UNDERSTANDING REASONING UNCERTAINTY IN WORKFLOW OPTIMIZATION FOR SELF-EVOLVING AGENTS: A STOCHASTIC SEARCH PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Workflow optimization for self-evolving agents uses execution feedback to search for better task-execution procedures under a limited budget. Uncertainty-guided reasoning methods, adapted to this search, concentrate proposals by lowering reasoning uncertainty, whereas program-search optimizers judge each candidate only by its own evaluation. Neither suffices: lower reasoning uncertainty is not better search, since concentration can discard every high-quality workflow, and per-candidate scores cannot show which unevaluated workflows are safe to skip. We give, to our knowledge, the first mathematical framework for reasoning uncertainty in workflow optimization, modeling it as stochastic search over history-conditioned proposal distributions, with reasoning uncertainty as their entropy over functionally distinct workflow groups and uncertainty-guided methods as operators that reshape them. Concentration helps only if it retains target-quality workflows, which we certify with a hypergraph over design choices: it pools evaluations across workflows sharing combinations of choices, bounds unevaluated workflows, and skips only groups whose bounds rule out any target. With high probability, this pruning retains every target workflow in its certified domain and reduces the expected draws to reach one; under sufficient conditions, it returns a workflow better than the unpruned search's at the same budget. On five systems tasks, it improves task scores over OpenEvolve by 4.7%–18.2%, outperforming uncertainty-guided variants.

1 INTRODUCTION

Workflow optimization for self-evolving agents uses execution feedback to optimize the procedures agents use to execute tasks (Hu et al., 2025; Zhang et al., 2025). Such workflow-level self-evolution provides a building block for recursive self-improvement (RSI), where the improvement mechanism itself becomes a target of optimization (Zelikman et al., 2024; Zhang et al., 2026). Automating the optimizations reduces manual workflow design, supports adaptation to different tasks, and improves the reusable procedure behind future responses (Hu et al., 2025; Zhang et al., 2025). However, realizing this automation requires ensuring *generality* across optimization methods, controlling the *complexity* of evaluations and total cost, and achieving *superiority* in returned workflow quality under a common budget (Yue et al., 2026).

Existing methods for automating workflow optimization fall into four categories, depending on the optimization targets: (1) Code-based self-improvement lets an agent rewrite its own code, including the code that carries out the improvement (Zelikman et al., 2024; Yin et al., 2025; Zhang et al., 2026); (2) Node-based optimization fixes the workflow scaffold and tunes local configurations such as prompts and demonstrations (Agrawal et al., 2026); (3) Graph-based optimization searches over workflow structures using execution feedback (Hu et al., 2025; Zhang et al., 2025); and (4) Others include textual-gradient optimization (Yuksekgonul et al., 2025) and program search with executable evaluation (Cheng et al., 2025; Sharma, 2025; Novikov et al., 2025). However, these optimizers judge each candidate only by its own evaluation, so they cannot tell which unevaluated workflows are safe to skip, even when those workflows share design choices with evaluated ones. A natural remedy is to concentrate proposals by lowering *reasoning uncertainty*, which correlates closely with agent performance in task execution (Han et al., 2024; Yin et al., 2024). Yet lower reasoning un-

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

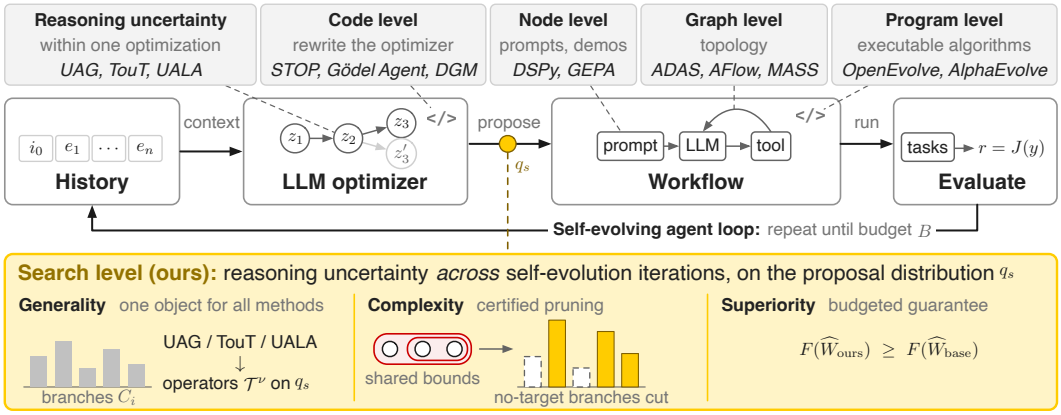

Figure 1: Overview of workflow optimization for self-evolving agents. **Top:** proposals and evaluation feedback form a search loop; gray tags indicate the levels targeted by prior work. **Bottom:** lowering reasoning uncertainty alone can discard high-quality workflows; we reshape the proposal distribution through one operator interface (generality), prune with pooled certified bounds (complexity), and return, with high probability, a workflow better than the baseline's at the same budget (superiority).

certainty is not better search: concentration without quality evidence can discard every high-quality workflow (Section 5.1). Moreover, reasoning uncertainty is defined differently across studies (Liu et al., 2025b) and remains open to model in self-improving agents (Oh et al., 2026), so uncertainty-guided methods (Yin et al., 2024; Mo & Xin, 2024; Han et al., 2024) cannot be compared directly. Figure 1 summarizes this gap: existing studies act on workflow components or on reasoning uncertainty within a single optimization attempt, but none certifies which workflows the search can skip across self-evolution iterations. We therefore ask this research question: *How can workflow optimization concentrate its proposals without discarding high-quality workflows, across methods (generality), with fewer evaluations (complexity), and with better returned workflows under a common budget (superiority)?*

Our approach is inspired by stochastic search theory: the sampling distribution of candidates (here, workflows) determines both the effort to reach good solutions and the quality of the returned solution (Spall, 2003; Zabinsky, 2003). Workflow optimization fits this view: the optimizer samples each proposed workflow from a distribution conditioned on the observable search history, as in adaptive random search (Solis & Wets, 1981). Reasoning uncertainty can then be read as the dispersion of this proposal distribution across semantic workflow branches, tying it to the same distributions that govern search effort and returned quality. Proposal distributions therefore provide a common object for a unified framework, turning the three challenges into questions about it: *generality* asks whether different methods can be expressed as transformations of these distributions, *complexity* how concentrating them changes the effort to reach good workflows, and *superiority* whether this change carries over to returned workflow quality under a common budget.

To ensure *generality*, we treat each uncertainty-guided method as reshaping the proposal distribution in a common space. Search histories live in standard Borel spaces, which support conditional proposal laws (Berberian, 1988; Kallenberg, 2021), and program proposals are mapped to workflows and conditioned on validity (Section 3). Each method then acts as a state-dependent operator on workflow distributions (Section 4), and reasoning uncertainty is the Shannon entropy (Shannon, 1948) of branch probabilities over a fixed semantic partition (Section 5.1), so methods can be compared by allocation effects, target retention, and resource use.

To control *complexity*, lowering reasoning uncertainty is not enough: concentrating proposals saves effort only if high-quality workflows are retained. We certify retention with a hypergraph over interacting workflow choices: under explicit modeling assumptions, a confidence ellipsoid (Abbasi-Yadkori et al., 2011) yields simultaneous quality bounds that let us exclude branches without losing target workflows, which reduces the evaluations needed to reach one (Section 5).

To achieve *superiority*, fewer expected evaluations are not enough, since returned quality depends on an adaptive search under a fixed evaluation budget. Branch-and-bound certificates close this gap:

the restricted search skips baseline candidates certified to be no better than the seed workflow and, under explicit sufficient conditions, returns with high probability a workflow that outperforms the baseline's under the same budget (Section 5.3).

Overall, our contributions are as follows:

- We propose the first unified framework that models workflow optimization as stochastic search over proposal distributions, with reasoning uncertainty as branch entropy and uncertainty-guided methods as operators on these distributions, making methods directly comparable and ensuring *generality*.
- We propose a hypergraph search operator that prunes branches while preserving target workflows. Under explicit sufficient conditions and with high probability, its expected target-hitting draw-count ratio equals the exponential of the branch-entropy change under uniform branch allocation, and it returns a workflow that outperforms the baseline at the same budget, controlling *complexity* and achieving *superiority*.
- On five systems optimization tasks (Cheng et al., 2025), our method outperforms OpenEvolve and its uncertainty-guided variants at matched budgets, improving task scores over OpenEvolve by 4.7%–18.2%, with the largest gain on *LLM–SQL reordering*.

2 Related Work

Recursive Self-Improvement. In RSI, the improvement mechanism itself becomes an optimization target, allowing optimizations to change how subsequent agent updates are generated or selected (Zelikman et al., 2024; Yin et al., 2025). Following the agent-harness taxonomy of Liu et al. (2026), prior studies fall into three categories: (1) Module-based self-improvement updates individual components such as memory, skills, tools, prompts, and context, *e.g.*, Promptbreeder (Fernando et al., 2024); (2) Orchestration-based self-improvement searches over how components are composed and executed, such as agentic units and workflow topologies, *e.g.*, ADAS (Hu et al., 2025) and AFlow (Zhang et al., 2025); and (3) Code-based self-improvement lets an agent rewrite its own source code or scaffold, including the code that carries out the improvement, *e.g.*, STOP (Zelikman et al., 2024), Gödel Agent (Yin et al., 2025), SICA (Robeyns et al., 2025), and DGM (Zhang et al., 2026). Workflow optimization for self-evolving agents falls within the first two categories but follows a fixed optimization procedure.

Workflow Optimization for Self-Evolving Agents. Workflow optimization for self-evolving agents uses execution feedback to generate and select better workflows under a finite evaluation budget (Hu et al., 2025; Zhang et al., 2025). Following Yue et al. (2026), prior static optimization studies, which fix the workflow before deployment, fall into four categories: (1) Node-based optimization tunes local configurations such as prompts and demonstrations within a fixed scaffold, *e.g.*, DSPy (Khattab et al., 2024) and GEPA (Agrawal et al., 2026); (2) Graph-based optimization searches a constrained space of workflow structures offline with execution feedback, *e.g.*, AFlow (Zhang et al., 2025) and ADAS (Hu et al., 2025); (3) Joint node- and graph-based optimization updates both together or alternately, *e.g.*, MASS (Zhou et al., 2026); and (4) Others include textual-gradient optimization of selected variables, *e.g.*, TextGrad (Yuksekgonul et al., 2025), and program search with executable evaluation for algorithms and systems, *e.g.*, FunSearch, AI-Driven Research for Systems (ADRS), OpenEvolve, and AlphaEvolve (Romera-Paredes et al., 2024; Cheng et al., 2025; Sharma, 2025; Novikov et al., 2025). These approaches turn past evaluations into new candidates with different representations and update rules. Our framework maps program candidates to workflow semantics and models changes as history-conditioned distributions, supporting analysis of conditions linking uncertainty mitigation to evaluation cost and returned workflow quality.

Reasoning Uncertainty. Reasoning uncertainty concerns the reliability of intermediate steps in multi-step inference (Liu et al., 2025b). Following Liu et al. (2025b), methods that quantify it fall into two categories: (1) Single-round-based methods use signals within one generation, such as token probabilities, *e.g.*, Uncertainty-aware Adaptive Guidance (UAG) (Yin et al., 2024); and (2) Multi-round-based methods use multiple sampled generations, *e.g.*, Tree of Uncertain Thoughts (TouT) (Mo & Xin, 2024). These and other methods that quantify or mitigate this uncertainty (Wang et al., 2023; Aggarwal et al., 2023; Kuhn et al., 2023; Farquhar et al., 2024; Han et al., 2024) operate within individual task executions, whereas reasoning uncertainty in workflow optimization evolves with the workflows. Applied directly, they may thus misestimate uncertainty and yield subopti-

mal workflows, and modeling such changes in interactive agents remains open (Oh et al., 2026). Our framework instead measures operational reasoning uncertainty over fixed semantic workflow branches of history-conditioned proposal distributions, so one operator formulation covers structural and allocation changes across iterations.

Branch-and-Bound. Branch-and-bound recursively partitions the search space, updates the best feasible solution found, and uses objective bounds to prune regions that cannot improve it (Land & Doig, 1960; Lawler & Wood, 1966). Its efficiency depends on how tight and cheap these bounds are, which has motivated extensive work on search, branching, and pruning strategies (Morrison et al., 2016; Gasse et al., 2019). In workflow optimization for self-evolving agents, branch-and-bound provides a principled way to focus a limited evaluation budget on promising workflow branches. With valid bounds that account for evaluation uncertainty, it eliminates branches that cannot improve the incumbent, reducing unnecessary evaluations while preserving the best attainable workflow quality.

3 Workflow Optimization for Self-Evolving Agents

We model workflow optimization for self-evolving agents as stochastic search, linking program proposals and execution feedback to the quality of the workflow returned under a fixed evaluation budget. Appendix B provides the full construction.

Programs and workflow semantics. We represent an agent's task-execution workflow by a program specifying algorithmic logic, tool use, coordination, and control flow (Hu et al., 2025). An outer optimizer generates and evaluates candidate programs, then uses execution feedback to select candidates and guide subsequent proposals, following the program-search pattern used in ADRS and OpenEvolve (Cheng et al., 2025; Sharma, 2025). To describe this search at both the program and workflow levels, let $\mathcal{C}$ denote the countable space of admitted finite-length program candidates. A semantic contract identifies the valid programs $\mathcal{P} \subseteq \mathcal{C}$ and maps each $c \in \mathcal{P}$ to its workflow semantics $\psi(c)$, yielding $\mathcal{W} := \psi(\mathcal{P}) \neq \emptyset$; distinct programs may represent the same workflow, as illustrated in Figure 2 (left). We equip these countable spaces with their discrete σ-algebras, making them standard Borel spaces for the probability model below (Berberian, 1988; Kallenberg, 2021).

Reference objective. To compare the quality of these programs and workflows, fix a reference evaluation protocol and a scoring function J, with larger scores preferred. For a valid program $c \in \mathcal{P}$, let Y_c denote its random evaluation record and define its reference quality by $f(c) := \mathbb{E}[J(Y_c)]$. Assume that programs representing the same workflow have equal reference quality. Then $F(\psi(c)) := f(c)$ defines workflow quality independently of the program chosen to represent it. During search, live evaluations instead yield random observed scores $r = J(y)$ from records y.

Observable search state. Although F depends only on workflow semantics, subsequent proposals can also depend on program source and accumulated feedback. We therefore use the observable history as the search state, $S_n := (i_0, (e_1, \ldots, e_n))$. Here i_0 records the initialization and optimization rules, and e_n is the n-th event recorded by the optimizer's controller: a proposal submission, evaluation result, scheduling decision, or archive update (Figure 2, right).

Workflow proposals. At a realized history snapshot s of S_n, let K_s denote the outcome distribution of a complete proposal attempt launched from that snapshot. Such an attempt, which may include generation retries within finite limits, returns a program or a no-program outcome $\perp_c$; extend ψ to map invalid programs and $\perp_c$ to $\perp$. To describe the resulting workflow proposals, push K_s through ψ, compute the probability of a valid outcome, and condition on validity: $q_s := \psi_{\#}K_s, \alpha_s := q_s(\mathcal{W}), q_s^+(A) := \frac{q_s(A)}{\alpha_s}, A \subseteq \mathcal{W}, \alpha_s > 0$. Here q_s includes the outcome $\perp$, while α_s is the probability of proposing a valid workflow. The conditional law q_s^+ describes how proposal probability is distributed among valid workflows.

Search dynamics. The law K_s describes a complete proposal attempt at a given state; to describe the evolving search, we also specify how controller events update that state. Let $T(S_n, \cdot)$ give the next-event distribution and let Φ append the observed event to the history:

$$e_{n+1} \mid S_n \sim T(S_n, \cdot), \qquad S_{n+1} = \Phi(S_n, e_{n+1}).$$

An attempt launched from snapshot s adds its outcome, drawn from K_s, as a later event, possibly after other events have been recorded; hence K_s is not the next-event law $T(S_n, \cdot)$. Subsequent proposal attempts use the updated history, allowing new feedback to influence their distributions.

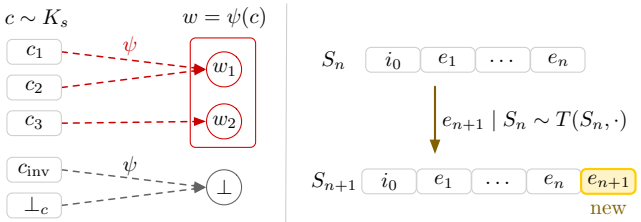

Figure 2: Two views of observable stochastic workflow search. **Left:** a complete proposal attempt launched from snapshot s has outcome law K_s. Dashed arrows illustrate the semantic map ψ: distinct valid programs can represent the same workflow, while an invalid program c_{inv} and a no-program outcome $\perp_c$ map to $\perp$. **Right:** the next-event law T describes the next recorded controller event. Appending this new event to the existing history forms S_{n+1}.

Budget and returned quality. To compare optimization methods under a common resource limit, give each method B evaluator starts after a shared pre-evaluated seed, plus finite generation and time limits that ensure termination (Appendix B.6). Evaluation retries, failures, and timeouts count toward B. Return eligibility requires a sound validity certification, unlike the analytical conditioning in q_s^+; the seed is certified. After settling outstanding evaluations, return the workflow $\widehat{W}_B$ with the highest observed score among the seed and certified evaluated candidates, with the seed as a fallback. We measure performance under such a budget by the expected reference quality of the returned workflow:

$$V(B) := \mathbb{E}\left[F(\widehat{W}_B)\right]. \tag{1}$$

4 From Uncertainty Methods to Workflow Search Operators

Under the fixed budget and return rule of Section 3, a method that changes only how proposals are generated affects $V(B)$ through the history-conditioned proposal laws it induces and the resources it consumes. We therefore adapt uncertainty-guided reasoning methods to this framework by specifying how each method transforms an input workflow distribution into an output distribution.

Program realization. At a fixed history snapshot s, let $q \in \mathsf{Prob}(\mathcal{W})$ be an input workflow law, such as q_s^+ from Section 3. To supply program inputs for workflows drawn from q, choose a measurable realization kernel $\Lambda_s(w, \cdot)$ on $\mathcal{P}$, supported on $\{c \in \mathcal{P} : \psi(c) = w\}$. Sampling a workflow from q and then a program from Λ_s gives the program input law $\mathcal{L}_s(q) := \int_{\mathcal{W}} \Lambda_s(w, \cdot)\, q(\mathrm{d}w)$ on $\mathcal{P}$; the support condition gives $\psi_{\#}\mathcal{L}_s(q) = q$, so program inputs reproduce q at the workflow level.

Method-induced proposals. For an input program c drawn from $\mathcal{L}_s(q)$, method ν encodes c, the task specification, and the available feedback in s as a request to optimize the workflow program; this bounded optimization attempt need not improve F or $V(B)$. Uncertainty signals act only within this attempt: rising token surprisal triggers UAG backtracking, value uncertainty guides TouT node selection, and answer uncertainty gates Uncertainty-Aware Language Agent (UALA) acceptance and tool use (Appendices C.2–C.4). With measurable transitions and decoding and finite attempt and duration limits, this procedure induces a terminal program-outcome kernel $\Gamma_s^\nu(c, \cdot)$ on $\mathcal{C}_\perp := \mathcal{C} \sqcup \{\perp_c\}$. Averaging this kernel over the program input law gives the program proposal law

$$K_s^\nu[q](D) := \int_{\mathcal{P}} \Gamma_s^\nu(c, D)\, \mathcal{L}_s(q)(\mathrm{d}c), \qquad D \subseteq \mathcal{C}_\perp.$$

As in Section 3, pushing $K_s^\nu[q]$ through ψ and conditioning on semantic validity gives a workflow law, which requires a positive probability of producing a valid program and thus determines the operator's domain.

Definition 4.1 (Adapted workflow search operator)**.** *For fixed realization and optimization procedures, let $\alpha_s^\nu(q) := K_s^\nu[q](\mathcal{P})$ be the probability of producing a valid program. On the admissible domain $\mathcal{D}_s^\nu := \{q \in \mathsf{Prob}(\mathcal{W}) : \alpha_s^\nu(q) > 0\}$, define*

$$\mathcal{T}_s^\nu(q)(A) := \frac{K_s^\nu[q](\psi^{-1}(A))}{\alpha_s^\nu(q)}, \qquad A \subseteq \mathcal{W}. \tag{2}$$

Role in the search. When $\alpha_s > 0$ and $q_s^+ \in \mathcal{D}_s^\nu$, applying the operator to q_s^+ gives the conditional workflow law $\mathcal{T}_s^\nu(q_s^+)$. In the comparison of Section 3, the adapted method is modeled as the optimizer whose proposal attempts have law $K_s^\nu[q_s^+]$. UAG, TouT, and UALA thus all instantiate Definition 4.1, so at a common snapshot s and input law q, their output laws $\mathcal{T}_s^\nu(q)$, validity probabilities $\alpha_s^\nu(q)$, and resource use can be compared directly (Appendix C.5).

5 Hypergraph Evidence for Workflow Optimization

Section 4 represents each uncertainty-guided method by how it changes the proposal law, but this representation does not say when the change benefits the search. We therefore construct quality evidence from joint relationships among workflow choices and turn it into a certified search operator on the same workflow-law interface (Figure 3).

5.1 From Uncertainty Signals to Joint Quality Evidence

The adapters provide *generality* at the workflow-law interface, allowing different reasoning procedures to act on a common workflow space. Their uncertainty signals, however, act only within an optimization attempt and do not by themselves bound workflow quality F. The other two goals therefore remain open: *complexity* concerns workflow evaluations and total cost, while *superiority* compares returned quality $V(B)$ under a common budget. This requires relating the changed proposals to workflow quality and resource use.

A natural summary of how a method changes proposals is their concentration across semantic branches. To quantify it, fix a finite semantic partition $(C_i)_{i=1}^m$ across methods and states and use entropy $H((q(C_i))_i)$; for $q = q_s^+$ with $\alpha_s > 0$, this is operational reasoning uncertainty $U_{\mathrm{reason}}(s)$ (Appendix E.1). Concentration alone, however, leaves the quality link unresolved: for two singleton branches with qualities 1 and 0, changing probabilities from $(1/2, 1/2)$ to $(0, 1)$ lowers entropy while eliminating every quality-1 proposal. To concentrate search while preserving targets, we therefore need evidence that excluded branches contain none of them.

Such evidence could come from evaluating every workflow separately, but this would not reuse information across combinations of shared choices. Such reuse must account for how the choices interact: for binary choices with $F(u, v) = \mathbf{1}_{\{u=v\}}$, every individual value has mean quality $1/2$ under a uniform partner, although pairs have quality 1 or 0. Marginal values therefore hide the distinction between good and bad combinations. A shared model of these joint relationships can retain combination effects while allowing evaluations to inform other workflows.

5.2 Hypergraph Representation and Shared Quality Bounds

To represent combinations that can overlap and involve more than two choices, we use a hypergraph (Berge, 1989; Zhou et al., 2006; Battiston et al., 2020). Fix categorical workflow attributes and a finite $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$: vertices encode attribute values and hyperedges encode compatible combinations. For selected-value indicators $x_v(w)$, let

$$x_e(w) := \prod_{v \in e} x_v(w), \quad \mathbf{x}(w) := (1, (x_e(w))_{e \in E_\mathcal{G}})^\mathsf{T}.$$

Each product identifies a recurs of values that recur across workflows.

To pool evaluations through these features, declare a nonempty certification domain $\mathcal{D} \subseteq \mathcal{W}$ and assume $F(w) = \mathbf{x}(w)^\mathsf{T}\theta_*$ on $\mathcal{D}$, with a known norm bound on the shared parameter. The same $1 + |E_\mathcal{G}|$ coefficients govern all workflows, so completed evaluations can constrain unevaluated workflows through their feature vectors. Each eligible evaluation contributes one feature–score pair to this joint fit. For the fit to support quality bounds, all included records must satisfy the same model, with a dictionary fixed before evidence collection, score-independent eligibility (Appendix D.1), predictable features, and conditionally calibrated scores with sub-Gaussian noise of known scale (Assumption D.1).

Under these conditions, regularized least squares gives a confidence ellipsoid with the radius of Abbasi-Yadkori et al. (2011). Projecting the ellipsoid onto workflow features and intersecting the

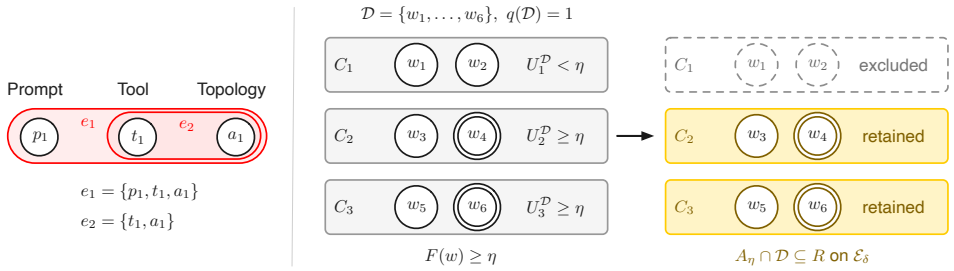

Figure 3: Three views of hypergraph evidence and certified workflow restriction. **Left:** overlapping hyperedges encode joint prompt, tool, and topology choices; each eligible evaluation contributes once to a shared quality model. **Middle:** branch upper bounds $U_i^{\mathcal{D}}$ on the best reference quality in each branch $C_i \cap \mathcal{D}$ guide exclusion from the recorded pool. Double rings mark illustrative targets $F(w) \geq \eta$, not inputs to selection. **Right:** $U_1^{\mathcal{D}} < \eta$ excludes $C_1 \cap \mathcal{D}$, while $C_2 \cap \mathcal{D}$ and $C_3 \cap \mathcal{D}$ are retained as unresolved branch domains. On the simultaneous-coverage event $\mathcal{E}_\delta$, the retained set R preserves all targets in $\mathcal{D}$.

resulting intervals over evidence times gives

$$\underline{F}_t(w) \leq F(w) \leq \overline{F}_t(w) \quad \text{for all } t,\ w \in \mathcal{D}$$

on one event $\mathcal{E}_\delta$ of probability at least $1 - \delta$, for predeclared $\delta \in (0, 1)$ (Proposition D.2). This simultaneous coverage permits choosing branches after inspecting the accumulated evidence.

5.3 Certified Allocation and Search Benefits

To turn these workflow bounds into an exclusion rule, fix a target η and $A_\eta := \{w : F(w) \geq \eta\}$. Excluding an entire branch requires a bound on its best possible quality, so at evidence time $t = t(s)$ construct $U_i^{\mathcal{D}}(s) \geq \sup_{w \in C_i \cap \mathcal{D}} \overline{F}_{t(s)}(w)$, using $-\infty$ for empty intersections. On $\mathcal{E}_\delta$, a branch with $U_i^{\mathcal{D}}(s) < \eta$ contains no target in $\mathcal{D}$. Retaining the remaining branch domains gives

$$R := R_\eta^{\mathcal{D}}(s) := \bigcup_{i : U_i^{\mathcal{D}}(s) \geq \eta} (C_i \cap \mathcal{D}), \qquad \zeta := q(R). \tag{3}$$

Hence $A_\eta \cap \mathcal{D} \subseteq R$ on $\mathcal{E}_\delta$; for an input law q with $q(\mathcal{D}) = 1$, this preserves all input target probability. We use this region to reallocate proposals by conditioning on R, optionally mixing with q to retain exploration support.

Definition 5.1 (Certified hypergraph search operator)**.** *For a supported certificate, $q(\mathcal{D}) = 1$, and a retained region R with $\zeta > 0$, choose $\varepsilon \in [0, 1)$ and define*

$$\mathcal{T}_{\mathcal{G},s}^{\varepsilon}(q)(A) := \varepsilon q(A) + (1 - \varepsilon)\frac{q(A \cap R)}{\zeta}. \tag{4}$$

Like the adapted operators of Definition 4.1, $\mathcal{T}_{\mathcal{G},s}^{\varepsilon}$ maps an input workflow law to an output law, but it reallocates according to certified quality bounds rather than native uncertainty signals, which supplies the quality evidence the adapters lack. On a recorded finite pool, its branch maxima and selection weights are directly computable, including bounds for unevaluated entries from existing evidence (Appendix D.3, Algorithm 1).

The benefit follows from target preservation. For a successful operator call with $q(A_\eta) > 0$, substituting A_η into the operator and using $q(A_\eta \cap R) = q(A_\eta)$ gives, on $\mathcal{E}_\delta$, $\frac{\mathcal{T}_{\mathcal{G},s}^{\varepsilon}(q)(A_\eta)}{q(A_\eta)} = \varepsilon + \frac{1-\varepsilon}{\zeta}$.

For independent draws from the respective frozen laws, the expected number of draws to the first target is the reciprocal of the target probability. The reciprocal of the displayed factor is therefore the expected target-hitting draw-count ratio, with strict benefit when $\zeta < 1$ (Proposition D.3).

This gain also clarifies when the concentration measure from Section 5.1 reflects search efficiency. Under target-preserving hard restriction with uniform probabilities on d positive-probability branches and k retained in full, the draw-count ratio is $k/d = e^{H(p') - H(p)}$, where p, p' are the

branch allocations before and after restriction. Thus $k < d$ lowers both reasoning uncertainty and expected draw count; general allocations need not lower entropy (Appendix E.2).

Since completed evaluations bound unevaluated workflows through shared interaction features, the excluded workflows need not be evaluated. With equal evaluation cost and a correct target test per draw, the reduction in expected draws then applies proportionally to expected evaluation cost, and total cost also decreases when certification and generation overhead stay below the evaluation cost saved (Appendix E.3). These savings address the complexity criterion of Section 5.1.

Superiority requires a further step, since $V(B)$ is defined by the adaptive search of Section 3, whose budget counts evaluator starts rather than independent draws from a frozen law. Appendix E.4 gives sufficient conditions for this step: the restricted search shares the baseline's candidate sequence with exact evaluation, skips only candidates certified to be no better than the seed, and spends no more on certification than it saves on evaluation. Under these conditions, it outperforms the baseline's $V(B)$ with exact certificates, and with the bounds of Section 5.2 its returned workflow outperforms the baseline's on $\mathcal{E}_\delta$ (Proposition E.6). The benefit chain thus runs from certified target preservation, through reduced target-hitting resources, to budgeted returned quality.

6 Evaluation

Benchmarks. We evaluate workflow optimization for self-evolving agents on five systems tasks from four *ADRS* case studies, using its public *OpenEvolve* artifacts (Cheng et al., 2025; UCB-ADRS, 2025). *Can't Be Late* (Wu et al., 2024) minimizes execution cost subject to deadlines in a single region; its ADRS multi-region extension adds region selection and migration. *Expert placement* optimizes load balance and execution overhead; *LLM-SQL reordering* improves prefix-cache reuse by ordering rows and fields with low overhead (Liu et al., 2025a); and *transaction scheduling* minimizes makespan for conflicting transactions (Cheng et al., 2024). Candidates implement workflows through fixed program interfaces and are evaluated on workload traces or problem instances, providing execution feedback to the optimizer; Appendix F details the interfaces and objectives.

Baselines. Our base optimizer is *OpenEvolve* (Sharma, 2025), with its native proposal, parent-selection, and archive-update rules under each task's declared configuration. We compare variants that adapt *UAG* (Yin et al., 2024), *TouT* (Mo & Xin, 2024), and *UALA* (Han et al., 2024) to workflow optimization through the interfaces in Section 4 and Appendix C. *Hypergraph* uses our evidence model and candidate-pool selection operator (Section 5). All methods share the generation backbone, initial program, evaluator, information sources, and resource limits.

Evaluation metrics. Our primary metric is $V(B)$, the expected reference quality of the returned workflow under a common budget B (Equation 1), estimated independently on held-out workloads. Table 1 reports available task-score improvements relative to *OpenEvolve* (shown as 100%). All analyses retain failed runs and seed fallbacks.

Experimental protocol. All methods start from the same pre-evaluated, semantically certified seed and share the evaluator, return rules, and generation and time limits. Budget B counts optimization iterations after the seed and is matched across methods for each task (Table 1).

6.1 Main Results

Relative-score improvements. Table 1(a) shows that *+Hypergraph* achieves the highest task score in all five cases, improving over *OpenEvolve* by 4.7%–18.2%. Its improvement exceeds that of the strongest uncertainty-guided variant by 2.9–10.1 percentage points, with the largest gap on *LLM–SQL reordering* (18.2% versus 8.1% for *+UAG*). The adapted methods vary across tasks: *+UAG* and *+TouT* show no gain at the reported precision on single-region *Can't Be Late*, while their gains reach 8.7% and 10.9%, respectively, on *expert placement*. Thus, adapting uncertainty-guided reasoning can improve workflow search, but the benefit depends on the task.

Runnable rate. In panel (b), *+Hypergraph* also achieves the highest runnable rate in every case, ranging from 75.0% to 93.3% and exceeding *OpenEvolve* by 12.0–16.1 percentage points. *+UAG* and *+TouT* have lower runnable rates than *OpenEvolve* across all five cases; *+UALA* exceeds it only on *LLM–SQL reordering* (69.0% versus 63.0%). Together with panel (a), these results show that

Table 1: Recorded score improvements and candidate executability. (a) Task-score improvements relative to *OpenEvolve* (shown as 100%, ↑). (b) Runnable candidates as a percentage of generated candidates for each case (↑).

(a) Relative score improvement (↑)

Method	*Can't Be Late*		*Expert placement*	*LLM–SQL reordering*	*Transaction scheduling*
	Single region	*Multi-region*			
Budget (iterations)	400	400	1000	100	100
OpenEvolve	100%	100%	100%	100%	100%
+UAG	+0.0%	+0.9%	+8.7%	+8.1%	+0.0%
+TouT	+0.0%	+1.8%	+10.9%	+7.9%	+3.2%
+UALA	+1.2%	+1.4%	+5.1%	+4.7%	+2.9%
+Hypergraph	+4.9%	+4.7%	+15.5%	+18.2%	+7.7%

(b) Runnable rate (↑)

Method	*Can't Be Late*		*Expert placement*	*LLM–SQL reordering*	*Transaction scheduling*
	Single region	*Multi-region*			
OpenEvolve	76.4%	71.9%	79.9%	63.0%	68.0%
+UAG	61.8%	59.3%	48.5%	49.0%	51.0%
+TouT	50.6%	54.8%	58.8%	52.0%	47.0%
+UALA	58.2%	61.1%	47.8%	69.0%	62.0%
+Hypergraph	90.2%	88.0%	93.3%	75.0%	80.0%

score improvement and candidate executability must be assessed separately: a higher task score can coexist with a lower runnable rate.

Case study. To illustrate joint evidence on single-region *Can't Be Late*, we extend the adaptive policy evolved by ADRS (Cheng et al., 2025, Figure 4(b)) with one check before each non-urgent switch from on-demand to spot instances. The original policy approves this switch from recent spot availability alone, yet its payoff also depends on the remaining work ω and the changeover overhead o: with on-demand and spot prices $\rho_{\mathrm{od}} > \rho_{\mathrm{spot}}$ per unit time, even a spot instance that survives until completion saves cost only if $\rho_{\mathrm{spot}}(o + \omega) < \rho_{\mathrm{od}}\,\omega$, *i.e.*, $\omega > \rho_{\mathrm{spot}}\,o/(\rho_{\mathrm{od}} - \rho_{\mathrm{spot}})$. The check therefore records context–action–outcome history rather than a longer availability window: each completed switch stores its gain under the hyperedge joining the binned values of its spot availability, remaining work, time margin, overhead, current state, and action. It vetoes a switch that violates this condition or, given enough records, whose average past gain under the same hyperedge is non-positive. For example, with $\rho_{\mathrm{od}} = 3\rho_{\mathrm{spot}}$, switches with $(\omega, o) = (10, 0.5)$ and $(1, 1)$ hours both satisfy this condition, saving 65% and 33% of the on-demand cost if the spot instance survives. A single preemption halfway, followed by a changeover back to back to an on-demand instance, still leaves the first with 27% savings but doubles the second's cost, so the first pays off at any preemption probability, whereas the second requires one below 25%. Under equally stable availability, a score summing per-factor credits would let the first switch's gains raise the second's score, whereas separate hyperedges keep the two apart, a distinction that marginal values hide (Section 5.1). Appendix G details both policies and the check.

7 Conclusion

Modeling workflow optimization as stochastic search, we showed that lower reasoning uncertainty is not better search: concentration helps only if it retains high-quality workflows. Our certified hypergraph operator pools evaluations across shared design-choice combinations and prunes only branches certified to hold no target, reducing the evaluations to reach one and, under sufficient conditions, returning with high probability a workflow no worse than the unpruned search's. On five systems tasks, it improves task scores over OpenEvolve by 4.7%–18.2%. Future work will relax these conditions and extend the framework to RSI.

AI USE STATEMENT

We used generative artificial intelligence (AI) tools to assist with drafting and revising sections of the manuscript, polishing language and presentation, retrieving and discovering related literature, and developing and checking mathematical arguments and proofs. This assistance included improving the organization of the exposition, checking notation and terminology consistency, and examining proof assumptions and intermediate derivations. The authors directed the use of these tools and take responsibility for the final content of the paper, including the accuracy of citations and the validity of its mathematical claims.

REPRODUCIBILITY STATEMENT

Appendices B–E provide the formal definitions, assumptions, and proofs underlying our theoretical results, together with the adapted workflow search operators and the certified hypergraph procedure. Section 6 and Appendix F describe the benchmark tasks, shared experimental controls, evaluation metrics, and optimization budgets; Appendix F also identifies the public ADRS artifact and its commit version. Appendix G presents condensed policy listings and explains the joint-context representation, veto rule, and outcome-recording procedure for the illustrative case study.

REFERENCES

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems*, volume 24, pp. 2312–2320, 2011. URL https://proceedings.neurips.cc/paper/2011/hash/e1d5be1c7f2f456670de3d53c7b54f4a-Abstract.html.

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.761. URL https://aclanthology.org/2023.emnlp-main.761/.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

Federico Battiston, Giulia Cencetti, Iacopo Iacopini, Vito Latora, Maxime Lucas, Alice Patania, Jean-Gabriel Young, and Giovanni Petri. Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874:1–92, 2020. doi: 10.1016/j.physrep.2020.05.004. URL https://doi.org/10.1016/j.physrep.2020.05.004.

S. K. Berberian. Borel spaces. The University of Texas at Austin, 1988. URL https://web.ma.utexas.edu/mp_arc/c/02/02-156.pdf. Revised online edition.

Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1989. URL https://www.sciencedirect.com/bookseries/north-holland-mathematical-library/vol/45/suppl/C.

Stephen Boyd and Jacob Mattingley. Branch and bound methods. Notes for EE364b, Stanford University, April 2018. URL https://web.stanford.edu/class/ee364b/lectures/bb_notes.pdf.

Audrey Cheng, Aaron Kabcenell, Jason Chan, Xiao Shi, Peter Bailis, Natacha Crooks, and Ion Stoica. Towards optimal transaction scheduling. *Proceedings of the VLDB Endowment*, 17 (11):2694–2707, 2024. doi: 10.14778/3681954.3681956. URL https://www.vldb.org/pvldb/vol17/p2694-cheng.pdf.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, and Ion Stoica. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025. doi: 10.48550/arXiv.2510.06189. URL https://arxiv.org/abs/2510.06189.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/d14c2267d848abeb81fd590f371d39bd-Abstract.html.

Jiuzhou Han, Wray Buntine, and Ehsan Shareghi. Towards uncertainty-aware language agent. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6662–6685. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.398. URL https://aclanthology.org/2024.findings-acl.398/.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Olav Kallenberg. *Foundations of Modern Probability*, volume 99 of *Probability Theory and Stochastic Modelling*. Springer, Cham, 3 edition, 2021. doi: 10.1007/978-3-030-61871-1. URL https://link.springer.com/book/10.1007/978-3-030-61871-1.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=VD-AYtP0dve.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. doi: 10.2307/1910129.

E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4): 699–719, 1966. doi: 10.1287/opre.14.4.699.

Shu Liu, Asim Biswal, Amog Kamsetty, Audrey Cheng, Luis Gaspar Schroeder, Liana Patel, Shiyi Cao, Xiangxi Mo, Ion Stoica, Joseph E. Gonzalez, and Matei Zaharia. Optimizing LLM queries in relational data analytics workloads. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025a. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/b5dc49f44db2fadc5c4d717c57f4a424-Paper-Conference.pdf.

Shuaiqi Liu, Zhengkai Lin, Yuxiang Zhang, Yuanyi Ren, Yue Wu, Yongbin Li, Zheng Wang, Zhihang Fu, and Jieping Ye. The path to recursive self-improving agents: Foundation, framework, and future directions. *Preprints*, 2026. doi: 10.20944/preprints202608.0051.v1. URL https://www.preprints.org/manuscript/202608.0051. Preprint.

Xiaoou Liu, Tiejin Chen, Longchao Da, Chacha Chen, Zhen Lin, and Hua Wei. Uncertainty quantification and confidence calibration in large language models: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 6107–6117. Association for Computing Machinery, 2025b. doi: 10.1145/3711896.3736569. URL https://doi.org/10.1145/3711896.3736569.

Shentong Mo and Miao Xin. Tree of uncertain thoughts reasoning for large language models. In *ICASSP 2024 – 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 12742–12746. IEEE, 2024. doi: 10.1109/ICASSP48485.2024.10448355. URL https://doi.org/10.1109/ICASSP48485.2024.10448355.

David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. doi: 10.1016/j.disopt.2016.01.005. URL https://www.sciencedirect.com/science/article/pii/S1572528616000062.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131. URL https://arxiv.org/abs/2506.13131.

Changdae Oh, Seongheon Park, To Eun Kim, Jiatong Li, Wendi Li, Samuel Yeh, Sean Du, Hamed Hassani, Paul Bogdan, Dawn Song, and Sharon Li. Uncertainty quantification in LLM agents: Foundations, emerging challenges, and opportunities. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16219–16250. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.738. URL https://aclanthology.org/2026.acl-long.738/.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025. doi: 10.48550/arXiv.2504.15228. URL https://arxiv.org/abs/2504.15228.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Cosma Shalizi. Stochastic processes (advanced probability ii). Lecture notes for 36-754, Carnegie Mellon University, 2007. URL https://www.stat.cmu.edu/~cshalizi/754/.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve/tree/411fb59c886c18704caaffb611e17cf9e7d824d2. Source snapshot: commit 411fb59 (July 18, 2026).

James E. Smith and Robert L. Winkler. The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006. doi: 10.1287/mnsc.1050.0451. URL https://doi.org/10.1287/mnsc.1050.0451.

Francisco J. Solis and Roger J.-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981. doi: 10.1287/moor.6.1.19. URL https://doi.org/10.1287/moor.6.1.19.

James C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley, 2003. doi: 10.1002/0471722138. URL https://doi.org/10.1002/0471722138.

UCB-ADRS. ADRS: Systems optimization artifacts. GitHub repository, 2025. URL https://github.com/UCB-ADRS/ADRS/tree/2139fd86c0be3fb41c75768033fa8af5e4872c1a. Artifact snapshot at commit 2139fd8.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993. doi: 10.7551/mitpress/3054.001.0001. URL https://mitpress.mit.edu/9780262731034/the-formal-semantics-of-programming-languages/.

Zhanghao Wu, Wei-Lin Chiang, Ziming Mao, Zongheng Yang, Eric Friedman, Scott Shenker, and Ion Stoica. Can't be late: Optimizing spot instance savings under deadlines. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 185–203. USENIX Association, 2024. URL https://www.usenix.org/conference/nsdi24/presentation/wu-zhanghao.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel Agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Zhiyuan Zeng, Xiaonan Li, Junqi Dai, Qinyuan Cheng, Xuanjing Huang, and Xipeng Qiu. Reasoning in flux: Enhancing large language models reasoning through uncertainty-aware adaptive guidance. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2401–2416. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.131. URL https://aclanthology.org/2024.acl-long.131/.

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for LLM agents. *arXiv preprint arXiv:2603.22386*, 2026. doi: 10.48550/arXiv.2603.22386. URL https://arxiv.org/abs/2603.22386.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://www.nature.com/articles/s41586-025-08661-4.

Zelda B. Zabinsky. *Stochastic Adaptive Search for Global Optimization*. Nonconvex Optimization and Its Applications. Springer, 2003. doi: 10.1007/978-1-4419-9182-9. URL https://doi.org/10.1007/978-1-4419-9182-9.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=46Zgqo4QIU.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/aa5f5e6eb6f613ec412f1d948dfa21a5-Paper-Conference.pdf.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems*, volume 19, pp. 1601–1608, 2006. URL https://proceedings.neurips.cc/paper/2006/hash/dff8e9c2ac33381546d96deea9922999-Abstract.html.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 15844–15872, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1ab4e0e8f35078109bc78d7b465d306f-Paper-Conference.pdf.

A NOTATION

The appendices follow the main argument: the stochastic-search model (Appendix B), adapted workflow search operators (Appendix C), hypergraph certificates (Appendix D), and their search benefits (Appendix E).

A.1 OBJECTS, DISTRIBUTIONS, AND BUDGETS

Table 2: Notation for programs, workflow proposals, and budgeted outcomes.

Symbol	Meaning
$\mathcal{C}, \mathcal{P}, \mathcal{W}$	Admitted programs, valid programs, and workflow semantics.
$\psi, \perp_c, \perp$	Extended semantics, no-program outcome, and non-workflow outcome.
E^0, J, f, F	Reference evaluation kernel, scoring function, and program/workflow reference qualities; $f = F \circ \psi$ on $\mathcal{P}$.
S_n, s	History after controller event n and a fixed launch snapshot.
K_s, q_s	Program proposal law and its semantic pushforward.
α_s, q_s^+	Validity probability and conditional workflow law ($\alpha_s > 0$).
T, Φ	Next-event kernel and history update map.
$\Lambda_s, \mathcal{L}_s(q)$	Program-realization kernel and the lift of workflow law q.
$\Gamma_s^\nu, K_s^\nu[q]$	Terminal program-outcome kernel and resulting program proposal law.
$\alpha_s^\nu(q), \mathcal{D}_s^\nu, \mathcal{T}_s^\nu$	Validity probability, admissible input laws, and adapted workflow search operator.
B, N_n, R_n	Evaluator-start budget, recorded starts, and reservations; $N_n + R_n \le B$.
G_B, τ_B	Generation-attempt limit and return-event index.
$\widehat{W}_B, V(B)$	Returned workflow and expected reference quality at budget B.

A.2 BRANCHES, INTERACTIONS, AND CERTIFICATES

Table 3: Notation for search allocation and quality certification.

Symbol	Meaning
$b, C_i, p_i, H(p)$	Branch map, partition cells, branch probabilities, and entropy.
η, A_η, g_η	Quality threshold, target set $\{w : F(w) \ge \eta\}$, and probability $q(A_\eta)$.
q, R, q^R	Input law, retained region, and conditional law $q(\cdot \mid R)$.
ζ, r_η	Retained proposal probability $q(R)$ and fraction of target probability retained, $q(A_\eta \cap R)/q(A_\eta)$.
τ_η, τ_η^R	First-target draw counts under frozen laws q and q^R.
$\mathcal{D}$	Workflow domain covered by the quality certificate.
$\mathcal{G}, \mathbf{x}(w)$	Hypergraph dictionary and interaction features.
$t, \widehat{\theta}_t, \beta_t$	Evidence count, coefficient estimate, and confidence radius.
$\underline{F}_t, \overline{F}_t$	Simultaneous lower and upper quality bounds.
$\delta, \mathcal{E}_\delta$	Failure-probability level and simultaneous-coverage event.
$U_i^{\mathcal{D}}(s)$	Upper bound on the best reference quality in $C_i \cap \mathcal{D}$.
$\varepsilon, \mathcal{T}_{\mathcal{G},s}^{\varepsilon}$	Exploration fraction and certified hypergraph search operator.
π_ℓ, π_ℓ'	Baseline and updated program-selection weights in a finite pool.

The evidence count t, controller-event index n, evaluator-start count N_n, and frozen-law draw count τ_η describe different processes.

B STOCHASTIC SEARCH AND BUDGETED RETURN

This appendix supplies the constructions and proofs behind Section 3: reference workflow quality, history-conditioned proposals, controller dynamics, and the budgeted return rule.

B.1 PROGRAMS, WORKFLOWS, AND OBSERVATIONS

Let $\mathcal{C} \subseteq \Sigma^*$ be the admitted finite-length programs over a finite alphabet Σ. A fixed semantic contract identifies the nonempty valid subset $\mathcal{P} \subseteq \mathcal{C}$ and assigns each valid program its task-execution workflow (Winskel, 1993; Hu et al., 2025; Zhang et al., 2025). Write $\mathcal{W} := \psi(\mathcal{P})$ and extend the semantics to

$$\psi : \mathcal{C}_\perp := \mathcal{C} \sqcup \{\perp_c\} \longrightarrow \mathcal{W}_\perp := \mathcal{W} \sqcup \{\perp\}, \qquad \psi(c) = \perp \quad (c \notin \mathcal{P}).$$

Here $\perp_c$ records an attempt that emits no program; invalid programs also map to $\perp$. Valid programs are semantically equivalent when their images under ψ agree. A valid program may still fail evaluation or receive a low score.

These spaces are countable; their discrete σ-algebras make ψ measurable. Let the evaluation-record space $\mathcal{Y}$ and controller-event space $\mathcal{E}$ be standard Borel, allowing numerical metrics, status labels, text, and missing-value symbols. The finite-history space

$$\mathcal{H} := \bigsqcup_{n \geq 0} \mathcal{E}^n, \qquad \mathcal{E}^0 := \{\emptyset\},$$

is then standard Borel under its product and disjoint-union σ-algebras (Berberian, 1988, Proposition 3.2.2). Throughout, $\mathcal{B}(\mathcal{X})$ denotes the specified σ-algebra on $\mathcal{X}$.

B.2 REFERENCE WORKFLOW QUALITY

Fix the workload distribution, execution environment, time limits, failure handling, and randomness of the reference evaluation protocol. Its kernel E^0 maps valid programs to evaluation records in $\mathcal{Y}$. For a measurable scoring function $J : \mathcal{Y} \to \mathbb{R}$, with larger values preferred, define

$$f(c) := \int_{\mathcal{Y}} J(y)\, E^0(c, \mathrm{d}y), \qquad \int_{\mathcal{Y}} |J(y)|\, E^0(c, \mathrm{d}y) < \infty, \quad c \in \mathcal{P}.$$

The protocol assigns scores to all terminal outcomes, including failures and timeouts.

Proposition B.1 (Objective factorization)**.** *There is a unique measurable $F : \mathcal{W} \to \mathbb{R}$ satisfying $f = F \circ \psi$ on $\mathcal{P}$ if and only if*

$$\psi(c) = \psi(c') \implies f(c) = f(c'), \qquad c, c' \in \mathcal{P}. \tag{5}$$

Under this condition, $\sup_{c \in \mathcal{P}} f(c) = \sup_{w \in \mathcal{W}} F(w)$.

Proof. Factorization implies the displayed invariance. Conversely, set $F(w) = f(c)$ for any representative with $\psi(c) = w$; invariance makes this well-defined. Surjectivity onto $\mathcal{W}$ gives uniqueness, its discrete σ-algebra gives measurability, and $f(\mathcal{P}) = F(\mathcal{W})$ gives equality of suprema. □

We assume Equation 5 throughout. Thus the semantic contract retains every distinction that affects reference quality. Live evaluations supply observed scores $J(y)$; they measure $f(c)$ without conditional bias only when their mean, given the search history and evaluated program, equals $f(c)$. Even then, selecting the largest noisy score need not select the highest reference quality (Smith & Winkler, 2006).

B.3 OBSERVABLE SEARCH HISTORY

Fix an initial record i_0 containing the seed, its observed score, budgets, configuration, and optimization rules. For recorded controller events $e_1, \ldots, e_n$, use

$$S_n := \big(i_0, (e_1, \ldots, e_n)\big) \in \mathcal{S} := \{i_0\} \times \mathcal{H}.$$

Records retain program snapshots, scores, statuses, feedback, archive decisions, and generation contexts used by subsequent proposals (Cheng et al., 2025; Sharma, 2025). They also retain the incumbent, resource counters, and, for asynchronous execution, pending-job identifiers, launch snapshots, and scheduling decisions. An arriving result remains linked to its launch snapshot even if other events have since changed the archive. After return, an idle event extends the history without starting further work.

Under a specified joint execution law, standard Borel histories admit regular conditional next-event distributions (Kallenberg, 2021, Chapter 8). Unobserved randomness is integrated into these conditional laws. Using the full history therefore retains the information needed for subsequent search decisions, including dependencies on source representations and earlier feedback.

B.4 Workflow Proposal Laws

A complete proposal attempt launched from history snapshot s returns a program or $\perp_c$. It may include generation retries, subject to finite attempt and duration limits; exhaustion without a program returns $\perp_c$. Let K be its measurable program-outcome kernel from $\mathcal{S}$ to $\mathcal{C}_\perp$, with $K_s(D) := K(s, D)$. The kernel includes randomness in context selection, generation, and program construction.

The semantic pushforward gives

$$q_s := \psi_\# K_s, \qquad q_s(A) = K_s\big(\psi^{-1}(A)\big), \quad A \subseteq \mathcal{W}_\perp. \tag{6}$$

In particular, $q_s(\{w\}) = \sum_{c \in \mathcal{P}:\, \psi(c)=w} K_s(\{c\})$: all program representations contribute their probability to the same workflow. Define

$$\alpha_s := K_s(\mathcal{P}) = q_s(\mathcal{W}), \qquad q_s^+(A) := \frac{q_s(A)}{\alpha_s}, \quad A \subseteq \mathcal{W}, \quad s \in \mathcal{S}_+ := \{s : \alpha_s > 0\}.$$

The pair (α_s, q_s^+) separates the probability of a valid outcome from the conditional distribution of valid workflows. At $\alpha_s = 0$, $q_s = \delta_\perp$ and q_s^+ is undefined.

Proposition B.2 (Measurable workflow proposal kernels)**.** *The family $(q_s)_{s\in\mathcal{S}}$ is a probability kernel from $\mathcal{S}$ to $\mathcal{W}_\perp$; $s \mapsto \alpha_s$ is measurable; and $(q_s^+)_{s\in\mathcal{S}_+}$ is a probability kernel from $\mathcal{S}_+$ to $\mathcal{W}$.*

Proof. For each s, inverse images under ψ preserve disjoint unions and the whole space, making q_s a probability measure. For each A, the kernel property of K makes $s \mapsto K_s(\psi^{-1}(A))$ measurable. Taking $A = \mathcal{W}$ proves measurability of α and $\mathcal{S}_+$. Restriction to $\mathcal{W}$ and division by the positive measurable α_s give the conditional kernel. □

B.5 Controller Dynamics

The controller records proposal submissions and outcomes, evaluation results, scheduling decisions, and archive updates. Each started evaluation produces a terminal record containing its immutable program snapshot, observed score, status, and feedback. Generation failure, evaluation failure, and rejection by selection remain distinct events.

Let $T(s, \cdot)$ be the conditional next-event kernel and let Φ append that event to the history:

$$e_{n+1} \mid S_n \sim T(S_n, \cdot), \qquad S_{n+1} = \Phi(S_n, e_{n+1}).$$

The proposal kernel K_s describes the eventual outcome of a complete attempt launched from s, whereas $T(S_n, \cdot)$ describes the next recorded event, which may concern another pending job. Both kernels belong to the same execution model, including evaluation and scheduling rules. Integrating over the event defines

$$M(s, A) := \int_{\mathcal{E}} \mathbf{1}_A\big(\Phi(s, e)\big)\, T(s, \mathrm{d}e), \qquad A \in \mathcal{B}(\mathcal{S}).$$

Proposition B.3 (Observable search dynamics)**.** *A measurable next-event kernel T and update Φ induce a transition kernel M. Together with $S_0 = (i_0, \emptyset)$, they determine a unique trajectory law satisfying*

$$\mathbb{P}(S_{n+1} \in A \mid S_0, \ldots, S_n) = M(S_n, A) \quad \textit{almost surely}.$$

When T is the recorded execution's conditional next-event law and Φ appends events, this law reproduces its observable histories.

Proof. For each s, $M(s,\cdot)$ is the pushforward of $T(s,\cdot)$ through $\Phi(s,\cdot)$; measurable kernel integration gives measurability in s. The Ionescu–Tulcea extension theorem yields the unique trajectory law (Shalizi, 2007, Theorem 33). Successive conditioning on the execution history gives the same finite-dimensional distributions. □

B.6 Budget and Return Rule

Evaluator starts. Each method starts from the same pre-evaluated, semantically certified seed $c_0 \in \mathcal{P}$, with workflow $w_0 = \psi(c_0)$ and observed score r_0. The budget $B \in \mathbb{N}_0$ counts subsequent starts of a common evaluator, including retries, failures, timeouts, and cancellations after start. The seed evaluation is excluded. For concurrent execution, let N_n count recorded starts and R_n count reservations for calls not yet started, maintaining

$$N_0 = R_0 = 0, \qquad N_n + R_n \le B.$$

Serialize budget updates: reserve an available slot before dispatch, convert it to a start at evaluator launch, and release it only if cancelled before launch. Completion never refunds a started call.

Stopping and settlement. Fix a finite generation-attempt limit G_B, counting initial attempts and retries even when no evaluation follows. Allow finitely many dispatch and bookkeeping steps per attempt or evaluator call, and finite deadlines for generation, dispatch, evaluation, validity certification, and cleanup. At G_B, start no further generation; at $N_n = B$, start no further generation or evaluation. Before return, settle all pending jobs, release unused reservations, and record terminal outcomes with finite scores for every started evaluator call. The return-event index τ_B is then a finite stopping time, with $N_{\tau_B} \le B$ and $R_{\tau_B} = 0$. For $B = 0$, return the seed immediately.

Eligible incumbent. For evaluator call j in start order, record its evaluated program c_j, terminal record y_j, and score $r_j = J(y_j)$. A fixed measurable, resource-bounded validity procedure provides a flag $a_j \in \{0,1\}$ satisfying $a_j = 1 \Rightarrow c_j \in \mathcal{P}$; rejection, timeout, or an inconclusive check gives $a_j = 0$. The seed has $a_0 = 1$. Preserve the highest-scoring eligible candidate independently of archive eviction and return

$$\mathcal{I}_B := \{0\} \cup \{j : 1 \le j \le N_{\tau_B},\ a_j = 1\}, \qquad j_B := \min \arg\max_{j \in \mathcal{I}_B} r_j, \qquad \widehat{W}_B := \psi(c_{j_B}).$$

The finite nonempty eligible set and fixed tie-breaking make $\widehat{W}_B$ measurable and valid, with the seed as fallback.

Reference return quality. Assuming $\mathbb{E}[|F(\widehat{W}_B)|] < \infty$, define

$$V(B) := \mathbb{E}[F(\widehat{W}_B)].$$

The expectation includes proposal, evaluation, and scheduling randomness. Compare methods with the same B, initialization, reference protocol, validity procedure, and generation and time limits. Selection uses observed scores, while $V(B)$ evaluates the returned workflow's reference quality.

C Uncertainty-Guided Workflow Search Operators

This appendix constructs the method-dependent terminal kernels used in Section 4. Equip $\mathsf{Prob}(\mathcal{W})$ with the σ-algebra generated by $q \mapsto q(A)$ for $A \subseteq \mathcal{W}$.

C.1 Program Realization and Terminal Laws

Here *optimization* means a bounded attempt to improve an input workflow program using the task specification and available feedback. Its internal states track progress toward a candidate program; the outer state s records search history. The attempt defines a proposal law; its effect on quality and resources is the subject of Section 5.

Fix a measurable realization kernel $\Lambda_s(w,\cdot)$ supported on $\{c \in \mathcal{P} : \psi(c) = w\}$. The program input law

$$\mathcal{L}_s(q)(D) := \int_{\mathcal{W}} \Lambda_s(w, D)\, q(\mathrm{d}w), \qquad D \subseteq \mathcal{P},$$

satisfies $\psi_{\#}\mathcal{L}_s(q) = q$. For method ν, encode each input program, task, and feedback as an optimization request, with fixed models, prompts, demonstrations, tool interfaces, and final-response decoding. Measurable transitions and decoding with finite attempt and duration limits induce a terminal kernel $\Gamma_s^\nu(c, \cdot)$ on $\mathcal{C}_\perp$. Failure to emit a program returns $\perp_c$. Averaging over inputs gives

$$K_s^\nu[q](D) := \int_{\mathcal{P}} \Gamma_s^\nu(c, D)\, \mathcal{L}_s(q)(\mathrm{d}c), \qquad D \subseteq \mathcal{C}_\perp.$$

Define the validity probability and admissible domain by

$$\alpha_s^\nu(q) := K_s^\nu[q](\mathcal{P}), \qquad \mathcal{D}_s^\nu := \{q \in \mathsf{Prob}(\mathcal{W}) : \alpha_s^\nu(q) > 0\}. \tag{7}$$

Semantic pushforward and conditioning give

$$\mathcal{T}_s^\nu(q)(A) := \frac{K_s^\nu[q](\psi^{-1}(A))}{\alpha_s^\nu(q)}, \qquad q \in \mathcal{D}_s^\nu, \quad A \subseteq \mathcal{W}. \tag{8}$$

Kernel integration and positive normalization yield a probability law; jointly measurable kernels make the operator measurable in (s, q) on its domain (Kallenberg, 2021, Chapter 3). In the transitions below, z denotes an internal optimization state and A a measurable set of such states.

C.2 Adaptive Backtracking and Guidance (UAG)

UAG uses rising token surprisal to trigger backtracking and demonstration guidance (Yin et al., 2024, Sections 4.1–4.2). Let z record the optimization trace and token probabilities, and let $\gamma_s(z)$ indicate that the fixed surprisal-increase threshold is exceeded. For nonterminal states,

$$\Xi_s^{\mathrm{UAG}}(z, \cdot) = (1 - \gamma_s(z))\Xi_s^{\mathrm{cont}}(z, \cdot) + \gamma_s(z)\Xi_s^{\mathrm{back}}(z, \cdot).$$

The first kernel continues generation; the second returns to a completed reasoning step, selects an optimization demonstration by relevance and originality, and resumes generation. Bounded execution and final decoding induce Γ_s^{UAG}.

C.3 Uncertainty-Aware Tree Selection (TouT)

TouT uses values and uncertainty from repeated value samples to guide tree search (Mo & Xin, 2024, Sections 3.2–3.3). In a breadth-first adapter, let $Q_s^{\mathrm{expand}}(z, \mathrm{d}t)$ expand intermediate program-optimization states into a tree record t containing sampled values v and empirical variances u. With fixed tie-breaking, $\mathrm{Select}_s(t)$ retains the prescribed number of nodes ranked by $v/\max\{u, \epsilon_{\mathrm{var}}\}$, where $\epsilon_{\mathrm{var}} > 0$ handles zero variance. Then

$$\Xi_s^{\mathrm{TouT}}(z, A) = \int \mathbf{1}_A\big(\mathrm{Select}_s(t)\big) Q_s^{\mathrm{expand}}(z, \mathrm{d}t).$$

Fixed expansion and depth limits, followed by decoding the selected terminal output, induce Γ_s^{TouT}.

C.4 Uncertainty-Gated Tool Use (UALA)

UALA uses answer uncertainty to gate acceptance and external assistance (Han et al., 2024, Section 3). The adapter accepts a program response when a fixed uncertainty test passes; otherwise it queries a tool and optimizes the response using its observation. For tool-observation kernel O_s and observation-conditioned optimization kernel Ξ_s^{opt}, this transition is

$$\Xi_s^{\mathrm{tool}}(z, A) = \int \Xi_s^{\mathrm{opt}}(z, o, A)\, O_s(z, \mathrm{d}o).$$

The controller rechecks the updated response and returns $\perp_c$ if its bounded rechecking ends without acceptance. This stopping rule and final decoding define Γ_s^{UALA}.

C.5 Comparison in the Search

Each terminal kernel enters Equation 8. At a common state s and admissible input q, compare the resulting workflow laws, validity probabilities $\alpha_s^\nu(q)$, and resource use. For $q' = \mathcal{T}_s^\nu(q)$, the

common semantic partition gives branch probabilities $p'_i = q'(C_i)$. Failed attempts remain in resource accounting: generation attempts count toward G_B, and evaluator starts toward B, under Appendix B.6.

After new feedback changes the history to s', subsequent optimization uses the operator indexed by s'.

D Hypergraph Construction and Guarantees

This appendix constructs the quality certificates and finite-pool implementation of Section 5, then proves the operator's target-preservation and efficiency properties.

D.1 Interaction Features and Evaluation Evidence

Fix categorical workflow attributes and the finite hypergraph dictionary $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ before collecting certificate evidence. Vertices encode attribute values; hyperedges encode compatible combinations, excluding conflicting values of one attribute. With the feature vector of Section 5.2, the dimension is $d_\mathcal{G} := 1 + |E_\mathcal{G}|$, including the intercept. Equivalent program representations have identical features. Overlapping hyperedges share one joint fit: an evaluation contributes one row, regardless of how many features it activates.

Each eligible record contains an immutable, semantically certified program c_j, its workflow $w_j = \psi(c_j)$, score $r_j = J(y_j)$, and evaluation-protocol tag. Include records under a comparable-protocol rule fixed before observing their scores, retaining failures and timeouts with their prescribed scores rather than filtering to surviving elites. Index included records by j, and let t count them; these are local evidence indices, distinct from controller events and evaluator starts in Section 3. All included records must satisfy the model below.

D.2 Simultaneous and Nested Quality Bounds

Declare a nonempty certification domain $\mathcal{D} \subseteq \mathcal{W}$.

Assumption D.1 (A reference model for certified hypergraph evidence)**.** *The feature dictionary is fixed and there is a parameter $\theta_* \in \mathbb{R}^{d_\mathcal{G}}$ with known norm bound $\|\theta_*\|_2 \le S_\theta$ such that*

$$F(w) = \mathbf{x}(w)^\mathsf{T}\theta_* \quad (w \in \mathcal{D}), \qquad r_j = \mathbf{x}(w_j)^\mathsf{T}\theta_* + \xi_j.$$

For a declared filtration $(\mathcal{F}_j)$, each $\mathbf{x}(w_j)$ is $\mathcal{F}_{j-1}$-measurable and, for all $a \in \mathbb{R}$,

$$\mathbb{E}[\exp(a\xi_j) \mid \mathcal{F}_{j-1}] \le \exp(a^2\sigma^2/2),$$

with known finite $\sigma \ge 0$. Every record used in the fit obeys this model.

These conditions require a realizable quality model and conditionally calibrated scores, beyond agreement with the observed data. A sequential implementation selects and evaluates one workflow before the next evidence update, with eligibility and evaluation calibrated to the declared filtration.

For $\lambda > 0$ and predeclared $\delta \in (0, 1)$, set

$$\mathbf{A}_t := \lambda I_{d_\mathcal{G}} + \sum_{j=1}^{t} \mathbf{x}(w_j)\mathbf{x}(w_j)^\mathsf{T},$$

$$\widehat{\theta}_t := \mathbf{A}_t^{-1} \sum_{j=1}^{t} \mathbf{x}(w_j) r_j,$$

$$\beta_t(\delta) := \sigma\sqrt{2\log\left(\frac{\det(\mathbf{A}_t)^{1/2}}{\lambda^{d_\mathcal{G}/2}\delta}\right)} + \sqrt{\lambda}S_\theta,$$

where $I_{d_\mathcal{G}}$ is the identity matrix and $\|z\|_Q := (z^\mathsf{T}Qz)^{1/2}$ for positive-definite Q. Regularization includes the intercept. The confidence-ellipsoid result of Abbasi-Yadkori et al. (2011, Theorem 2) gives an event $\mathcal{E}_\delta$ of probability at least $1 - \delta$ on which

$$\|\widehat{\theta}_t - \theta_*\|_{\mathbf{A}_t} \le \beta_t(\delta) \quad \text{for all } t \ge 0.$$

Define workflow bounds by projecting this ellipsoid and intersecting intervals over time:

$$u_t(w) := \mathbf{x}(w)^\mathsf{T}\widehat{\theta}_t + \beta_t(\delta)\|\mathbf{x}(w)\|_{\mathbf{A}_t^{-1}},$$
$$\ell_t(w) := \mathbf{x}(w)^\mathsf{T}\widehat{\theta}_t - \beta_t(\delta)\|\mathbf{x}(w)\|_{\mathbf{A}_t^{-1}},$$
$$\overline{F}_t(w) := \min_{0\le r\le t} u_r(w), \qquad \underline{F}_t(w) := \max_{0\le r\le t} \ell_r(w).$$

Proposition D.2 (Simultaneous and nested workflow bounds)**.** *Under Assumption D.1, on $\mathcal{E}_\delta$,*

$$\underline{F}_t(w) \le F(w) \le \overline{F}_t(w) \quad \textit{for every } t \ge 0 \textit{ and } w \in \mathcal{D}.$$

For each workflow, upper bounds are nonincreasing and lower bounds nondecreasing.

Proof. Cauchy–Schwarz on the confidence event gives $|\mathbf{x}(w)^\mathsf{T}(\widehat{\theta}_t - \theta_*)| \le \beta_t(\delta)\|\mathbf{x}(w)\|_{\mathbf{A}_t^{-1}}$ simultaneously for all t, w. Every interval contains $F(w)$, so their running intersection preserves containment and is nested. □

D.3 Finite-Pool Bounds and Exact Selection

For a recorded nonempty pool of M_s semantically certified program entries c_ℓ^{pool}, choose known weights $\pi_\ell > 0$ summing to one and record

$$q := \sum_{\ell=1}^{M_s} \pi_\ell \delta_{\psi(c_\ell^{\mathrm{pool}})}, \qquad \mathcal{D} := \{\psi(c_\ell^{\mathrm{pool}}) : 1 \le \ell \le M_s\}. \tag{9}$$

Then $q(\mathcal{D}) = 1$. At the evidence index $t = t(s)$, compute

$$U_i^{\mathcal{D}}(s) := \max_{w \in C_i \cap \mathcal{D}} \overline{F}_t(w),$$

with value $-\infty$ for empty intersections. The maximum uses all pool entries, including unevaluated candidates whose bounds follow from the shared fit. For this fixed domain, these branch bounds tighten monotonically. On $\mathcal{E}_\delta$, Equation 3 therefore retains every target in $\mathcal{D}$.

For $R = R_\eta^{\mathcal{D}}(s)$ with $\zeta = q(R) > 0$ and $\varepsilon \in [0, 1)$, update the program-selection weights by

$$\begin{aligned} \zeta &= \sum_{\ell:\psi(c_\ell^{\mathrm{pool}})\in R} \pi_\ell, \\ \pi'_\ell &:= \pi_\ell \left[\varepsilon + (1-\varepsilon)\frac{\mathbf{1}_R(\psi(c_\ell^{\mathrm{pool}}))}{\zeta}\right]. \end{aligned} \tag{10}$$

Their semantic pushforward is exactly the operator in Equation 4; equivalent programs contribute their combined probability. Selection uses the recorded bounds and input weights, without access to unknown reference qualities.

Complete the operator's execution rule as follows: if a supported certificate is unavailable, $q(\mathcal{D}) \ne 1$, or $\zeta = 0$, return q unchanged and record the corresponding status. Otherwise sample using π'_ℓ. Before invoking the operator, an empty pool triggers the controller's bounded refresh or certified-seed return rule. Record the pool, bounds, target, and selection law in s; in the sequential implementation, incorporate each terminal evaluation record before the next certificate update. Algorithm 1 summarizes a sequential run that applies this operator to each recorded pool.

For uniform allocation across the d represented branches, choose within-branch weights $v_\ell > 0$ summing to one in each branch and set $\pi_\ell = v_\ell/d$. Hard restriction ($\varepsilon = 0$) to k represented branches then realizes Proposition E.5. General weights use retained proposal probability ζ directly. All evaluator starts, including failures, count toward B, except the shared pre-evaluated seed; generation attempts count toward G_B.

Algorithm 1 Certified hypergraph search on recorded pools

Input: Pre-evaluated certified seed c_0; budget B; generation-attempt limit G_B; dictionary $\mathcal{G}$; partition $(C_i)_{i=1}^m$; target η; exploration weight $\varepsilon \in [0,1)$; $\lambda, \delta, S_\theta, \sigma$ of Assumption D.1

Output: Returned workflow $\widehat{W}_B$

$\mathbf{A}_0 \leftarrow \lambda I_{d_\mathcal{G}}$, $\mathbf{b}_0 \leftarrow \mathbf{0}$, $t \leftarrow 0$, $N \leftarrow 0$ ▷ evidence and evaluator starts
while $N < B$ and fewer than G_B generation attempts **do**
  Generate or refresh a pool $\{c_\ell^{\mathrm{pool}}\}_{\ell=1}^{M_s}$ with weights $\pi_\ell > 0$; if it is empty, apply the refresh or seed-return rule
  $q \leftarrow \sum_\ell \pi_\ell \delta_{\psi(c_\ell^{\mathrm{pool}})}$ and $\mathcal{D} \leftarrow \{\psi(c_\ell^{\mathrm{pool}})\}_\ell$ ▷ Equation 9
  $\widehat{\theta}_t \leftarrow \mathbf{A}_t^{-1}\mathbf{b}_t$ and compute $\beta_t(\delta)$ ▷ Appendix D.2
  for each $w \in \mathcal{D}$ **do**
    $\overline{F}_t(w) \leftarrow \min_{0 \le r \le t}\big[\mathbf{x}(w)^\mathsf{T}\widehat{\theta}_r + \beta_r(\delta)\|\mathbf{x}(w)\|_{\mathbf{A}_r^{-1}}\big]$
  end for
  $U_i \leftarrow \max_{w \in C_i \cap \mathcal{D}} \overline{F}_t(w)$ for each branch i, with $-\infty$ for empty intersections
  $R \leftarrow \bigcup_{i:U_i \ge \eta}(C_i \cap \mathcal{D})$ and $\zeta \leftarrow \sum_{\ell:\psi(c_\ell^{\mathrm{pool}}) \in R} \pi_\ell$
  if the certificate is unsupported or $\zeta = 0$ **then**
    $\pi' \leftarrow \pi$ and record the fallback status ▷ identity fallback
  else
    $\pi'_\ell \leftarrow \pi_\ell\big[\varepsilon + (1-\varepsilon)\mathbf{1}_R(\psi(c_\ell^{\mathrm{pool}}))/\zeta\big]$ for all ℓ ▷ Equation 10
  end if
  Sample $\ell \sim \pi'$, start the evaluator on c_ℓ^{pool}, and set $N \leftarrow N+1$
  Record the score r and validity flag; update the eligible incumbent ▷ Appendix B.6
  if the record is eligible as evidence **then** ▷ Appendix D.1
    $\mathbf{x} \leftarrow \mathbf{x}(\psi(c_\ell^{\mathrm{pool}}))$, $\mathbf{A}_{t+1} \leftarrow \mathbf{A}_t + \mathbf{x}\mathbf{x}^\mathsf{T}$, $\mathbf{b}_{t+1} \leftarrow \mathbf{b}_t + r\,\mathbf{x}$, $t \leftarrow t+1$
  end if
end while
return $\widehat{W}_B$, the highest-scoring eligible workflow, with the seed as fallback

D.4 Operator Properties and Search Benefits

Proposition D.3 (Certified allocation and target efficiency)**.** *For a successful operator call, let $q' = \mathcal{T}^{\varepsilon}_{\mathcal{G},s}(q)$, with $q(\mathcal{D}) = 1$, $\zeta = q(R) > 0$, and $g_\eta = q(A_\eta) > 0$. The output is normalized and preserves the conditional workflow law within each branch receiving positive output probability. Hard restriction ($\varepsilon = 0$) restricts input support to R; positive ε preserves the null sets of q. On $\mathcal{E}_\delta$,*

$$\frac{q'(A_\eta)}{q(A_\eta)} = \varepsilon + \frac{1-\varepsilon}{\zeta}.$$

Let τ_η and τ'_η count independent draws from the respective frozen laws until the first target hit, including the successful draw. Their expected draw-count ratio is the reciprocal of this factor. With one equal-cost evaluation and a correct target test per draw, the same ratio holds for expected evaluation costs. Under hard restriction of a uniform baseline over d positive-probability branches to k such branches,

$$\frac{\mathbb{E}[\tau'_\eta]}{\mathbb{E}[\tau_\eta]} = \frac{k}{d} = e^{H(b_\# q') - H(b_\# q)}.$$

Proof. The operator is a mixture of probability measures with density $\varepsilon + (1-\varepsilon)\mathbf{1}_R/\zeta$ relative to q. Since $q(\mathcal{D}) = 1$ and R is a union of branch intersections with $\mathcal{D}$, this density is q-almost surely constant within each branch, giving conditional-law preservation and the support statements. Valid bounds give $q(A_\eta \setminus R) = 0$, which yields the target ratio. The draw-count and entropy identities follow from Propositions E.4 and E.5 and the mixture formula in Appendix E.3. □

These identities freeze a realized input law and certificate before taking fresh independent draws; they do not assert independent proposals in the adaptive run. Matrix inversion, finite maxima, comparisons, and the stated fallback make the finite-pool construction measurable in the recorded state. For Proposition E.6, exact seed evaluation permits skipping a candidate when its branch upper bound

is at most $F(w_0)$. The simultaneous certificate then supplies the required safety on $\mathcal{E}_\delta$; the shared candidate sequence, exact evaluation, and overhead condition remain necessary for the budgeted-quality comparison.

E Search-Reduction Theory and Proofs

This appendix proves the benefit chain in Section 5: certified target preservation improves target-hitting resources, and saved evaluations improve budgeted returned quality under an explicit execution coupling.

E.1 Branch Allocation and Reasoning Uncertainty

The valid-workflow proposal law q_s^+ from Section 3 induces an allocation over fixed semantic design categories.

Definition E.1 (Semantic search branches)**.** *Fix $m \in \mathbb{Z}^+$ and a surjective map $b : \mathcal{W} \to \{1, \dots, m\}$. The semantic branches are the nonempty partition*

$$C_i := b^{-1}(\{i\}), \qquad \mathcal{W} = \bigsqcup_{i=1}^{m} C_i.$$

Keep the semantic contract and b fixed across methods and states. Since b acts on workflow semantics, equivalent program representations have the same branch label. A branch may contain workflows with different quality and may have zero proposal probability. For $s \in \mathcal{S}_+$, define

$$p_i(s) := q_s^+(C_i) = \frac{K_s(\psi^{-1}(C_i))}{\alpha_s}, \qquad p(s) := (p_i(s))_{i=1}^m \in \Delta_m,$$

where $\Delta_m := \{p \in [0,1]^m : \sum_i p_i = 1\}$. Equivalently, $p(s) = b_\# q_s^+$ is the law of the branch label of $W \sim q_s^+$. The coordinates are measurable because q^+ is a probability kernel and every subset of $\mathcal{W}$ is measurable.

Definition E.2 (Operational reasoning uncertainty)**.** *At a state with $\alpha_s > 0$, define*

$$U_{\text{reason}}(s) := H(p(s)) = -\sum_{i=1}^{m} p_i(s) \log p_i(s),$$

using natural logarithms and $0 \log 0 := 0$.

Thus $0 \le U_{\text{reason}}(s) \le \log m$, with zero entropy for a single positive-probability branch and maximum entropy for uniform allocation (Shannon, 1948, Section 6). The continuous entropy function makes U_{reason} measurable. Both $p(s)$ and $U_{\text{reason}}(s)$ are undefined when $\alpha_s = 0$. Entropy describes proposal concentration across the declared branches; it does not involve F and therefore supplies no target-preservation guarantee. That guarantee must come from quality evidence.

E.2 Certified Restriction and Target-Hitting Resources

Fix the target η, with $A_\eta := \{w : F(w) \ge \eta\}$, and a nonempty certification domain $\mathcal{D} \subseteq \mathcal{W}$. The branch bounds in Section 5.3 imply

$$U_i^{\mathcal{D}}(s) \ge \sup_{w \in C_i \cap \mathcal{D}} F(w) \tag{11}$$

on the simultaneous-coverage event $\mathcal{E}_\delta$, with $\sup \emptyset = -\infty$. Write

$$I_\eta(s) := \{i : U_i^{\mathcal{D}}(s) \ge \eta\}, \qquad R_\eta^{\mathcal{D}}(s) := \bigcup_{i \in I_\eta(s)} (C_i \cap \mathcal{D}).$$

Proposition E.3 (Safe exclusion and nested retained regions)**.** *Valid branch bounds give $A_\eta \cap \mathcal{D} \subseteq R_\eta^{\mathcal{D}}(s)$. For fixed $\mathcal{D}$, partition, and target, if $U_i^{\mathcal{D}}(s') \le U_i^{\mathcal{D}}(s)$ for every i, then*

$$I_\eta(s') \subseteq I_\eta(s), \qquad R_\eta^{\mathcal{D}}(s') \subseteq R_\eta^{\mathcal{D}}(s).$$

Proof. For $w \in A_\eta \cap C_i \cap \mathcal{D}$, validity gives $U_i^{\mathcal{D}}(s) \geq F(w) \geq \eta$, so w is retained. If $i \in I_\eta(s')$, then $U_i^{\mathcal{D}}(s) \geq U_i^{\mathcal{D}}(s') \geq \eta$, proving nesting. □

To quantify the effect on proposals, freeze an input law $q \in \mathsf{Prob}(\mathcal{W})$ with $g_\eta := q(A_\eta) > 0$. For a retained region R, define

$$\zeta := q(R) > 0, \qquad r_\eta := \frac{q(A_\eta \cap R)}{g_\eta} > 0, \qquad q^R(A) := \frac{q(A \cap R)}{\zeta}.$$

Here ζ is retained proposal probability and r_η is the fraction of target probability retained. Certified restriction with $q(\mathcal{D}) = 1$ gives $r_\eta = 1$.

Proposition E.4 (Target-hitting improvement under restriction)**.** *The restricted target probability is $q^R(A_\eta) = r_\eta g_\eta/\zeta$. For independent draws from the respective frozen laws until the first target hit, including the successful draw,*

$$\mathbb{E}[\tau_\eta] = \frac{1}{g_\eta}, \qquad \frac{\mathbb{E}[\tau_\eta^R]}{\mathbb{E}[\tau_\eta]} = \frac{\zeta}{r_\eta}.$$

Restriction strictly improves target probability and expected draw count exactly when $r_\eta > \zeta$. Certified target preservation gives strict benefit whenever $\zeta < 1$.

Proof. The target-probability identity follows by conditioning. Independence gives $\mathbb{P}(\tau_\eta > n) = (1 - g_\eta)^n$ and $\mathbb{P}(\tau_\eta^R > n) = (1 - r_\eta g_\eta/\zeta)^n$ for $n \geq 0$. Summing these tails gives the geometric means and their ratio. □

Proposition E.5 (Uniform-branch entropy and target-hitting draw count)**.** *Suppose q is uniform across its d positive-probability semantic branches and R is the union of $k \geq 1$ of those branches, with $r_\eta > 0$. For $p = b_\# q$ and $p^R = b_\# q^R$,*

$$\zeta = \frac{k}{d}, \qquad H(p) = \log d, \qquad H(p^R) = \log k.$$

Under the sampling assumptions of Proposition E.4,

$$\frac{\mathbb{E}[\tau_\eta^R]}{\mathbb{E}[\tau_\eta]} = \frac{k}{d\, r_\eta} = \frac{\exp(H(p^R) - H(p))}{r_\eta}.$$

For target-preserving restriction, an entropy decrease $\Delta H := H(p) - H(p^R)$ therefore reduces expected draw count by $e^{-\Delta H}$.

Proof. The retained proposal probability is k/d and each retained branch has conditional probability $1/k$. Substitute these probabilities into entropy and Proposition E.4. □

For nonuniform allocations, safe restriction need not lower entropy. For example, removing a target-free branch from probabilities $(0.9, 0.05, 0.05)$, when the other branches contain only targets, raises target probability from 0.1 to 1 and entropy from approximately 0.394 to $\log 2$. Target preservation, rather than entropy ordering, establishes the general benefit.

E.3 Sampling Costs and Exploration Mixtures

With one evaluation of cost $\kappa > 0$ and a correct target test per draw, the expected evaluation costs are κ/g_η and $\kappa\zeta/(r_\eta g_\eta)$. If additional screening costs C_{scr} are charged once, net expected savings require

$$C_{\mathrm{scr}} < \frac{\kappa}{g_\eta}\left(1 - \frac{\zeta}{r_\eta}\right).$$

All generation and certification overhead must also be charged. In particular, rejection sampling from q needs $1/\zeta$ raw proposals per retained draw on average, giving $1/(r_\eta g_\eta)$ raw proposals per target hit; target-preserving rejection alone does not reduce generation calls.

For $0 < \varepsilon < 1$, the exploration mixture $q_\varepsilon := \varepsilon q + (1-\varepsilon)q^R$ has the same null sets as q. Linearity and the geometric mean formula give

$$\frac{q_\varepsilon(A_\eta)}{q(A_\eta)} = \varepsilon + (1-\varepsilon)\frac{r_\eta}{\zeta}, \qquad \frac{\mathbb{E}[\tau_\eta^\varepsilon]}{\mathbb{E}[\tau_\eta]} = \left[\varepsilon + (1-\varepsilon)\frac{r_\eta}{\zeta}\right]^{-1}.$$

Here τ_η^ε again counts independent draws from a frozen law. These identities compare sampling resources; the adaptive evaluator-start budget B requires the execution comparison below.

E.4 Saved Evaluations and Budgeted Quality

Branch-and-bound excludes candidates whose certified upper bounds cannot improve a retained solution (Land & Doig, 1960; Boyd & Mattingley, 2018). The following coupling connects such exclusions to the returned-quality objective of Section 3.

Proposition E.6 (Budget dominance under safe skipping)**.** *Couple two searches through the same possibly random candidate sequence $(W_j)_{j\ge1}$ and seed w_0, with exact evaluation and retention of the best evaluated workflow. Every candidate is realized by a semantically certified program eligible for return. Skipping an evaluation leaves the later candidate sequence and order unchanged, and the restricted search skips a candidate only with a certificate that $F(W_j) \le F(w_0)$.*

For a prefix of length n, let G_n be common generation and other common costs, $\kappa_j \ge 0$ the evaluation costs, J_n the skipped indices, and $D_n \ge 0$ all additional costs of bound construction, screening, and bookkeeping. The cumulative resource charges are

$$C_0(n) := G_n + \sum_{j\le n}\kappa_j, \qquad C_{\mathrm{pr}}(n) := G_n + D_n + \sum_{\substack{j\le n\\ j\notin J_n}}\kappa_j.$$

Suppose the baseline completes a finite prefix N within total-resource allowance $\mathfrak{b}$, the restricted run can execute the charged operations sequentially, and

$$D_N \le \sum_{j\in J_N}\kappa_j \quad \textit{almost surely}. \tag{12}$$

Then the restricted run can complete the same prefix within $\mathfrak{b}$. Spending remaining resources on additional candidates while retaining its best workflow yields

$$F(\widehat{W}_{\mathfrak{b}}^{\mathrm{pr}}) \ge F(\widehat{W}_{\mathfrak{b}}^{0}) \quad \textit{almost surely}.$$

Expected returned quality is therefore no smaller whenever both quantities are integrable.

Proof. The overhead condition gives $C_{\mathrm{pr}}(N) \le C_0(N) \le \mathfrak{b}$. Every skipped candidate is no better than the retained seed, and both runs evaluate every candidate in the prefix that improves it. Their best qualities agree after the prefix; further evaluations cannot lower the restricted run's retained quality. Taking expectations proves the final claim. □

For the evaluator-start budget, set $\kappa_j = 1$, $G_n = D_n = 0$, and $\mathfrak{b} = B$, assuming the common generation and time limits permit the compared prefixes. The proposition then gives $V_{\mathrm{pr}}(B) \ge V_0(B)$, where $V_{\mathrm{pr}}(B) := \mathbb{E}[F(\widehat{W}_B^{\mathrm{pr}})]$ and $V_0(B) := \mathbb{E}[F(\widehat{W}_B^{0})]$. Strict improvement requires positive probability that an additional evaluated candidate exceeds the baseline's returned quality. Comparisons of total cost use actual charges in Equation 12.

With statistical certificates, the same pathwise proof holds on the simultaneous-coverage event $\mathcal{E}_\delta$, provided the execution and cost conditions still hold there. Its probability of at least $1-\delta$ establishes a high-probability quality comparison. Unconditional expected-quality dominance additionally requires control of certificate failures; selecting the highest noisy observed score also requires a separate final-selection guarantee.

F EXPERIMENTAL PROTOCOLS

F.1 SYSTEMS OPTIMIZATION TASKS

The five tasks in Table 4 use the public ADRS artifact at commit `2139fd8` (Cheng et al., 2025; UCB-ADRS, 2025). Their initial programs, evaluators, and configurations are in `openevolve/examples/ADRS`, under `cant-be-late`, `cant-be-late-multi`, `eplb`, `llm_sql`, and `txn_scheduling`, respectively.

Table 4: Systems tasks selected for workflow optimization. Each candidate supplies the algorithm used by the corresponding task evaluator.

Task	Candidate program	Objectives and constraints
Can't Be Late: single region	A policy choosing when to use spot or on-demand instances	Minimize execution cost while completing jobs before their deadlines.
Can't Be Late: multiple regions	A policy choosing instance type, region, and migration	Minimize execution and migration cost while meeting deadlines.
Expert placement	An algorithm replicating and assigning experts across GPUs	Improve the artifact's load-balance score while reducing the combined runtime of placement and simulation.
LLM-SQL reordering	An algorithm ordering table rows and fields	Improve prefix-cache hit rate and reduce reordering runtime while preserving table contents.
Transaction scheduling	A scheduler ordering transactions with conflicts	Minimize makespan while respecting the evaluator's conflict and execution constraints.

F.2 BASELINE CONFIGURATIONS AND SHARED CONTROLS

The adapted methods use the operator constructions in Appendix C with the common OpenEvolve host, and Hypergraph applies certified restriction to each recorded pool (Appendix D.3). All methods share the seed, generation backbone and base decoding settings, evaluator, information sources, and generation and time limits, and follow the settlement and return rules in Appendix B.6.

F.3 EVALUATION METRICS

Returned quality. For each independent run $j \in \{1, \dots, n_{\mathrm{run}}\}$, let $\widehat{F}^{(j)}$ estimate reference quality from held-out workloads. Assess the returned workflow $\widehat{W}_B^{(j)}$ and estimate $V(B)$ by

$$\widehat{V}(B) = \frac{1}{n_{\mathrm{run}}} \sum_{j=1}^{n_{\mathrm{run}}} \widehat{F}^{(j)}\left(\widehat{W}_B^{(j)}\right).$$

Retain every run, including failures and seed fallbacks, and assess the program actually returned. Repeated workload measurements refine a run's quality estimate but do not increase the number of independent search runs.

Score comparisons. Compare scores within tasks. Use percentage-point differences for rates, relative differences for positive cost baselines, and absolute differences when the baseline is zero.

Budget. Budget B counts optimization iterations after the shared seed and is matched across methods for each task, as listed in Table 1.

G CASE STUDY: JOINT SWITCHING EVIDENCE IN CAN'T BE LATE

This case study details the policy extension in Section 6.1. In single-region *Can't Be Late*, a policy decides at each tick whether a deadline-constrained job runs on a spot instance, runs on an on-demand instance, or waits; spot instances are cheaper but can be preempted, and each switch between instances incurs a changeover overhead.

Original policy. Listing 1 condenses the adaptive policy evolved by ADRS (Cheng et al., 2025, Figure 4(b)). It summarizes spot availability in a sliding window by the available fraction, the longest availability run, and the trailing run, and sets its parameters by classifying the situation. Its urgent branches hold on-demand instances when the slack cannot absorb further delay: they lock on-demand instances near the deadline and rebuild the slack buffer after it shrinks. Otherwise, it uses spot instances only when they look stable and waits while enough slack remains. When the job runs on demand, the non-urgent switch to spot instances depends only on the availability summary, through `safe_to_switch`.

Where the check acts. Whether this switch pays off depends on more than availability: the remaining work must amortize the changeover overhead, and the time margin must absorb a possible preemption. Listing 2 therefore keeps all other branches and adds a joint check after `safe_to_switch` approves a switch. The check can only keep the job on demand: it never triggers a switch, never forces a return from spot instances, and never overrides an urgent branch.

Joint context. The check represents the scheduling context at the switch as one hyperedge, built by `make_joint_edge` in Listing 2. Its vertices are binned attribute values: the availability fraction in four intervals cut at 0.25, 0.5, and 0.75; the longest and trailing runs, remaining work, and changeover overhead by $\lfloor \log_2(1+x) \rfloor$ for value x; and the time margin, the ratio of spare slack to the time still needed, in five intervals cut at 0.1, 0.5, 1, and 2. Together with the current state and the action, the hyperedge stands for one complete condition–action combination, *e.g.*, frequently available spot instances with a long longest run but a short trailing run, little remaining work, an ample time margin, and a high overhead, while running on demand and switching to spot. Switches whose contexts fall into the same bins share one hyperedge and hence one history.

Veto rule. The check, `joint_switch_worthwhile`, applies two tests. The first asks whether the switch could pay off in the best case, where the spot instance survives until completion. With remaining work ω, changeover overhead o, and on-demand and spot prices $\rho_{\mathrm{od}} > \rho_{\mathrm{spot}}$ per unit time, switching then costs $\rho_{\mathrm{spot}}(o+\omega)$ and staying costs $\rho_{\mathrm{od}}\,\omega$, assuming equal compute speed on both instance types, fixed prices, and startup billed at the spot price. The test thus rejects the switch unless $\omega > \rho_{\mathrm{spot}}\,o/(\rho_{\mathrm{od}} - \rho_{\mathrm{spot}})$: the remaining work needed to repay a switch grows with its overhead, so neither factor decides alone. The second test consults history: with at least five records under the hyperedge, the check vetoes the switch if their average gain is nonpositive; with fewer records, it keeps the original decision. The check therefore acts as a veto: when the original policy deems a switch safe but similar past switches did not pay off, the job stays on demand.

Outcome records. The hyperedge is frozen as `pending_switch_edge` when the switch is made. When the tracked episode closes, either because the job finishes or after it returns to an on-demand instance and recovers, the episode's gain is stored under this hyperedge. The gain is the computation completed during the episode minus its billed cost converted to on-demand compute time, where the cost includes spot startup, runtime, and any setup for returning; it measures the savings over performing the same computation on the running on-demand instance and is negative when preemptions and overheads outweigh the lower spot price. Records enter the history only after their episodes close, so no decision reads future traces, and each hyperedge keeps its 32 most recent gains.

Why joint records matter. The added history thus records complete context–action–outcome tuples rather than a longer availability window. Consider two switches under equally stable spot availability, with prices $\rho_{\mathrm{spot}} = 1$ and $\rho_{\mathrm{od}} = 3$ per hour: the first with $\omega = 10$ and $o = 0.5$ hours, the second with $\omega = o = 1$ hour. Both pass the best-case test, since $\omega > o/2$. Staying on demand costs 30 and 3; if the spot instance survives, switching costs 10.5 and 2, saving 65% and 33%. If the spot instance is preempted halfway, each switch pays another changeover to return on demand: the first then costs $5.5 + 3 \times 5.5 = 22$ and still saves 27%, whereas the second costs $1.5 + 3 \times 1.5 = 6$, twice the cost of staying. With preemption probability p, the expected savings are thus $19.5 - 11.5p > 0$ for the first and $1 - 4p$ for the second, which is positive only if $p < 1/4$; the history-based veto therefore tends to block the second switch when $p \geq 1/4$ but never blocks the first. A score that credits each factor separately and sums the credits would let the first switch's gains raise the credit of stable availability, and hence the score of the second switch. Joint records place the two contexts

in different hyperedges, so the second switch is judged only by outcomes of switches like it. This mirrors Section 5.1, where marginal values hide the distinction between good and bad combinations, and Section 5.2, where evaluations sharing hyperedge features pool their evidence. Unlike the certified operator of Section 5.3, the check uses per-hyperedge averages without confidence bounds; its bin boundaries, history length, and minimum record count are illustrative settings, and its gain measures local switching savings rather than total job cost.

Listing 1: Adaptive policy evolved by ADRS for single-region *Can't Be Late*, condensed from Cheng et al. (2025, Figure 4(b)). `hold_on_demand` merges its deadline-locking and buffer-rebuilding branches; `SPOT`, `ON_DEMAND`, and `NONE` abbreviate the instance choices.

```
def AdaptiveStrategy(self, has_spot, state, env, task):
    # Availability summary: fraction, longest run, trailing run.
    alpha, streak, tail = summarize_window(self.window, has_spot)
    params = get_params(classify_situation(alpha, streak, tail))
    need, slack = ticks_needed(task, env), ticks_remaining(env)
    safety = safety_margin(task, params)

    # Urgent branches: deadline locking and buffer rebuilding.
    if hold_on_demand(has_spot, tail, need, slack, safety, params):
        return ON_DEMAND
    if safety >= slack:
        return SPOT if has_spot else ON_DEMAND

    # Non-urgent switch from on-demand to spot.
    if state == ON_DEMAND:
        if safe_to_switch(has_spot, streak, tail, params):
            return SPOT
        return ON_DEMAND

    # Selective spot use, waiting, or buffer rebuilding.
    if has_spot and looks_stable(streak, tail, params):
        return SPOT
    if slack > safety + params.wait_margin:
        return NONE
    start_rebuilding(params.dwell)
    return ON_DEMAND
```

Listing 2: Extended policy. Only the non-urgent switch from on-demand to spot instances changes; highlighted identifiers belong to the joint check.

```
def JointAdaptiveStrategy(self, has_spot, state, env, task):
    alpha, streak, tail = summarize_window(self.window, has_spot)
    params = get_params(classify_situation(alpha, streak, tail))
    need, slack = ticks_needed(task, env), ticks_remaining(env)
    safety = safety_margin(task, params)

    # Urgent branches are unchanged.
    if hold_on_demand(has_spot, tail, need, slack, safety, params):
        return ON_DEMAND
    if safety >= slack:
        return SPOT if has_spot else ON_DEMAND

    # Non-urgent switch, now gated by the joint check.
    if state == ON_DEMAND:
        if not safe_to_switch(has_spot, streak, tail, params):
            return ON_DEMAND
        # NEW: represent the complete scheduling context.
        edge = self.make_joint_edge(
            alpha, streak, tail, need, slack, state, task)
        # NEW: veto switches that are unlikely to pay off.
        if not self.joint_switch_worthwhile(edge, task):
            return ON_DEMAND
        # Freeze the context for later outcome recording.
        self.pending_switch_edge = edge
        return SPOT

    # Selective spot use, waiting, and rebuilding are unchanged.
    if has_spot and looks_stable(streak, tail, params):
        return SPOT
    if slack > safety + params.wait_margin:
        return NONE
    start_rebuilding(params.dwell)
    return ON_DEMAND
```