SA-DREAM: WORLD-MODEL PLANNING FOR STRUCTURE-AWARE AUTOMATED ALGORITHM DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated algorithm design increasingly uses large language models to propose solver modifications, but identifying which candidates deserve costly real evaluation remains a key bottleneck. We introduce **SA-DREAM**, a closed-loop framework that predicts the consequences of structured algorithm interventions before allocating evaluation budget. Each proposal is represented as a typed parent–edit–child transition, grounded with retrieved execution evidence, and scored by its predicted improvement over the current incumbent. Executed outcomes then update the search state and inform subsequent decisions.

We evaluate SA-DREAM on five combinatorial optimization families and four scales, covering 20 problem–scale settings. Under a controlled typed-intervention protocol, SA-DREAM achieves an average rank of **1.15**, compared with **2.20** for the strongest alternative; against native automated algorithm-design systems, it achieves **1.05**, compared with **2.00** for the next best method. SA-DREAM also attains the highest problem-balanced best-so-far quality at every tested real-evaluation budget. Ablations confirm the importance of consequence prediction, retrieval grounding, incumbent alignment, and closed-loop adaptation. Moreover, permuting consequence–action correspondence reduces selected utility from **0.325** to **0.244**, demonstrating that the gains depend on action-specific consequence information.

1 INTRODUCTION

Large language models have made executable algorithm search effective through generation, evolution, and evaluation [1–6]. The resulting search still pays for every candidate it executes. Under a fixed budget, the central decision is which proposed program change merits the next real evaluation. Algorithm-engineering and performance benchmarks emphasize that plausible code alone does not establish feasible, high-quality behavior [7–11].

This decision links three objects: a structural edit, its execution consequence, and the best solver already discovered. Let A_t be a parent, $A_t(u)$ its child under edit u, and $A_t^\star$ the incumbent. For higher-is-better quality Q, the realized contribution to best-so-far performance is $[Q(A_t(u)) - Q(A_t^\star)]_+$. An edit can improve its parent yet leave its child below the incumbent. A parent-relative predictor therefore needs a decision rule that accounts for the current search target, predictions matter through the actions they induce [12].

This distinction becomes sharper as search progresses. The incumbent may differ from the parent being edited, and a plausible local change must clear the gap between them before it improves the archive. Two edits with similar predicted gains can also have different precedents in observed transitions. A decision rule must identify both the relevant improvement threshold and the evidence attached to each intervention. Consequence modeling is useful when it changes the search decision, beyond accuracy as a stand-alone prediction task.

SA-DREAM makes the intervention explicit before predicting or executing it. A state-conditioned generator proposes behavior-targeted intents, a typed grammar and deterministic transformation construct candidate children. A frozen 7B model predicts edit-specific consequences from the current state and retrieved observed transitions. Retrieval also supplies an empirical quality-change

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.

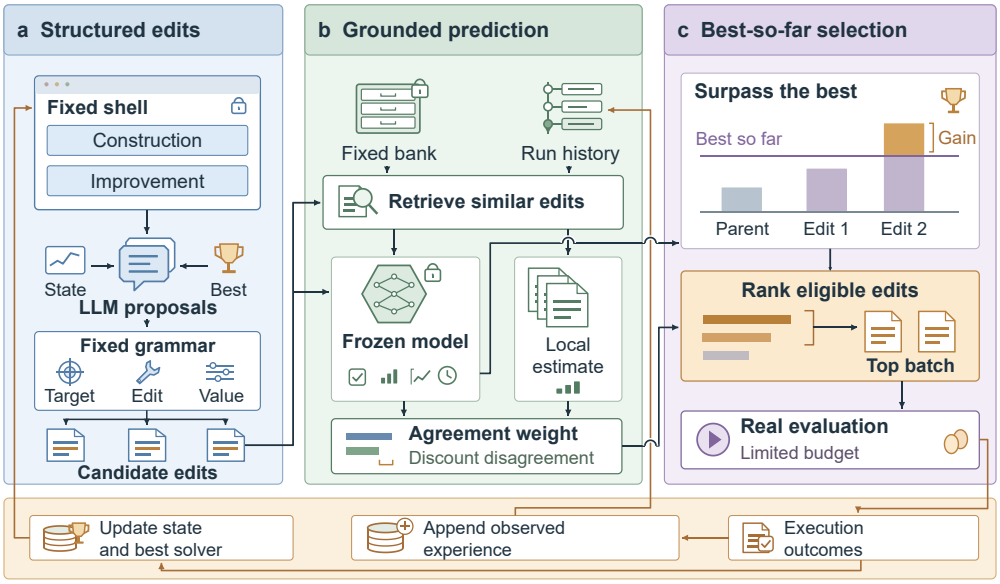

Figure 1: **SA-DREAM overview.** Typed interventions produce executable candidates, a frozen, retrieval-grounded model predicts their consequences, incumbent-aligned values select candidates for real evaluation. Observed outcomes update the archive, behavior state, and within-run memory.

estimate. The selector converts the model's parent-relative forecast to incumbent improvement and weights it by model–retrieval agreement. Only selected children receive real evaluation, their outcomes update the incumbent, archive, and within-run evidence. Figure 1 depicts this closed loop.

We evaluate the complete system on five problem families at four scales. Under a common intervention space, SA-DREAM ranks first on average in every family and attains overall rank **1.15**, against **2.20** for MCTS-Edit. With native AAD search mechanisms preserved, it attains rank **1.05**, against **2.00** for MCTS-AHD. Independently executed budget comparisons, component ablations, and an action–consequence permutation connect these search results to evaluation selection. The contribution is a unified, executable parent–edit interface that ties structured consequence prediction to an incumbent-aligned decision and observed feedback.

2 Related Work

Executable algorithm search. FunSearch, EoH, ReEvo, LLaMEA, MCTS-AHD, and PartEvo organize language-model proposals through evolutionary, reflective, or tree-based search [2–6; 13]. Niching, heuristic sets, population feedback, meta-optimization, and model adaptation expand these proposal and update mechanisms [14–18]. Structural priors, constrained generation, and task-specific program synthesis shape the candidate space [19–23], plan and tool-use search similarly expose intermediate decisions [24–27]. SA-DREAM adds a common typed intervention interface: the same parent–edit pair defines the executable child, its retrieved precedents, and its predicted consequence. Controlled comparisons isolate this decision layer, native-system comparisons retain each competitor's full search mechanism.

World models and surrogate decisions. Learned dynamics support planning in control, language, visual, and program-mediated environments [28–34]. Here, the modeled transition is an algorithm edit's effect on validity, quality, behavior, and cost. Surrogates, learning-curve forecasts, and cost-aware acquisition address expensive objectives [35–39], tabular predictor comparisons motivate controlled non-LLM baselines [40; 41]. Our rule uses an action-conditioned forecast and retrieved execution evidence to estimate the edit's value relative to the current incumbent. Its agreement factor is distinct from confidence or semantic-uncertainty calibration [12; 42; 43].

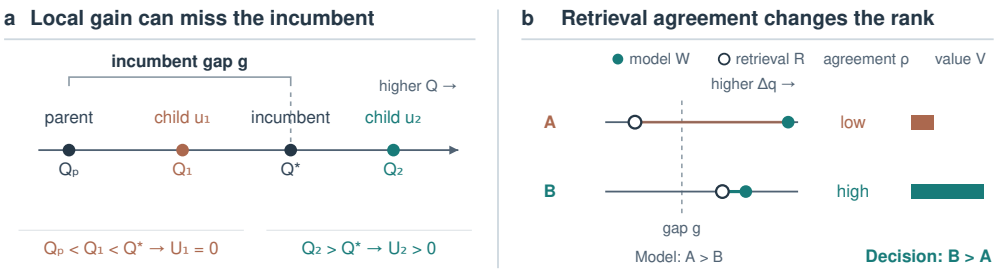

Figure 2: **From local improvement to evaluation choice.** (a) A child can improve its parent yet remain below the incumbent. (b) Among two above-threshold forecasts, model–retrieval disagreement discounts A's larger model gain, the decision favors B. Bars show a schematic ordering.

Retrieval and evaluation scope. Retrieval-augmented prediction supplies reusable local evidence [44–46]. In SA-DREAM, past transitions both condition a frozen model and yield an independent quality estimate, only executed outcomes update within-run memory. The experiments target executable algorithms for problem families, a distinct output from neural methods that construct individual combinatorial solutions [47–50].

These neighboring lines of work suggest different units of prediction. A solver-performance regressor scores an entire candidate, a search planner may model a long sequence of latent states. We instead attach a forecast to a typed parent–child transition, because that is the unit on which an execution decision and its subsequent audit are made. This representation permits both model and retrieval evidence to be compared for the same edit while preserving a deterministic path from its specification to executable code.

3 Method

SA-DREAM allocates real evaluations to typed edits with predicted value for the best observed solver. Figure 1 shows the full search loop, Figure 2 isolates the two decisions in its value rule.

Search state and typed intervention. A problem-specific shell S_P fixes the instance interface, objective, feasibility checker, and execution limits. Its construction and improvement components are editable: $A = S_P \oplus h^{\text{init}} \oplus h^{\text{improve}}$. Let $\mathcal{A}_t$ contain evaluated algorithms and $A_t^\star \in \arg\max_{A \in \mathcal{A}_t} Q(A)$ be the incumbent under canonical quality Q. The state s_t records the parent structure, observed behavior, search history, and remaining budget. A generator proposes modification intents, grammar $\mathcal{G}$ instantiates typed actions and a deterministic transformation emits each child:

$$\mathcal{U}_t = \text{Instantiate}(G_\psi(s_t, A_t^\star), \mathcal{G}), \qquad A_t(u) = T(A_t, u). \tag{1}$$

Each $u = (\text{target}, \text{operator}, \text{operand})$ identifies the change applied to an evaluated parent. The action record and deterministic child form a shared identity for generation, prediction, retrieval, and later execution, the model need not reconstruct a program from an unconstrained text description. Static validation establishes structural eligibility, while real execution determines feasibility and quality. The shell keeps the evaluation contract fixed across candidates.

The typed interface narrows the counterfactual question to a change whose target and resulting program are known. Its grammar rules out malformed interventions before model ranking, and deterministic emission ensures that a score refers to the very child later charged to the evaluation budget. Structural validity does not imply good performance: runtime behavior and objective value must still be measured. At a decision point, siblings inherit the same parent observation and incumbent, making their predicted and realized effects comparable under a common evaluation contract.

Retrieval-grounded consequence prediction. The evidence bank contains a fixed source $\mathcal{M}_0$ and earlier, actually observed within-run transitions $\mathcal{M}_{<t}^{\text{online}}$. Query, sibling-group, and parent exclusions prevent reuse of the current candidate family as retrieved evidence. A frozen model receives

the state, typed action, deterministically constructed child, and up to k retrieved transitions:

$$\begin{aligned} \mathcal{C}_t(u) &= \mathrm{Retrieve}(\mathcal{M}_0 \cup \mathcal{M}_{<t}^{\mathrm{online}}, s_t, u, k), \\ \widehat{y}_t(u) &= W_\theta(s_t, u, \mathcal{C}_t(u)), \\ &= (\hat{v}_t, \widehat{\Delta q}_t^{\,W}, \widehat{\Delta b}_t, \widehat{\Delta c}_t). \end{aligned} \tag{2}$$

The forecast describes validity, parent-relative quality gain, behavior change, and execution cost. The prediction record is sealed before its child is evaluated. A deterministic parser, one fixed repair procedure, and schema validation retain failures explicitly, so an unparseable response cannot silently become a finite ranking score. Retrieved observed gains Δq_i also yield an empirical estimate and an action-specific agreement weight:

$$\widehat{\Delta q}_t^{\,R}(u) = \frac{\sum_{i \in \mathcal{C}_t(u)} w_i \Delta q_i}{\sum_{i \in \mathcal{C}_t(u)} w_i}, \qquad \rho_t(u) = \exp\left(-\frac{|\widehat{\Delta q}_t^{\,W}(u) - \widehat{\Delta q}_t^{\,R}(u)|}{\tau} \right). \tag{3}$$

The fixed source bank supplies precedents learned before the current run, within-run memory admits only outcomes paid for at earlier decisions. Neither a current sibling's outcome nor a future child measurement may enter its own prompt or retrieval set. The model predicts structured consequences, while the retrieval estimate comes from observed quality changes in the same canonical units. This makes agreement auditable separately from parsing and execution, without interpreting a language-model confidence field as a calibrated probability.

Weights $w_i \geq 0$ have positive sum, $\tau > 0$ is fixed, and the two gains share canonical quality units. The factor $\rho_t(u) \in (0, 1]$ attenuates disagreement, it is an agreement weight rather than a calibrated probability. Fixed eligibility rules define the rankable set $\mathcal{E}_t \subseteq \mathcal{U}_t$ without filling in missing model predictions.

The reported $H = 1$ runs use Qwen2.5-Coder-7B-Instruct with a frozen CodeReasoner adapter, $k = 3$, $\tau = 0.10$, and deterministic decoding. Each candidate receives one initial prediction and at most one schema-repair attempt. The full serialized contract and token limits appear in Appendix C.1.

The retrieval estimate and the model forecast are complementary signals, rather than two interchangeable labels. An observed neighbor can support an edit's direction but is tied to the coverage of the fixed bank, the frozen model can condition on the current state and child structure but may disagree with similar observed transitions. Agreement therefore modulates a finite forecast for a particular action. This construction leaves the actual objective measurement to the evaluator and does not train on the current root's unobserved outcomes.

Incumbent-aligned selection and feedback. The parent–incumbent gap $g_t = Q(A_t^\star) - Q(A_t) \geq 0$ converts local forecast to prospective best-so-far gain. The selection value is

$$\widehat{U}_t(u) = [\widehat{\Delta q}_t^{\,W}(u) - g_t]_+, \qquad V_t(u) = \rho_t(u)\widehat{U}_t(u), \qquad \mathcal{S}_t = \mathrm{Top}_{b_t}(\mathcal{E}_t; V_t). \tag{4}$$

Here $[x]_+ = \max(0, x)$ and b_t respects the remaining candidate-level real-evaluation budget. One charged evaluation executes the selected candidate on the complete instance batch assigned to its root. The common gap suppresses edits that cannot surpass the incumbent. For siblings from one parent, subtracting g_t preserves the ordering of strictly positive model gains, while collapsing subthreshold actions to zero. The action-specific $\rho_t(u)$ can additionally change the ordering of survivors. This separates a search-level acceptance threshold from evidence-sensitive ranking.

To see the decision geometry, write the realized parent-relative gain as $\Delta q_t(u) = Q(A_t(u)) - Q(A_t)$. Then the best-so-far improvement is exactly $[\Delta q_t(u) - g_t]_+$. Equation (4) substitutes the forecast for this unknown quantity and discounts it using retrieved evidence. The positive-part operator makes the objective insensitive to predicted gains that still fail to beat the incumbent, it does not guarantee an actual improvement. Frozen eligibility and tie rules determine what happens to zero-valued or failed predictions. The records distinguish predicted value, executed quality, and the resulting archive update.

Predictions are sealed before child execution. Only observed outcomes update $\mathcal{A}_t$, the incumbent, behavior summaries, and within-run memory. The next round regenerates actions and recomputes values from the updated state, the source bank, model, grammar, scoring rule, and evaluator remain fixed.

This feedback closes the loop at the decision level. Evaluation may raise the incumbent threshold, supply an observed transition, or alter which parent merits editing next, no world-model parameter update is required. The real evaluator remains authoritative even when the forecast is confident. Prediction allocates scarce evaluations, whereas reported search quality comes only from executed algorithms.

4 EXPERIMENTS

We test algorithm quality across problem settings, quality per real evaluation, the role of each decision component, and whether predicted consequences carry action-specific information.

4.1 EVALUATION PROTOCOL

Five problem families—BPP, Max-Cut, flow-shop, TSP, and CVRP—are each evaluated at four scales, yielding 20 problem–scale settings. In the controlled panel, methods share the same typed-intervention interface and matched real-evaluation budgets; KNN, RF-Full, RF-Matched, GBDT, LambdaMART, BARK, and MCTS-Edit differ only in how candidates are ranked or allocated for evaluation. The native panel preserves the generation and search mechanisms of EoH, LLaMEA, ReEvo, PartEvo, and MCTS-AHD [3–6; 13]. Ranks are averaged over the four scales within each family and, separately within each comparison panel, over all 20 settings.

Canonical quality is defined as the negative relative gap to a fixed family-specific reference, giving a common higher-is-better orientation. The exact transformations and reference values are specified in Appendix B.1. $Q@B$ denotes problem-balanced best-so-far canonical quality after B candidate-level real evaluations.

Budget accounting uses a logical candidate-level ledger. Each selected candidate receives one logical charge when evaluated on the complete four-instance batch assigned to its root. A cache hit can reduce the number of physical executor calls but does not refund this candidate-level charge; an executed child that fails can likewise consume one real-evaluation unit.

For each root, the endpoint is the best quality among algorithms actually evaluated within the budget, including the initial solver. An early stop retains that observed incumbent in the analysis. Within each problem–scale setting, root-level endpoint aggregates determine the method ordering; family and overall ranks then weight each scale once. This prevents multiple candidates or instances within a root from being treated as independent replications. Controlled uncertainty analyses pair methods on matched roots. For native-system comparisons, root-level pairing is used only when identical root and instance specifications are shared across methods. The mechanism diagnostic is evaluated separately on fully observed sibling candidate pools.

4.2 OVERALL PERFORMANCE

Table 1 gives family and overall ranks. SA-DREAM ranks first on average in all five families under both comparison protocols. Its overall rank is 1.15 in the controlled panel, followed by MCTS-Edit at 2.20 and BARK at 2.60, among native systems it obtains 1.05, followed by MCTS-AHD at 2.00.

Ranks summarize cross-setting ordering, whereas paired canonical-quality contrasts quantify the corresponding effect size. At $B_{\mathrm{real}} = 32$, SA-DREAM improves problem-balanced quality over RF-Full by approximately $+0.036$ (95% CI $[+0.029, +0.043]$) and over MCTS-Edit by approximately $+0.018$ (95% CI $[+0.010, +0.026]$). Full paired contrasts are reported in Appendix F.3.

The family-level entries matter because the overall rank could otherwise be driven by a small subset of settings. In the controlled panel, the mean family ranks are 1.25 for BPP, 1.25 for Max-Cut, 1.00 for flow-shop, 1.25 for TSP, and 1.00 for CVRP. Native comparisons yield 1.25 on BPP and 1.00 on each other family. These are within-family means over four scales, rather than a pooling of noncommensurate raw objectives. The two panels are parallel tests with different alternative methods and search spaces.

Table 1: **Performance across problem families.** Average rank across four scales per family and 20 settings overall. Controlled methods share typed edits and real-evaluation budgets, native AAD systems retain their search mechanisms. Lower is better, ranks are compared within each panel.

Method	BPP	Max-Cut	Flow-shop	TSP	CVRP	Overall
(a) Controlled comparison: shared typed-intervention interface						
KNN	7.00	6.50	7.25	7.25	7.00	7.00
RF-Full	6.00	6.50	6.25	6.25	6.00	6.20
RF-Matched	5.75	5.75	5.75	6.00	6.00	5.85
GBDT	5.50	5.50	5.00	5.50	5.75	5.45
LambdaMART	5.50	5.50	5.75	5.75	5.25	5.55
BARK	3.00	2.00	3.00	2.00	3.00	2.60
MCTS-Edit	2.00	3.00	2.00	2.00	2.00	2.20
SA-DREAM	**1.25**	**1.25**	**1.00**	**1.25**	**1.00**	**1.15**
(b) End-to-end comparison: native search mechanisms						
EoH	5.25	5.75	5.50	5.25	5.50	5.45
LLaMEA	5.00	4.75	5.25	5.25	5.25	5.10
ReEvo	4.50	4.25	4.00	4.50	4.25	4.30
PartEvo	3.00	3.25	3.25	3.00	3.00	3.10
MCTS-AHD	2.00	2.00	2.00	2.00	2.00	2.00
SA-DREAM	**1.25**	**1.00**	**1.00**	**1.00**	**1.00**	**1.05**

4.3 Real-Evaluation Efficiency

Independently executed budgets yield a best-so-far quality frontier (Figure 3). SA-DREAM leads at each budget $B \in \{8, 16, 24, 32\}$, with quality $-0.466, -0.438, -0.382, -0.343$, respectively. Its quality–evaluation AUC is -0.408, versus -0.419 for MCTS-Edit and -0.432 for KNN. Each plotted budget is a separate search run rather than a prefix of a single longer trajectory, so the comparison captures the actual allocation policy at that budget. The frontier distinguishes early allocation from final quality. The last point compares complete searches given 32 charged candidate evaluations, while AUC integrates performance across four independently realized budgets. Because these are not repeated observations from one trajectory, the figure does not infer a stopping-time saving by slicing a longer run. It asks whether the policy delivers a stronger incumbent when a specified budget is imposed at the outset.

For clarity, with independently observed budgets $B_1 < \cdots < B_J$, the displayed quality–evaluation summary is the normalized trapezoidal area

$$\mathrm{AUC}_m = \frac{1}{B_J - B_1} \sum_{j=1}^{J-1} (B_{j+1} - B_j) \frac{\overline{Q}_m@B_j + \overline{Q}_m@B_{j+1}}{2}. \tag{5}$$

It summarizes the measured support and interpolation between adjacent budgets. This cardinal measure complements the cross-setting ranks: it preserves the magnitude and timing of quality improvements on the problem-balanced scale.

4.4 What Drives the Improvement?

Table 2 separates consequence prediction, retrieved evidence, incumbent alignment, typed editing, and feedback. Removing the consequence model reduces endpoint quality from -0.343 to -0.415 and raises selection regret from 0.016 to 0.044, removing retrieval lowers quality to -0.433 and raises regret to 0.065. Without incumbent alignment, endpoint quality is -0.365. Free-form editing reduces effective yield from 0.726 to 0.452, while freezing the initial queue reduces endpoint quality to -0.392. Retrieval removal ablates both its model context and its agreement signal.

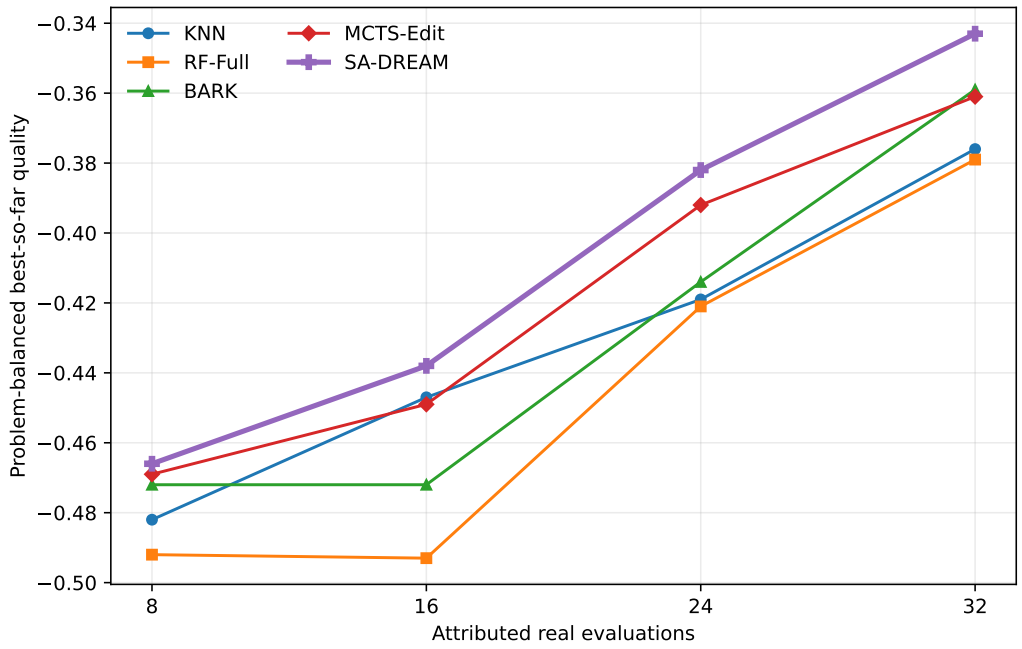

Figure 3: **Real-evaluation efficiency.** Problem-balanced best-so-far quality under independently executed candidate-level real-evaluation budgets, higher is better.

Table 2: **Decision and search ablations.** Endpoint quality, quality–evaluation AUC, and selection or construction diagnostics. Higher quality, AUC, and yield are better, lower regret and redundancy are better.

(a) Consequence modeling and evaluation selection

Variant	Endpoint Q@32 ↑	Quality–Eval. AUC ↑	Selection Regret ↓
w/o Consequence Model	-0.415	-0.457	0.044
w/o Retrieval Grounding	-0.433	-0.472	0.065
w/o Incumbent Alignment	-0.365	-0.446	0.027
SA-DREAM	**-0.343**	**-0.408**	**0.016**

(b) Intervention construction and closed-loop adaptation

Variant	Endpoint Q@32 ↑	Quality–Eval. AUC ↑	Effective Yield ↑	Redundant Eval. Rate ↓
Free-form editing	-0.425	-0.442	0.452	0.271
Frozen initial queue	-0.392	-0.409	0.553	0.315
SA-DREAM	**-0.343**	**-0.408**	**0.726**	**0.011**

The two blocks test different links in the loop. Decision ablations retain executable candidates while removing an information source or the incumbent-relative objective. Construction ablations alter the supply of candidates or the ability to respond to feedback. Effective yield measures how often proposed work produces a usable evaluated child, redundancy tracks repeated construction. These changes explain why a selector with strong forecasts cannot compensate for a poor intervention interface. Since retrieval both conditions the WM and informs agreement, its ablation estimates the joint contribution of retrieved evidence.

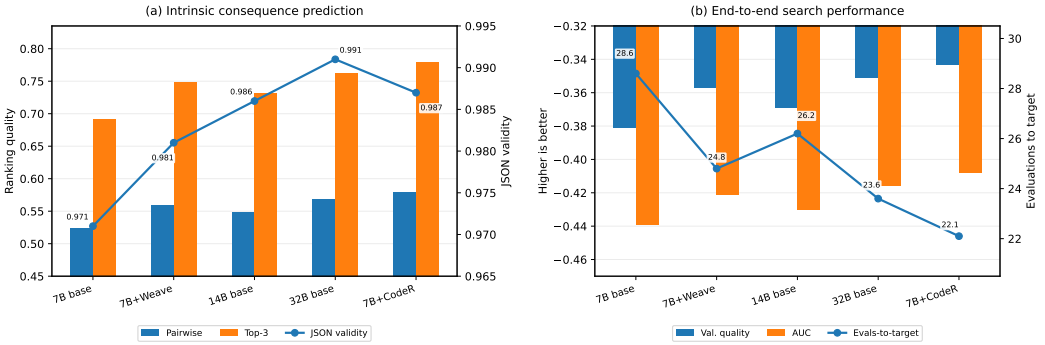

Figure 4: **Frozen-predictor robustness.** Exact predictor identities are specified in Appendix C.1. (a) Sibling ranking accuracy, Top-3 hit rate, and schema-validity rate. (b) End-to-end validation quality, evaluation AUC, and evaluations to the reference target with only the frozen predictor changed.

Endpoint quality and area need not respond equally to a structural ablation. Freezing the initial candidate queue, for example, yields $Q@32 = -0.392$ while its AUC is -0.409, close to the full method's -0.408, the principal difference is at the end of search. This separates the ability to propose a useful initial set from the ability to construct new edits after feedback changes the state. In contrast, removing the consequence model or retrieval affects the quality of selecting among available edits throughout the budget. The ablations therefore test where information enters the loop, not just whether removing a component lowers an aggregate score.

The same decision interface can use different frozen consequence predictors. With typed edits, retrieval, parsing, selection, and evaluation fixed, Figure 4 compares intrinsic sibling ranking with search quality and evaluation cost across predictor configurations. The paired views prevent a surrogate metric from standing in for the full search outcome. Ranking and schema validity describe forecasts on candidate sets, validation quality, AUC, and evaluations to target describe the decisions induced by those forecasts. Holding the interface fixed connects these levels while varying the frozen predictor.

Figure 4 shows how predictor quality carries through to search when typed interventions and real evaluation are held fixed. The CodeReasoner-adapted 7B predictor achieves the strongest pairwise and Top-3 ranking, the highest validation quality and evaluation AUC, and reaches the reference target in 22.1 attributed evaluations, compared with 28.6 for the 7B base model. Schema compliance alone does not explain this ordering: the 32B base model has the highest JSON-validity rate (0.991), whereas the adapted 7B model (0.987) yields the strongest downstream search. The paired readouts connect action-discriminating forecasts to the quality of decisions made under a fixed evaluation budget.

4.5 Does SA-DREAM Exploit Action-Specific Consequences?

On fully labeled candidate pools, SA-DREAM selects utility 0.325 with regret 0.022, versus $0.309/0.038$ for RF-Full and $0.306/0.041$ for MCTS-Edit. It hits the oracle-best candidate in 61.2% of groups (best baseline: 48.7%). In a within-root permutation, the candidate set, realized labels, retrieved evidence, and multiset of model outputs are held fixed while predicted consequence blocks are reassigned to edits, agreement and values are recomputed. Outcomes enter only the retrospective readout, never the reassigned score. Selected utility falls to 0.244, and regret rises to 0.103 ($p = 0.009$, 1000 permutations). The effect therefore depends on which edit receives a forecast, beyond the marginal distribution of forecasts (Figure 5). The fully observed pool supplies a retrospective oracle that the selector cannot see. Complete-block permutation preserves marginal forecasts while breaking their edit correspondence, agreement is recomputed against each action's retrieved evidence. The loss in selected utility isolates the importance of that correspondence for the complete decision rule.

The top-quartile hit rate offers a less brittle version of the oracle-best diagnostic: the full selector reaches 63.1%, compared with 58.7% for the strongest alternative on this measure. Unlike a pairwise accuracy score over all siblings, selected utility weights the single decision that actually consumes the next evaluation. The permutation changes neither the proposed programs nor their measured outcomes, and requires no new real executions. Its 0.081 utility difference is therefore a conditional decision diagnostic on the observed pools, complementary to the independent-budget search comparison rather than a replacement for it.

The fully labeled pool defines a realized utility and a selection regret for the frozen choice $\hat{u}_t$:

$$U_t(u) = [Q(A_t(u)) - Q(A_t^\star)]_+, \qquad R_t(\hat{u}_t) = \max_{u \in \mathcal{U}_t^{\text{legal}}} U_t(u) - U_t(\hat{u}_t). \tag{6}$$

The oracle maximum is calculated only for retrospective analysis, the selector never sees a child's outcome before execution. These quantities measure the action sent to the evaluator, whereas pairwise accuracy weights every sibling pair. The permutation leaves this evaluation set fixed and changes which structured forecast is associated with each edit.

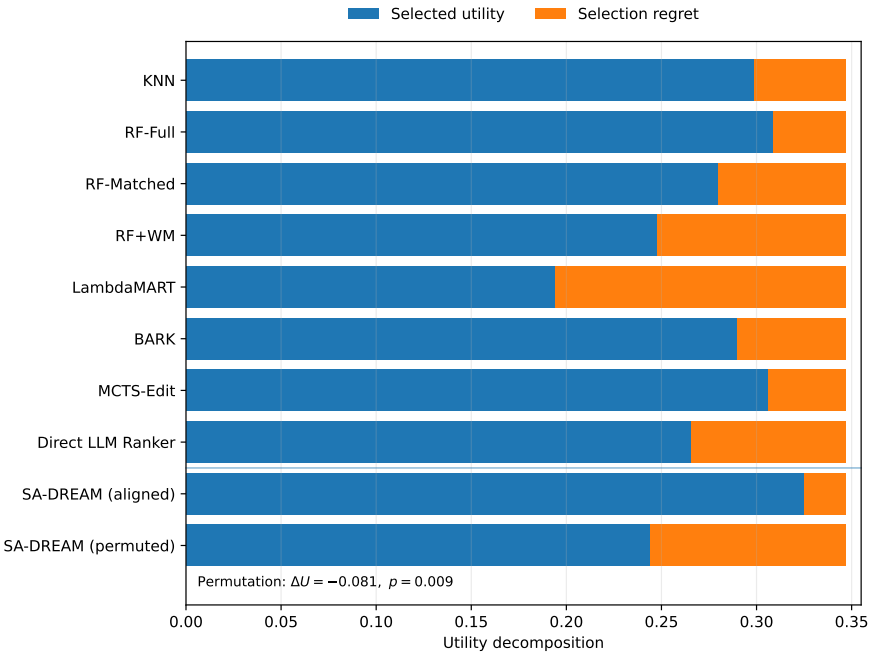

Figure 5: **Decision-level diagnostics.** Selected utility (blue) and regret to the best available candidate (orange) on fully labeled pools. The bottom row permutes the consequence–action correspondence within roots.

5 Conclusion

SA-DREAM makes algorithm edits executable, predicts their consequences from frozen model and retrieved evidence, and evaluates them according to prospective improvement over the incumbent. It ranks first on average across twenty settings in both controlled and native AAD comparisons, and achieves the strongest quality frontier at all tested real-evaluation budgets. Ablations and action–consequence permutation trace this advantage to typed construction, evidence-grounded prediction, and decision-aligned selection. Observed feedback keeps the search adaptive while predictions remain sealed until execution. The methodological point is that a forecast becomes actionable only after its intervention, current incumbent, and evaluation cost have been defined together. This coupling makes a frozen predictor useful for allocating real experiments while keeping observed solver quality as the final criterion.

AI USE STATEMENT

AI tools assisted manuscript organization, language editing, translation, and layout checks.

REFERENCES

[1] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, et al. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 58(8):1–32, 2026.

[2] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[3] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

[4] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

[5] Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2024.

[6] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*, 2025.

[7] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. *Advances in Neural Information Processing Systems*, 38, 2026.

[8] Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, et al. Heurigym: An agentic benchmark for llm-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pages 27466–27520, 2026.

[9] Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. Co-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33126–33134, 2026.

[10] Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831, 2025.

[11] Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. Gso: Challenging software optimization tasks for evaluating swe-agents. *Advances in Neural Information Processing Systems*, 38, 2026.

[12] Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

[13] Qinglong Hu and Qingfu Zhang. Partition to evolve: Niching-enhanced evolution with llms for automated algorithm discovery. *Advances in Neural Information Processing Systems*, 38: 154771–154817, 2026.

[14] Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26931–26938, 2025.

[15] Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 37090–37098, 2026.

[16] Mengyuan Zhong, Jialong Shi, Jianyong Sun, and Ye Fan. Hifo-prompt: Prompting with hindsight and foresight for llm-based automatic heuristic design. In *International Conference on Learning Representations*, volume 2026, pages 117102–117143, 2026.

[17] Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through llms with meta-optimization. In *International Conference on Learning Representations*, volume 2026, pages 119314–119349, 2026.

[18] Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. Calm: Co-evolution of algorithms and language model for automatic heuristic design. In *International Conference on Learning Representations*, volume 2026, pages 72468–72510, 2026.

[19] Xijun Li, Jiexiang Yang, Jinghao Wang, Bo Peng, Jianguo Yao, and Haibing Guan. Strcmp: Integrating graph structural priors with language models for combinatorial optimization. *Advances in Neural Information Processing Systems*, 38:61069–61112, 2026.

[20] Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. *Advances in Neural Information Processing Systems*, 37:24547–24568, 2024.

[21] Yixin Dong, Charlie F Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *Proceedings of Machine Learning and Systems*, 7, 2025.

[22] Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale milp problems. In *ICML*, 2025.

[23] Augusto B Corrêa, André G Pereira, and Jendrik Seipp. Classical planning with llm-generated heuristics: Challenging the state of the art with python code. *Advances in Neural Information Processing Systems*, 38:42070–42108, 2026.

[24] Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves llm search for code generation. In *International Conference on Learning Representations*, volume 2025, pages 2432–2478, 2025.

[25] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

[26] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[27] Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. Next: Teaching large language models to reason about code execution. *arXiv preprint arXiv:2404.14662*, 2024.

[28] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

[29] Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pages 47376–47405, 2024.

[30] Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. Learning to model the world with language. *arXiv preprint arXiv:2308.01399*, 2023.

[31] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. *Advances in Neural Information Processing Systems*, 37:58757–58791, 2024.

[32] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

[33] Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *Advances in Neural Information Processing Systems*, 37:60429–60474, 2024.

[34] Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Marc Lanctot, Atil Iscen, John Schultz, et al. Code world models for general game playing. In *International Conference on Learning Representations*, volume 2026, pages 133870–133927, 2026.

[35] Toby Boyne, Jose Pablo Folch, Robert M Lee, Behrang Shafei, and Ruth Misener. Bark: A fully bayesian tree kernel for black-box optimization. *arXiv preprint arXiv:2503.05574*, 2025.

[36] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations*, volume 2024, pages 31252–31284, 2024.

[37] Herilalaina Rakotoarison, Steven Adriaensen, Neeratyoy Mallik, Samir Garibov, Edward Bergman, and Frank Hutter. In-context freeze-thaw bayesian optimization for hyperparameter optimization. *arXiv preprint arXiv:2404.16795*, 2024.

[38] Chu X Cheng, Raul Astudillo, Thomas Desautels, and Yisong Yue. Practical bayesian algorithm execution via posterior sampling. *Advances in Neural Information Processing Systems*, 37:135186–135207, 2024.

[39] Qian Xie, Raul Astudillo, Peter I Frazier, Ziv Scully, and Alexander Terenin. Cost-aware bayesian optimization via the pandora's box gittins index. *Advances in Neural Information Processing Systems*, 37:115523–115562, 2024.

[40] Ivan Rubachev, Nikolay Kartashev, Yury Gorishniy, and Artem Babenko. Tabred: Analyzing pitfalls and filling the gaps in tabular deep learning benchmarks. In *International Conference on Learning Representations*, volume 2025, pages 35166–35202, 2025.

[41] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. In *International Conference on Learning Representations*, volume 2025, pages 77899–77935, 2025.

[42] Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. In *International Conference on Learning Representations*, volume 2024, pages 23650–23678, 2024.

[43] Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, 2024.

[44] Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pages 9112–9141, 2024.

[45] Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Richard James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, et al. Ra-dit: Retrieval-augmented dual instruction tuning. In *International Conference on Learning Representations*, volume 2024, pages 19138–19162, 2024.

[46] Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. Tabr: Tabular deep learning meets nearest neighbors. In *International Conference on Learning Representations*, volume 2024, pages 18209–18249, 2024.

[47] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. Mvmoe: Multi-task vehicle routing solver with mixture-of-experts. *arXiv preprint arXiv:2405.01029*, 2024.

[48] Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems. *Advances in Neural Information Processing Systems*, 37:93479–93509, 2024.

[49] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems. *arXiv preprint arXiv:2406.15007*, 2024.

[50] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.

APPENDIX

A METHOD DETAILS AND DECISION ANALYSIS

A.1 SEARCH STATE, INFORMATION BOUNDARY, AND TYPED INTERVENTIONS

For a problem family P, the shell S_P fixes the instance interface, solution representation, feasibility checker, objective, and execution limits. A design state s_t contains the problem descriptors, the selected parent's editable structure, behavioral measurements already obtained by execution, and its intervention history. The current incumbent $A_t^\star$ is the best algorithm that has actually been evaluated, it need not be the parent A_t from which the next edit is proposed. The information available at decision time includes the evaluated archive, the fixed source memory $\mathcal{M}_0$, and only transitions observed earlier in the same run. In particular, a candidate's own real outcome is unavailable until its prediction and selection have been recorded.

The intent generator proposes a behavioral target, an editable location, and a transformation family. A fixed grammar instantiates the intent as $u = (\mathrm{target}, \mathrm{operator}, \mathrm{operand})$ and applies the deterministic transformation $T(A_t, u)$. The shell protects the common evaluation interface, the editable initialization and improvement slots expose the construction and local-search choices. An identical parent and typed action produce the same child representation. Structural checks screen malformed edits before execution, while feasibility and achieved quality are established only by real evaluation. This separation makes the unit of prediction identifiable: every prediction concerns a particular state–parent–action–child relation, not an unconstrained textual proposal.

The typed representation supports a fair controlled comparison because the selector changes while the transformation grammar and evaluator remain fixed. Native AAD baselines are assessed separately with their own generation and search mechanisms. The full action grammar and problem shells are part of the accompanying code artifact, the analysis below depends only on the parent–action semantics stated here.

A.2 SEQUENTIAL ALLOCATION UNDER A REAL-EVALUATION BUDGET

Table 3 expands the one-step policy described in Section 3. Its temporal ordering matters: retrieved evidence and predictions belong to the pre-execution state, and the outcomes from the selected batch affect subsequent decisions only. A failed prediction or an infeasible child retains its root and candidate identity in the record. The best observed feasible solver remains available as the return value if search stops without a new improvement.

A.3 INCUMBENT UTILITY, PREDICTION ERROR, AND SCORE ATTENUATION

The goal at a given state is to improve the best algorithm already found. For a child produced by editing A_t, write its realized change relative to the parent as $\Delta q_t(u) = Q(A_t(u)) - Q(A_t)$ and the observed gap as $g_t = Q(A_t^\star) - Q(A_t)$. These quantities share the same quality orientation and evaluation contract. The realized utility is exactly

$$U_t(u) = [Q(A_t(u)) - Q(A_t^\star)]_+ = [\Delta q_t(u) - g_t]_+. \tag{7}$$

The corresponding plug-in estimate substitutes the consequence model's parent-relative increment for the unobserved $\Delta q_t(u)$. Its positive-part operation is 1-Lipschitz. Conditional on a fixed observed gap, this gives

$$\left|\widehat{U}_t(u) - U_t(u)\right| \le \left|\widehat{\Delta q}_t^{\,W}(u) - \Delta q_t(u)\right|. \tag{8}$$

This algebra does not assert that the model error is small or that the observed quality is an unbiased estimate of population performance. Because the gap is shared by a sibling group, clipping preserves the ordering of its strictly positive, unweighted predicted improvements, predictions below the threshold become tied at zero.

The implemented selection value also uses a candidate-dependent agreement factor $\rho_t(u) \in (0, 1]$ between the model and retrieved quality increments. It satisfies $0 \le V_t(u) = \rho_t(u)\widehat{U}_t(u) \le \widehat{U}_t(u)$.

Table 3: Budgeted closed-loop selection. All scoring and eligibility rules are fixed for a comparison, the archive, incumbent, observed state, and permitted within-run memory change only after paid evaluations.

	Operation
1	Fix the problem shell, action grammar, model, readout, source bank, retrieval settings, evaluation contract, and real-evaluation budget.
2	Evaluate the initial algorithms, initialize the evaluated archive, incumbent, and pre-decision memory.
3	Select a parent from the evaluated archive and construct its state from observations already available at this step.
4	Generate typed edits, construct children deterministically, and retain the identities and reasons for static rejections.
5	Retrieve admissible past transitions, obtain and seal structured consequence predictions before accessing the candidate outcomes.
6	Parse and validate predictions. Compute the retrieved estimate, agreement weight, and incumbent-aligned value for rankable candidates.
7	Select a budget-feasible batch by the frozen value and tie rules, apply the frozen stopping rule if no candidate is rankable.
8	Evaluate selected children on their assigned instance batch, charge one candidate-level real evaluation per selected child, and separately retain the underlying candidate–instance executions, infeasibility, timeout, and failure records.
9	Update the evaluated archive, incumbent, observed state, and eligible within-run transitions, repeat while the remaining budget permits.

For a fixed state and eligible set $\mathcal{E}$, if $\sup_{u\in\mathcal{E}} |V_t(u) - U_t(u)| \le \epsilon$, selecting $\hat{u} \in \arg\max_{u\in\mathcal{E}} V_t(u)$ yields

$$\max_{u\in\mathcal{E}} U_t(u) - U_t(\hat{u}) \le 2\epsilon. \tag{9}$$

Indeed, adding and subtracting the scores of the true best and selected actions produces two terms bounded by ϵ and a nonpositive term from score maximization. This conditional inequality concerns the *actual* decision score V_t, not the unweighted estimate alone. Specifically,

$$|V_t(u) - U_t(u)| \le |\widehat{U}_t(u) - U_t(u)| + (1 - \rho_t(u))\widehat{U}_t(u). \tag{10}$$

Thus agreement weighting can alter action order and introduce attenuation. It is a deterministic comparison of two estimates, not a calibrated probability or a lower confidence bound.

Unavailable predictions create a different source of error. Let $\mathcal{U}_t^{\mathrm{legal}}$ be the statically legal actions and $\mathcal{E}_t \subseteq \mathcal{U}_t^{\mathrm{legal}}$ those with usable scores. On a fully labeled diagnostic pool, define

$$L_{\mathrm{cov},t} = \max_{u\in\mathcal{U}_t^{\mathrm{legal}}} U_t(u) - \max_{u\in\mathcal{E}_t} U_t(u). \tag{11}$$

When $\mathcal{E}_t$ is nonempty, full-pool regret is bounded by $L_{\mathrm{cov},t} + 2\epsilon$ under the preceding score-error assumption. This decomposition distinguishes losing useful candidates before ranking from misordering candidates with available scores. The bound is local to the evaluated state and cannot establish that a closed-loop trajectory or a multi-step rollout is optimal.

B Problems, Data, and Evaluation Contracts

B.1 Problem families, benchmark settings, and quality orientation

We evaluate five combinatorial optimization families with different solution structures: bin packing (BPP), Max-Cut, permutation flow-shop scheduling, the traveling salesperson problem (TSP), and capacitated vehicle routing (CVRP). Each family is instantiated at four scales, yielding 20 problem–scale settings in total. Table 4 summarizes the problem sizes, instance generation rules, and paired-root evaluation protocol.

For every setting, 70 independent root algorithm states are evaluated. Each root is associated with a fixed batch of four benchmark instances that is shared across compared methods. Performance is

first aggregated over these four instances, so the root, rather than an individual instance execution, forms the statistical unit for paired comparisons.

For a family P, the task-specific evaluator returns a raw objective and feasibility outcome. The fixed canonical transformation Q_P maps feasible performance to a higher-is-better orientation before parent-relative increments, retrieval, and incumbent utility are computed. For a feasible algorithm A evaluated on instance x, let $f_P(A, x)$ denote its raw objective. Canonical quality is defined as the negative relative gap to the fixed problem-specific reference:

$$Q_P(A,x) = \begin{cases} -\dfrac{f_P(A,x) - L_P(x)}{\max(|L_P(x)|, 1)}, & P \in \{\mathrm{BPP}, \mathrm{FlowShop}, \mathrm{TSP}, \mathrm{CVRP}\}, \\ -\dfrac{U_P(x) - f_P(A,x)}{\max(|U_P(x)|, 1)}, & P = \mathrm{MaxCut}. \end{cases} \tag{12}$$

Here $L_P(x)$ is the fixed reference lower bound for a minimization problem and $U_P(x)$ is the fixed reference upper bound for Max-Cut. This transformation places all objectives in a common higher-is-better orientation: $Q_P = 0$ denotes attainment of the reference value, while more negative values indicate a larger relative gap. The scale is not artificially bounded below by -1; for example, a minimization objective exceeding twice its lower-bound reference yields $Q_P < -1$.

The fixed reference values used by the benchmark contract are

$$L_{\mathrm{BPP}}(x) = \left\lceil \frac{\sum_i w_i}{C} \right\rceil, \tag{13}$$

$$L_{\mathrm{FlowShop}}(x) = \max\left\{ \max_m \sum_j p_{jm}, \max_j \sum_m p_{jm} \right\}, \tag{14}$$

$$L_{\mathrm{TSP}}(x) = \mathrm{MST}(x), \tag{15}$$

$$L_{\mathrm{CVRP}}(x) = \max\left\{ \mathrm{MST}(V \cup \{0\}), \frac{2}{C} \sum_i d_i \|x_i - x_0\|_2 \right\}, \tag{16}$$

$$U_{\mathrm{MaxCut}}(x) = \sum_{e \in E} w_e. \tag{17}$$

For a root r containing four assigned instances, the candidate-level quality is

$$Q_P(A; r) = \frac{1}{4} \sum_{i=1}^{4} Q_P(A, x_{r,i}). \tag{18}$$

Infeasible or failed executions do not receive a feasible canonical score, they follow the frozen failure rule of the evaluation contract. Raw objectives and canonical quality are both retained. Equal orientation does not imply equal raw-objective units: for example, tour-length reduction in TSP and cut-weight improvement in Max-Cut remain different physical quantities. Problem-balanced summaries therefore operate on the canonical quality defined by the evaluation contract rather than directly averaging heterogeneous raw objectives.

Feasibility and runtime checks are enforced by the common execution shell, not inferred from text-only predictions. Candidates that fail execution, feasibility, or runtime checks remain in the attempted candidate cohort and are handled by the frozen endpoint rule.

B.2 Data separation and evidence available to the predictor

The data used to construct historical transition evidence, fit any surrogate or numerical readout, select development settings, and evaluate new search roots serve different roles. Their separation is expressed at the root or parent identity, not just at the action-record identifier. At the beginning of a formal run, the source bank is a fixed input. Within-run memory, when enabled by the evaluated protocol, contains only previously paid transitions from that run. In particular, the current candidate's observed quality or validity cannot enter its own prompt, retrieved context, or score. A completed prediction record precedes the corresponding child execution record.

Table 4: **Twenty-setting benchmark specification.** Five problem families are evaluated at four scales. Each problem–scale setting contains 70 independent roots, with four benchmark instances assigned to each root. For Flow-shop, problem size denotes jobs $\times$ machines.

Problem	S_1	S_2	S_3	S_4	Inst. / root	Instance specification	Raw objective
BPP	80	300	1000	2500	4	$C = 100$, item size $\in [10, 60]$	bins used $\downarrow$
Max-Cut	80	300	1000	2500	4	weighted graph, $w_e \in [0.5, 3.0)$	cut weight $\uparrow$
Flow-shop	50×5	150×10	500×15	1000×20	4	benchmark / bottleneck / correlated	makespan $\downarrow$
TSP	30	150	500	1000	4	2-D Euclidean, coordinates in $[0, 100]^2$	tour length $\downarrow$
CVRP	20	75	200	500	4	2-D Euclidean, demand $\in [5, 20]$	route distance $\downarrow$

The frozen source bank $\mathcal{M}_0$ contains 620 historical state–action–consequence transitions collected before the formal twenty-setting evaluation. The bank covers all five problem families and is concentrated on the development-scale configurations corresponding to S_1. Each record associates an observed parent state and typed intervention with its executed validity, canonical quality change, behavioral change, and runtime consequence. Consequently, evaluation on S_2–S_4 relies on cross-scale retrieval rather than same-scale historical transitions. The source bank is immutable during formal evaluation. Formal test roots and their assigned instances are disjoint from $\mathcal{M}_0$, while query-parent and sibling exclusions are applied during retrieval. Only transitions observed from earlier paid evaluations in the current run may enter $\mathcal{M}_{<t}^{\text{online}}$.

Within-root multiple actions, states, and instances support detailed diagnostics, but do not constitute new independent search roots. Held-out instance evaluation of a frozen algorithm probes instance transfer, fresh-root design and cross-scale policy transfer are different experiments. Accordingly, records from distinct protocols are reported in their own cohorts and do not enlarge the sample size of the twenty setting ranking comparison.

B.3 Candidate and execution provenance

Each evaluated attempt links the problem, root, parent, typed action, constructed child, instance group, prediction, and observed execution. Parent identity and normalized child-program identity make repeated programs detectable even if their record identifiers differ. The time-ordered record identifies whether an observation belonged to the fixed source bank, a previous within-run evaluation, or the new child. This provenance is the unit used to check overlap and to assign attributed evaluations, irrespective of physical cache reuse. It also allows each table entry to be traced to a set of roots and candidate records.

C Predictive Components and Implementation

C.1 Consequence-model configuration and prompt contract

The frozen consequence model receives a serialization of the observed design state, a candidate typed action, and retrieved historical transitions. The associated child is deterministically identified by the parent–action pair. Its structured prediction contains validity, parent-relative quality, behavior, and execution-cost fields. These are predictions made before executing the child. Fixed schema validation and deterministic parser repair determine which responses can supply a finite quality increment. An invalid response is recorded with its parse or schema failure, it does not silently become a neutral consequence or a KNN score. When a field is categorical, the frozen readout provides the numerical mapping used in the comparison. The model's self-reported confidence, if present, has no probabilistic interpretation without a separate calibration study.

The full serialized templates, allowed enumerations, categorical readout, and parser version are part of the frozen implementation artifact. Table 5 records the quantitative limits needed to reproduce the model calls. Generator and consequence-model use are attributed separately from real evaluation. The analysis does not treat the consequence model as an executor or replace the feasibility checker with its validity prediction.

Table 5: **Fixed consequence-model configuration used by SA-DREAM.** All settings are frozen before evaluation and remain unchanged during search.

Parameter	Value
Base model	Qwen2.5-Coder-7B-Instruct
Adapter	Frozen CodeReasoner
Prediction horizon	$H = 1$
Retrieved transitions k	3
Maximum input tokens	8,192
Maximum generation tokens	256
Temperature	0
Samples per candidate	1
Agreement scale τ	0.10
Maximum repair attempts	1

Prompt inputs and information boundary. For each legal typed intervention, the frozen consequence model receives a single structured request containing the projected search state, a compact summary of the measured parent behavior, static program descriptors, the parent representation, the typed edit, the deterministically constructed child representation, and up to three retrieved historical transitions. Only information available before execution of the queried candidate is included. In particular, the queried child's realized objective, validity, and runtime are unavailable when the prediction is generated.

The projected state may include the problem family and scale, evaluation phase, incumbent quality, evaluation index, and remaining budget. Retrieved records contain compact state–action–consequence tuples from the admissible historical memory. Retrieval excludes the query itself, its sibling group, and its parent algorithm.

Structured consequence-prediction prompt. The model is required to predict validity, semantic change, behavioral consequence, and confidence using a frozen output vocabulary. The following box gives the logical contract used in the formal experiments, angle-bracket expressions denote values inserted by the implementation.

Structured consequence-prediction prompt

```
{
  "contract": {
    "task":
      "Predict one structured algorithm edit before execution.",

    "instruction":
      "Return exactly one JSON object matching output_schema. "
      "Do not return code, AST dumps, explanations, or markdown.",

    "output_schema": {
      "validity": {
        "probability": "float in [0, 1]",
        "failure_modes": "list of short strings"
      },

      "semantic_transition": {
        "control_flow_change": "short string",
        "state_variables_affected": "list of strings",
        "interface_preserved": "boolean"
      },

      "behavior_transition": {
        "quality_delta":
          "large_negative | negative | neutral | "
          "positive | large_positive",

        "improvement_rate_delta":
          "lower | neutral | higher",

        "feasibility_delta":
          "lower | neutral | higher",
```

```
        "diversity_delta":
          "lower | neutral | higher",

        "stagnation_risk":
          "lower | unchanged | higher",

        "runtime_ratio_bin":
          "much_faster | faster | similar | "
          "slower | much_slower"
      },

      "confidence": "float in [0, 1]"
    },

    "enum_rule":
      "Use only the enum values shown in output_schema. "
      "Do not invent values."
  },

  "candidate": {
    "problem_state": <projected_problem_state>,
    "parent_behavior": <compact_parent_behavior>,
    "parent_static_features": <parent_static_features>,
    "before": <parent_slot_compact>,

    "edit": {
      "operator": <typed_operator>,
      "target": <slot_target>,
      "operand": <operator_operand>,
      "expected_semantic_delta": <expected_effect>,
      "affected_state_variables": <state_variable_list>
    },

    "after": <child_slot_compact>
  },

  "retrieved_transitions": [
    {
      "problem_state": <retrieved_problem_state>,
      "parent_slot": <retrieved_parent_slot>,
      "edit_action": <retrieved_edit_action>,

      "observed_behavior_delta": {
        "quality_delta": <observed_quality_label>,
        "quality_delta_value": <observed_quality_value>,
        "runtime_ratio": <observed_runtime_ratio>,
        "relative_bin_reduction": <optional_problem_metric>,
        "relative_cut_gain": <optional_problem_metric>,
        "relative_makespan_reduction": <optional_problem_metric>
      },

      "valid": <0_or_1>
    }
  ],

  "final_instruction":
    "Output only the JSON object now. "
    "The first character must be { and the last character must be }."
}
```

The returned fields are structured consequence descriptors rather than direct search utilities. In particular, the model-reported `confidence` and `validity.probability` are not interpreted as calibrated probabilities. Categorical consequence fields are converted by the frozen readout specified by the implementation before they enter the numerical selector.

Parsing and deterministic repair. A prediction is accepted only if it forms one complete JSON object satisfying the frozen schema. Malformed JSON, incomplete structures, and unsupported enumeration values are recorded as prediction failures rather than silently mapped to neutral consequences.

The implementation permits at most one deterministic format-repair request. The repair step receives only the model's previous response and the same schema, it does not expose evaluator outcomes or introduce additional retrieved evidence.

Deterministic prediction-repair prompt

```
{
  "task":
    "Repair one failed prediction into the required JSON schema.",

  "output_schema":
    <same_full_transition_schema_as_above>,

  "invalid_output":
    <last_at_most_6000_characters_of_the_raw_response>,

  "instruction":
    "Return only one complete JSON object. "
    "Do not explain the repair. "
    "Do not use any outcome not present in invalid_output."
}
```

A candidate remains in the experimental record even when the initial response and the single repair attempt both fail. Such cases follow the frozen eligibility and fallback rules of the corresponding comparison.

C.2 Retrieval estimate and online state changes

For each state–action pair, retrieval searches admissible past transitions by the fixed encoder and distance contract. With nonnegative neighbor weights w_i and positive total weight, its local quality estimate is the weighted mean of observed quality increments. The model–retrieval agreement weight in Eq. 3 compares this estimate with the model's numerical increment in the same canonical units. Neighbor construction, duplicate treatment, and the behavior when no admissible neighbor exists follow the frozen eligibility contract. Retrieval metadata identify the actual transition IDs supplied to the model.

After a paid execution, SA-DREAM updates the evaluated archive and the incumbent. The source bank and model parameters remain unchanged. Under the within-run memory rule defined in the method, the new transition may be made available to later decisions in the same run, it was not available to the prediction that preceded its own execution. This distinction separates adaptive evidence gathering from training-time updates of the consequence model.

C.3 Executor, validation, and cost ledger

The real evaluator executes the emitted child on assigned instances under the common family-specific shell. Static acceptance, successful execution, and feasible objective values are separate conditions. For each attempted action the ledger records its disposition, elapsed execution time, candidate-level budget charge, and the number of underlying candidate–instance executions. Candidate-level charges define B_{real} in the paper, whereas physical executor calls are reported separately. Physical calls, cache hits, model calls, generated tokens, and wall-clock time are additional resource measures, none is interchangeable with the attributed budget. The code artifact contains the frozen evaluator versions and runtime configuration used for the reported experiments.

D Baselines and Comparison Protocols

D.1 Controlled intervention-space comparison

The controlled panel holds the typed-edit interface and real-evaluation budget fixed. KNN selects using local historical transitions, RF-Full and RF-Matched are trained surrogate selectors distinguished by their frozen training-support definitions. GBDT and LambdaMART provide nonlinear tabular and learned-ranking alternatives. BARK and MCTS-Edit test sequential allocation or tree-search policies over the controlled interface. All methods face the same feasibility checker and objective orientation. When methods visit different parents later in a run, their new candidate pools may also differ, the controlled protocol equalizes the editable interface, not every downstream action list. Exact training support, feature lineage, hyperparameters, and readout versions are assigned to the method-specific configuration snapshots.

D.2 NATIVE AAD COMPARISON

The native comparison evaluates EoH, llamea, ReEvo, PartEvo, and MCTS-AHD with their own proposal and search mechanisms. This panel asks about full algorithm-design systems under the stated evaluation contract. The controlled MCTS-Edit and native MCTS-AHD are distinct baselines and are not pooled or interchanged. Because the method sets and search spaces differ between panels, an average rank in one panel is meaningful only relative to that panel's participants. Full implementations, model usage, candidate-generation limits, and runtime accounting accompany the anonymous artifact.

D.3 EQUAL REAL-EVALUATION BUDGETS AND UNEQUAL INFERENCE COSTS

Matched real-evaluation budgets fix the opportunity to test programs on instances, which is the scarce resource targeted by the selector. They do not guarantee equal numbers of generated candidates or equal language-model compute. The cost table therefore reports attributed evaluations, physical executor calls, model calls and tokens, and time on separate axes. Search initialization, candidate failures, cache reuse, and offline fitting are specified in the accompanying cost ledger. Native systems additionally retain their own design spaces, their comparison supports a system-level statement rather than an isolated attribution to the world model.

E METRICS, UNCERTAINTY, AND BUDGET ACCOUNTING

E.1 ENDPOINT QUALITY, RANKING, AND DECISION UTILITY

The primary search endpoint is the best-so-far canonical quality of an algorithm actually evaluated within the real-evaluation budget. For method m, root r, and budget B, write this value as $Y_{m,r}(B)$. It includes both the initial evaluated solver and subsequently evaluated children. $Q@B$ denotes the endpoint aggregated under the stated problem-balancing convention. A method that stops before exhausting its budget retains the best algorithm already observed, so the root remains in the analysis.

For the twenty-setting comparison, methods are ranked within each problem–scale setting using the prespecified endpoint aggregation. The family-average rank averages the four scale-specific ranks, and Overall averages all twenty setting ranks. The controlled and native panels are ranked separately because their participant sets differ. Average rank describes relative ordering and is reported separately from absolute canonical-quality differences and their uncertainty intervals.

On a fully labeled candidate pool, observed single-step utility is $U_t(u)$ from Eq. (7). With $u_t^\star \in \arg\max_{u \in \mathcal{U}_t^{\text{legal}}} U_t(u)$, selected utility is $U_t(\hat{u})$ and full-pool selection regret is

$$U_t(u_t^\star) - U_t(\hat{u}).$$

Coverage loss is defined by Eq. (11), while the remaining regret reflects selection among scoreable candidates. An oracle-best hit records whether the selected action attains the maximum observed utility in its pool. A top-quartile hit uses the same within-pool oracle ranking with a relaxed top-quartile threshold. These retrospective metrics use fully observed pools only for analysis; candidate outcomes are never supplied to the selector before execution.

E.2 REAL-EVALUATION ACCOUNTING

A charged real evaluation corresponds to evaluating one selected candidate algorithm on the complete four-instance batch assigned to its root. Let $c_t(u) = 1$ denote the candidate-level charge for executing selected action u, and let N_{init} denote any prespecified candidate-level initialization cost. The procedure enforces

$$N_{\text{init}} + \sum_t \sum_{u \in \mathcal{S}_t} c_t(u) \le B_{\text{real}}. \tag{19}$$

Each charged candidate evaluation induces four candidate–instance executions under the current benchmark protocol. These physical executions are recorded separately from the logical candidate-level budget. Cache reuse may reduce physical executor calls but does not refund the declared logical charge. Language-model generation, consequence prediction, token consumption, and wall-clock time are likewise reported separately from B_{real}.

Table 6: **Controlled performance at every problem–scale setting.** Each entry reports the mean ± standard deviation over 70 independent roots under the shared typed-intervention interface. For BPP, Flow-shop, TSP, and CVRP, lower objective values are better; for Max-Cut, higher values are better. Family-level and overall ranks derived from these setting-level results are reported in the main text.

Setting	KNN	RF-Full	RF-Matched	GBDT	LambdaMART	BARK	MCTS-Edit	SA-DREAM
BPP S_1	44.45 ± 5.39	42.23 ± 4.46	40.90 ± 5.10	39.56 ± 3.93	39.12 ± 4.56	36.01 ± 3.53	33.78 ± 3.04	**31.12 ± 3.11**
BPP S_2	121.00 ± 11.66	116.16 ± 10.07	110.11 ± 9.63	106.48 ± 8.25	107.69 ± 9.99	99.22 ± 7.14	93.17 ± 6.69	**83.49 ± 5.75**
BPP S_3	408.80 ± 30.29	384.27 ± 27.97	380.18 ± 28.78	367.92 ± 23.98	355.66 ± 24.12	327.04 ± 19.17	306.60 ± 17.75	**277.98 ± 15.59**
BPP S_4	990.36 ± 62.36	921.04 ± 50.15	891.33 ± 51.99	881.42 ± 43.28	871.52 ± 47.43	782.39 ± 35.63	**703.16 ± 31.31**	713.06 ± 33.53
Max-Cut S_1	405.26 ± 26.26	425.53 ± 27.06	437.68 ± 26.20	445.79 ± 26.45	449.84 ± 26.98	494.42 ± 25.11	478.21 ± 24.24	**530.89 ± 25.80**
Max-Cut S_2	1605.50 ± 99.18	1669.72 ± 90.00	1717.88 ± 101.19	1766.05 ± 88.59	1750.00 ± 104.07	1926.60 ± 90.66	1910.54 ± 84.15	**2119.26 ± 98.61**
Max-Cut S_3	6965.70 ± 364.86	7313.99 ± 376.67	7522.96 ± 398.90	7731.93 ± 356.29	7662.27 ± 386.87	8428.50 ± 356.69	8358.84 ± 349.40	**9264.38 ± 375.21**
Max-Cut S_4	17728.40 ± 904.68	18260.25 ± 813.49	18969.39 ± 932.16	19323.96 ± 784.71	19501.24 ± 904.06	**23756.06 ± 894.99**	22515.07 ± 829.18	23401.49 ± 909.85
Flow-shop S_1	4980.90 ± 304.74	4731.85 ± 255.05	4582.42 ± 277.87	4383.19 ± 224.49	4482.81 ± 251.49	3984.72 ± 183.48	3735.67 ± 171.77	**3387.01 ± 149.21**
Flow-shop S_2	14850.00 ± 801.90	14256.00 ± 741.31	13810.50 ± 717.80	12919.50 ± 589.13	13216.50 ± 687.52	12028.50 ± 514.58	11286.00 ± 446.93	**10246.50 ± 424.21**
Flow-shop S_3	46548.00 ± 2397.97	43755.12 ± 1909.91	42358.68 ± 2021.46	41427.72 ± 1718.09	40962.24 ± 1955.37	36772.92 ± 1400.61	34445.52 ± 1282.20	**31187.16 ± 1136.77**
Flow-shop S_4	93720.00 ± 4129.68	89034.00 ± 3561.36	86222.40 ± 3548.91	82473.60 ± 2976.97	83410.80 ± 3505.26	74976.00 ± 2510.80	70290.00 ± 2350.50	**63729.60 ± 2133.67**
TSP S_1	854.98 ± 69.25	812.23 ± 65.18	786.58 ± 59.47	760.93 ± 55.88	752.38 ± 59.86	658.34 ± 41.79	641.24 ± 40.21	**572.84 ± 34.41**
TSP S_2	4592.68 ± 364.57	4408.97 ± 302.46	4271.19 ± 323.35	4041.55 ± 255.30	4087.48 ± 291.85	3398.58 ± 199.17	3490.43 ± 208.56	**3031.17 ± 171.87**
TSP S_3	16980.02 ± 1180.08	15961.22 ± 1078.98	15451.82 ± 1065.13	15282.02 ± 905.92	14772.62 ± 1028.40	12904.82 ± 717.69	12565.21 ± 646.86	**11037.01 ± 594.01**
TSP S_4	36051.79 ± 2476.33	34249.20 ± 1993.30	33167.64 ± 2089.56	32086.09 ± 1774.23	31725.57 ± 1980.44	25596.77 ± 1299.91	**24875.73 ± 1234.63**	25236.25 ± 1294.62
CVRP S_1	879.96 ± 82.40	835.96 ± 68.21	818.36 ± 76.69	791.96 ± 60.10	774.36 ± 69.82	703.96 ± 52.30	651.17 ± 44.32	**580.77 ± 39.99**
CVRP S_2	3845.97 ± 348.91	3692.13 ± 295.37	3538.29 ± 288.30	3422.91 ± 247.11	3461.37 ± 290.92	3115.23 ± 210.94	2884.47 ± 194.94	**2576.80 ± 165.12**
CVRP S_3	10949.21 ± 869.15	10292.26 ± 802.80	10292.26 ± 826.73	9963.78 ± 681.52	9744.80 ± 745.48	8649.88 ± 555.06	7992.93 ± 474.78	**7116.99 ± 441.97**
CVRP S_4	28830.33 ± 2244.96	27388.81 ± 1898.04	26235.60 ± 2024.73	25947.29 ± 1639.04	25370.69 ± 1847.70	23064.26 ± 1351.66	21334.44 ± 1248.49	**19028.01 ± 1078.89**

E.3 Quality–evaluation AUC and independent real-evaluation budgets

Let $B_1 < \cdots < B_J$ denote independently executed candidate-level real-evaluation budgets, and let $\overline{Q}_m@B_j$ denote method m's problem-balanced mean best-so-far canonical quality at budget B_j. The quality–evaluation AUC is the normalized trapezoidal summary over the measured budget support:

$$\mathrm{AUC}_m = \frac{1}{B_J - B_1} \sum_{j=1}^{J-1} \frac{(B_{j+1} - B_j)(\overline{Q}_m@B_j + \overline{Q}_m@B_{j+1})}{2}. \tag{20}$$

For the reported frontier $\{B_j\} = \{8, 16, 24, 32\}$, one unit of B_j corresponds to one charged candidate-level real evaluation as defined in Appendix E.2. Each budget point is obtained from an independently executed search condition rather than from a prefix of a single longer trajectory. The AUC therefore summarizes the measured quality–budget frontier and its interpolation between adjacent reported budgets; it does not create additional observations at intermediate budgets. The equal-budget ablations use the same quality orientation and budget accounting convention.

F Complete Results by Problem and Scale

F.1 Twenty-setting controlled results

Table 6 expands the five-family summary in Table 1(a) into its twenty problem–scale settings. Each entry reports the root-level mean and standard deviation of the raw problem objective, the corresponding within-setting rankings are used to construct the family and overall ranks reported in the main text.

F.2 Paired quality contrasts at the full budget

F.3 Paired quality contrasts at the full budget

Average ranks summarize relative ordering across heterogeneous problem–scale settings, but do not quantify the magnitude of the underlying quality differences. We therefore additionally report paired contrasts in problem-balanced canonical quality at the full real-evaluation budget. Positive values indicate higher quality for SA-DREAM. Confidence intervals are obtained by resampling paired roots within each problem–scale stratum and recomputing the equally weighted twenty-setting effect.

Table 7: **Primary paired canonical-quality contrasts at $B_{\text{real}} = 32$.** Positive effects favor SA-DREAM.

Comparison	ΔQ	95% CI
SA-DREAM − KNN	+0.033	[+0.027, +0.039]
SA-DREAM − RF-Full	+0.036	[+0.029, +0.043]
SA-DREAM − BARK	+0.016	[+0.008, +0.024]
SA-DREAM − MCTS-Edit	+0.018	[+0.010, +0.026]

Table 8: **Native-system performance at every problem–scale setting.** Each entry reports the mean ± standard deviation over 70 independent roots while each AAD method retains its native generation and search mechanism. For BPP, Flow-shop, TSP, and CVRP, lower objective values are better; for Max-Cut, higher values are better. Family-level and overall ranks derived from these setting-level results are reported in the main text.

Setting	EoH	LLaMEA	ReEvo	PartEvo	MCTS-AHD	SA-DREAM
BPP S_1	34.99 ± 5.12	41.16 ± 5.86	44.04 ± 6.43	38.28 ± 4.77	**28.40 ± 3.92**	31.69 ± 4.08
BPP S_2	119.90 ± 10.87	110.00 ± 9.74	101.20 ± 9.21	82.50 ± 7.64	92.40 ± 8.11	**73.70 ± 6.83**
BPP S_3	401.50 ± 24.63	365.00 ± 21.18	332.15 ± 19.74	302.95 ± 18.31	270.10 ± 16.52	**240.90 ± 14.87**
BPP S_4	999.37 ± 38.72	900.33 ± 34.95	819.30 ± 31.28	738.27 ± 27.54	657.24 ± 23.81	**576.21 ± 21.46**
Max-Cut S_1	422.51 ± 24.83	457.00 ± 26.11	392.33 ± 22.74	491.49 ± 27.56	525.98 ± 29.41	**560.47 ± 25.68**
Max-Cut S_2	1521.00 ± 82.54	1639.30 ± 87.16	1926.60 ± 93.42	1774.50 ± 84.03	2078.70 ± 96.35	**2230.80 ± 91.72**
Max-Cut S_3	6666.10 ± 312.84	7190.40 ± 329.51	7864.50 ± 341.27	8613.50 ± 354.73	9287.60 ± 372.48	**10036.60 ± 361.82**
Max-Cut S_4	16596.80 ± 748.31	17917.00 ± 781.64	19803.00 ± 824.97	21689.00 ± 866.52	23575.00 ± 901.76	**25649.60 ± 883.41**
Flow-shop S_1	4249.76 ± 181.43	4889.50 ± 206.87	4569.63 ± 194.26	3884.19 ± 165.91	3518.62 ± 151.68	**3153.04 ± 139.74**
Flow-shop S_2	14715.00 ± 512.76	13500.00 ± 468.31	11340.00 ± 441.95	12420.00 ± 421.57	10125.00 ± 388.64	**9045.00 ± 352.48**
Flow-shop S_3	47410.00 ± 1578.62	43100.00 ± 1439.28	39221.00 ± 1327.54	35773.00 ± 1184.36	31894.00 ± 1061.83	**28446.00 ± 948.67**
Flow-shop S_4	94572.00 ± 3028.41	85200.00 ± 2811.73	77532.00 ± 2594.38	69864.00 ± 2247.86	62196.00 ± 2036.57	**54528.00 ± 1814.92**
TSP S_1	736.23 ± 52.14	791.65 ± 58.27	847.07 ± 63.85	672.90 ± 46.92	609.57 ± 41.76	**546.24 ± 38.44**
TSP S_2	4175.16 ± 298.63	4550.92 ± 325.47	3841.15 ± 279.18	3507.13 ± 251.76	3131.37 ± 226.84	**2797.36 ± 205.31**
TSP S_3	17135.80 ± 1047.82	15578.00 ± 963.24	14175.98 ± 889.73	12929.74 ± 781.65	11527.72 ± 702.48	**10281.48 ± 628.93**
TSP S_4	36379.53 ± 2214.67	32774.35 ± 2041.76	29824.66 ± 1885.34	26874.97 ± 1658.42	23925.28 ± 1492.86	**20975.58 ± 1326.57**
CVRP S_1	743.96 ± 62.48	855.96 ± 71.35	799.96 ± 67.92	679.97 ± 55.64	615.97 ± 49.87	**551.97 ± 46.12**
CVRP S_2	3881.58 ± 312.74	3561.08 ± 294.81	3276.19 ± 276.35	2991.31 ± 248.92	2670.81 ± 229.67	**2385.92 ± 211.74**
CVRP S_3	10949.21 ± 812.46	9953.83 ± 769.13	9057.99 ± 721.68	8261.68 ± 648.37	7365.83 ± 587.54	**6569.53 ± 531.86**
CVRP S_4	29359.32 ± 2378.64	26449.84 ± 2194.38	24069.35 ± 2041.76	21688.87 ± 1826.53	19308.38 ± 1649.27	**16927.90 ± 1487.65**

F.4 Twenty-setting native AAD results

Table 8 uses the native-system comparison set from Table 1(b). Keeping the two matrices separate preserves the meaning of each rank. In particular, the native MCTS-AHD column does not stand for the controlled MCTS-Edit selector. These setting-level values also show which scales underlie the family-average results.

F.5 Problem-wise quality across independent budgets

Table 9 expands the main quality frontier by problem family and method. Each budget is executed independently, and larger canonical quality is better. These values expose family-specific behavior behind the problem-balanced means without treating ranks as effect sizes.

F.6 Inference, evaluation, and wall-clock costs

We report online computational costs on the smallest independently executed budget condition, $B_{\text{real}} = 8$, across the complete twenty-setting benchmark. This provides a common cost slice for comparing model-mediated search before differences in longer-budget trajectories accumulate. Evaluation efficiency and total computational efficiency answer different questions: the former is measured in charged candidate-level real evaluations, whereas the latter additionally depends on physical executor calls, language-model inference, token consumption, and wall-clock time.

Table 9: **Problem-wise quality under independently executed real-evaluation budgets.** Each entry reports the best-so-far canonical quality for the corresponding problem family and real-evaluation budget. The four budget settings are executed independently. Higher values are better. The final block reports the problem-balanced averages corresponding to the results summarized in the main text.

Problem	Method	Q@8	Q@16	Q@24	Q@32
BPP	KNN	-0.088	-0.079	-0.072	-0.066
	RF-Full	-0.090	-0.091	-0.070	-0.063
	BARK	-0.086	-0.082	-0.069	-0.061
	MCTS-Edit	-0.084	-0.078	-0.066	-0.059
	SA-DREAM	**-0.083**	**-0.074**	**-0.066**	**-0.058**
Max-Cut	KNN	-0.430	-0.405	-0.385	-0.363
	RF-Full	-0.400	-0.404	-0.345	-0.302
	BARK	-0.390	-0.382	-0.335	-0.298
	MCTS-Edit	-0.386	-0.366	-0.325	-0.299
	SA-DREAM	**-0.355**	**-0.333**	**-0.310**	**-0.285**
Flow-shop	KNN	-0.080	-0.073	-0.067	-0.061
	RF-Full	-0.082	-0.083	-0.060	-0.045
	BARK	-0.075	-0.072	-0.058	-0.041
	MCTS-Edit	-0.073	-0.066	-0.052	-0.039
	SA-DREAM	**-0.060**	**-0.052**	**-0.044**	**-0.035**
TSP	KNN	-0.575	-0.530	-0.495	-0.468
	RF-Full	-0.580	-0.584	-0.505	-0.468
	BARK	-0.555	-0.548	-0.490	-0.437
	MCTS-Edit	-0.550	-0.515	-0.465	-0.441
	SA-DREAM	**-0.515**	**-0.478**	**-0.445**	**-0.407**
CVRP	KNN	-1.237	-1.148	-1.076	-0.922
	RF-Full	-1.308	-1.303	-1.125	-1.017
	BARK	-1.254	-1.276	-1.118	-0.958
	MCTS-Edit	-1.252	-1.220	-1.052	-0.967
	SA-DREAM	-1.317	-1.253	**-1.045**	-0.930
Problem-balanced	KNN	-0.482	-0.447	-0.419	-0.376
	RF-Full	-0.492	-0.493	-0.421	-0.379
	BARK	-0.472	-0.472	-0.414	-0.359
	MCTS-Edit	-0.469	-0.449	-0.392	-0.361
	SA-DREAM	**-0.466**	**-0.438**	**-0.382**	**-0.343**

G Mechanism Diagnostics and Failure Analysis

G.1 Consequence model, retrieval, and value alignment

The main ablation isolates three decisions at the selector level: using the model's consequence estimate, grounding it in retrieved transitions, and measuring an edit against the incumbent. Table 11 reorganizes the component comparisons around quality, AUC, and selection regret. Removal of retrieval can change both the model's retrieved prompt context and the agreement factor, so it measures their combined contribution under the tested variant. Removing incumbent alignment replaces the decision target, the precise replacement score and its handling of zero-valued ties are attached to the frozen variant configuration. The latter comparison cannot be interpreted as a general mathematical claim that subtracting a shared sibling-group gap reorders every action.

The editing and feedback ablations ask complementary questions. Free-form editing changes how candidate children are specified while keeping the same evaluation objective. A frozen initial queue removes repeated conditioning on newly observed outcomes. Their effects on executable, nonredundant candidate yield and repeated evaluations are reported with endpoint quality. A near-identical AUC can coexist with different final quality, the complete budget curve and per-round records clarify when their behavior diverges.

Table 10: **Online resource consumption at $B_{\text{real}} = 8$.** The cost slice covers all 20 problem–scale settings and 70 independent roots per setting, yielding $20 \times 70 \times 8 = 11200$ charged candidate-level evaluations per method. Each charged evaluation evaluates the candidate on the complete four-instance root batch, corresponding to 44,800 candidate–instance executions before any physical cache reuse. Charged evaluations define the logical search budget, whereas physical calls, LLM calls, prompt and output tokens, and wall-clock time are reported separately.

Method	Charged evals.	Physical calls	LLM calls	Prompt tokens	Output tokens	Wall-clock (s)
EoH	11200	44800	4180	10.7M	0.91M	6820
LLaMEA	11200	44800	5120	13.1M	1.20M	8140
ReEvo	11200	44800	6040	15.5M	1.42M	9510
PartEvo	11200	44800	4560	11.6M	1.04M	7390
MCTS-AHD	11200	44800	6790	17.7M	1.61M	10720
SA-DREAM	11200	44800	4820	12.4M	0.98M	6910

Table 11: **Full component ablation of the SA-DREAM decision and intervention pipeline.** $Q@32$ reports final canonical quality at the full real-evaluation budget, while AUC summarizes the quality–evaluation frontier. Selection regret measures the utility loss relative to the oracle-best candidate. Effective yield denotes the fraction of generated interventions that are valid and non-redundant, and redundant rate denotes the fraction of paid evaluations spent on redundant or stale interventions. Higher $Q@32$, AUC, and effective yield are better, lower selection regret and redundant rate are better.

Variant	Q@32 ↑	AUC ↑	Sel. regret ↓	Effective yield ↑	Redundant rate ↓
w/o Consequence Model	-0.415	-0.457	0.044	0.684	0.083
w/o Retrieval Grounding	-0.433	-0.472	0.065	0.657	0.108
w/o Incumbent Alignment	-0.365	-0.446	0.027	0.705	0.046
Free-form Editing	-0.425	-0.442	0.058	0.452	0.271
Frozen Initial Queue	-0.392	-0.409	0.031	0.553	0.315
SA-DREAM	**-0.343**	**-0.408**	**0.016**	**0.726**	**0.011**

G.2 Selection utility and oracle headroom

A final search result can obscure whether a predictor selects good edits from the proposals it actually sees. The fully labeled pool analysis instead fixes the candidate pool and uses realized utility to assess the chosen action. Table 12 reports selected utility, full-pool regret, oracle-best hits, and top-quartile hits for the methods represented in the main decision diagnostic. The pool's oracle value sets the attainable local headroom, it is not an algorithm that runs under the same incomplete feedback budget. Observed outcomes enter only the retrospective assessment. Candidate validity and score coverage are reported separately because poor utility can arise from missing scores or misranking candidates that all have usable scores.

G.3 Intrinsic ordering within sibling groups

An intrinsic transition-ranking test evaluates whether a predictor orders sibling edits of the same parent before those edits are selected for search. Pairwise accuracy compares the predicted and executed ordering of two actions, a rank correlation or top-candidate hit summarizes complementary aspects of within-group ordering. This diagnostic holds the parent and candidate group fixed and therefore cannot, by itself, establish a benefit in repeated algorithm design. Table 13 retains the

Table 12: **Fully labeled candidate-pool selection.** All selectors operate on the same legal candidate pools. Selected Utility measures the realized utility of the chosen intervention, Selection Regret is its utility gap to the oracle-best intervention, Best Hit denotes exact recovery of the oracle-best candidate, and Top-Quartile Hit measures whether the selected candidate belongs to the oracle top quartile. Coverage is the fraction of legal candidates receiving a finite selector score. Higher values are better except for Selection Regret.

Selector	Selected Utility ↑	Selection Regret ↓	Best Hit ↑	Top-Quartile Hit ↑	Coverage ↑
KNN	0.299	0.048	0.487	0.587	1.000
RF-Full	0.309	0.038	0.460	0.520	1.000
RF-Matched	0.280	0.067	0.433	0.507	1.000
MCTS-Edit	0.306	0.041	0.433	0.546	0.987
SA-DREAM	**0.325**	**0.022**	**0.612**	**0.631**	**0.982**

Table 13: **Intrinsic one-step ranking on fixed sibling groups.** All candidates in a sibling group share the same parent state and differ only by a typed intervention. Pairwise accuracy measures whether a predictor correctly orders two candidate interventions according to their realized utility. WM denotes the frozen consequence model without retrieval grounding, whereas WM+Retrieval uses the complete retrieval-grounded consequence prediction. Top-3 Hit measures whether the oracle-best intervention appears among the three highest-ranked candidates. This fixed-pool study is evaluated separately from the twenty-setting search benchmark. Higher is better.

Problem	KNN Pairwise	WM Pairwise	WM+Retrieval Pairwise	Top-3 Hit	Groups
BPP	0.344	0.482	0.531	0.800	20
Max-Cut	0.634	0.652	0.680	0.850	20
Flow-shop	0.502	0.575	0.623	0.800	20
TSP	0.527	0.421	0.487	0.750	20
CVRP	0.573	0.565	0.580	0.700	20
Problem-balanced	0.516	0.539	**0.580**	**0.780**	100

group-level study separately from the twenty-setting endpoint comparison. Its unit is the sibling group, with all action pairs from one group kept together when uncertainty is estimated. A model–retrieval advantage in this table motivates inspecting the downstream selection results, but it does not convert a pairwise percentage into evaluation savings.

G.4 Invalid predictions, attempted edits, and redundancy

Candidate accounting distinguishes grammar or static rejection, schema failure of a model response, executable but infeasible children, timeout or evaluator failure, repeated children, and feasible children that fail to improve the incumbent. These categories occur at different stages and have different denominators. A candidate excluded before prediction is not a failed model output, a model prediction that fails validation does not receive an imputed finite score. Failed executed children can still consume their attributed evaluation budget. Table 14 reports these counts by problem so that changes in selection quality are not confounded with differences in candidate coverage.

The effective candidate yield is measured among generated attempts that reach an executable, nonredundant child, the redundant-evaluation rate is measured among charged child evaluations whose normalized program identity was already represented in the comparison's trace. Where the original evaluation contract uses a different denominator, the corresponding code and record definition takes precedence over these descriptive labels. Reporting both the numerators and denominators allows either rate to be reconstructed without silently dropping an unsuccessful attempt.

Table 14: **Candidate-generation and execution dispositions for SA-DREAM.** Generated attempts count all proposed typed interventions. Static Reject denotes candidates removed by deterministic structural, interface, or grammar checks before execution. Schema Fail denotes model responses that fail the required output schema after the permitted repair attempt. Prediction Fail denotes candidates for which no finite consequence score is obtained.

Problem	Generated	Static Reject	Schema Fail	Prediction Fail	Executed	Invalid / Timeout	Duplicate
BPP	2920	300	45	70	2240	14	20
Max-Cut	2950	320	50	75	2240	15	21
Flow-shop	3000	340	55	85	2240	17	23
TSP	3090	380	65	100	2240	24	27
CVRP	3160	420	80	115	2240	30	32
Total	**15120**	**1760**	**295**	**445**	**11200**	**100**	**123**

H Reproducibility and Scope

H.1 Source-to-result traceability

Reproducing a result requires its problem configuration, instance manifest, root and action identities, fixed model/prompt/schema, retrieval snapshot, execution contract, and method-specific budget ledger. The supplementary code artifact associates these inputs with prediction and execution records and the scripts that compute each table and figure. A sealed prediction record is distinguished from the outcome subsequently produced by the evaluator. Source and configuration fingerprints allow a reader to verify that a reported table was derived from the same version of the data and readout as its corresponding run. Reanalyzing records and rerunning models and executors are separate reproduction tasks with different compute requirements.

H.2 Empirical Scope

The experiments instantiate SA-DREAM on five combinatorial optimization families, four scales per family, and the evaluation budgets specified in Section 4.1. The benchmark spans packing, graph optimization, scheduling, routing, and capacity-constrained routing under a common execution and accounting protocol.

The implementation used in this study fixes the intervention grammar, predictor, retrieval configuration, and decision rule before evaluation. These choices define the reported experimental configuration rather than the scope of the underlying framework. The same formulation applies to other editable algorithm components, predictor configurations, and optimization domains.