifies and evaluates the code with Monte Carlo tree search to find effective workflows. Although their search spaces and optimization strategies differ, these methods mainly search for an effective architecture at the task or dataset level.

More recently, MaAS (Zhang et al., 2025a) introduces an agentic supernet to construct query-specific architectures. Rather than searching for a single fixed design, it learns a distribution over candidate agentic operators and sequentially samples an architecture for each query. RAAS (Yang et al., 2026) further improves controller optimization by comparing multiple architectures sampled for the same query and exploiting their relative performance. These approaches, however, assign the same architecture-level signal to all operator decisions within a sampled architecture, regardless of their individual contributions to the final outcome. In contrast, FCA introduces operation-level advantages for individual operator decisions, enabling more fine-grained controller optimization.

3 METHOD

In this section, we first describe preliminaries on agent architecture search (Sec. 3.1) and analyze the limitation of architecture-level credit assignment (Sec. 3.2). We then introduce our FCA framework (Sec. 3.3).

3.1 PRELIMINARIES

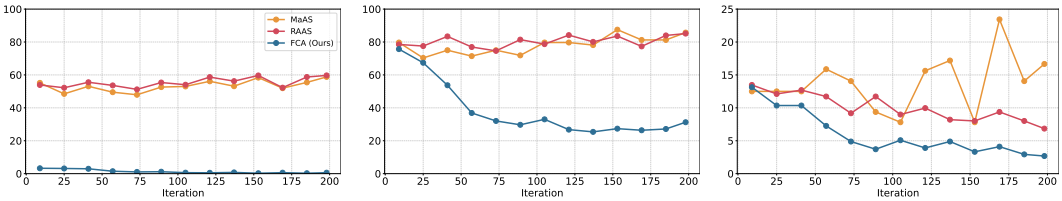
We consider the agent architecture search based on an agentic supernet (Zhang et al., 2025a; Yang et al., 2026). Let $\mathcal{O}$ denote the set of candidate agentic operators. Given a query q, a controller π_θ with parameters θ selects operators from $\mathcal{O}$ sequentially at each layer to construct an architecture. For the i-th sampled architecture, we denote by $o_{i,\ell} \in \mathcal{O}$ the operator selected at the ℓ-th layer, and by $o_{i,<\ell}$ the preceding operator choices. At each layer, the controller predicts a distribution over the candidate operators conditioned on the query and the preceding operator choices:

$$p_{i,\ell}(o) = \pi_\theta \left(o \mid q, o_{i,<\ell}\right), \tag{1}$$

where $p_{i,\ell}(o)$ is the probability of selecting the operator o at the ℓ-th layer of the i-th architecture. The architecture terminates when the early-stop operator $o^{\mathrm{ES}} \in \mathcal{O}$ is selected or the maximum number of layers is reached. For each sampled architecture, the reward is determined by its task performance and computational cost. Let t_i and c_i denote the task score and computational cost of the i-th architecture, respectively. The corresponding architecture-level reward is given by

$$R_i^{\mathrm{arch}} = t_i - \lambda c_i, \tag{2}$$

where λ controls the trade-off between task performance and computational cost.



(a) Operation-level credit error (b) Over-processing rate (c) Correct-then-wrong rate

Figure 1: Analysis of architecture-level credit assignment on Qwen3-8B (Yang et al., 2025) using the BBH benchmark (Suzgun et al., 2023). We compare MaAS (Zhang et al., 2025a), RAAS (Yang et al., 2026), and FCA in terms of (a) operation-level credit error, (b) over-processing rate, and (c) correct-then-wrong rate during training. FCA shows lower values across all three metrics than MaAS and RAAS. For more details, please refer to the main text. (Best viewed in color.)

3.2 Analysis on Architecture-Level Credit Assignment

Existing agent architecture search methods optimize individual operator decisions using architecture-level credit computed from the final outcome of a sampled architecture. Specifically, MaAS (Zhang et al., 2025a) samples a single architecture for each query and uses its architecture-level reward R_i^{arch} in Eq. (2) as the learning signal. RAAS (Yang et al., 2026) instead samples multiple architectures for the same query and computes the architecture-level advantage, which measures the relative performance of each architecture compared to the group average as follows:

$$A_i^{\text{arch}} = R_i^{\text{arch}} - \frac{1}{N}\sum_{j=0}^{N-1} R_j^{\text{arch}}, \tag{3}$$

where N is the number of architectures sampled for the same query. In both methods, the same signal is assigned to all operator decisions within a sampled architecture, regardless of their individual contributions to the final outcome. Such coarse supervision can be inaccurate, since individual operator decisions may have different effects on intermediate outputs and the final answer.

To analyze this, we train the controllers of MaAS (Zhang et al., 2025a) and RAAS (Yang et al., 2026) on BBH (Suzgun et al., 2023) with Qwen3-8B (Yang et al., 2025). Specifically, we extract an intermediate answer from the output of each operation, and compare it with the ground truth. We define an operation as beneficial if it changes an incorrect intermediate answer into a correct one, and detrimental in the opposite case. Based on this, we measure the following three metrics during training: (1) the operation-level credit error, the ratio of operations receiving credit with an incorrect sign, *i.e.*, positive credit despite degrading the answer or negative credit despite improving it; (2) the over-processing rate, the ratio of execution trajectories that continue applying operations after already obtaining a correct answer; and (3) the correct-then-wrong rate, the ratio of execution trajectories in which a correct intermediate answer becomes incorrect after subsequent operations. We show in Fig. 1 these metrics during training. We can see from Fig. 1(a) that MaAS and RAAS show large operation-level credit errors, since the same architecture-level credit is assigned to all operations regardless of their individual effects. This makes it difficult for the controller to distinguish beneficial operation choices from unnecessary ones. Consistent with this observation, both methods frequently continue processing after obtaining a correct answer, resulting in high over-processing rates (Fig. 1(b)). More importantly, redundant operations can also deteriorate an already correct answer, as shown by the correct-then-wrong rates in Fig. 1(c). These results show that architecture-level credit assignment can provide inaccurate supervision for individual operations, leading to unnecessary computation and potentially degrading task performance. This suggests the need for fine-grained credit assignment at the operation level.

3.3 Fine-Grained Credit Assignment (FCA)

Motivated by the above analysis, we introduce FCA, which augments the architecture-level advantage with operation-level advantages for individual operator decisions (Fig. 2). FCA estimates

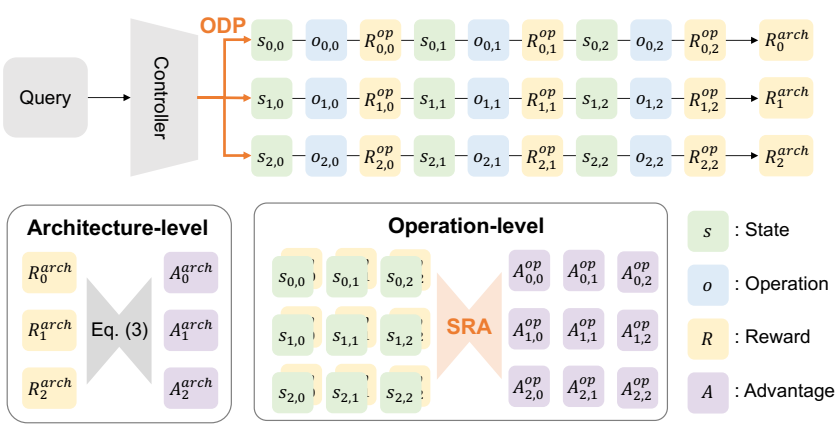


Figure 2: Overview of FCA. Given a query, the controller samples agent architectures using ODP to maintain sufficient operator diversity. Architecture-level advantage provides a shared learning signal based on the final architecture reward, whereas SRA provides operation-level advantage by comparing rewards under semantically similar reasoning states. FCA exploits both signals for fine-grained controller optimization.

operation-level advantages using SRA and maintains operator diversity using ODP. We combine the architecture- and operation-level advantages to optimize the controller.

Operation-level reward. To provide a learning signal for each operator decision, we use the intermediate output produced after applying the selected operator at each layer (see Appendix B for details). For the i-th architecture, let ℓ_i^* denote the first layer that produces a correct answer. We assign a task score of 1 to the operation at ℓ_i^*, and decay the scores of preceding operations by a factor of γ according to their distances from ℓ_i^*, since operations closer to ℓ_i^* contribute more directly to the answer. Operations after ℓ_i^* receive a score of zero, as further processing is unnecessary. The only exception is the early-stop operator, o^{ES}, which receives a score of 1 when terminating the architecture with a correct answer. Formally, the operation-level task score can be expressed as follows:

$$G_{i,\ell} = \begin{cases} \gamma^{\ell_i^* - \ell}, & \ell \le \ell_i^*, \\ 1, & \ell > \ell_i^* \text{ and } o_{i,\ell} = o^{\mathrm{ES}} \text{ with a correct answer}, \\ 0, & \text{otherwise}. \end{cases} \tag{4}$$

If an architecture does not produce a correct answer at any operation, we set $G_{i,\ell} = 0$ for all ℓ. We further account for the computational cost of each operation. Denoting by $c_{i,\ell}$ the cost of the operator $o_{i,\ell}$, we define the operation-level reward as follows:

$$R_{i,\ell}^{\mathrm{op}} = G_{i,\ell} - \lambda c_{i,\ell}, \tag{5}$$

where λ is the hyperparameter that balances task performance and computational cost. Note that we set $\lambda = 0$ in the architecture-level reward in Eq. (2) for FCA when computing A_i^{arch}, since the computational cost is already incorporated into the operation-level reward. This prevents the cost from being penalized twice.

State-conditioned relative advantage (SRA). Given the operation-level rewards, we estimate a relative advantage for each operator decision by comparing its reward with those of other decisions. The reward of an operation, however, depends not only on the selected operator but also on the preceding reasoning process. For example, an operation applied to an almost complete solution obtains a high reward regardless of which operator is selected. Comparing rewards across different reasoning states thus does not reflect the contribution of the selected operator. To address this problem, we introduce SRA, which compares each operation with the operations performed under similar reasoning states. Specifically, SRA compares each operation with those in the other sampled architectures, placing larger weights on operations with more similar reasoning states. The weighted

average of their rewards serves as a reference reward, and the operation-level advantage is defined as the difference between the reward of the operation and this reference.

We represent the reasoning state of an operator decision by the intermediate output before the decision.[1] For the ℓ-th layer of the i-th sampled architecture, let $s_{i,\ell}$ denote the intermediate output before selecting $o_{i,\ell}$. We encode $s_{i,\ell}$ using a pretrained MiniLM (Wang et al., 2020) encoder $f(\cdot)$, which is kept frozen during training, and define the similarity between two intermediate outputs $s_{i,\ell}$ and $s_{j,k}$ as follows:

$$\sigma_{(i,\ell),(j,k)} = \frac{f(s_{i,\ell})^\top f(s_{j,k})}{\|f(s_{i,\ell})\|_2 \|f(s_{j,k})\|_2}. \tag{6}$$

We convert the state similarity into a non-negative kernel weight so that only positively similar states contribute to the comparison:

$$\kappa_{(i,\ell),(j,k)} = \begin{cases} \max\left(\sigma_{(i,\ell),(j,k)}, 0\right), & j \neq i, \\ 0, & j = i. \end{cases} \tag{7}$$

We set the weights for all the operations in the i-th architecture to zero, since their rewards depend on $o_{i,\ell}$ through ℓ_i^*, and including them would bias the advantage of $o_{i,\ell}$ (Sutton & Barto, 2018). We then normalize the weights as follows:

$$\omega_{(i,\ell),(j,k)} = \frac{\kappa_{(i,\ell),(j,k)}}{\sum_{j'=0}^{N-1} \sum_{k'=0}^{T_{j'}-1} \kappa_{(i,\ell),(j',k')}}, \tag{8}$$

where N denotes the number of architectures sampled for the same query, and T_j is the number of operator decisions in the j-th architecture. The weight $\omega_{(i,\ell),(j,k)}$ indicates how much the operation at the k-th layer of the j-th architecture contributes to the comparison for the operation at the ℓ-th layer of the i-th architecture. Operations performed under more similar reasoning states thus receive larger weights. Using these weights, we aggregate the operation-level rewards obtained under similar reasoning states as follows:

$$B_{i,\ell}^{\mathrm{op}} = \sum_{j=0}^{N-1} \sum_{k=0}^{T_j-1} \omega_{(i,\ell),(j,k)} R_{j,k}^{\mathrm{op}}, \tag{9}$$

where $B_{i,\ell}^{\mathrm{op}}$ estimates the expected reward of operator decisions made under a similar reasoning state. The operation-level advantage is then computed relative to this reference reward:

$$A_{i,\ell}^{\mathrm{op}} = R_{i,\ell}^{\mathrm{op}} - B_{i,\ell}^{\mathrm{op}}. \tag{10}$$

A positive advantage indicates that the operator decision achieves a higher reward than expected under similar reasoning states, and vice versa.

Operator distribution projection (ODP). SRA relies on comparisons among different operator choices under similar reasoning states. As training proceeds, however, the controller distribution can become concentrated on a small number of operators (orange line in Fig. 3), since the operators that receive positive advantages are sampled more frequently and thus reinforced repeatedly (Cui et al., 2025). This reduces the alternative operator choices available for SRA, limiting relative advantage estimation. To address this problem, we introduce ODP, which projects the controller distribution onto the nearest distribution satisfying a diversity constraint.

[1] We use the intermediate output rather than the preceding operator choices, since different operator choices can lead to similar intermediate outputs.

Let $\mathbf{p}_{i,\ell} = [p_{i,\ell}(o)]_{o\in\mathcal{O}}$ denote the probability vector of the controller distribution, and $\tilde{\mathbf{p}}_{i,\ell}$ the sampling distribution after projection. We measure the concentration of the sampling distribution using its squared ℓ_2 norm:

$$\|\tilde{\mathbf{p}}_{i,\ell}\|_2^2 = \sum_{o\in\mathcal{O}} \tilde{p}_{i,\ell}(o)^2. \tag{11}$$

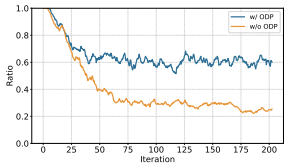


Figure 3: Ratio of distinct operators selected by the controller at each training iteration on BBH (Suzgun et al., 2023) with Qwen3-8B (Yang et al., 2025). A higher value indicates more diverse operator sampling.

Since two independent draws both select the operator o with probability $\tilde{p}_{i,\ell}(o)^2$, this quantity is the probability that they select the same operator, where a smaller value indicates more diverse operator sampling. ODP constrains this concentration while keeping the sampling distribution close to the original controller distribution:

$$\begin{aligned} \tilde{\mathbf{p}}^*_{i,\ell} = \arg\min_{\tilde{\mathbf{p}}_{i,\ell}} \quad & \frac{1}{2}\|\tilde{\mathbf{p}}_{i,\ell} - \mathbf{p}_{i,\ell}\|_2^2 \\ \text{s.t.} \quad & \|\tilde{\mathbf{p}}_{i,\ell}\|_2^2 \le \tau_p. \end{aligned} \tag{12}$$

Here, τ_p controls the degree of operator diversity, with a smaller value imposing a stronger diversity constraint. Let $\mathbf{u}$ denote the uniform distribution over the candidate operators, which has the smallest ℓ_2 norm among all distributions. The constraint thus bounds the distance from $\mathbf{u}$, and the projection moves the controller distribution toward $\mathbf{u}$ when it violates the constraint, yielding the following closed-form solution:

$$\tilde{\mathbf{p}}^*_{i,\ell} = \mathbf{u} + \frac{\sqrt{\tau_p - 1/|\mathcal{O}|}}{\|\mathbf{p}_{i,\ell} - \mathbf{u}\|_2}(\mathbf{p}_{i,\ell} - \mathbf{u}). \tag{13}$$

Otherwise, we retain the original controller distribution, preserving its learned operator preferences when sufficient diversity is already maintained. We provide the detailed derivation in Appendix A.

Controller optimization. We optimize the controller using both architecture- and operation-level advantages, allowing the update to reflect both the overall architecture outcome and the contribution of individual operator decisions. Specifically, we define the loss for each operator decision as follows:

$$L^{\text{FCA}}_{i,\ell} = -r_{i,\ell}\left(A^{\text{arch}}_i + A^{\text{op}}_{i,\ell}\right)\log p_{i,\ell}(o_{i,\ell}), \tag{14}$$

where $r_{i,\ell}$ is the ratio used for importance sampling (Degris et al., 2012) to account for the change in sampling probability caused by ODP:

$$r_{i,\ell} = \frac{p_{i,\ell}(o_{i,\ell})}{\tilde{p}^*_{i,\ell}(o_{i,\ell})}. \tag{15}$$

Note that no gradient is propagated through $r_{i,\ell}$ (*i.e.*, stop-gradient). Finally, we aggregate the losses over all operator decisions to update the controller as follows:

$$\mathcal{L}_{\text{FCA}} = \frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{T_i}\sum_{\ell=0}^{T_i-1} L^{\text{FCA}}_{i,\ell}. \tag{16}$$

4 Experiments

In this section, we first describe the experimental setup (Sec. 4.1). We then provide quantitative comparisons of FCA with existing agent architecture search methods across different benchmarks and backbone LLMs (Sec. 4.2), and present detailed analyses of FCA (Sec. 4.3).

Table 1: Quantitative results on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q).

Method	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
Vanilla	49.0%	789	88.8%	293	84.0%	282	68.0%	113	75.1%	408
CoT (Wei et al., 2022)	49.6%	819	89.1%	313	81.7%	438	68.6%	262	76.5%	852
Self-Consistency (Wang et al., 2023)	54.1%	2401	89.8%	932	83.2%	1305	69.5%	774	80.5%	2517
MaAS (Zhang et al., 2025a)	58.8%	1495	91.8%	2815	86.3%	3608	71.0%	1640	75.8%	922
RAAS (Yang et al., 2026)	59.7%	1468	92.7%	2515	86.3%	1592	78.0%	804	76.5%	1378
FCA (Ours)	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706

Table 2: Quantitative results on GPT-4o-mini (OpenAI, 2024). We report task performance (%) and the average number of tokens per query (Tok./Q).

Method	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
Vanilla	47.3%	835	87.4%	356	87.0%	604	69.5%	382	73.6%	363
CoT (Wei et al., 2022)	44.2%	986	87.7%	548	87.8%	494	69.5%	330	75.3%	654
Self-Consistency (Wang et al., 2023)	50.8%	2814	88.4%	1631	88.5%	1402	71.6%	986	79.5%	1932
MaAS (Zhang et al., 2025a)	50.0%	3096	92.5%	3005	85.5%	528	79.2%	616	77.3%	1304
RAAS (Yang et al., 2026)	50.4%	2733	93.1%	2832	86.3%	287	79.5%	576	78.3%	923
FCA (Ours)	52.5%	2171	93.6%	2056	88.5%	298	80.4%	506	81.5%	710

4.1 Experimental Setup

We evaluate FCA using two backbone LLMs, Qwen3-8B (Yang et al., 2025) and GPT-4o-mini (OpenAI, 2024), on five benchmarks covering mathematical reasoning (MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021)), code generation (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)), and general reasoning (BBH (Suzgun et al., 2023)). We report accuracy for MATH, GSM8K, and BBH, and Pass@1 for HumanEval and MBPP according to the standard evaluation protocols. For comparison, we reproduce MaAS (Zhang et al., 2025a) and RAAS (Yang et al., 2026) based on their official implementations under the same experimental setting, using the same candidate operator set for all methods.

Following prior works (Zhang et al., 2025a; Yang et al., 2026), we use a controller consisting of a MiniLM (Wang et al., 2020) encoder and multiple linear layers. We sample eight agent architectures for each query, each consisting of at most four layers. We use the same values of γ and τ_p across all benchmarks, setting them to 0.5 and 0.5, respectively. For both backbone LLMs, we measure computational cost using token consumption, and set λ to 5e-5. Further implementation details are provided in Appendix B.

4.2 Results

We compare in Tables 1 and 2 FCA with MaAS (Zhang et al., 2025a) and RAAS (Yang et al., 2026) using Qwen3-8B (Yang et al., 2025) and GPT-4o-mini (OpenAI, 2024), respectively. We summarize the findings as follows: (1) FCA achieves a better trade-off between task performance and computational cost than MaAS and RAAS. For example, on MATH with GPT-4o-mini, FCA improves the performance of RAAS from 50.4% to 52.5%, while reducing the number of tokens per query from 2733 to 2171. This suggests that leveraging operation-level credit in FCA helps the controller reinforce useful operator choices while avoiding redundant computation. (2) The improvement is particularly pronounced on BBH. Specifically, on Qwen3-8B, FCA improves the performance of RAAS from 76.5% to 81.2%, while reducing the number of tokens per query from 1378 to 706. A plausible reason is that BBH covers diverse reasoning tasks with different reasoning requirements, making query-specific architecture selection particularly important. FCA better identifies such architectures through fine-grained credit assignment. (3) FCA consistently improves a trade-off between performance and efficiency over existing agent architecture search methods (Zhang et al., 2025a;

Table 3: Ablation study of FCA on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q). The baseline denotes the model without SRA and ODP, using the architecture-level advantage only.

Method	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
Baseline	59.3%	1436	92.0%	2611	85.5%	1718	77.7%	870	76.5%	1874
+ SRA	60.5%	1342	93.2%	2449	87.0%	1668	80.1%	782	78.3%	824
+ SRA + ODP (FCA)	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706

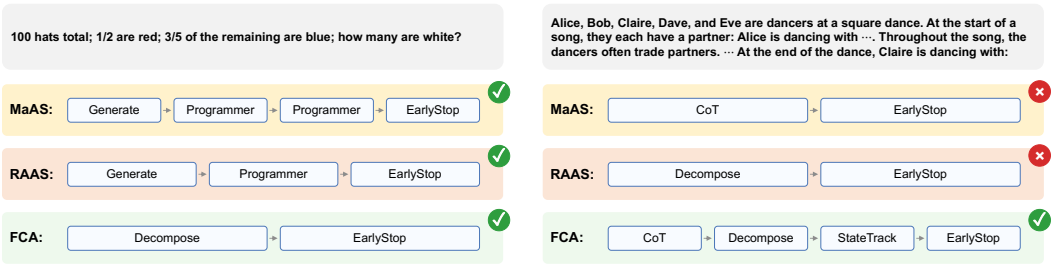


Figure 4: Qualitative comparison of agent architectures constructed by MaAS (Zhang et al., 2025a), RAAS (Yang et al., 2026), and FCA for two queries. For the simpler query (left), all methods produce the correct answer, while FCA constructs a more compact architecture with fewer operators. For the more challenging query (right), FCA selects additional operators to perform the required reasoning and produces the correct answer. See Table 4 in the Appendix for descriptions of the operators.

Yang et al., 2026) across all five benchmarks with both Qwen3-8B and GPT-4o-mini. This demonstrates the generalization ability of FCA across different benchmarks and backbone LLMs.

4.3 Analysis

Ablation study. We show in Table 3 an ablation study of SRA and ODP. From the first two rows, we can see that SRA improves the performance of the baseline. In particular, on BBH, SRA improves the performance from 76.5% to 78.3%, while reducing the number of tokens per query from 1874 to 824. This suggests that the operation-level advantages allow the controller to suppress redundant operations, which cannot be identified by the architecture-level advantage alone. The last row shows that ODP further provides a better trade-off between task performance and computational cost on all the benchmarks, indicating that maintaining diverse operator choices provides informative comparisons for SRA.

Qualitative analysis. We show in Fig. 4 agent architectures searched by MaAS (Zhang et al., 2025a), RAAS (Yang et al., 2026), and FCA for two queries. For the simpler query (left), we can see that all methods produce the correct answer, while FCA constructs a more compact architecture with fewer operators. This suggests that fine-grained credit assignment helps avoid redundant operator choices when additional reasoning is unnecessary. In contrast, for the more challenging query (right), FCA uses additional operators to obtain the correct answer. This indicates that FCA does not simply favor shorter architectures, but adjusts the computation according to the reasoning required for each query.

5 Conclusion

We have presented FCA for agent architecture search, addressing the limitation of architecture-level credit that assigns the same learning signal to all operator decisions within an architecture. Specifically, we have introduced SRA to estimate operation-level advantages through reward comparisons under similar reasoning states, and ODP to maintain diverse operator choices for these comparisons. We have demonstrated that our approach improves a trade-off between task performance and computational efficiency over existing agent architecture search methods, across different benchmarks and backbone LLMs.

REFERENCES

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. *arXiv*, 2021.

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

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*, 2024.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling language modeling with pathways. *JMLR*, 2023.

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. *arXiv*, 2021.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv*, 2025.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv*, 2025.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *ICML*, 2012.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *NeurIPS*, 2025.

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 *NeurIPS Datasets and Benchmarks*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *ICLR*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *COLM*, 2025.

OpenAI. GPT-4 technical report. *arXiv*, 2023.

OpenAI. GPT-4o mini: advancing cost-efficient intelligence. OpenAI, 2024.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *ACL*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic LLM agent search in modular design space. In *ICLR*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv*, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of ACL*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv*, 2023.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *NeurIPS*, 2020.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ICLR*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv*, 2025.

Jiayi Yang, Guancheng Wan, Man Zhang, and Mang Ye. RAAS: LLM agentic system architecture search with GRPO. In *CVPR*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *NeurIPS*, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *ICML*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *ICLR*, 2025b.

A Derivation of Operator Distribution Projection

We provide the derivation of the closed-form solution of ODP in Eq. (13). We omit the architecture and layer indices, and denote by $\mathbf{p}$ and $\tilde{\mathbf{p}}$ the controller distribution and the sampling distribution, respectively. Note that $\tilde{\mathbf{p}}$ in Eq. (12) is a distribution, and thus satisfies the nonnegativity and sum-to-one constraints in addition to the diversity constraint.

Equivalent ball constraint. Let $\mathbf{u}$ denote the uniform distribution over $\mathcal{O}$. For any $\tilde{\mathbf{p}}$ whose elements sum to one, we have

$$\begin{aligned}\|\tilde{\mathbf{p}} - \mathbf{u}\|_2^2 &= \sum_{o\in\mathcal{O}} \tilde{p}(o)^2 - \frac{2}{|\mathcal{O}|}\sum_{o\in\mathcal{O}} \tilde{p}(o) + \sum_{o\in\mathcal{O}} \frac{1}{|\mathcal{O}|^2} \\ &= \|\tilde{\mathbf{p}}\|_2^2 - \frac{1}{|\mathcal{O}|}. \end{aligned} \tag{17}$$

The diversity constraint in Eq. (12) is thus equivalent to

$$\|\tilde{\mathbf{p}} - \mathbf{u}\|_2 \le r_p, \qquad r_p = \sqrt{\tau_p - \frac{1}{|\mathcal{O}|}}. \tag{18}$$

Eq. (12) therefore finds the distribution closest to $\mathbf{p}$ within the ball centered at $\mathbf{u}$ with radius r_p.

Closed-form solution. If $\mathbf{p}$ already satisfies the diversity constraint, $\mathbf{p}$ itself is the solution of Eq. (12), and ODP leaves the controller distribution unchanged. Otherwise, $\mathbf{p}$ lies outside the ball, that is,

$$r_p < \|\mathbf{p} - \mathbf{u}\|_2 . \tag{19}$$

We first find the closest point to $\mathbf{p}$ in the ball, and then show that it is a distribution. For any $\tilde{\mathbf{p}}$ in the ball, the triangle inequality gives

$$\|\tilde{\mathbf{p}} - \mathbf{p}\|_2 \ge \|\mathbf{p} - \mathbf{u}\|_2 - \|\tilde{\mathbf{p}} - \mathbf{u}\|_2 \ge \|\mathbf{p} - \mathbf{u}\|_2 - r_p. \tag{20}$$

Intuitively, the closest point lies in the direction from $\mathbf{u}$ toward $\mathbf{p}$. We thus consider the point at distance r_p from $\mathbf{u}$ in this direction:

$$\tilde{\mathbf{p}}^* = \mathbf{u} + \frac{r_p}{\|\mathbf{p} - \mathbf{u}\|_2}(\mathbf{p} - \mathbf{u}). \tag{21}$$

Eq. (21) can be rewritten as follows:

$$\tilde{\mathbf{p}}^* = (1 - \epsilon^*)\,\mathbf{p} + \epsilon^*\,\mathbf{u}, \qquad \epsilon^* = 1 - \frac{r_p}{\|\mathbf{p} - \mathbf{u}\|_2}, \tag{22}$$

where ϵ^* lies between zero and one by Eq. (19). The distance from $\tilde{\mathbf{p}}^*$ to $\mathbf{p}$ is then given by

$$\|\tilde{\mathbf{p}}^* - \mathbf{p}\|_2 = \|\epsilon^*(\mathbf{u} - \mathbf{p})\|_2 = \epsilon^* \|\mathbf{p} - \mathbf{u}\|_2 = \|\mathbf{p} - \mathbf{u}\|_2 - r_p, \tag{23}$$

which equals the lower bound in Eq. (20). The point $\tilde{\mathbf{p}}^*$ is thus the closest point to $\mathbf{p}$ in the ball. Since ϵ^* lies between zero and one, and both $\mathbf{p}$ and $\mathbf{u}$ are distributions, $\tilde{\mathbf{p}}^*$ also satisfies the nonnegativity and sum-to-one constraints, and is therefore the solution of Eq. (12). Substituting r_p in Eq. (18) into Eq. (21), we obtain the closed-form solution in Eq. (13):

$$\tilde{\mathbf{p}}^* = \mathbf{u} + \frac{\sqrt{\tau_p - 1/|\mathcal{O}|}}{\|\mathbf{p} - \mathbf{u}\|_2}(\mathbf{p} - \mathbf{u}). \tag{24}$$

Table 4: Agentic operators used in our search space. We report a summary of the corresponding prompt for each operator.

Operator	Prompt
Generate	Give *only* the final answer (or only the code block), with no reasoning or explanation.
CoT	Work through the problem step by step, using the kind of reasoning it requires (logic, ordering, symbol tracking, language, or arithmetic). For code: state the I/O spec and the approach, then implement.
SelfConsistency	Sample $K{=}3$ chain-of-thought solutions at temperature 0.6; an LLM vote then picks the answer that appears most often.
Programmer	Write a self-contained `solve()` function that computes the exact answer; execute it; on an error, retry with the traceback as feedback (up to 3 attempts).
Decompose	(1) List the minimal ordered sub-problems; (2) solve them in order, reusing earlier results; (3) combine the sub-results into the final answer.
DeepReason	Stage 1: describe the algorithm, the I/O specification, and the edge cases in plain English, with no code. Stage 2: write the implementation conditioned on the stage-1 analysis.
StateTrack	Write the initial state, then process each event one at a time, rewriting the *full* updated state after each; read the answer off the final state.
RuleCheck	State the explicit rule(s) that decide the problem, check each option against them one by one, and choose the option that satisfies them.
OptionCompare	Name the one criterion that separates the correct option from the rest, evaluate every option on it side by side, and eliminate those that fail.
SelfRefine	Check the draft for concrete errors. If it is correct, keep its final answer unchanged; fix only genuine mistakes.
Verify	Assume the draft is wrong until proven otherwise: re-derive the key steps independently, test the answer on counterexamples and edge cases, and fix any error found.
EarlyStop	No LLM call; the current draft becomes the final answer.

B Additional Implementation Details

For HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), GSM8K (Cobbe et al., 2021), and MATH (Hendrycks et al., 2021), we follow the standard protocol (Zhang et al., 2025a; Yang et al., 2026) and use the data splits provided by AFlow (Zhang et al., 2025b). We train the controller for four epochs on these benchmarks. We additionally evaluate on BBH (Suzgun et al., 2023), which consists of 27 subtasks covering diverse reasoning problems. Since BBH provides a single evaluation split only, we construct training and test sets from it. Specifically, we randomly sample 45 questions from each subtask, and split them into 30 training and 15 test questions. This results in 810 training and 405 test questions in total. We train the controller for one epoch on BBH.

To compute the operation-level task score, we extract an intermediate answer from the output of each operation and verify it against the ground truth provided in the training set, using the same evaluation procedure as for the final answer. For MATH, GSM8K, and BBH, we extract the answer from the output and compare it with the ground-truth answer. For HumanEval and MBPP, we extract the code block from the output and execute it on the ground-truth test cases, regarding it as correct if it passes all the tests.

We use the same candidate operator set for all methods to ensure a fair comparison, and summarize the operators in Table 4. The released implementations of MaAS (Zhang et al., 2025a) and RAAS (Yang et al., 2026) specify operator configurations only for a subset of benchmarks. We thus reproduce both methods under our common search space. Their implementations also include several benchmark-specific procedures. For example, in code generation tasks, a generated solution is always tested and, if it fails, refinement is subsequently applied. We omit such predefined procedures so that these operator choices remain part of the search process rather than being fixed for specific benchmarks.

The baseline in Table 3 uses the architecture-level advantage only, without SRA and ODP. It differs from RAAS (Yang et al., 2026) in that it does not use multi-trial assessment synthesis (MTAS), which executes each sampled architecture multiple times and aggregates the results. We exclude MTAS from both the baseline and FCA to isolate the effect of fine-grained credit assignment.

C More Analyses

SRA. SRA estimates operation-level advantages by softly weighting operator rewards according to the similarity between reasoning states. We compare in Table 5 SRA with hard grouping, which

Table 5: Quantitative comparison between hard grouping and SRA on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q) for different similarity thresholds δ.

Method	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
Hard grouping ($\delta = 0.3$)	59.5%	1458	93.3%	2472	85.5%	1734	79.5%	782	75.8%	1064
Hard grouping ($\delta = 0.5$)	61.1%	1318	92.8%	2625	87.0%	1661	78.6%	856	78.3%	904
Hard grouping ($\delta = 0.7$)	60.1%	1396	93.3%	2510	86.3%	1812	80.1%	738	79.0%	826
Hard grouping ($\delta = 0.9$)	58.6%	1612	91.9%	2814	88.5%	1594	78.0%	914	76.8%	1247
SRA	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706

Table 6: Quantitative comparison between ϵ-greedy exploration and ODP on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q) for different values of ϵ.

Method	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
ϵ-greedy ($\epsilon = 0.05$)	60.3%	1307	93.9%	2461	87.0%	1604	79.8%	737	78.3%	812
ϵ-greedy ($\epsilon = 0.10$)	60.5%	1288	93.3%	2502	88.5%	1667	80.1%	776	78.5%	759
ϵ-greedy ($\epsilon = 0.20$)	60.9%	1396	93.6%	2587	86.3%	1741	79.8%	821	79.3%	936
ϵ-greedy ($\epsilon = 0.30$)	59.9%	1514	93.0%	2668	85.5%	1793	80.6%	714	77.8%	1016
ODP	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706

Table 7: Sensitivity analysis of the discount factor γ on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q). We fix $\tau_p = 0.5$ for all experiments.

γ	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
0.25	60.3%	1298	93.5%	2319	87.0%	1694	79.8%	728	79.8%	764
0.50	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706
0.75	61.1%	1276	93.4%	2461	88.5%	1586	80.6%	789	80.5%	681
1.00	60.5%	1379	93.7%	2415	87.0%	1722	80.1%	739	80.0%	832

Table 8: Sensitivity analysis of the diversity constraint τ_p on Qwen3-8B (Yang et al., 2025). We report task performance (%) and the average number of tokens per query (Tok./Q). We fix $\gamma = 0.5$ for all experiments.

τ_p	MATH		GSM8K		HumanEval		MBPP		BBH	
	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q	Perf.	Tok./Q
0.30	60.7%	1284	93.9%	2448	87.0%	1607	80.6%	773	80.5%	741
0.40	60.9%	1229	93.6%	2362	87.8%	1671	80.1%	736	80.7%	692
0.50	60.9%	1245	93.8%	2388	87.8%	1638	80.9%	751	81.2%	706
0.60	60.7%	1261	93.5%	2327	87.0%	1619	80.4%	765	79.8%	758

instead treats reasoning states whose similarity exceeds a threshold as the same group and assigns them equal weights. Specifically, we replace the similarity weight in SRA with

$$\kappa^{\text{hard}}_{(i,\ell),(j,k)} = \mathbb{I}\left[\sigma_{(i,\ell),(j,k)} \geq \delta\right], \tag{25}$$

where δ is the similarity threshold. We can see that the best threshold for hard grouping varies across benchmarks, since the similarities between reasoning states are distributed differently across tasks.

A fixed threshold thus groups too many or too few operations on some benchmarks, requiring the threshold to be tuned for each benchmark. In contrast, SRA provides comparable or better performance on most benchmarks without per-benchmark tuning, by assigning larger weights to more similar states.

ODP. ODP maintains operator diversity by projecting the controller distribution when it becomes overly concentrated, while retaining the original distribution otherwise. A simple alternative is ϵ-greedy exploration (Sutton & Barto, 2018), which introduces random operator choices by mixing the controller distribution with the uniform distribution as follows:

$$\tilde{p}^{\epsilon}_{i,\ell}(o) = (1-\epsilon)p_{i,\ell}(o) + \frac{\epsilon}{|\mathcal{O}|}, \qquad o \in \mathcal{O}, \tag{26}$$

where ϵ controls the degree of random exploration. We compare in Table 6 ODP with ϵ-greedy using different values of ϵ. We can see that ODP outperforms ϵ-greedy in most cases. Since ϵ is fixed throughout training, ϵ-greedy introduces the same amount of random exploration regardless of the current distribution. It can thus be excessive at early iterations, where the distribution is still diverse, and insufficient at later iterations, where the distribution becomes concentrated. In contrast, ODP adjusts the distribution only when its diversity is insufficient, and by the minimum amount required. ODP thus maintains operator diversity more effectively, providing robust performance across all the benchmarks without tuning.

Analysis on γ. We show in Table 7 the results of FCA with different values of the discount factor γ in Eq. (4). We can see that FCA provides comparable performance across different values of γ, indicating that it is robust to the choice of γ. Note that $\gamma = 1$ assigns the same score to all the operations before ℓ_i^*, regardless of their distances from ℓ_i^*. This degrades the performance on all the benchmarks compared to $\gamma = 0.5$, while increasing the number of tokens per query in most cases, suggesting that decaying the scores of the operations far from the correct answer is beneficial.

Analysis of τ_p. We show in Table 8 the results of FCA with different values of τ_p. We can see that the performance changes only marginally on all the benchmarks except BBH, indicating that FCA is robust to the choice of τ_p. On BBH, the performance is more sensitive to τ_p, and the largest value (*i.e.*, $\tau_p = 0.6$) degrades the performance the most. This is because a large value of τ_p weakens the diversity constraint, while BBH consists of diverse reasoning tasks that require different operator choices, making diverse operator sampling particularly important.

Training cost. FCA introduces two additional components during training, that is, encoding the intermediate outputs for SRA and projecting the controller distribution for ODP. We measure their computational overhead on MATH (Hendrycks et al., 2021) with Qwen3-8B (Yang et al., 2025). Sampling agent architectures from the backbone LLM takes 196 seconds per batch, requiring 0.65 PFLOPs. In contrast, encoding the intermediate outputs with the MiniLM (Wang et al., 2020) encoder takes 0.24 seconds and 0.74 TFLOPs, corresponding to roughly 0.1% of the backbone LLM. Estimating the operation-level advantages in SRA and projecting the controller distribution in ODP take 5 and 2.5 milliseconds, respectively. Note that SRA and ODP are used only for optimizing the controller, and they introduce no additional computation at test time. This indicates that fine-grained credit assignment improves the controller with a negligible computational overhead.

Limitation. SRA measures the similarity between reasoning states using a frozen MiniLM encoder, which may not fully capture fine-grained differences between intermediate outputs, *e.g.*, two solutions that differ only in the final answer. Adopting task-specific or learned state representations could provide more accurate comparisons. Following existing methods based on an agentic supernet (Zhang et al., 2025a; Yang et al., 2026), FCA also constructs an architecture as a sequence of operators, where each operator processes the output of the preceding one. It thus does not represent richer interactions among operators, such as exchanging intermediate results or revisiting earlier operations. We believe that addressing these limitations is a promising direction for future work.

D AI USE STATEMENT

We used generative AI tools solely for language editing to improve the readability of the manuscript, including grammar, wording, and sentence structure. All AI-assisted edits were reviewed by the authors, who take full responsibility for the final content of the paper.