SPRUNE: TRAINING-FREE CREDIT-AND-ACT FOR AGENTIC WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic workflows are graphs of operators, prompts, and edges that pass drafts between operators. Automated methods improve such workflows mostly by searching over candidate graphs and returning one, which leaves open how much each component contributes and discards components that later tasks may still need. A growing number of methods instead tie actions to per-component credit, but this coupling has not been named or treated as a design space of its own. We call it credit-and-act (CRACT) and describe it by three parts: a credit estimator, an accumulation rule, and an action set. We then present SPRUNE, a training-free CRACT method that values operators, prompts, and edges jointly with Owen values over three a priori unions, estimated by permutation sampling and smoothed into running scores by an exponential moving average. The running score archives weak components without deleting them, restores them when their score recovers, and, in a second variant, selects prompts to rewrite and decides whether operators proposed by a language model stay. Because every action reads this score, the attribution that SPRUNE reports is the signal behind its decisions. Across six benchmarks in code generation, mathematical reasoning, and question answering and three open-weight backbones, SPRUNE is competitive with seven workflow-search methods that share its backbone. An analysis of the credit recorded in nearly a thousand runs, most of them from the hyperparameter search, shows that operators carry most of the credit, that credit rankings are stable across seeds on five of the six benchmarks, and that most operators ending with the highest credit had been archived and restored along the way. Code is available at `https://anonymous.4open.science/r/sprune-iclr`.

1 INTRODUCTION

Agentic systems are usually built as a harness: a fixed scaffold of operators, prompts, and orchestration connections around a language model. In the graph representation, these connections are called edges, and they pass a draft from one operator to the next. Orchestration frameworks have made such a scaffold a standard building block for agent systems (LangChain, 2024; Moura, 2023), and continuously running self-modifying agents have turned it into a standalone product (Steinberger & OpenClaw contributors, 2025; Razzhigaev et al., 2026; Nous Research, 2026; DeepSeek-AI, 2026). Harness engineering has therefore become an explicit practice (Weng, 2026). That practice is still largely done by hand, even though automated workflow-search methods now search over the same three parts in order to improve performance.

These methods search over agents, graphs, or modules and commit to the result: DyLAN drops low-rated agents (Liu et al., 2023c), GPTSwarm learns which edges to keep and rewrites prompts (Zhuge et al., 2024), ADAS writes new agents in code (Hu et al., 2025), AgentSquare recombines modules (Shang et al., 2025), AFlow searches over workflows with Monte Carlo tree search (Zhang et al., 2025c), MaAS samples a subgraph per query from a trained supernet (Zhang et al., 2025a), and HarnessX accepts or rejects typed edits proposed from execution traces (Chen et al., 2026). Across these methods, search changes the graph, yet operators, prompts, and candidate edges are not attributed together. Closer methods still miss that joint view. HiveMind (Xia et al., 2026) computes training-free DAG-Shapley values over agents, but it only rewrites the weakest agent's prompt and leaves the topology fixed, and HarnessDev (Wu et al., 2026) reports that unconstrained whole-harness rewriting is unstable because the rewrite cannot separate which component caused 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.

drop. Moreover, none of the search-based methods archives a removed part, so the part cannot be restored if later tasks still need it.

Several of these methods nevertheless tie an action to a contribution estimate. DyLAN keeps the agents with the highest importance scores, HiveMind rewrites the prompt of the agent with the lowest Shapley value, and recent prompt optimizers turn per-agent credit into prompt updates (Zehle, 2026; Li et al., 2026). This coupling recurs from paper to paper, yet it has not been named or treated as a design space: the credit estimator, whether credit is accumulated, and the actions it may trigger are chosen implicitly. We call this pattern **credit-and-act** (CRACT) and describe it by three interchangeable parts: a credit estimator, an accumulation rule, and an action set.

We therefore propose SPRUNE, a training-free CRACT method whose Owen credit (Owen, 1977) covers operators, prompts, and candidate edges jointly. Our contributions are as follows:

- **Credit-and-act (CRACT):** We name the loop in which per-component credit is estimated, accumulated, and used as the only signal for edits to a workflow, and we describe it by a credit estimator, an accumulation rule, and an action set.
- **SPRUNE:** Owen credit over three unions, estimated by permutation sampling, an exponential moving average (EMA) as the running score, and a reversible action set of archive, restore, prompt rewrite, and operator expansion.
- **Interpretability by construction:** The running score that drives every action is also the reported attribution.
- **Evaluation and analysis:** Seven published baselines on six benchmarks and three open-weight backbones, and an analysis of the credit recorded in nearly a thousand runs. Code is available at `https://anonymous.4open.science/r/sprune-iclr`.

2 RELATED WORK

Automated Workflow Search. Automated workflow search casts an agent system as a directed graph and looks for a higher-performing topology and prompts: GPTSwarm (Zhuge et al., 2024) trains a REINFORCE policy over edge weights, MaAS (Zhang et al., 2025a) keeps a supernet and draws a query-specific subgraph, AgentSquare (Shang et al., 2025) evolves planning, reasoning, tool, and memory blocks, HarnessX (Chen et al., 2026) applies typed edits from execution traces, AFlow (Zhang et al., 2025c) grows workflows by Monte Carlo tree search, ADAS (Hu et al., 2025) synthesizes agents as programs, and DyLAN (Liu et al., 2023c) drops low-rated agents during a task. Related work chooses the team and its communication topology (Zhang et al., 2024a; 2026; Wang et al., 2025). These methods return a stronger graph or a cheaper team but leave open which operators, prompts, and edges drive the outcome (Appendix N discusses each in more detail).

Agent-Component Attribution. Shapley values (Shapley et al., 1953; Lundberg & Lee, 2017) also attribute the score of a workflow to its parts. AgentsMatter (Lu et al., 2026a) and CapaBench (Yang et al., 2025a) estimate removal-based Shapley values of agents, ShapleyFlow (Yang et al., 2025b) ranks modules to upgrade, and Prompt Valuation (Liu et al., 2023a) and ADOPT (Zhao et al., 2025) value prompts. Some methods act on such credit: CANTANTE (Zehle, 2026) and temporal–structural credit assignment (Li et al., 2026) turn it into prompt updates, and HiveMind (Xia et al., 2026) rewrites the prompt of the weakest agent. Where attribution drives an action, it credits one component type and acts on agents or prompts; no method jointly credits operators, prompts, and candidate edges and acts on the structure they form.

Table 1 compares the seven workflow-search methods that we benchmark along training-free status, operator expansion, pruning, prompt rewriting, and interpretability. To the best of our knowledge, none covers the full set of capabilities that SPRUNE does.

3 PRELIMINARY

In this section we first define the workflow credit-assignment problem in Section 3.1 and then give an overview of SPRUNE, its scope, and its operating choices in Section 3.2. Figure 1 illustrates the three player types that the rest of the paper builds on.

Table 1: Capability coverage of related workflow methods.

Method	Training-free	Expand	Prune	Rewrite	Interpretability
AFlow (Zhang et al., 2025c)	✓	✓	×	✓	none
ADAS (Hu et al., 2025)	✓	✓	×	×	none
GPTSwarm (Zhuge et al., 2024)	×	×	✓	✓	none
AgentSquare (Shang et al., 2025)	partial	✓	×	✓	none
DyLAN (Liu et al., 2023c)	✓	×	✓	×	peer ratings
MaAS (Zhang et al., 2025a)	×	×	×	×	none
HarnessX (Chen et al., 2026)	×	✓	✓	✓	none
SPRUNE (Ours)	✓	✓	✓	✓	Owen credit

3.1 PROBLEM DEFINITION

Agentic Workflow. We define an agentic workflow W as a directed graph of nodes linked by edges that set the execution order (Zhang et al., 2025c). Each node is a single call to a language model, parameterized by a **model** M (the LLM that executes the call), a **prompt** π (the instruction the model receives), a **temperature** ρ (the sampling randomness), and an **output format** F (how the response is parsed). In our implementation, F is determined by the node's role (for example, generating or revising a draft) and by the benchmark, rather than set independently.

An **edge**, in turn, is the directed link between two nodes that determines whether a draft may pass from one to the other. Prior work represents edges as graphs (Zhuge et al., 2024; Zhang et al., 2025a), neural networks (Liu et al., 2023c), or code (Hu et al., 2025; Zhang et al., 2025c) (Appendix J). We restrict our representation to a directed graph: each edge is one link between two nodes, and a missing edge passes no draft, so conditional branches and loops are outside our scope.

Workflow Credit Assignment. Within this representation, we call each node an **operator**, because in our setting every operator is a single LLM call—unlike methods whose operators can wrap multiple nodes into one composite block. A workflow is then written $W = (\mathcal{O}, E)$, where $\mathcal{O} = \{o_1, \ldots, o_N\}$ is its operator set, π_i is the prompt of operator o_i, and E is drawn from the candidate edges $\mathcal{E} = \{e_{ij} = (o_i, o_j) : i \neq j\}$, the ordered pairs of distinct operators. Throughout a run, the model, the temperature, and the output format, together with the backend configuration of the executor, stay fixed (Figure 1a); only which operators run, which prompts they receive, and which edges connect them can change. Automated workflow optimization treats these choices as a search for the workflow with the highest expected score $\mathbb{E}_T[G(W, T)]$, where the evaluation function G scores a workflow on a task T (Zhang et al., 2025c; Hu et al., 2025; Zhang et al., 2025a). We study the question that such a search leaves implicit: given one workflow and only its system-level score, how much does each component contribute? We refer to this question as **workflow credit assignment**. SPRUNE uses its answer both as the attribution it reports and as the signal that decides which components stay active (Section 3.2), and the resulting workflows are evaluated on the same expected score, which keeps them comparable with optimization methods.

Workflow Components as Players. To make this question precise, we treat every component that can be removed while the rest of the workflow still runs as a **player** p: each operator, each operator's prompt, and each candidate edge. The fixed parameters, by contrast, are not players. The **player set** is

$$\mathcal{P} = \{p_{o_i}\}_{i=1}^{N} \cup \{p_{\pi_i}\}_{i=1}^{N} \cup \{p_{e_{ij}}\}_{e_{ij} \in \mathcal{E}}.$$

A prompt is counted separately from its operator because the two can be removed independently: an operator can run without its prompt, whereas a prompt has no effect without its operator.

Workflow Interpretation as a Cooperative Game. With the players defined, we cast credit assignment as a cooperative game. A **coalition** $C \subseteq \mathcal{P}$ is any subset of the players, and $W|_C$ is the workflow restricted to C: absent operators are bypassed and issue no call, absent prompts leave their operator running with an empty instruction, and absent or dangling edges pass no draft (Appendix A.1). The **characteristic function** v_T then maps each coalition to the score that its restricted workflow achieves on task T,

$$v_T(C) = G(W|_C, T),$$

so that $v_T(\emptyset)$ is the score with every component absent and $v_T(\mathcal{P})$ is the score of the full workflow. Credit in this game is therefore measured in the units of the task score itself.

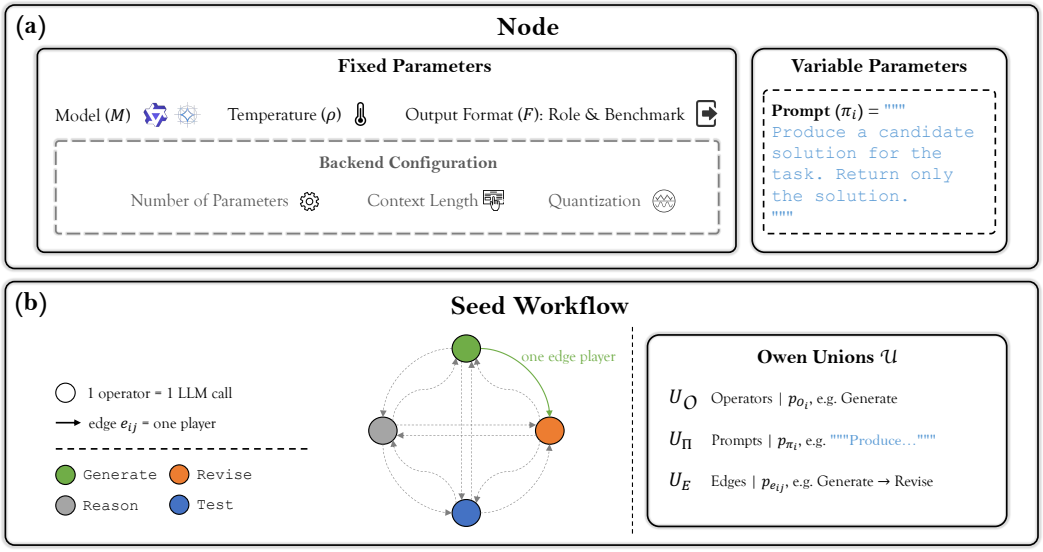

Figure 1: **(a)** A node is a single LLM call. The model, the temperature, the output format, and the backend configuration are fixed parameters; the prompt is the only variable parameter and is therefore a player. **(b)** The seed workflow links every ordered pair of operators by a candidate edge. Its operators, prompts, and edges form the three Owen unions U_O, U_Π, and U_E.

To divide $v_T(\mathcal{P}) - v_T(\emptyset)$ among the players, we use the **Owen value** (Owen, 1977), a refinement of the Shapley value (Shapley et al., 1953) for games with **a priori unions**. Here the players are partitioned by type into three unions: operators U_O, prompts U_Π, and edges U_E (Figure 1b). Credit is then allocated in two stages, first a quotient game among the unions and then a division within each union, so that an operator is compared only with other operators, a prompt only with other prompts, and an edge only with other edges. We write ϕ_p for the Owen value of player p in the game v_T; its formal definition and the sampling estimator we build on are given in Appendix A.

3.2 SPRUNE OVERVIEW

Credit-and-Act. Credit becomes useful for adapting a workflow once an action is tied to it. Search-based methods do not rely on such a tie: they evaluate candidate workflows, or learn a distribution over them, and return the result (Zhuge et al., 2024; Hu et al., 2025; Shang et al., 2025; Zhang et al., 2025c;a; Chen et al., 2026). We call a loop that repeatedly estimates per-component credit and edits the workflow on that credit alone a **credit-and-act** (CRACT) loop. A CRACT loop has three interchangeable parts: a *credit estimator*, which turns the game of the current task into a value for each player; an *accumulation rule*, which folds these values into a running score across tasks; and an *action set*, whose actions are selected by that score. Several prior methods can be read as points in this design space, although none is described in these terms. DyLAN scores agents from peer ratings and keeps the top-ranked ones (Liu et al., 2023c); HiveMind computes Shapley values over agents and rewrites the prompt of the lowest-valued agent (Xia et al., 2026); and contrastive or temporal–structural credit assignment turns a system-level score into per-agent prompt updates (Zehle, 2026; Li et al., 2026). Each of these methods credits a single component type, agents or their prompts, and none credits the edges between them.

SPRUNE instantiates CRACT with a Monte Carlo estimate of the Owen value over the three unions of Section 3.1 as the credit estimator, an exponential moving average as the accumulation rule, and archiving, restoring, prompt rewriting, and operator expansion as the action set. It keeps crediting archived players, so an archived component can return once its running score recovers. To our knowledge, SPRUNE is the first CRACT loop whose credit covers operators, prompts, and edges jointly.

One CRACT Round. SPRUNE keeps the player set partitioned into a **live set** L, whose players run, and an **archive** $\mathcal{A} = \mathcal{P} \setminus L$, whose players are held out; the graph that runs on each task is $W|_L$. Figure 2(a) shows one round on the validate task stream, which has four steps. (1) *Score*: $W|_L$

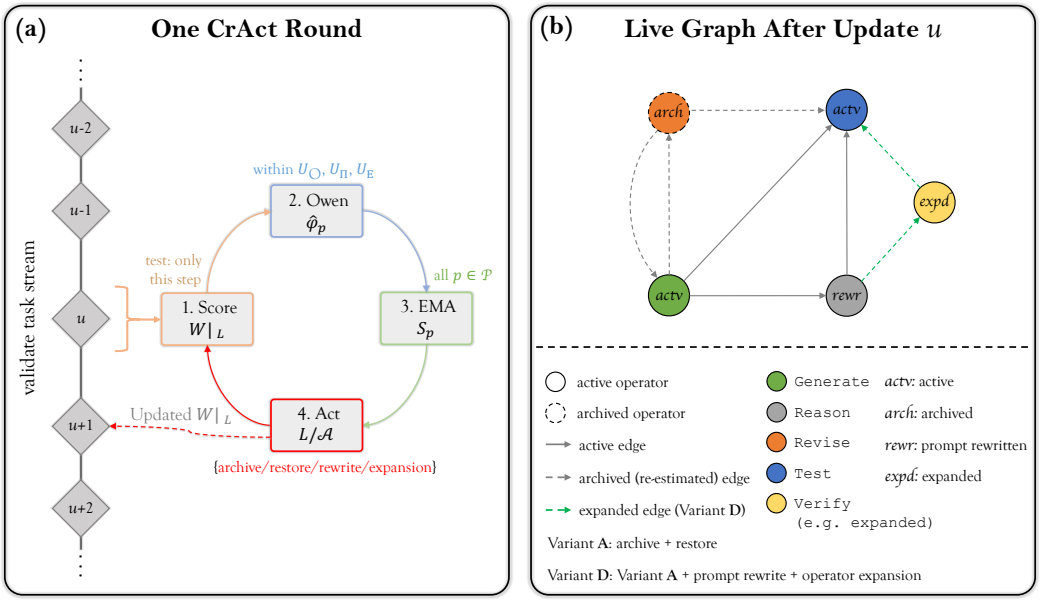

Figure 2: **(a)** One CrAct round of Sprune at update u on the validate task stream: Score, Owen, EMA, and Act (Section 3.2). On the test split, only Step 1 runs. **(b)** An example live graph after an update: the archived operator and its edges are held out but still re-estimated, one prompt has been rewritten, and an expanded operator enters with its new edges (variant D).

answers the current task before any credit is updated, so the reported answer is not influenced by that task's label. (2) *Owen*: Sprune estimates the Owen value $\hat{\phi}_p$ of every player in $\mathcal{P}$, archived players included. (3) *EMA*: because per-task estimates are noisy, they are folded into a **running score** S_p by an exponential moving average. (4) *Act*: players whose running score stays low for several consecutive updates move to $\mathcal{A}$, archived players whose score recovers return to L, and variant D may also rewrite a prompt and expand the operator set. The updated $W|_L$ runs on the next task; on the test split the final live graph is frozen and only Step 1 runs.

An archived player is thus **not deleted**: its credit is re-estimated at every update, which is what allows it to return. And because the running score is the only signal that the act step reads, the attribution reported after an update is the signal that drove it, not a separate post-hoc estimate; in this sense, Sprune is *interpretable by construction*.

Task Scope, Operations, and Variants. We apply Sprune to reasoning tasks whose evaluation function G is numerical, covering code generation, mathematical reasoning, and question answering (Section 5.1). The seed operator set, extracted from existing literature, contains Generate, Reason, Revise, and Test; each operator has one prompt player, and the seed workflow uses every candidate edge (Appendix C). We study two action sets. Variant **A** only archives and restores players and is used for the main results. Variant **D** adds two language-model actions: it rewrites the live prompt with the lowest running score, and it can add one operator with its prompt and candidate edges, up to six operators (Appendix B). New players are credited from the next update on, so the running score decides whether they stay live, while the language model only writes the new text.

4 The Design Details of Sprune

This section details each step of the CrAct round of Figure 2(a), following Algorithm 1. No step takes a gradient or updates a model weight, which keeps Sprune training-free.

Initialization and Evaluation. Sprune starts from the seed workflow of Figure 1: four operators, their four prompts, and the twelve candidate edges, which gives $n = 20$ players (Eq. 4). Every player starts in the live set L with running score $S_p = 0$ and streak counter $\kappa_p = 0$. On each validate task T, Sprune first runs $W|_L$ and records its score $h_t = v_T(L)$ before any credit update; the same

Algorithm 1 SPRUNE on the validate task stream.

Require: validate tasks $T_0, T_1, \ldots$; player set $\mathcal{P}$ with unions $\mathcal{U} = \{U_O, U_\Pi, U_E\}$; evaluator G; (α, τ, k, m, B)
Ensure: final live graph $W|_L$ and reported scores h

$L \leftarrow \mathcal{P}$; $S_p \leftarrow 0, \kappa_p \leftarrow 0$ for all $p \in \mathcal{P}$; $h \leftarrow ()$ ▷ Initialization
for $t = 0, 1, 2, \ldots$ **do**
  $h_t \leftarrow G(W|_L, T_t)$; append h_t to h ▷ Score, before any update
  if $\neg$SHOULDTRIGGER(t, h) **then**
    continue ▷ Eq. 7
  end if
  $\hat{\phi}_p \leftarrow$ OWENMC$(v_{T_t}, \mathcal{U}, m)$ for all $p \in \mathcal{P}$ ▷ Owen, Eq. 1
  $S_p \leftarrow (1-\alpha)\, S_p + \alpha\, \hat{\phi}_p$ for all $p \in \mathcal{P}$ ▷ EMA, Eq. 2
  if $\hat{\phi} \not\equiv 0$ **then**
    update κ_p and move players between L and $\mathcal{A}$ ▷ Act, Eq. 3
  end if
  FEASIBILITYREPAIR(L)
  if variant D **and** $\hat{\phi} \not\equiv 0$ **then**
    rewrite one prompt, then expand $\mathcal{P}$ ▷ Algorithms 2 and 3
  end if
end for
return $W|_L, h$ ▷ $W|_L$ is frozen for the test split

evaluator, applied to other coalitions, supplies the game v_T. On the test split, the final live graph is frozen and only this step runs.

Owen Credit Estimation. Exact Owen values require thousands of workflow executions per task even for the seed player set (Appendix J). SPRUNE therefore samples m orderings $\omega_1, \ldots, \omega_m$ uniformly from $\Omega(\mathcal{U})$, the orderings in which the players of each union U_O, U_Π, and U_E form one contiguous block, and averages the marginal contribution of each player:

$$\hat{\phi}_p = \frac{1}{m} \sum_{\ell=1}^{m} \Big[v_T\big(\omega_\ell^{\leq p}\big) - v_T\big(\omega_\ell^{<p}\big) \Big], \tag{1}$$

where $\omega_\ell^{<p}$ is the set of players that precede p in the ℓ-th ordering and $\omega_\ell^{\leq p} = \omega_\ell^{<p} \cup \{p\}$. This estimator is unbiased for the Owen value (Saavedra-Nieves et al., 2018). Each ordering is evaluated along its $n+1$ prefixes, so one update issues $m(n+1)$ queries to v_T, and recurring prefixes are memoized (Appendix A.2). Because the sum runs over all of $\mathcal{P}$, archived players are credited as well.

Score Accumulation. Per-task estimates are noisy, since a player that sits in weak coalitions on one task may still be essential on the next. SPRUNE therefore folds each estimate into a running score,

$$S_p^{(u)} = (1-\alpha)\, S_p^{(u-1)} + \alpha\, \hat{\phi}_p^{(u)}, \qquad S_p^{(0)} = 0, \tag{2}$$

where u counts credit updates and a larger $\alpha \in [0, 1]$ adapts faster but is more sensitive to noise. The average is taken for every player, archived players included, and S_p is the only quantity that the act step reads.

Archive and Restore. For each live player, a streak counter κ_p tracks how many consecutive updates its running score has stayed below a threshold τ. A live player is archived when its streak reaches k, and an archived player is restored as soon as one update gives $S_p \geq \tau$:

$$\kappa_p^{(u)} = \big(\kappa_p^{(u-1)} + 1\big)\, \mathbf{1}\big[S_p^{(u)} < \tau\big] \quad \text{for } p \in L^{(u-1)},$$
$$p \in L^{(u)} \iff \begin{cases} \kappa_p^{(u)} < k, & p \in L^{(u-1)}, \\ S_p^{(u)} \geq \tau, & p \in \mathcal{A}^{(u-1)}. \end{cases} \tag{3}$$

A restored player starts a new streak at zero, so a noisy dip does not remove a player, but a genuine recovery brings it back quickly. Safeguards against empty unions and degenerate updates are described in Appendix J.

Expansion in Variant D and Scheduling. Variant D adds two language-model actions to the same update, applied only when the credits are not all zero: a prompt rewrite of the live operator whose

Table 2: Frozen-test scores (%) on six benchmarks and three local Q8 backbones; every method uses the same backbone within a column. Gold, silver, and bronze mark the first three per column; subscripts are the sample standard deviation over three seeds when the seeds differ. SPRUNE cells on the 9B and 27B backbones are the best test score over the searched configurations (one seed each), an upper bound (Section 5.2); on the 31B backbone, SPRUNE (A) uses its default configuration over three seeds. ‡ADAS returned no valid DROP answer. A dash marks an unfinished run.

	Code Generation pass@1 ↑						Mathematical Reasoning solve rate ↑						Question Answering F1 ↑					
	HumanEval			MBPP			GSM8K			MATH			HotpotQA			DROP		
Method	9B	27B	31B	9B	27B	31B	9B	27B	31B	9B	27B	31B	9B	27B	31B	9B	27B	31B
AFlow (Zhang et al., 2025c)	–	–	**82.4±13.8**	–	–	80.9±6.4	–	–	–	5.8±7.1	–	–	–	–	–	–	–	–
ADAS (Hu et al., 2025)	74.8	75.6	77.9	72.1	**82.1**	**81.5**	–	–	–	12.3	**18.3**	8.4	**57.6**	**58.0**	**63.5**	0.0‡	–	–
GPTSwarm (Zhuge et al., 2024)	64.9±0.8	74.3±1.6	77.9±0.8	66.6	73.9	78.3	28.7	31.4	73.3	4.1	4.7	2.9	45.1	48.0	52.1	52.8	67.4	69.7
AgentSquare (Shang et al., 2025)	**88.5**	67.9	**82.4**	**73.9**	78.3	77.4	66.3	74.0	**93.1**	**15.4**	4.7	4.3	53.0	51.1	49.3	**68.0**	68.7	**69.8**
DyLAN (Liu et al., 2023c)	33.1±28.9	62.6±17.2	76.3±0.8	71.3	75.4	79.5	15.4	5.7	32.1	–	–	–	44.9	28.0	32.8	56.5	54.0	59.7
MaAS (Zhang et al., 2025a)	–	26.5±7.1	27.5±12.8	65.1	75.1	75.4	38.4±2.7	41.0±3.6	–	–	–	–	30.1	25.3	26.2	49.7	50.2	59.3
HarnessX (Chen et al., 2026)	64.9	72.5	69.5	65.1	75.4	76.2	3.9	4.3	4.0	4.7	8.6	**12.6**	3.6	4.7	16.9	14.2	24.3	16.3
SPRUNE (A, Ours)	80.9	72.5	79.1±15.7	58.9	65.7	72.7±0.5	**71.3**	70.0	–	12.1	5.6	–	47.4	55.7	24.5±16.2	53.4	**70.7**	65.6±0.1
SPRUNE (D, Ours)	80.9	**79.4**	–	60.7	66.0	–	71.1	**80.7**	–	12.3	14.8	–	46.7	54.7	–	51.1	68.2	–

prompt player has the lowest running score (Algorithm 2), followed by an operator expansion that proposes one new operator with its prompt and candidate edges (Algorithm 3). A rewrite changes only the text π_i; an expansion extends $\mathcal{P}$, and the new players receive their first estimate on the next update. Updates run every B validate tasks under the default `fixed_batch` schedule, and the main results use $B = 1$ (Appendix A.3).

5 EXPERIMENTS

We ask how a credit-and-act loop compares with search-based methods on task score, what its credit reveals, and whether the language-model actions of variant D help.

5.1 SETUP

Benchmarks and Baselines. Following AFlow (Zhang et al., 2025c), we use HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) (pass@1), GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) (solve rate), and HotpotQA (Yang et al., 2018) and DROP (Dua et al., 2019) (F1), each split at a fixed seed into 20% validate and 80% test tasks (Appendix G). The baselines are AFlow, ADAS, GPTSwarm, AgentSquare, DyLAN, MaAS, and HarnessX, each run from its official repository with its published budget; each searches on the validate split and its returned workflow is scored once on the test split (Appendix H).

Backbones. All methods run on three open-weight backbones served locally at 8-bit quantization through llama.cpp (Gerganov & ggml contributors, 2023): Qwen3.5-9B (Team, 2026b), Qwen3.8-27B (Qwen Team, 2026), and Gemma 4 31B-it (Team, 2026a). Within one backbone, every method uses the same model for every call, including those of its optimizer, so the numbers are comparable to each other but not to published results. Appendix D lists the decoding settings and hardware, and Appendix H the baseline budgets.

SPRUNE Configuration and Protocol. SPRUNE runs one credit update per validate task ($B = 1$), scores each task before the update it triggers, and freezes the live graph for the test split. Its hyperparameters (α, τ, k, m) are drawn from an 81-point grid: variant A was run with 45 to 48 grid points per benchmark and backbone and variant D with 9 to 31, each on one seed. On the 9B and 27B backbones, the table reports for each variant the best test score over these configurations; on the 31B backbone, where no search was run, variant A uses its default configuration $(0.1, 0.05, 3, 5)$ over three seeds. We report frozen-test scores in percent, as means over three seeds with the sample standard deviation when the seeds differ, and leave unfinished cells empty.

5.2 MAIN RESULTS

Overall. Table 2 shows that no single method leads across benchmarks and backbones. AgentSquare is the strongest baseline, first in 7 of its 18 cells and among the top three in 14, and ADAS is first in

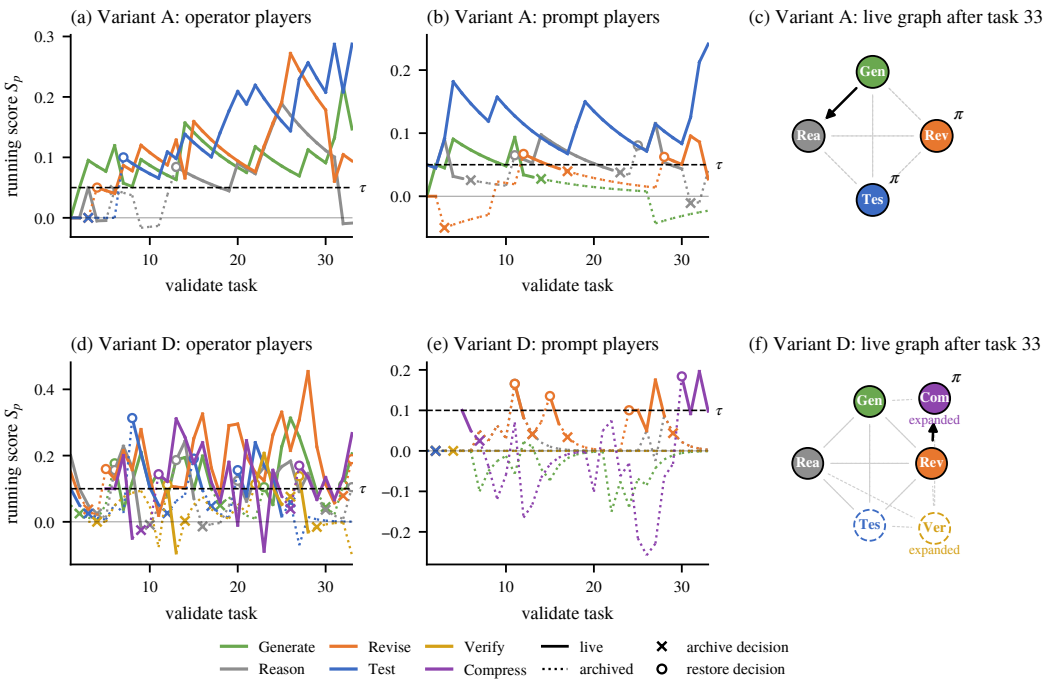

Figure 3: Credit trajectories in two SPRUNE runs on HumanEval: variant A with the 9B backbone (top, test score 80.9) and variant D with the 27B backbone (bottom, 79.4). **(a, b, d, e)** Running scores; dotted segments are archived, and crosses and circles mark archive and restore decisions. **(c, f)** Frozen live graphs; dashed circles are archived operators, and π marks a live prompt.

6 of the 13 cells it finished. SPRUNE, taking the better of its two variants, is first in 4 of the 16 cells where it finished and among the top three in 12, leading GSM8K on the 9B and 27B backbones and HumanEval and DROP on the 27B backbone. MBPP is its weakest benchmark, and MATH scores stay below 20 for every method. Appendix K discusses the results by task family, the missing cells, and the results across backbones.

Effect of the Action Set. Where both variants finished, the best D configuration improves on the best A configuration by 6.9 points on HumanEval, 10.6 on GSM8K, and 9.3 on MATH on the 27B backbone, and it stays within 2.6 points of A elsewhere. The median D configuration, however, scores below the median A configuration in 11 of the 12 cells, most clearly on the 9B backbone (Appendix E). The language-model actions thus raise the best achievable score with the larger backbone, while archive and restore alone are the more reliable choice with the smaller one.

Hyperparameter Selection. Each SPRUNE cell on the 9B and 27B backbones reports the best test score over the searched configurations, whereas each baseline runs one configuration, so hyperparameter selection confounds the comparison in SPRUNE's favor; these cells are an upper bound, and a three-seed replay of the validate-selected configuration is the protocol-conformant number.

5.3 CASE STUDY AND CREDIT ANALYSIS

Credit Trajectories. Figure 3(a–c) follows the variant A run behind the HumanEval score of 80.9 with the 9B backbone. Because running scores start below τ, the third update archives most players, and restoring then undoes much of this: Test, archived after the third task and restored after the seventh, ends with the highest running score among the operators and runs last, so its output is the scored answer. In the variant D run of Figure 3(d–f), the language model adds Verify, which asks for a critique rather than a solution and is archived after negative credit, and Compress, which ends with the highest running score and runs last (Appendix M). Each point on these curves is the quantity that the act step read.

Credit Across Benchmarks. Figure 4(a) averages the final running scores over all variant A runs. Operators carry most of the credit, receiving between 52% and 100% of the total on every benchmark

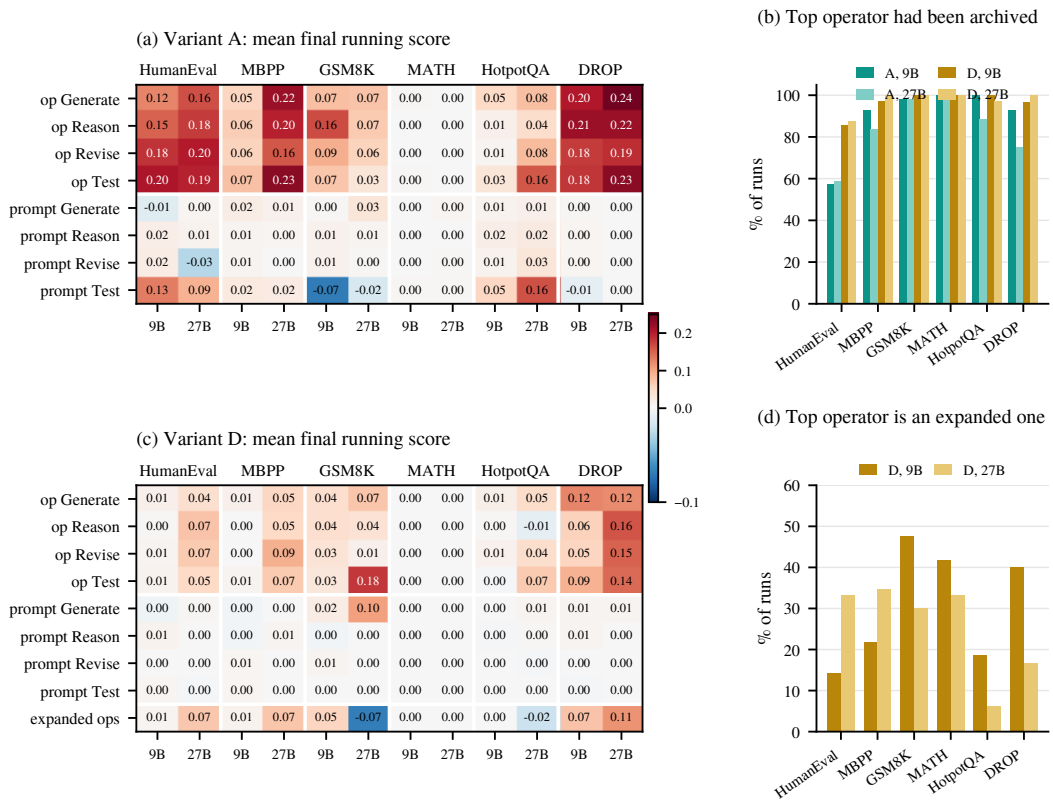

Figure 4: Credit across all variant A (618) and variant D (320) runs of the hyperparameter search. **(a, c)** Mean final running scores; for variant D, the last row averages the proposed operators. **(b)** Share of runs whose top operator had been archived before. **(d)** Share of variant D runs whose top operator was proposed by the language model.

except MATH, where all credits are close to zero. Prompts matter mainly through the prompt of Test, which receives positive credit on HumanEval and HotpotQA and negative credit on GSM8K, marking it as harmful there, and the leading operator depends on the task, for example Test on HumanEval and Reason on GSM8K with the 9B backbone. The rankings are stable across seeds (Appendix K). In variant D (Figure 4c, d), part of the credit moves to the operators proposed by the language model, one of which ends with the highest running score in 24–30% of the runs.

Why Restoring Matters. In 86.9% of the 618 variant A runs and 96.6% of the 320 variant D runs, the operator with the highest final running score had been archived at least once before it was restored (Figure 4b); an irreversible rule with the same thresholds would have lost it. Where variant D fails, on the 9B backbone, the live graph has usually collapsed to a single operator whose rewritten or proposed instruction no longer asks for the solution alone (Appendix F).

6 Conclusion

We named credit-and-act (CrAct) and presented Sprune, a training-free instance whose Owen credit covers operators, prompts, and edges jointly and drives reversible archive and restore decisions. Sprune is competitive with seven workflow-search methods, and its credit explains its own behavior: operators carry most of the credit, single prompts can help or hurt depending on the task, and most operators that end with the highest credit had been archived and restored.

Limitations. The Sprune cells on the 9B and 27B backbones are best-of-search upper bounds on one seed, several cells are missing, running scores start below τ, and the faithfulness of the credit and the token cost of the baselines remain untested (Appendix L).

STATEMENTS

Reproducibility. An anonymized code repository is provided for double-blind review (`https://anonymous.4open.science/r/sprune-iclr`). It contains the implementation of SPRUNE, the adapters for the seven baselines, the split scripts, and the experiment configurations. Section 5.1 and Appendix D give the backbones, the decoding settings, the hardware, the hyperparameter grid, and the variant D parameters; Appendix G gives the split sizes; Appendix H gives the repositories, budgets, and entry points of the baselines; and Appendices A, B, and C give the definitions, the variant D algorithms, and the seed prompts. Every method runs on the same local open-weight backbone within a column of Table 2, served through llama.cpp at 8-bit quantization, which removes hosted-API version drift and rate limits as sources of variation. Baseline cells and the default SPRUNE cells are means over three seeds; the SPRUNE cells on the 9B and 27B backbones are the best test scores of single-seed search runs, as stated in Section 5.1. One replication detail is easy to miss: the prompt cache keys on hashes of the realized prompt and instruction text, so cached calls stay correct across the coalitions of one update.

Ethics. This work involves no human subjects and no personally identifying or otherwise sensitive data, and all experiments use publicly available research benchmarks. SPRUNE changes which components of an agent workflow run, so it shares the general dual-use considerations of automating software and agent development. It adds no capability beyond reallocating effort inside a workflow that the host already runs, and its archive-and-restore design keeps every structural decision inspectable and reversible. We are aware of no further ethical concerns raised by this work.

Use of AI. We used generative AI tools for code development, for literature search and the cross-checking of implementation details, for writing analysis scripts and figures, and for drafting and editing the text, including the formal definitions, always under the direction of the authors. Generative AI also appears inside the method: the prompt rewrite and the operator expansion of variant D query a language model as a component of SPRUNE. We did not use generative AI to generate or alter experimental results, to produce synthetic data for training or evaluation, or to fabricate citations. We reviewed all AI-assisted work, testing the code and checking the text and figures against the implementation, the run logs, and the cited literature, and we take responsibility for the final content of this work.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Javier Castro, Daniel Gómez, and Juan Tejada. Polynomial calculation of the shapley value based on sampling. *Computers & operations research*, 36(5):1726–1730, 2009.

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

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

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

DeepSeek-AI. Deepseek harness: Everything is a plugin. `https://github.com/deepseek-ai/deepseek-harness`, 2026.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Bradley Efron and Carl N. Morris. Data analysis using stein's estimator and its generalizations. *Journal of the American Statistical Association*, 70:311–319, 1975. URL `https://api.semanticscholar.org/CorpusID:53002187`.

Georgi Gerganov and ggml contributors. llama.cpp: LLM inference in C/C++. `https://github.com/ggml-org/llama.cpp`, 2023.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Daniel G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–685, 1952. URL `https://api.semanticscholar.org/CorpusID:120274071`.

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

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. Whale: A simple recipe for joint harness-weight optimization. *arXiv preprint arXiv:2609.00196*, 2026.

LangChain. LangGraph. `https://github.com/langchain-ai/langgraph`, 2024.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Wenwu Li, Yuran Song, Mingze Zhao, Bo Jin, and Wenhao Li. Unifying temporal and structural credit assignment in llm-based multi-agent prompt optimization, 2026. URL https://arxiv.org/abs/2605.30227.

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*, volume 2024, pp. 39578–39601, 2024.

Hanxi Liu, Xiaokai Mao, Haocheng Xia, Jian Lou, Jinfei Liu, and Kui Ren. Prompt valuation based on shapley values. *arXiv preprint arXiv:2312.15395*, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023b.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023c.

Mingyu Lu, Yushan Huang, Chris Lin, and Su-In Lee. Agents that matter: Optimizing multi-agent llms via removal-based attribution. *arXiv preprint arXiv:2605.27621*, 2026a.

Yuxing Lu, Yicheng Chen, Shanchan Wu, and Sercan Ö Arık. Procedural graphs: Self-evolving execution structures for llm agents. *arXiv preprint arXiv:2609.09153*, 2026b.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. Compositional questions do not necessitate multi-hop reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4249–4257, 2019.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 94743–94765, 2025.

João Moura. CrewAI: Framework for orchestrating role-playing, autonomous AI agents. https://github.com/crewAIInc/crewAI, 2023.

Nous Research. Hermes Agent. https://github.com/NousResearch/hermes-agent, 2026.

Guilliermo Owen. Values of games with a priori unions. In *Mathematical economics and game theory: Essays in honor of Oskar Morgenstern*, pp. 76–88. Springer, 1977.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Peng Qi, Xiaowen Lin, Leo Mehr, Zijian Wang, and Christopher D Manning. Answering complex open-domain questions through iterative query generation. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2590–2602, 2019.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Anton Razzhigaev, Andrei Gritsaev, Andrei Kaznacheev, Nikita Dragunov, Roman Yampolskiy, and Andrei Kuznetsov. Ouroboros: A self-developing frontier coding agent with reviewed core evolution. *arXiv preprint arXiv:2608.08311*, 2026.

Alejandro Saavedra-Nieves, Ignacio García-Jurado, and M. Gloria Fiestras-Janeiro. *Estimation of the Owen Value Based on Sampling*, pp. 347–356. Springer International Publishing, 2018. ISBN 9783319738482. doi: 10.1007/978-3-319-73848-2_33. URL http://dx.doi.org/10.1007/978-3-319-73848-2_33.

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 *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Lloyd S Shapley et al. A value for n-person games. *Contributions to the Theory of Games (AM-28)*, 1953.

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

Peter Steinberger and OpenClaw contributors. OpenClaw. https://github.com/openclaw/openclaw, 2025.

Gemma Team. Gemma 4 technical report, 2026a. URL https://arxiv.org/abs/2607.02770.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026b. URL https://qwen.ai/blog?id=qwen3.5.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. Vero: A harness for agents to optimize agents. *arXiv preprint arXiv:2602.22480*, 2026.

Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. Metagen: Self-evolving roles and topologies for multi-agent llm reasoning. *arXiv preprint arXiv:2601.19290*, 2026.

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

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

Lilian Weng. Harness engineering for self-improvement. *lilianweng.github.io*, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Xinping Lei, Qingshui Gu, Yuxuan Zhang, Zexuan Wang, Chen He, Chen Huang, Maojia Song, et al. Harnessdev: Can llms create and evolve their own agent harness? *arXiv preprint arXiv:2609.01437*, 2026.

Yihan Xia, Taotao Wang, Shengli Zhang, Zhangyuhua Weng, Bin Cao, and Soung Chang Liew. Hivemind: Contribution-guided online prompt optimization of llm multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29767–29774, 2026.

Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, et al. Answering complex open-domain questions with multi-hop dense retrieval. *arXiv preprint arXiv:2009.12756*, 2020.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs. *arXiv preprint arXiv:2602.07755*, 2026.

Yingxuan Yang, Bo Huang, Siyuan Qi, Chao Feng, Haoyi Hu, Yuxuan Zhu, Jinbo Hu, Haoran Zhao, Ziyi He, Xiao Liu, Zongyu Wang, Lin Qiu, Xuezhi Cao, Xunliang Cai, Yong Yu, and Weinan Zhang. Who's the mvp? a game-theoretic evaluation benchmark for modular attribution in llm agents, 2025a. URL https://arxiv.org/abs/2502.00510.

Yingxuan Yang, Bo Huang, Siyuan Qi, Chao Feng, Haoyi Hu, Yuxuan Zhu, Jinbo Hu, Haoran Zhao, Ziyi He, Xiao Liu, et al. Understanding and optimizing agentic workflows via shapley value. *arXiv preprint arXiv:2502.00510*, 2025b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Tom Zehle. Cantante: Optimizing agentic systems via contrastive credit attribution, 2026. URL https://arxiv.org/abs/2605.13295.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024a.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025b.

Guibin Zhang, Haiyang Yu, Kaiming Yang, Bingli Wu, Fei Huang, Yongbin Li, and Shuicheng Yan. Evoroute: Experience-driven self-routing llm agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 38213–38225, 2026.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Charlotte Zhuang, Dylan Slack, et al. A careful examination of large language model performance on grade school arithmetic. *Advances in Neural Information Processing Systems*, 37:46819–46836, 2024b.

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

Minjun Zhao, Xinyu Zhang, Shuai Zhang, Deyang Li, and Ruifeng Shi. Adopt: Adaptive dependency-guided joint prompt optimization for multi-step llm pipelines. *arXiv preprint arXiv:2512.24933*, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A DEFINITIONS

This appendix makes precise the objects that Section 3 introduces informally: the players and the restricted workflow (Appendix A.1), the Owen value and its sampling estimator (Appendix A.2), and the update trigger (Appendix A.3). It closes with the break-even cost model (Appendix A.4) and notes on the sampler (Appendix A.5).

A.1 PLAYERS AND THE RESTRICTED WORKFLOW

Player identifiers. In the implementation, each player carries a string identifier: `op:`o_i for the operator player p_{o_i}, `prompt:`o_i for the prompt player p_{π_i}, and `edge:`$o_i{\to}o_j$ for the edge player $p_{e_{ij}}$. With N operators and every ordered pair of distinct operators as a candidate edge, the number of players is

$$n = |\mathcal{P}| = \underbrace{N}_{U_O} + \underbrace{N}_{U_\Pi} + \underbrace{N(N-1)}_{U_E} = 2N + N(N-1). \tag{4}$$

The seed workflow of Figure 1 has $N = 4$ operators and therefore $4 + 4 + 12 = 20$ players.

Restricted workflow. For a task T and a coalition $C \subseteq \mathcal{P}$, the restricted workflow $W|_C$ is built as follows. An operator whose player p_{o_i} is absent is bypassed: it issues no model call and passes no draft to its successors. An operator whose prompt player p_{π_i} is absent runs with an empty instruction but keeps its role and its inputs, while a prompt player whose operator is absent has no effect. An absent edge player removes its edge, and edges with a bypassed endpoint are dropped as well. The remaining operators then run in the topological order given by Kahn's algorithm, with ties broken by the seed operator order; operators left inside a cycle are appended in the same order, so execution always terminates. Each operator o_j reads the draft of every predecessor o_i with e_{ij} in $W|_C$ that has already run, the most recent one taking precedence, and the output of the last operator to run is the answer that G scores.

Characteristic function. $v_T(C)$ is the benchmark score of $W|_C$ on T. In particular, $v_T(\emptyset)$ is the score when no operator runs, and $v_T(\mathcal{P})$ is the score of the full workflow. Evaluations are memoized per task, so a coalition that recurs within one update is executed only once.

A.2 OWEN VALUE AND ITS SAMPLING ESTIMATOR

Let $\mathcal{U} = \{U_1, \ldots, U_q\}$ partition $\mathcal{P}$ into unions; in SPRUNE, $q = 3$ and the unions are U_O, U_Π, and U_E. Write $[q] = \{1, \ldots, q\}$, $U_R = \bigcup_{j \in R} U_j$ for $R \subseteq [q]$, and $w(X, Y) = |X|!\,(|Y| - |X| - 1)!\,/\,|Y|!$ for finite sets $X \subset Y$. For a player $p \in U_j$, the Owen value (Owen, 1977) is

$$\phi_p^{\mathcal{U}}(v) = \sum_{R \subseteq [q] \setminus \{j\}} \; \sum_{C \subseteq U_j \setminus \{p\}} w(R, [q])\, w(C, U_j) \left[v(U_R \cup C \cup \{p\}) - v(U_R \cup C) \right]. \tag{5}$$

The outer sum plays a Shapley game among the unions, and the inner sum divides each union's share among its members. The Owen value is efficient, so the values of all players sum to $v(\mathcal{P}) - v(\emptyset)$. In the main text, ϕ_p stands for $\phi_p^{\mathcal{U}}(v_T)$ and $\hat{\phi}_p$ for its estimate of Eq. 1.

Equivalently, $\phi_p^{\mathcal{U}}(v)$ is the expected marginal contribution $v(\omega^{\leq p}) - v(\omega^{<p})$ when the ordering ω is drawn uniformly from the orderings in which every union forms one contiguous block, where $\omega^{<p}$ denotes the players that precede p and $\omega^{\leq p} = \omega^{<p} \cup \{p\}$ (Owen, 1977). Averaging this marginal over sampled orderings therefore gives an unbiased Monte Carlo estimator of the Owen value (Saavedra-Nieves et al., 2018), the counterpart for a priori unions of permutation sampling for the Shapley value (Castro et al., 2009). The sampler of Eq. 1 follows this construction: it draws a uniform order of the q unions and, independently, a uniform order inside each union, and it concatenates the blocks.

Efficiency of the estimate. Along one ordering, the marginals telescope to $v_T(\mathcal{P}) - v_T(\emptyset)$, and the per-task memo returns the same two endpoint values for every ordering. The Monte Carlo estimate is therefore efficient without correction. The rescaling applied before accumulation,

$$\tilde{\phi}_p = \begin{cases} \hat{\phi}_p \cdot \dfrac{v_T(\mathcal{P}) - v_T(\emptyset)}{\sum_{p' \in \mathcal{P}} \hat{\phi}_{p'}}, & \left|\sum_{p'} \hat{\phi}_{p'}\right| > \varepsilon, \\ \big(v_T(\mathcal{P}) - v_T(\emptyset)\big)/n, & \text{otherwise}, \end{cases} \qquad \varepsilon = 10^{-12}, \tag{6}$$

is thus a numerical safeguard whose scale factor is one up to floating-point error; its second branch assigns zero to every player when $v_T(\mathcal{P}) = v_T(\emptyset)$.

Query count. Each sampled ordering is evaluated along its $n + 1$ prefixes, from $\emptyset$ to $\mathcal{P}$, so one update with m orderings issues $m(n + 1)$ queries to v_T, and the two endpoint values that Eq. 6 needs are among them. By Eq. 4, this count grows as $O(mN^2)$ in the operator count. How many operators run inside a query sets its token price rather than the number of queries, and the cost model of Appendix A.4 charges that price separately. A subset-sampling estimator that queries both $v(C)$ and $v(C \cup \{p\})$ for m coalitions per player would instead need $2mn$ queries; it is kept in the code as a reference implementation and prices no reported run.

A.3 Update Trigger

Let h be the reported scores through the zero-based task index t, so $|h| = t + 1$. Let h^{-H} be its last $\min(H, |h|)$ entries for a window length H, and h^{+H} the entries preceding them, falling back to all but the current score when the window already covers the whole history. Let h^{-k} and h^{+k} be the analogous split at depth $\max(k, 2)$, and let $\bar{\cdot}$ and Var denote the sample mean and the population variance. Then

$$\textsc{ShouldTrigger}(t, h) = \begin{cases} \mathbf{1}\big[(t+1) \bmod B = 0\big], & \texttt{fixed_batch}, \\ \mathbf{1}\big[\bar{h}^{+H} - \bar{h}^{-H} \geq \delta\big] \vee \mathbf{1}\big[\mathrm{Var}(h^{-k}) > \nu \,\mathrm{Var}(h^{+k})\big], & \texttt{event}, \end{cases} \tag{7}$$

where the event branch fires unconditionally while fewer than two scores are available, the variance test requires at least two entries on each side, and, when $\mathrm{Var}(h^{+k}) = 0$, it fires whenever $\mathrm{Var}(h^{-k}) > 0$. The main campaign uses only the `fixed_batch` branch, at $B = 1$.

A.4 Break-even

Write c for the tokens of one operator call, N for the number of operators in the full graph, and a_t for the number of operators executed on task t. Charging each query of an update at $N/2$ operator calls in expectation, the per-task costs of the full graph and of SPRUNE are

$$\mathrm{Cost}_t^{\text{base}} = N\,c, \qquad \mathrm{Cost}_t^{\text{s}} = a_t\,c + \mathbf{1}\big[\textsc{ShouldTrigger}(t, h)\big] \cdot m(n+1)\,\frac{N}{2}\,c, \tag{8}$$

and the break-even point is the first task at which the cumulative cost of SPRUNE falls below that of the full graph,

$$N_{\text{breakeven}} = \min\Big\{t : \sum_{s \leq t} \mathrm{Cost}_s^{\text{s}} < \sum_{s \leq t} \mathrm{Cost}_s^{\text{base}}\Big\}. \tag{9}$$

Under the `fixed_batch` branch of Eq. 7, updates fire $\lfloor t/B \rfloor$ times in t tasks. Write $\Lambda = m(n + 1)\,N/2$ for the per-update charge in units of c. Let t_0 be the first task at which archiving takes effect, so that $a_s = N$ for $s \leq t_0$; since a player needs k consecutive sub-threshold updates before it is archived (Eq. 3) and updates arrive every B tasks, $t_0 \geq kB$. Let $\bar{a}$ be the mean number of executed operators for $s > t_0$ and $\Delta = N - \bar{a}$. For $t > t_0$, the cumulative advantage of SPRUNE is $D(t) = c\,(t - t_0)\,\Delta - c\,\Lambda\lfloor t/B \rfloor$, and relaxing $\lfloor t/B \rfloor$ to t/B gives

$$D(t) > 0 \iff t\big(\Delta - \Lambda/B\big) > t_0\,\Delta. \tag{10}$$

If $\Delta \leq \Lambda/B$, the left-hand side is non-positive for every $t \geq 0$ while the right-hand side is non-negative, so no crossing exists and Eq. 9 is undefined, which is what the cost ledger reports. If $\Delta > \Lambda/B$, the smallest solution is

$$N_{\text{breakeven}} = \left\lceil \frac{t_0\,\Delta}{\Delta - \Lambda/B} \right\rceil. \tag{11}$$

Two bounds decide the $B = 1$ case. At least one operator executes on every task, because the feasibility repair keeps one operator live, so $\Delta \leq N - 1$; and $\Lambda \geq N$ for all $m, n \geq 1$. Hence at $B = 1$ we have $\Lambda/B > \Delta$, and no crossing can occur for the schedule that produced our results. A crossing requires $B > \Lambda/\Delta$, that is, fewer than Δ/Λ updates per task, which is the quantity that the event branch of Eq. 7 controls. On the frozen test split no update fires, so the per-task cost is $a_t\,c$ with $a_t \leq N$. The ledger also supports an observed-token accounting in which the update charge is the measured cost of the coalition executions rather than Λc; memoization places that measurement below Λc, and the algebra above applies with Λ replaced by the measured mean charge per update.

Algorithm 2 Variant D, prompt rewrite.

Require: operators $o_1, \ldots, o_N$ with prompts $\pi_1, \ldots, \pi_N$; live set L; running scores S; rewrite threshold τ_{rw}; retry cap $r_{\max}$; retry counts $r_1, \ldots, r_N$

$I \leftarrow \{i : p_{o_i} \in L,\ p_{\pi_i} \in L,\ \pi_i \neq \epsilon\}$
if $I = \emptyset$ **then**
  return prompts unchanged
end if
$i^\star \leftarrow \arg\min_{i \in I} S_{p_{\pi_i}}$ ▷ ties broken by operator name
if $S_{p_{\pi_{i^\star}}} \geq \tau_{\mathrm{rw}}$ **or** $r_{i^\star} \geq r_{\max}$ **then**
  return prompts unchanged
end if
$J \leftarrow \textsc{ParseJSON}\big(\mathrm{LLM}(\textsc{RewritePrompt}(o_{i^\star}, \pi_{i^\star}))\big)$ ▷ {`operator, prompt`}
if J is malformed **or** J.operator $\neq o_{i^\star}$ **or** J.prompt $= \epsilon$ **then**
  return prompts unchanged
end if
$\pi_{i^\star} \leftarrow J$.prompt; $\quad r_{i^\star} \leftarrow r_{i^\star} + 1$ ▷ $\mathcal{P}, \mathcal{U}$, and n unchanged

A.5 Sampler Notes

Variance reduction, not attempted. The credits are the plain permutation average of Eq. 1, with no weighting and no shrinkage. Two standard devices remain untested. Unequal sampling probabilities with an inverse-probability correction in the manner of Horvitz & Thompson (1952) would let the edge union, which dominates $\mathcal{P}$ and carries the noisiest marginals, receive more samples at the same total budget. Shrinking each player's credit toward the mean of its union in the manner of Efron & Morris (1975) would trade a small bias for lower variance at small m, which is the regime that the grid of Appendix D searches ($m \in \{2, 5, 10\}$). Either device would change how often the archive rule fires by mistake at fixed k and τ, so neither is a free improvement, and we report no measurement of either.

Degenerate updates and feasibility repair. An update whose credits are all zero indicates a constant game or a failed evaluation. Its estimates still enter the running scores through Eq. 2, but the archive step is skipped, so such an update cannot archive every player at once. Because restoring is always available, archiving is not a monotone contraction of the player set. The feasibility repair of Appendix J then visits the unions in the order operators, prompts, edges; it restores the prompt of a live operator whenever one exists, and an edge only between live operators.

Trigger rationale. The first condition of the `event` branch of Eq. 7 detects a drop in the level of the reported scores, and the second an increase in their variance; the implemented constants are given in Appendix D. A head-to-head comparison of the two schedules is future work.

Order of the variant D actions. Within one update, the prompt rewrite (Algorithm 2) runs before the operator expansion (Algorithm 3), so a newly added operator is never rewritten in the update that adds it. An expansion proposal is a declarative template of a name, an instruction, and a list of edges, whose cost can be priced exactly before it is executed, unlike the executable program text searched by AFlow (Zhang et al., 2025c) and ADAS (Hu et al., 2025).

Magnitudes in the cost model. With the campaign defaults $m = 5$, $n = 20$, and $N = 4$, the update term of Eq. 8 is $5 \cdot 21 \cdot 2c = 210c$ per triggered update, against $4c$ per task for the full graph. A smaller a_t and an update rate below one per task are therefore the only routes to a crossing, and the frozen test split matters to the token ledger. An offline search such as AFlow's, by contrast, pays a one-time search cost and then one full-graph pass per task, so a sufficiently long stationary stream favors it.

B Variant D Algorithms

Variant D runs two language-model actions after the archive step of Algorithm 1, on a triggered update whose credits are not all zero. The prompt rewrite (Algorithm 2) runs first and the operator expansion (Algorithm 3) second; the live graph is then rebuilt from the possibly grown player set. Below, ϵ denotes the empty string.

Algorithm 3 Variant D, operator expansion.

Require: operators $o_1, \ldots, o_N$ with prompts and edges E; task index t; accepted expansions b; caps $b_{\max}$ and $N_{\max} = 6$; period $B_{\exp}$; acceptance threshold $\tau_{\exp}$; live score h_t

if $b \geq b_{\max}$ **or** $N \geq N_{\max}$ **or** $t \bmod B_{\exp} \neq 0$ **or** $h_t \geq \tau_{\exp}$ **then**
  return unchanged
end if
$J \leftarrow \textsc{ParseJSON}\big(\text{LLM}(\textsc{ExpandPrompt}(o_1, \ldots, o_N))\big)$ ▷ {`name, prompt, edges`}
if J is malformed **or** J.name is not a new valid name **or** J.prompt $= \epsilon$ **then**
  return unchanged
end if
$o_{N+1} \leftarrow J$.name; $\pi_{N+1} \leftarrow J$.prompt
$E_{\text{new}} \leftarrow$ proposed edges whose endpoints lie in $\{o_1, \ldots, o_{N+1}\}$, without self-loops or duplicates
if $E_{\text{new}} = \emptyset$ **then**
  $E_{\text{new}} \leftarrow \{(o_i, o_{N+1}), (o_{N+1}, o_i) : i \leq N\}$ ▷ default: both directions
end if
$E \leftarrow E \cup E_{\text{new}}$; add $p_{o_{N+1}}$, $p_{\pi_{N+1}}$, and one edge player per new edge to $\mathcal{P}$; $b \leftarrow b + 1$

Prompt rewrite. Among the live operators whose prompt player is live and non-empty, Algorithm 2 selects the one whose prompt player has the lowest running score, breaking ties by operator name. It acts only if that score lies below the rewrite threshold τ_{rw} and the operator's retry budget is not spent. A single model call must return a JSON object that names the operator and its replacement instruction; a response that is malformed, names a different operator, or carries an empty instruction is rejected, and the prompts are left unchanged. Because only the text of $\pi_{i^\star}$ changes, the player identifiers, the unions, and n are identical before and after a rewrite.

Operator expansion. Four guards precede any model call in Algorithm 3: the number of accepted expansions, the operator ceiling $N_{\max} = 6$, the expansion period, and an acceptance threshold on the live score of the current task, so that expansion is attempted only while the live graph falls short on that task. A proposal is accepted only when its name is a new valid identifier and its instruction is non-empty. Proposed edges are kept only if both endpoints exist in the grown operator set, with self-loops and duplicates discarded, and a proposal that leaves no valid edge defaults to edges in both directions between the new operator and every existing one. An accepted expansion thus adds two players plus one per new edge; under the default, one expansion takes N from 4 to 5 and n from 20 to 30, as Eq. 4 gives. The new identifiers are registered in the store, so the next update credits them.

C Seed Operators

Each operator of the seed workflow is one model call with a fixed role and one seed instruction, which is its prompt player (Table 3). At run time, the call receives the role, the instruction, the task, and the draft of a predecessor, assembled in the template below. The instruction line is omitted when the prompt player is absent, the context block appears only for tasks with a context passage, and the draft block appears only when a live predecessor has already produced a draft; for MATH, the task text also states the required answer format. Angle brackets mark the parts filled at run time, and Appendix M shows the template filled in for one HumanEval task.

```
Role: <role>

<instruction>

Task:
<task>

Context:
<context>

Current draft:
<draft>

Your output:
```

The output of each call becomes the operator's draft, and the benchmark's parser extracts the final answer from the draft of the last operator to run.

Table 3: Seed operators, their roles, and their seed instructions.

Operator	Role	Seed instruction π_i
Generate	generate	*Produce a candidate solution for the task. Return only the solution.*
Reason	reason	*Check the draft. Improve reasoning if needed. Return only the solution.*
Revise	revise	*Fix errors in the draft. Return only the corrected solution.*
Test	test	*If the draft fails the task, correct it. Return only the solution.*

Variant A leaves this operator set fixed. Variant D may add operators, up to six in total, together with edges to the operators that already exist (Algorithm 3).

D Hyperparameters and Models

D.1 Search Grid

The four core hyperparameters of SPRUNE and their discrete grid are listed in Table 4; the Cartesian product has $3^4 = 81$ points. Every search run draws points from this grid uniformly without replacement under a fixed seed and evaluates each point on one seed. The `bayesian` mode of the released entry point uses the same without-replacement draw as the `random` mode, so no model-based search is involved. Variant A was run with 45 to 48 points per benchmark on the 9B and 27B backbones, and variant D with 9 to 31 points; variant D searches the same four hyperparameters and keeps its mechanism parameters at their defaults (Appendix D.3). No search was run on the 31B backbone, where SPRUNE (A) uses the default configuration of Table 4 over three seeds.

Table 4: SPRUNE search grid. Campaign defaults are in the last column. `trigger_mode` is `fixed_batch` at batch size one for every reported run.

Parameter	Symbol	Grid	Default
EMA rate	α	$\{0.1, 0.5, 1.0\}$	0.1
Archive threshold	τ	$\{0.01, 0.05, 0.10\}$	0.05
Streak length	k	$\{2, 3, 5\}$	3
Owen permutations	m	$\{2, 5, 10\}$	5

Table 2 reports, for each variant, the best test score over the searched configurations on the 9B and 27B backbones. As discussed in Section 5.2, this is an upper bound; a three-seed replay of the configuration selected on the validate split is not part of this submission.

The event-triggered schedule of Appendix A.3 uses a window of 15 tasks, a performance-drop threshold of 0.1, and a variance multiplier of 2.0. It is implemented but was not used for any reported result.

D.2 Executors

Table 5 lists the three executors. Each is served through llama.cpp at 8-bit quantization with an output cap of 2048 tokens, and every method uses the same executor within a column of Table 2. Qwen sampling follows the instruct model card (temperature 0.7, $\mathrm{top_p} = 0.8$, $\mathrm{top_k} = 20$, presence penalty 1.5), and Gemma 4 follows its instruction-tuned model card (temperature 1.0, $\mathrm{top_p} = 0.95$, $\mathrm{top_k} = 64$).

Hardware. Every job runs its own llama.cpp server, built with CUDA, on one NVIDIA GPU with all layers offloaded to the GPU, together with 6 CPU cores and 90 GB (9B backbone) or 100 GB (27B and 31B backbones) of host memory. Jobs were scheduled as arrays on a shared Slurm cluster and placed on whichever eligible GPU was free: an A100 with 20, 40, or 80 GB or an H100 with 80 GB for the 9B backbone, and a card with at least 40 GB for the 27B and 31B backbones. The weights are the Q8_0 GGUF files listed in Table 5. Because the GPU type varied between jobs, wall-clock times are not comparable across methods, and we do not report them as a cost measure.

Table 5: Executors. All three are served through llama.cpp at 8-bit quantization; context is the llama.cpp `--ctx-size`, and the output cap is 2048 tokens for all three.

Id	Model	Context	Temperature
`qwen3.5-9b-q8`	Qwen3.5-9B (Team, 2026b)	8192	0.7
`qwen3.8-27b-q8`	Qwen3.8-27B (Qwen Team, 2026)	32768	0.7
`gemma4-31b-q8`	Gemma 4 31B-it (Team, 2026a)	32768	1.0

D.3 Variant D Parameters

Variant D adds the parameters of its two language-model actions (Appendix B). The prompt rewrite uses a rewrite threshold of 1.0, so every live prompt is eligible, and a retry cap of three per operator. The operator expansion uses an expansion period of one update, an acceptance threshold of 1.0 on the live score, a cap of two accepted expansions, and a ceiling of six operators. These values are fixed in every reported run.

E Hyperparameter Sensitivity

Table 6 summarizes the test scores of every configuration that the hyperparameter search evaluated, and Table 7 averages them by hyperparameter value. Three observations follow.

Spread across configurations. On several cells the searched configurations span tens of points, for example from 12.2 to 80.9 on HumanEval and from 3.4 to 47.4 on HotpotQA with the 9B backbone, and at most a fifth of the configurations come within five points of the best one on HumanEval, MBPP, GSM8K, MATH, and HotpotQA with that backbone. Other cells are stable: on DROP, and on MBPP and HotpotQA with the 27B backbone, at least 70% of the configurations lie within five points of the best. On the unstable cells, the best-of scores of Table 2 therefore sit well above a typical configuration, which is why we read them as an upper bound.

Marginal effects. Averaged over benchmarks, no single value moves the score by more than three points relative to the median of its backbone and benchmark. A streak length of $k = 2$ is the weakest choice on both backbones, which is consistent with archiving on too little evidence, and $m = 10$ permutations give the largest gain on the 9B backbone.

Variant D. The best configurations of variant D exceed those of variant A on HumanEval, GSM8K, and MATH with the 27B backbone, but the median configuration of D is below that of A on most cells, most clearly with the 9B backbone (for example 16.0 against 65.6 on HumanEval), where several D configurations score close to zero. The two language-model actions therefore raise the best achievable score with the larger backbone while lowering the score of a typical configuration with the smaller one.

F Live Graphs, Variant D Examples, and Cost

This appendix summarizes the search dumps of every finished SPRUNE run: the variant A and variant D searches on the 9B and 27B backbones and the default variant A runs on all three backbones.

F.1 Final Live Graphs and Archive Activity

Table 8 reports the size of the frozen live graph and the number of archive and restore decisions per run. Three patterns stand out. First, the frozen graphs are small: variant A keeps about three of the four operators, one to two of the four prompts, and one to two of the twelve edges, so a test task costs about three model calls instead of the four of the seed workflow. Second, restoring is used in practice: every run except a handful restores at least one player, with tens of restores per run on average. Much of this activity is players whose running scores oscillate around τ; relative to the number of players of each type, prompts are archived most often and edges least often. Third, variant D prunes the 9B graphs much further, to about two operators and half an edge on average, which matches its low median scores on that backbone (Appendix E).

Table 6: Test scores (%) over all searched configurations of SPRUNE (one seed each). For each backbone and benchmark: the number of configurations, the minimum, median, and maximum test score, and, for variant A, the percentage of configurations within five points of the best. The maxima are the SPRUNE cells of Table 2.

		Variant A					Variant D			
Backbone	Benchmark	n	min	median	max	≤ 5 pts	n	min	median	max
9B	HumanEval	47	12.2	65.6	80.9	20	31	0.0	16.0	80.9
	MBPP	48	22.3	50.6	58.9	20	26	0.0	30.2	60.7
	GSM8K	47	30.6	61.0	71.3	20	18	15.6	60.1	71.1
	MATH	47	1.0	4.1	12.1	20	31	0.4	1.2	12.3
	HotpotQA	48	3.4	16.7	47.4	10	29	2.4	4.3	46.7
	DROP	48	42.9	49.3	53.4	70	27	22.1	45.9	51.1
27B	HumanEval	47	23.7	61.8	72.5	10	21	9.9	60.3	79.4
	MBPP	48	43.7	62.8	65.7	90	21	51.3	63.0	66.0
	GSM8K	45	27.0	49.0	70.0	20	9	45.2	48.3	80.7
	MATH	47	0.6	2.3	5.6	100	18	0.6	1.9	14.8
	HotpotQA	47	16.9	54.3	55.7	70	28	5.6	30.7	54.7
	DROP	48	52.1	67.7	70.7	90	22	60.9	66.4	68.2

Table 7: Marginal effect of each hyperparameter value for variant A: the mean, over all searched configurations with that value and over the six benchmarks, of the test score minus the median test score of the same backbone and benchmark (in points).

Parameter	Value	9B	27B
α	0.1	0.0	$+0.5$
	0.5	-1.0	-2.4
	1.0	-1.6	-0.8
τ	0.01	-2.6	-0.8
	0.05	0.0	-0.7
	0.10	$+0.2$	-1.4
k	2	-2.7	-2.5
	3	-0.6	$+0.3$
	5	$+0.7$	-0.7
m	2	-2.7	-1.4
	5	-1.4	-0.9
	10	$+2.4$	-0.4

F.2 VARIANT D EXAMPLES

Appendix M lists the prompts that the language model rewrote and the operators it proposed in the best variant D run of each benchmark with the 27B backbone, together with their final running scores. In the run of Figure 3(d–f), Verify, whose instruction asks for a critique rather than a solution, receives negative credit and is archived, while Compress is kept and runs last, after Revise, in the frozen graph of Generate, Reason, Revise, and Compress.

F.3 WHERE VARIANT D FAILS

Figure 5 relates the number of live operators in the frozen graph to the test score of the run, expressed as a percentage of the best run of the same backbone and benchmark; MATH is left out because all of its scores are close to zero. With the 9B backbone, 87 of the 150 variant D runs end with a single live operator, at a median of 32.5% of the best run, whereas variant A reaches a median of 83.6% with four live operators. With the 27B backbone, single-operator runs of variant D keep a median of 88.0%, and variant D more often keeps five or six operators, including those proposed by the language model.

Of the 186 variant D runs with the 9B backbone, 34 score below 2%; 27 of them are on MATH, where every method scores low, and 7 on HumanEval and MBPP. In all 34, the frozen graph con-

Table 8: Frozen live graphs and archive activity, averaged over runs. Columns give the number of runs; the live operators, prompts, and edges of the frozen graph (out of 4, 4, and 12 in the seed workflow); the archive and restore decisions per run; the share of runs with at least one restore; and the median number of live model calls per validate task. Model calls per test task equal the number of live operators.

Backbone	Runs	n	Ops	Prompts	Edges	Archive	Restore	≥ 1 restore	Calls/task
9B	A search	318	2.96	1.65	2.00	101.0	87.6	100%	33.2
	A default	18	3.00	1.00	0.72	56.6	41.3	100%	34.4
	D search	186	1.95	1.03	0.48	81.4	56.7	99%	57.5
27B	A search	300	3.23	1.42	1.46	89.8	75.9	99%	26.2
	A default	18	3.39	1.11	0.78	44.9	30.2	100%	26.7
	D search	134	3.22	1.05	1.15	110.0	87.4	99%	43.4
31B	A default	15	3.33	1.00	0.87	38.6	23.8	100%	25.0

sists of a single operator, and in 32 of them its instruction, rewritten or newly proposed, no longer asks for the solution alone, as in *Distill the provided information to its fundamental truth*. Single-operator graphs also occur among successful variant D runs, so the collapse alone does not explain the failures, and we have not tested the effect of the instruction change in isolation.

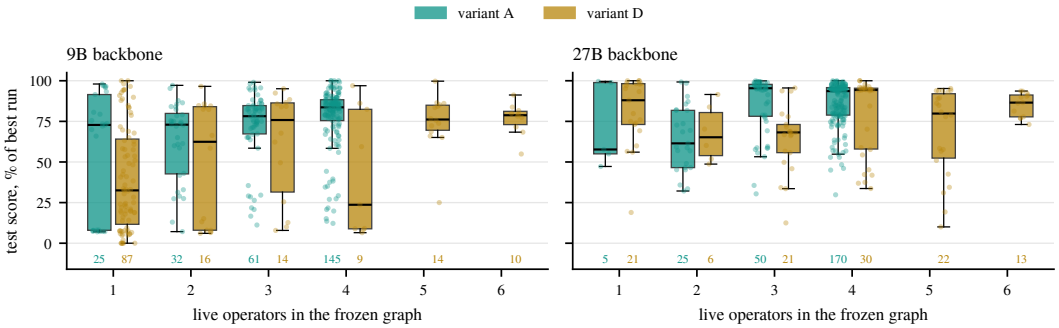

Figure 5: Test score, as a percentage of the best run of the same backbone and benchmark, against the number of live operators in the frozen graph, for all variant A and variant D runs outside MATH. Numbers below the boxes count the runs.

F.4 Cost

Model calls per test task equal the number of live operators, about three for variant A against four for the seed workflow (Table 8). The validate phase is more expensive: with memoization, variant A issues a median of 26 to 34 live model calls per validate task, about 8×10^4 tokens, and variant D issues 43 to 58 calls, about 2×10^5 tokens, likely because its rewritten and added prompts limit cache reuse. Both figures are approximate: the backend counters also include the test phase, which we subtract from the call counts as one call per live operator per test task and do not subtract from the token counts.

A matched cost comparison with the baselines is not possible from our logs. Among the baseline implementations, only AFlow prints a token count for each model call; the usage accounting of MaAS fails with the local backend, and the remaining methods do not report tokens. Counting the tokens of every method requires a proxy in front of the inference server, which we leave to future work.

G Benchmarks and splits

Validate and test are 20 percent and 80 percent of the size column at a fixed seed, AFlow's convention, applied identically to all eight methods.

Table 9: Benchmark sizes as materialized in this campaign, with the metric and one known caveat per benchmark.

Benchmark	Size	n_{val}	n_{test}	Metric and known caveat
HumanEval (Chen et al., 2021)	164	33	131	pass@1; smallest validate split, largest overfit risk, weak tests (Liu et al., 2023b)
MBPP (Austin et al., 2021)	427 (sanitized)	86	341	pass@1; sanitized split, not comparable to full-set numbers
GSM8K (Cobbe et al., 2021)	1319 (test)	264	1055	solve rate; near-saturated, contamination (Zhang et al., 2024b; Mirzadeh et al., 2025)
MATH (Hendrycks et al., 2021)	605 (L5, 4 cat.)	119	486	solve rate; risk of confusion with the MATH-500 subset (Lightman et al., 2024)
HotpotQA (Yang et al., 2018)	1000 (sampled)	200	800	F1; shortcut-solvable subset (Min et al., 2019), which later multi-hop benchmarks were built to avoid (Trivedi et al., 2022)
DROP (Dua et al., 2019)	1000 (sampled)	200	800	F1; numeric and date grading fragility

H BASELINES

Every baseline runs from its official repository: AFlow (Zhang et al., 2025c), `https://github.com/FoundationAgents/AFlow`; ADAS (Hu et al., 2025), `https://github.com/ShengranHu/ADAS`; GPTSwarm (Zhuge et al., 2024), `https://github.com/metauto-ai/GPTSwarm`; AgentSquare (Shang et al., 2025), `https://github.com/tsinghua-fib-lab/AgentSquare`; DyLAN (Liu et al., 2023c), `https://github.com/SALT-NLP/DyLAN`; MaAS (Zhang et al., 2025a), `https://github.com/bingreeky/MaAS`; and HarnessX (Chen et al., 2026), `https://github.com/Darwin-Agent/HarnessX`. All seven share the backbone of Section 5, and none of them is given a larger or a different executor than SPRUNE. Vendor trees are left unmodified: copies and search dumps are written to a separate workspace directory.

Table 10: Native coverage, entry point, and budget per baseline. A cell outside a method's native coverage is evaluated through the shared unified-task graph on the same validate and test splits, and is recorded as such in the released job records.

Method	Native benchmarks	Entry point and budget
AFlow	all six	stock `run.py`; `--max_rounds 20`, `validation_rounds=5`, official `check_convergence` left on; then the official test evaluation
ADAS	DROP, GSM8K (via MGSM)	`--n_generation 30`, then evaluate
DyLAN	HumanEval, MATH	official HumanEval parts 0–3; listwise chain-of-thought configuration for MATH
GPTSwarm	HumanEval	`experiments/run_humaneval.py`, not the optimized-swarm entry point, which targets a different benchmark
MaAS	HumanEval, GSM8K, MATH	`--round 1 --sample 4`, Graph then test mode
AgentSquare	none of the six	unified-task graph over AgentSquare's four module types (planning, reasoning, tool use, memory)
HarnessX	none of the six	unified-task graph over HarnessX's nine-dimension pipeline

Table 10 lists the coverage and the budget we give each method. Where a vendor ships a native runner for a benchmark we use it at the published budget, which is not always the vendor command-line default: AFlow's `validation_rounds` is set to 5, the value used in the AFlow paper and README, rather than the command-line default of 1, and its own `check_convergence` stopping rule is left at the vendor default. The AFlow test phase copies the best validate graph into the

first round of the test workspace and then runs the official test evaluation, a line upstream leaves commented out. No task-count truncation is applied to any split.

Two conventions are needed to read AFlow's round logs. AFlow's Graph phase is a search mode, and the quantity it logs per round is the benchmark metric on validate for that round's workflow, not a separate quantity. A round that scores 0.0 indicates a broken generated `graph.py`, for example one in which a required operator was never constructed, rather than a meaningful measurement, and a later round can recover; the validate score we record for such a run is therefore the best round rather than the last.

One difference between our setup and AFlow's published one deserves emphasis, because it affects how AFlow's position in Table 2 should be read. AFlow's published Table 1 pairs a strong proprietary optimizer, Claude-3.5-Sonnet, with a `gpt-4o-mini` executor. Our reproduction instead gives all eight methods the same fixed local backbone, which is what makes a difference in score attributable to the method rather than to the budget of the model driving the search. The consequence is asymmetric across methods: AFlow's search operator asks a model to write and revise executable workflow code, and the quality of that search degrades disproportionately when the model doing the writing is weakened, whereas methods that select among predefined components degrade more gracefully. AFlow is therefore not expected to reproduce its published standing here, and we do not read that as evidence against AFlow as a method. An exact reproduction of the original configuration is in any case unavailable to us: the two Claude model identifiers named in AFlow's published setup no longer resolve, so the optimizer of record cannot be instantiated.

Two artifacts in the repository are kept outside every table in this paper. AFlow's published table is retained for reference only and is never mixed with our measurements. A separate probe on hosted models, run on AFlow's initial workflow to separate model strength from method strength, is likewise excluded from every table here, because it evaluates the validate split at a single round and is not comparable to the test-split results we report.

I Results Across Backbones

Figure 6 plots the cells of Table 2 for each method across the three backbones. The horizontal axis is not a pure scale axis: the 31B backbone belongs to a different model family, and on it SPRUNE (A) uses its default configuration instead of the best searched one (hollow markers).

Three observations follow. First, only GPTSwarm improves from one backbone to the next on almost every benchmark: 10 of its 11 changes of more than one point are gains. The other methods change unevenly; AgentSquare, the strongest baseline, loses 20.6 points on HumanEval and 10.7 points on MATH from the 9B to the 27B backbone. Second, MBPP is the benchmark on which nearly every method gains with the larger backbones; SPRUNE (A) gains 6.8 and then 7.0 points but stays below the baselines. Third, from the 9B to the 27B backbone, SPRUNE (D) gains on five of the six benchmarks and loses 1.5 points on HumanEval, whereas SPRUNE (A) gains on MBPP, HotpotQA, and DROP and loses on HumanEval, GSM8K, and MATH. The drop of SPRUNE (A) on HotpotQA with the 31B backbone comes with a seed standard deviation of 16.2 points under the default configuration (Table 2), in line with the configuration sensitivity reported in Appendix E.

J Details Deferred from the Main Text

Edge representations in prior work. Graph-based methods (Zhuge et al., 2024; Zhang et al., 2025a) use hierarchical, sequential, or parallel relations among nodes. Neural-network representations (Liu et al., 2023c) introduce non-linear links so that the wiring can adapt to the input during the run. Code-based representations (Hu et al., 2025; Zhang et al., 2025c) express edges as a program that can include sequences, conditions, and loops. SPRUNE uses the first kind and does not search over executable control flow.

Seed operators and their origins. Generate produces a candidate solution. Reason checks the draft by step-by-step reasoning, extending chain-of-thought prompting (Wei et al., 2022). Revise fixes errors in the draft, following the self-refinement paradigm (Madaan et al., 2023; Shinn et al., 2023). Test checks whether the draft satisfies the task and corrects it if it fails, with the check performed by

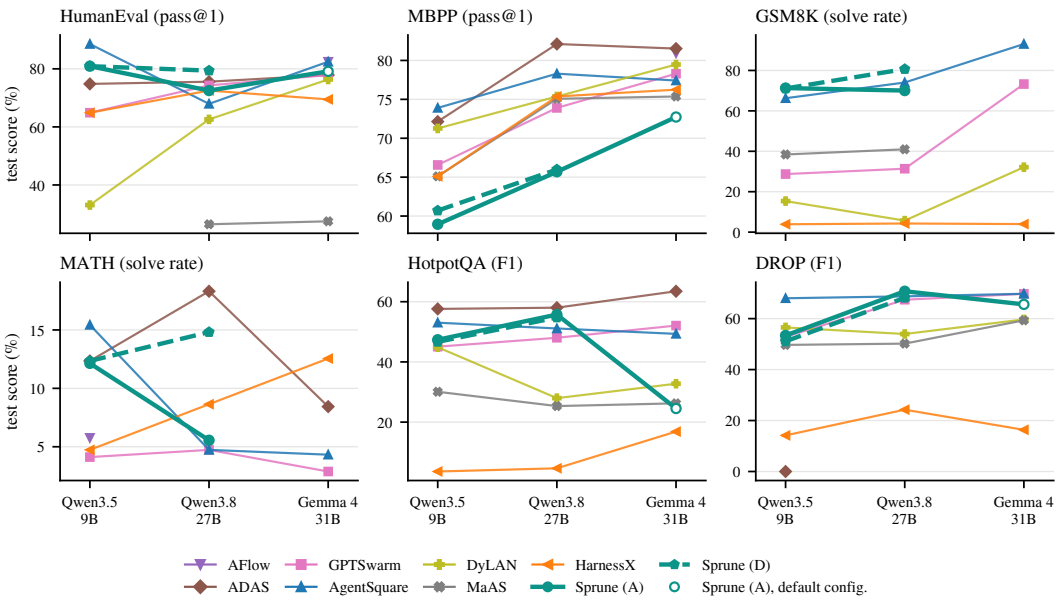

Figure 6: Test scores of every method across the three backbones, taken from Table 2. Missing points are unfinished runs. Hollow markers are SPRUNE (A) with its default configuration, the only SPRUNE runs on the 31B backbone.

the model rather than by executing code. Their prompt template and seed instructions are given in Appendix C.

Cost of exact Owen values. Exact Owen values sum over every coalition that is compatible with the three unions. For the seed player set this amounts to about 1.6×10^4 workflow executions per task, almost all from the twelve-edge union, against $m(n+1) = 105$ queries for the sampled estimate at $m = 5$. The sampled estimates sum exactly to $v_T(\mathcal{P}) - v_T(\emptyset)$, as the Owen value does; the rescaling of Eq. 6 only guards against floating-point drift.

Safeguards of the archive step. If every credit of an update is zero, which signals a constant game or failed evaluations, the archive step is skipped for that update, although the zero estimates still enter the running scores. A feasibility repair then keeps each union non-empty: if no operator remains live, the operator with the highest running score is restored; if no prompt remains live, the prompt of the first remaining operator in seed order is restored; and if at least two operators remain but no edge between them is live, the first such edge in seed order is restored.

Variant D actions. The prompt rewrite considers the live operators whose prompt player is live and non-empty and is skipped when the lowest running score is at least the rewrite threshold or the operator's retry budget is spent. The operator expansion is subject to a limit on accepted expansions, a maximum of six operators, and an acceptance threshold on the live score of the current task. Both run in this order within one update (Appendix B), with the parameters of Appendix D.3.

Scheduling. Besides the default `fixed_batch` schedule, an event-triggered schedule fires when recent scores drop or their variance grows beyond a threshold (Appendix A.3); it was not used for any reported result.

K EXTENDED RESULTS

Results by task family. On code generation, AgentSquare leads HumanEval on the 9B backbone and shares the lead with AFlow on the 31B backbone, where SPRUNE is second on both (80.9 and 79.1); on the 27B backbone, SPRUNE (D) is first with 79.4. MBPP is the weakest benchmark for SPRUNE, which trails most baselines there on all three backbones. On mathematical reasoning, SPRUNE leads GSM8K on the 9B and 27B backbones (71.3 and 80.7), ahead of AgentSquare (66.3 and 74.0). Absolute MATH scores stay below 20 for every method with these backbones, so we compare them only across methods; there, SPRUNE (D) is second on the 27B backbone. On question

answering, ADAS leads HotpotQA on all three backbones and AgentSquare leads DROP on the 9B and 31B backbones, while SPRUNE (A) is first on DROP (70.7) and second on HotpotQA (55.7) on the 27B backbone. Across backbones (Appendix I), only GPTSwarm improves on almost every benchmark, while AgentSquare loses 20.6 points on HumanEval from the 9B to the 27B backbone.

Missing cells. AFlow finished only 3 of its 18 cells, and ADAS, MaAS, DyLAN, and SPRUNE also have unfinished cells, including every SPRUNE (D) cell on the 31B backbone. The counts in Section 5.2 are therefore taken over the cells that each method finished.

Further credit analysis. Summed over players, the operator union receives between 52% and 100% of the total credit on every benchmark except MATH. Test leads among the operators on HumanEval with the 9B backbone and on MBPP and HotpotQA with the 27B backbone, and Reason leads on GSM8K with the 9B backbone. The stability of the rankings is measured on the default runs of the 27B and 31B backbones as the mean Spearman correlation between the final running scores of the eight operator and prompt players in pairs of seeds. In variant A, 92.3% of the players that are live at the end had been archived at some point. In variant D, an operator proposed by the language model ends with the highest running score in 30% of the runs with the 9B backbone and in 24% with the 27B backbone. We do not measure the score that an irreversible rule would reach.

L EXTENDED LIMITATIONS

First, the SPRUNE cells on the 9B and 27B backbones report the best test score over the searched configurations, each on one seed, whereas each baseline runs one configuration; they are an upper bound, and a three-seed replay of the configuration selected on the validate split is the protocol-conformant number. Second, several cells are missing, most of them for AFlow, and absolute MATH scores are low for every method, so we compare MATH only across methods. Third, running scores start at zero, below τ, which archives most players during the first k updates; restoring repairs this, but initializing the scores at τ would avoid it, and players whose scores stay near τ are archived and restored often. Fourth, we did not test the faithfulness of the credit directly, for example by removing high- and low-credit players and measuring the change in score, and we could not match the token cost of the baselines, whose implementations do not report tokens uniformly (Appendix F). Fifth, all backbones are local open-weight models at 8-bit quantization, and whether the comparison holds for frontier models is unknown. Finally, CRACT is a descriptive frame: naming the design space does not show that every instance is effective, and we evaluate only two points in it. The three-seed replay, the missing cells, a faithfulness test, and a token-counting proxy for all methods are the next steps, followed by experiments under distribution shift, where the continued credit updates of a CRACT loop are meant to help.

M PROMPT EXAMPLES

This appendix shows what the operators see and what the language model writes in variant D, in the spirit of the prompt listings of prior workflow-search papers.

M.1 ONE OPERATOR CALL

The first validate task of HumanEval, as Generate receives it in the seed workflow; the instruction line is the prompt player of Generate (Appendix C).

```
Role: generate

Produce a candidate solution for the task. Return only the solution.

Task:
def can_arrange(arr):
    """Create a function which returns the largest index of an element which
    is not greater than or equal to the element immediately preceding it. If
    no such element exists then return -1. The given array will not contain
    duplicate values.

    Examples:
    can_arrange([1,2,4,3,5]) = 3
    can_arrange([1,2,3]) = -1
    """
```

```
Your output:
```

A later operator such as Test receives the same template with its own role and instruction and a `Current draft:` block that holds the output of its most recent live predecessor. When the prompt player of an operator is archived, the instruction line is omitted and the role line remains.

M.2 Prompts Rewritten by the Language Model

Table 11 lists rewritten prompts from the best variant D run of each benchmark with the 27B backbone. Every rewrite keeps an explicit instruction to return only the solution, unlike the instructions of the failed runs on the 9B backbone (Appendix M.4).

Table 11: Prompts rewritten by the language model in the best variant D run of each benchmark with the 27B backbone, and the final status of the prompt player.

Benchmark	Operator	Rewritten instruction	Prompt
HumanEval	Reason	*Review the current draft for logical consistency and depth. Refine the argumentation to ensure robustness and clarity. Output the final refined solution only.*	archived
GSM8K	Reason	*Critically evaluate the logical consistency and validity of the current draft. Identify and resolve any gaps, contradictions, or weak arguments in the reasoning process. Refine the explanation to ensure it is rigorous, coherent, and directly supports the final solution. Return only the improved solution.*	live
DROP	Revise	*Correct any errors found in the draft and return only the revised solution.*	archived
HotpotQA	Test	*Evaluate the draft against the original task requirements. If it fails to meet them, provide a corrected version that fully satisfies the task. Return only the final solution.*	archived
MATH	Reason	*Critically evaluate the current draft for logical gaps, inconsistencies, or weak reasoning. Strengthen the argumentation by ensuring every claim is supported by sound logic and evidence. Refine the text to be coherent, rigorous, and clearly justified. Output only the final, improved solution.*	archived

M.3 Operators Proposed by the Language Model

Table 12 lists the operators that the language model proposed in the same runs, with the final running score and status of their operator players. Operators whose instruction asks for a critique rather than a solution, such as Verify and Critique, end archived in every run shown, while Compress, which returns a condensed solution, stays live in both runs where it appears. The run of Figure 3(d–f) is the HumanEval row.

M.4 Instructions in Failed Runs

In the lowest-scoring variant D runs with the 9B backbone, the frozen graph consists of a single operator whose instruction no longer asks for the solution alone (Appendix F). Three examples, each from a run that scores zero on the test split: Generate on MBPP, rewritten to *Develop a detailed, step-by-step solution approach to address the specified task.*; Reason on MBPP, rewritten to *Critically evaluate the provided draft for logical fallacies, contradictions, or unsupported claims; then reconstruct the argument to maximize clarity, coherence, and validity before presenting the final conclusion.*; and a proposed operator on HumanEval, Abstract, with *Distill the provided information to its fundamental truth, stripping away specific examples and contextual details to reveal the overarching universal principle.*

N Additional related work

This appendix expands the neighbors named in Section 1 that the main text compresses, and closes with what closing the attribution loop costs.

Table 12: Operators proposed by the language model in the best variant D run of each benchmark with the 27B backbone, with the final running score and status of their operator players.

Benchmark	Operator	Instruction written by the language model	S_p	Status
HumanEval	Verify	*Analyze the output from 'Reason' and 'Test'. If inconsistencies or errors are found, generate a detailed critique explaining the flaws. If the output is valid, confirm its correctness and finalize the result.*	-0.101	archived
HumanEval	Compress	*Condense the input text to its core semantic essence while preserving all critical information and logical structure.*	0.267	live
GSM8K	Validate	*Verify the logical consistency and factual accuracy of the current state before proceeding to execution.*	0.012	live
GSM8K	Refine	*Edit the text for maximum clarity and concision by removing redundancies and resolving ambiguities, while strictly preserving the original meaning.*	0.014	live
MBPP	Verify	*Critically evaluate the current output for factual accuracy, logical consistency, and adherence to constraints. Identify any errors or ambiguities.*	0.000	archived
HotpotQA	Critique	*Analyze the input text to identify logical inconsistencies, factual errors, and stylistic flaws. Provide a structured list of specific issues along with actionable recommendations for improvement.*	-0.005	archived
MATH	Compress	*Condense the provided reasoning steps into a concise, high-density summary that retains essential logical inferences and final conclusions while removing redundancy and unnecessary elaboration.*	0.053	live

Workflow search methods in detail. Automated workflow search casts an agent system as a directed graph and looks for a higher-performing topology and prompts. GPTSwarm (Zhuge et al., 2024) trains a REINFORCE policy over communication-edge weights and, in parallel, rewrites prompts. By contrast, MaAS (Zhang et al., 2025a) avoids committing to one graph: it keeps a supernet and draws a query-specific subgraph. A modular alternative is AgentSquare (Shang et al., 2025), which evolves planning, reasoning, tool, and memory blocks and uses an in-context predictor to skip a full run of low-ranked combinations. In a trace-driven loop, HarnessX (Chen et al., 2026) starts from execution traces and applies typed insert, replace, or delete steps, keeping only accepted edits. AFlow (Zhang et al., 2025c), in turn, grows candidate workflows by Monte Carlo tree search. Unlike those graph edits, ADAS (Hu et al., 2025) synthesizes new agents as programs. Last in this set, DyLAN (Liu et al., 2023c) collects peer ratings among agents while a task is running and drops the low-rated agents, so the team shrinks before the task ends.

Team and topology selection. A second decision in the same literature is the team and its communication topology. G-Designer (Zhang et al., 2024a) builds a task-conditioned communication topology. Rather than designing that topology, EvoRoute (Zhang et al., 2026) routes each subtask to an agent using past runs. A cheaper alternative is AgentDropout (Wang et al., 2025), which drops agents at inference time to cut tokens. In each case, the chosen team or wiring is then reused on later queries. Taken together, automated workflow search returns a stronger graph or a cheaper team. It leaves open which operators, prompts, and edges drive the outcome.

Attribution methods in detail. Attribution asks which part of a finished workflow drives the outcome. Cooperative-game values (Shapley et al., 1953) assign each feature a contribution to a model's prediction (Lundberg & Lee, 2017). The same principle of local attribution extends to agent systems: rather than asking which input feature drives a single prediction, one asks which operator, prompt, or edge drives the workflow's score. AgentsMatter (Lu et al., 2026a) estimates each agent by removal-based Shapley values. In benchmark form, CapaBench (Yang et al., 2025a) turns that estimator into a Monte Carlo test of capability attribution. ShapleyFlow (Yang et al., 2025b) uses the same family of values to ask which module to upgrade while the workflow skeleton stays fixed. After that ranking, some methods also apply the change, for example by swapping a backbone or rewriting a prompt while the rest of the workflow stays as it was.

When the players are prompts rather than agents, the same pattern appears. Prompt Valuation (Liu et al., 2023a) assigns a value to each prompt in an ensemble and marks which help or hurt, without modifying the workflow graph. ADOPT (Zhao et al., 2025) uses Kernel SHAP per step and spends more of the rewrite budget on high-contribution prompts. CANTANTE (Zehle, 2026) contrasts rollouts of several joint configurations to derive per-agent prompt updates, and temporal–structural credit assignment (Li et al., 2026) focuses prompt updates on the rounds and roles that constrain the outcome; both act on prompts only. On the agent side, HiveMind (Xia et al., 2026) applies DAG-Shapley to agents and then rewrites only the weakest agent's prompt, leaving operators and edges untouched. Where attribution does drive an action, it therefore credits one component type and acts on agents or prompts; no method jointly credits operators, prompts, and candidate edges and acts on the structure they form.

Procedural Graphs. Lu et al. (2026b) maintain a procedural guidance graph for a single agent and evolve it offline for a fixed number of rounds, accepting a candidate only when it passes a validation gate, after which the graph is frozen. Two differences matter here. The unit of optimization is sequential decision guidance for one agent rather than the topology of a multi-agent workflow, so there is no analogue of our edge union; and credit is assigned by contrasting trajectories and by asking a model which guidance helped, rather than by a cooperative-game solution concept, so there is no axiomatic guarantee and no scalar per-component score that accumulates across tasks.

MetaGen. Wang et al. (2026) report self-evolving roles and topologies with inter-task accumulation and a role cache, and describe the method as training-free, which places it close to SPRUNE on two axes. We keep our claims narrow: the available description does not specify the attribution signal at a level that would let us state how credit reaches an individual role or connection, so we claim neither that it overlaps with our contribution nor that it does not.

G-Designer, EvoRoute, and AgentDropout. Zhang et al. (2024a) generate task-conditioned communication topologies, Zhang et al. (2026) route among agents from accumulated experience, and AgentDropout (Wang et al., 2025) removes agents dynamically to save tokens, sharing our efficiency motivation. All three change which components participate, which makes them the closest structural analogues to our pruning step. The distinction is again the decision rule. None of them computes a marginal-contribution value with the symmetry and dummy-player properties of the Shapley (Shapley et al., 1953) or Owen (Owen, 1977) value, so a component can be dropped for correlating with low performance rather than for contributing little; and none keeps a reversible archive that lets a dropped component return on later evidence.

Memory-centric continual methods. MemGPT (Packer et al., 2023) treats the context window as the top tier of a managed memory hierarchy and lets the model page information in and out through tool calls; MemEvolve (Zhang et al., 2025b) meta-evolves the memory system itself; and Xiong et al. (2026) meta-learn agentic memory designs. All three target persistence across tasks, which is the property that separates SPRUNE from episodic offline search, and they differ from us in what persists: a memory representation and its update rule in their case, a credit vector over operators, prompts, and edges in ours. Persistent memory changes what an operator sees, whereas persistent attribution changes which operators exist. The two are compatible rather than competing, and we have not explored the combination.

Harness-level optimization. Meta-Harness (Lee et al., 2026) optimizes a model harness end to end, and WHALE (Kim et al., 2026) alternates harness search with weight learning so that harness and model co-adapt, which places WHALE outside the training-free regime. HarnessDev (Wu et al., 2026) contributes a benchmark rather than a method and reports that free-form self-rewriting is unstable, a result we read as support for constraining the edit space: SPRUNE archives, restores, or proposes a declaratively described component, and every change traces back to a numeric credit estimate.

VeRO. Ursekar et al. (2026) supply an outer harness for harness optimization: versioned snapshots of a target agent, budget-controlled evaluation, structured execution traces, and a benchmark suite of target agents and tasks, used to compare optimizers against each other. That makes VeRO infrastructure and measurement for the problem SPRUNE addresses, not a competing credit signal.

It scores a whole edited agent against a reference procedure; SPRUNE scores an operator, a prompt, or an edge inside one agent. The two compose: a per-component credit rule is exactly the kind of optimizer such a harness is built to evaluate.

DSPy. Khattab et al. (2023) compile declarative model calls into self-improving pipelines, optimizing prompt content and few-shot demonstrations against a program-level metric. The optimized object overlaps with our prompt union, so DSPy is the natural comparison for a reader who interprets SPRUNE's prompt scores as prompt optimization. Two differences follow. Compilation happens once and the compiled program is then fixed; and the metric is evaluated on the program as a whole, so an improvement cannot be attributed to a particular module, prompt, or edge. SPRUNE's scores are per-component by construction and are re-estimated as the task stream continues.

Hand-designed multi-step pipelines are the objects we price. On the multi-hop reading benchmarks in our suite, the strongest published systems are not workflow search methods at all but hand-designed multi-step pipelines: interleaving retrieval with chain-of-thought reasoning (Trivedi et al., 2023), iterative query generation (Qi et al., 2019), multi-hop dense retrieval (Xiong et al., 2020), decomposition into self-asked follow-up questions (Press et al., 2023), and graph-structured long-term memory (Gutiérrez et al., 2024). We do not compare against them, because they are workflows rather than methods for searching over workflows. They are worth naming here for a different reason: each is exactly the kind of fixed scaffold of operators, prompts, and edges whose components SPRUNE's player abstraction is built to price, and each was assembled by hand from an intuition about which step earns its keep. That intuition is the quantity this paper estimates.

Attribution without an optimizer. CapaBench (Yang et al., 2025a), Agents that Matter (Lu et al., 2026a), the Shapley analysis of agentic workflows of Yang et al. (2025b), and prompt valuation (Liu et al., 2023a) all produce attribution scores without acting on them. ADOPT (Zhao et al., 2025) does act, but on the allocation of optimization effort across the steps of a prompt pipeline rather than on structure, using a Shapley-based allocation rule in the lineage of Lundberg & Lee (2017); we discuss ADOPT from its paper alone, having located no official code release. Read as measurement instruments, all of these are complementary to this work rather than competing with it.

What closing the loop costs. The step SPRUNE adds to the instruments above is closure: an attribution estimate changes the workflow that produces the next estimate. Closure is also what creates the three problems this paper has to solve. The estimator must be cheap enough to run on a live task stream rather than inside an offline search budget; structural decisions must be reversible, because an early estimate is noisy, which is why components are archived instead of deleted; and the extra coalition evaluations have to pay for themselves, which is the break-even question of Eq. 9 and Appendix F. None of the work discussed above faces all three at once, because none of it closes the loop on structure after the search returns.