AUTOMATED ALGORITHM DISCOVERY THROUGH MATHEMATICAL REFORMULATION: A STUDY IN VISUAL-TOKEN REDUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated algorithm discovery typically searches within a fixed understanding of a task, mutating programs or recombining familiar heuristics. For algorithm-design tasks that admit useful mathematical abstraction, the mathematical formulation shapes algorithm design before implementation. Such a task may be expressed as different mathematical problems, each revealing different properties to preserve and different algorithmic structures. Each formulation admits multiple solution procedures with different states, updates, and outcomes. Our central insight is therefore to make both the mathematical formulation and its solution procedure explicit objects of search. We introduce a formulation-centric framework in which each proposal records a formulation–solution pair. An AI–Math–AI loop generates diverse formulations and a growing toolbox of mathematical mechanisms, retrieves relevant external knowledge, derives solution procedures, and implements them as executable programs. Its feedback mechanism balances exploitation and diversity: it allocates more search effort to mathematical areas associated with stronger proposals, while discouraging repeated exploration and redirecting attempts toward underexplored areas and tools. We study visual-token reduction in frozen multimodal large language models, where the framework discovers effective selectors grounded in reconstruction, information gain, and coverage. Their performance across tasks, token budgets, and model backbones demonstrates the practical value of exploring alternative mathematical views of the same problem. These results support searching over mathematical formulations and solution procedures as a productive direction for automated algorithm discovery.

1 INTRODUCTION

Language-model agents increasingly automate algorithm discovery by proposing ideas, generating code, and evaluating candidates (Lu et al., 2024; Yamada et al., 2025). Existing approaches often search through program mutations or heuristic recombination, using execution feedback to refine subsequent proposals (Romera-Paredes et al., 2024; Novikov et al., 2025; Liu et al., 2024a). For tasks that admit useful mathematical abstraction, choosing the mathematical formulation opens an additional dimension of algorithm design. A formulation determines which properties an algorithm seeks to preserve and which computational mechanisms become relevant. Searching over formulations brings this design choice into automated discovery.

Our central insight is that *one task admits multiple mathematical formulations, and each formulation admits multiple solution procedures*. For visual-token selection, reconstruction emphasizes preserving the span of the full feature set, information gain rewards complementary directions, and coverage seeks representatives close to every feature. These perspectives motivate different algorithmic states and updates, including residual projections, inverse-matrix updates, and nearest-representative distances. Even within a single formulation, different procedures can produce substantially different outcomes. Effective discovery therefore requires exploring both the mathematical objective and the computations used to pursue it.

We introduce a *formulation-centric framework* that makes both choices explicit through an *AI–Math–AI* loop. An LLM generates candidate formulations and a dynamic toolbox of mathematical

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.

mechanisms. Each attempt selects a formulation and a tool, retrieves relevant mathematical knowledge, and derives a *formulation–solution pair* specifying the objective, constraints, scoring rules, and state updates. This pair guides executable implementation and repair, connecting mathematical reasoning to evaluated behavior. Feedback combines evaluation outcomes with coverage, repetition, and failure records to inform subsequent proposals and encourage diverse tools and procedures. Separately selecting formulations and tools also enables mathematical mechanisms to transfer across objectives.

We evaluate the framework on visual-token reduction in frozen multimodal large language models, where competing formulations share a token budget and downstream evaluator. Automated discovery followed by component analysis identifies three families finalized through analysis grounded in reconstruction, information gain, and coverage. Under a matched budget of 150 attempts, our framework achieves a best discovery score 9.1 percentage points above the strongest evolutionary baseline. At 5% token retention, the final selectors preserve 72.3–73.3% of full-token performance on average across 14 benchmarks, compared with 63.1% for MMTok. Further evaluations show gains on benchmarks outside development, across token budgets, and on two additional backbones.

Our contributions are threefold: (i) a formulation-centric approach searching over mathematical formulations and solution procedures; (ii) an AI–Math–AI loop integrating mathematical retrieval, executable synthesis, and feedback-guided exploration; and (iii) an empirical study demonstrating improved discovery quality and effective visual-token reduction across tasks and backbones.

2 Related Work

Automating Research Workflows. The AI Scientist series connects idea generation, coding, experiments, analysis, and writing, including agentic tree search (Lu et al., 2024; Yamada et al., 2025). Agent Laboratory and AI-Researcher follow similar workflows (Schmidgall et al., 2025; Tang et al., 2025), while AutoSOTA coordinates agents to reproduce and improve models (Li et al., 2026). We organize discovery around alternative mathematical formulations of the same task.

Execution-Guided Algorithm Search. Execution-guided methods use measured performance to choose subsequent attempts. FunSearch evolves functions within a scaffold, AlphaEvolve edits code directly, and EoH jointly evolves heuristic descriptions and implementations (Romera-Paredes et al., 2024; Novikov et al., 2025; Liu et al., 2024a). ShinkaEvolve combines parent selection balancing exploration and reuse, code-similarity filtering, and feedback-driven generator-model selection (Lange et al., 2026). Autoresearch repeatedly modifies a training program under a fixed budget (Karpathy, 2026). Our *formulation–solution pairs* complement program evolution by deriving selection and update rules from objectives and constraints.

Knowledge-Guided Research and Discovery. MOOSE-Chem retrieves scientific inspirations for chemistry hypotheses; its successors add hierarchical search and simulated feedback (Yang et al., 2025c;b; Liu et al., 2025b). EvoScientist builds reusable memories from previous attempts (Lyu et al., 2026), while DeepEvolve combines retrieval with algorithm evolution, implementation, debugging, and evaluation (Liu et al., 2025a). We use retrieved mathematics to derive task-specific procedures, with a growing toolbox supporting tool reuse across formulations.

Visual-Token Reduction. VisionZip retains tokens receiving high vision-encoder attention and merges the remainder by feature similarity (Yang et al., 2025a). DivPrune maximizes minimum pairwise cosine distance, while CDPruner maximizes a DPP determinant based on visual similarity and question relevance (Alvar et al., 2025; Zhang et al., 2025). MMTok greedily selects visual tokens that jointly cover text and visual tokens (Dong et al., 2026). Our discovered selectors connect to prior methods as follows: ORGS targets feature reconstruction through largest-residual selection and orthogonal projection updates. Leverage maintains an inverse regularized information matrix, whereas CDPruner uses a question-reweighted pairwise kernel. FPS follows DivPrune's farthest-point rule with variance-scaled squared Euclidean distances and a different initialization.

3 Background and Preliminaries

Visual-token reduction. A vision–language model encodes an image into N visual tokens. Let $[N] = \{1, \ldots, N\}$. Let $X \in \mathbb{R}^{N \times d}$ denote the features available for selection and $V \in \mathbb{R}^{N \times d_v}$

the aligned embeddings supplied to the language model. Row i of both matrices corresponds to the same image position. Given an optional question q and a token budget K, a selector a returns $S_a = a(X, V, q, K) \subseteq [N]$, with $|S_a| = K$. The fixed surrounding model gathers V_{S_a} in the original spatial order and updates its positions and attention mask. All model parameters remain frozen, allowing selectors to be compared through downstream answer quality under the same token budget.

Mathematical formulations. A *formulation* translates an AI task into a mathematical problem by specifying its variables, objective, and constraints. Different formulations emphasize different properties to preserve. For visual-token reduction, let $Z = g(X) \in \mathbb{R}^{N\times d}$ denote transformed selection features, with row $z_i^\top$, and let Z_S contain the selected rows. Representative formulations include coverage, $\min_{S\subseteq[N],\,|S|=K} \max_{i\in[N]} \min_{j\in S} \|z_i - z_j\|_2^2$; reconstruction, $\min_{S\subseteq[N],\,|S|=K,\,A\in\mathbb{R}^{N\times K}} \|Z - AZ_S\|_F^2$; and information gain, $\max_{S\subseteq[N],\,|S|=K} \log\det(I_d + \lambda^{-1}\sum_{i\in S} z_i z_i^\top)$, where $\lambda > 0$. Coverage seeks representatives close to every token, reconstruction asks whether selected vectors can rebuild the full feature set, and information gain rewards complementary feature directions. While a formulation defines *what* to preserve, constructing S requires choosing the transformation g, distance measure, optimization strategy, and numerical parameters.

From a formulation to an algorithm. A *mathematical area* is a field such as geometry, algebra, or information theory. A *mathematical tool* is a finer-grained concept or algorithmic mechanism from such an area, such as a metric-space distance, orthogonal projection, spectral filter, or matrix inverse update. Each tool can support multiple formulations, and each formulation can use multiple tools.

A *solution procedure* specifies a formulation's initialization, candidate scoring, selection, state updates, and stopping. Its *state* comprises quantities retained between selection steps. A *proposal* records an instantiated formulation and its solution procedure as a *formulation–solution pair*. Different solution procedures can pursue the same preservation goal.

4 Method: Discovering Algorithms through Mathematical Reformulation

4.1 Workflow Overview

Our framework varies task formulations and solution procedures through initialization followed by an iterative AI–Math–AI loop. During initialization, an LLM generates candidate formulations and constructs a mathematical toolbox organized into areas and tools. During each ordinary search attempt, the system performs the following steps: (*i*) selects a formulation and a tool; (*ii*) expands the tool into retrieval queries and retrieves relevant mathematical knowledge; (*iii*) uses the retrieved knowledge to instantiate the formulation and derive a solution procedure; (*iv*) implements and validates the procedure; and (*v*) evaluates the candidate and updates the search records.

Each attempt produces either an evaluated candidate or a recorded failure. Repairs remain within the attempt that generated the candidate, and all attempts count toward the search budget. The automated loop returns executable candidates together with their formulations, solution procedures, retrieved evidence, code, and evaluation outcomes. The matched attempt comparison uses a breadth-oriented search topology: ordinary candidates are sibling attempts generated from independently selected formulation–tool pairs.

4.2 Opening the Search Space through Multiple Formulations

Insight: one task admits multiple mathematical problems. We open the algorithm search space by varying the mathematical problem that the program is intended to solve. Given the task description, inputs, output constraints, and token budget, an LLM generates formulations expressing preservation goals. The reported search begins with 23 formulations, including coverage, reconstruction, information-theoretic, transport, control, and robustness objectives.

These formulations define a problem-level basis for algorithm discovery. For example, coverage naturally raises questions about representatives and distances, reconstruction motivates spans and residuals, and information gain motivates determinants and sequential matrix updates. The same downstream evaluator compares candidates across these mathematical views.

The scheduler alternates between visual-only formulations, which use image features alone, and question-dependent formulations, which may additionally use q and its embeddings. Within each mode, formulations are explored according to a fixed schedule, while saturated formulation–procedure groups can be temporarily paused to leave budget for other mathematical views. Appendices E.2 and E.3 list the formulations and scheduling settings.

4.3 Constructing a Dynamic Mathematical Toolbox

Insight: a formulation admits multiple solution procedures. Mathematical tools offer alternative ways to instantiate and solve the same formulation. We therefore factor proposal generation into a formulation and a separately selected mathematical tool. This allows the same formulation to yield multiple algorithms and allows a useful tool to be transferred across formulations.

The LLM first identifies broad mathematical areas and generates tools associated with them. The reported primary run begins with 158 seed tools. Examples from geometry include metric-space distances, proximity ordering, ϵ-net sampling, coresets, and geometric projection; examples from algebra include iterative refinement, spectral filtering, matrix-based updates, and quantization. The toolbox stores concise descriptions of reusable mathematical tools. Tools are organized by area, while their later use in proposals can combine concepts from multiple areas.

The toolbox is dynamic. The system tracks tool and operation attempts, including retrieval and execution failures. When initial tools become heavily used, few untried tools remain, or a periodic refresh is reached, the LLM proposes additional tools. Expansion emphasizes lightly explored mathematical areas and new combinations of operations, while duplicate tools and tools similar to recent failures receive lower priority. A tool is normally set aside after limited use and can be revisited to derive a different solution procedure.

4.4 Retrieving Knowledge for a Selected Tool

Insight: retrieval turns a mathematical direction into usable algorithmic ingredients. A short tool description may correspond to several theoretical concepts, problem settings, and solution strategies. For a selected tool, the LLM therefore applies query expansion to generate three to five complementary *retrieval queries*. The queries target definitions, theoretical properties, related optimization problems, established algorithms, update rules, and computational guarantees. They avoid software-library and task-specific implementation terms so that retrieval remains focused on reusable mathematics.

The queries retrieve titles and abstracts from arXiv, OpenAlex, and OpenReview, supplemented by definition articles from Wikipedia. Retrieval removes duplicate titles, entries without usable content, and material unrelated to the selected tool. If the resulting evidence is insufficient, the search widens its publication window and retries.

An extraction call is instructed to extract only source-supported objectives, selection or update operations, intermediate quantities, data structures, and computational costs. Unresolved attributes are explicitly marked as uncertain. The resulting record supplies external mathematical ingredients for deriving a task-specific algorithm.

4.5 Deriving a Formulation–Solution Pair

The proposal model receives the selected formulation, mathematical tool, retrieved evidence, task interface, and summaries of relevant previous outcomes. It first instantiates the formulation by specifying task-dependent choices such as the feature transformation g, distance or similarity measure, regularization, and numerical parameters. It then derives a solution procedure with:

1. an initialization of the selected set and algorithmic state;
2. a score or decision rule for choosing the next token;
3. an update rule for the maintained state;
4. a stopping condition at K selected tokens; and
5. pseudocode, complexity, and numerical precautions.

This ordering separates mathematical reasoning from code generation. The formulation specifies the desired property, the selected tools and retrieved evidence suggest ways to pursue it, and the solution procedure converts these ingredients into concrete computations over the task tensors.

For example, consider the coverage formulation. A geometry tool concerning divergence-based distances leads retrieval to quadratic Bregman divergences. Proposal generation instantiates the coverage distance using variance-scaled squared Euclidean distance and combines it with farthest-point selection. The resulting procedure maintains the selected set S and a distance vector δ, where δ_i is token i's squared distance to its nearest selected representative. It repeatedly chooses:

$$c = \arg\max_{i \in [N] \setminus S} \delta_i, \qquad S \leftarrow S \cup \{c\}, \qquad \delta_i \leftarrow \min\{\delta_i, D(i, c)\}, \tag{1}$$

until $|S| = K$. The formulation determines the coverage goal, while the retrieved distance and sequential update determine how that goal is pursued.

A formulation–tool pair can yield multiple solution procedures. Retrieved sources, feature mappings, initialization choices, or state-update rules can yield distinct procedures for the same formulation. Conversely, the same mathematical tool may play roles under different formulations. This many-to-many relationship provides the main source of algorithmic diversity in the framework.

4.6 Broad Exploration and Feedback-Guided Toolbox Expansion

Insight: broad mathematical coverage complements evolutionary depth. The ordinary search allocates attempts across formulations, mathematical areas, and computational operations. A search-space coverage table records how often combinations such as geometry with distance computation or algebra with projection have been explored. During the initial coverage phase, the scheduler prioritizes untried or lightly explored combinations. It then samples tools using coverage, usage, and failure records, with preference for new mathematical mechanisms.

The system also records an operation signature for each proposed procedure, including its selection rule and ordered state updates. Proposals substantially repeating recent computations can be rejected or revised, while computation-group records guide subsequent exploration. These mechanisms encourage revisiting a formulation or tool to develop distinct selection rules and state updates.

Feedback has three roles. First, coverage and tool-usage records identify where additional mathematical exploration is needed. Second, retrieval failures and execution diagnostics reduce repeated unsuccessful attempts and guide toolbox expansion or repair. Third, downstream scores identify empirically useful procedures and provide outcome summaries for later proposal generation. Coverage drives the breadth-oriented phase, while scores inform subsequent proposals within its sibling-attempt topology.

4.7 Implementation, Validation, and Evaluation

A candidate program implements the proposed solution procedure inside a fixed token-selection block. The code generator receives the proposal, available tensors and their shapes, output requirements, resource constraints, and numerical precautions. Fixed surrounding code gathers the selected rows of V and updates the model inputs; the generator operates within the selection block while the model and evaluator remain fixed.

Generated programs pass source and execution checks before downstream evaluation. Source checks enforce syntax, the editable-code boundary, the token budget, permitted operations, and numerical requirements. Synthetic probes test termination, tensor compatibility, and the return of exactly K unique, in-range indices. Using different dimensions for X and V exposes invalid operations that mix their feature spaces. Programs producing the same selections as a previous candidate on every fixed probe can be screened as behavioral duplicates.

Failures trigger bounded repair. Deterministic rules handle recurring mechanical errors, while an LLM receives the proposal and execution diagnostics to repair remaining issues. The proposal acts as the reference for preserving the intended scores and state updates during repair. Each revised program re-enters the validation sequence.

A candidate that passes the active checks is evaluated with the frozen vision–language model, producing a common discovery score U. The outcome record stores the formulation, tool, retrieved

evidence, proposed procedure, scored code version, validation results, and discovery score or failure status. These artifacts link each proposal and implemented procedure to its evaluation outcome.

5 Representative Discovered Selection Algorithms

FPS: coverage through scaled distances. For the coverage formulation, *Variance-Scaled Farthest-Point Selection* (FPS) applies coordinate-wise variance scaling and uses the resulting squared Euclidean distance for farthest-point selection. Starting from a fixed-seed token, it repeatedly selects the point with the largest distance to its nearest selected representative and updates these nearest-representative distances until K tokens are selected. It measures coverage using squared Euclidean distance between the scaled features, $D(i,j) = \|z_i - z_j\|_2^2$, stores nearest-representative squared distances δ_i and applies the selection and update rules in Equation (1) until $|S| = K$. Each step fills the largest remaining coverage gap.

ORGS: reconstruction through residuals. For the reconstruction formulation, retrieval on centroid selection supports a procedure expanding the span of retained vectors: *Orthogonal Residual Greedy Selection* (ORGS). It standardizes each feature coordinate by subtracting its mean and dividing by its clamped sample standard deviation (Appendix C.3). It initializes residuals $r_i = z_i$ and an empty orthonormal basis of mutually perpendicular unit vectors for the selected span. Each step selects the unselected token maximizing $\|r_i\|_2^2$, applies Gram–Schmidt orthogonalization to its residual against the basis, and normalizes it to direction v. The update $r_i \leftarrow r_i - (z_i^\top v)v$ removes this direction; under exact orthogonality, $z_i^\top v = r_i^\top v$. The score heuristically chooses a direction; projection reduces reconstruction error.

Leverage: information preservation through ridge updates. Motivated by an information gain formulation that preserves reconstructive information while limiting redundancy, *Sequential Information Gain* (Leverage) uses ridge-leverage scores and sequential Sherman–Morrison updates motivated by log-determinant information gain (Appendix C.3). For raw features $x_i = X_i^\top$, it maintains a matrix state P, initialized as $P = \lambda^{-1} I_d$ with $\lambda > 0$. Each step selects the unselected token maximizing $x_i^\top P x_i$ and updates $P \leftarrow P - (Px_c)(Px_c)^\top / (1 + x_c^\top P x_c)$. The score favors weakly represented directions, while the update discounts each newly selected direction.

6 Experiments

6.1 Experimental Setup

Discovery and development. Discovery uses the frozen Qwen2.5-VL-7B model on 1,000 OCR-Bench samples, with fixed input resolution and 5% token retention. We compare the full framework with three evolutionary controls (FunSearch-style (Romera-Paredes et al., 2024), EoH-style (Liu et al., 2024a), and ShinkaEvolve-style (Lange et al., 2026)) and three ablations: direct mutation, retrieval removal, and removal of feedback-guided strategy selection. Direct mutation and the evolutionary controls use the MMTok (Dong et al., 2026) coverage selector as their seed program due to its strong performance.

An attempt covers proposal generation, implementation, validation, bounded repair, and evaluation; failed attempts still count toward the budget. All configurations share the generator (`deepseek-v4-pro`), initial MMTok coverage selector, task interface, token budget, and evaluator. Appendix E describes implementation, validation, and search controls; Appendix A.5 specifies the matched-search comparison protocol and accounting rules. Representative families are evaluated on OCRBench, ChartQA, and DocVQA to determine the final versions of ORGS, Leverage, and FPS; these three benchmarks are therefore treated as development tasks.

Final evaluation. We evaluate the final selectors at 20%, 10%, and 5% token retention on 14 benchmarks with Qwen2.5-VL-7B and on seven benchmarks each with SAIL-VL-2B and InternVL3.5-4B. Eleven benchmarks are outside development. VisionZip (Yang et al., 2025a), DivPrune (Alvar et al., 2025), CDPruner (Zhang et al., 2025), and MMTok (Dong et al., 2026) are compared on seven common benchmarks; MMTok additionally provides the common reference on the remaining benchmarks and transfer backbones. All methods share the input pipeline and per-image budget K, using native dynamic resolution for Qwen and native tiling for the transfer backbones.

Metrics. The discovery score U is OCRBench accuracy reported as a percentage. Component comparisons equally average OCRBench, ChartQA, and DocVQA scores, all reported as percentages. Final evaluation reports mean performance retention, $R_{a,b} = 100 s_{a,b}/s_{\text{full},b}$, where both scores use the same model and protocol. Throughout the paper, accuracy, ANLS, F1, their component-comparison means, and performance retention are reported as percentages; differences are expressed in percentage points. Tables omit percentage signs from individual task-score entries; mean performance-retention entries may retain percentage signs. MME retains its native score scale. Appendix A gives dataset and decoding details; Appendix C.4 specifies selector settings; and Appendices E.3 and E.10 provide the search and control configurations.

6.2 Discovery Quality and Framework Contributions

Better discovery under the same budget. Within 150 attempts, the full framework reaches 49.3%, exceeding the strongest evolutionary control, ShinkaEvolve-style (Lange et al., 2026) at 40.2%, by 9.1 percentage points (Table 1). It first exceeds 40% at attempt 36 and reaches its best score at attempt 44. Appendix B.1 reports yield and hit rates at multiple score thresholds; the framework also produces six candidates with $U \geq 40\%$, compared with three without retrieval, two for ShinkaEvolve-style search, and none for direct mutation, FunSearch-style (Romera-Paredes et al., 2024), or EoH-style (Liu et al., 2024a) search. The framework improves both the best score and high-score yield. Appendix B.6 analyzes the evolutionary controls' outcomes.

Table 1: **Automated algorithm discovery baseline comparison.** Best accuracy on OCRBench.

Method	EoH-style	FunSearch-style	ShinkaEvolve-style	**Ours**
Discovery score U (%) ↑	39.20	39.90	40.20	**49.30**

Diverse high-performing solution procedures. Candidates use different solution procedures if their token scores or maintained states and updates differ. Specifically, residual methods update the unrepresented component of each feature, leverage methods update an inverse information matrix, and kernel methods update determinants or Cholesky factors.

Under this definition, the six high-scoring candidates from the full framework form five algorithm types: two residual-based procedures, one-shot SVD leverage scoring, kernel-determinantal selection with sampling refinement, and kernel power-function selection with incremental Cholesky updates. Without retrieval, three high-scoring types are found; the two high-scoring ShinkaEvolve candidates both use greedy coverage. Formulation-centric search therefore yields high-performing procedures spanning residual, leverage, and kernel-based selection. Appendix B.2 lists the high-scoring candidates and their algorithm and output clusters.

Table 2: **Framework ablations and extended exploration.** Best discovery accuracy on OCRBench. The first four variants use 150 attempts; the extended run reaches 52.1%.

Variant	Direct mutation	w/o retrieval	w/o guided strategy selection	Ours 150 attempts	**Ours** extended run
Discovery score U (%) ↑	39.90	44.90	39.20	49.30	**52.10**

Contributions of formulation, retrieval, and feedback. Under the same 150-attempt budget, replacing mathematical reformulation with direct mutation reduces the best score from 49.3% to 39.9%, a 9.4-point drop (Table 2). This ablation directly mutates the best program, bypassing formulation and tool selection, retrieval, and formulation–solution derivation; the resulting drop demonstrates the value of formulation-guided discovery (Appendix B.5). Even without retrieval, reformulation and solution derivation yield better-performing, more diverse algorithms from the LLM's internal mathematical knowledge, raising the best score to 44.9%. Adding retrieval raises the score to 49.3%, doubles the number of candidates above 40% from three to six, and increases the number of high-scoring algorithm types from three to five. Retrieval therefore improves both discovery quality and the variety of successful mechanisms. Appendix B.3 provides the retrieval comparison. Feedback supplies evaluation outcomes to subsequent proposals, while coverage and reuse records discourage repeated exploration. Removing history-based guidance and randomizing strategy choices reduces the best score from 49.3% to 39.2% (Appendix B.4).

Toolbox expansion and program repair. Extended exploration with toolbox expansion improves the discovery score from 49.3% within 150 attempts to 52.1% during extended exploration at attempt 441. Added tools produce 24 candidates with $U \geq 45\%$, a best score of 52.1%, and nine behavioral clusters covering subspace, residual, density-based, and inverse-matrix procedures. This gain is not explained solely by additional search attempts: under the same extended-search budget, the variant without toolbox expansion does not show a comparable improvement over the 150-attempt result. Appendix B.8 summarizes toolbox expansion and high-scoring output clusters. Proposal-guided repair recovers ideas from failed implementations: 78 of 140 traceable repair episodes yield valid evaluations (55.7%), including eleven candidates with $U \geq 40\%$ (Appendix E.12.1).

6.3 Why the Discovered Selectors Work

Table 3: **Key component ablations.** Selector performance and component ablations at $\rho = 0.05$ under dynamic resolution. Scores and their three-task mean are reported in percent.

Family	Variant	OCRBench	ChartQA	DocVQA	Mean
ORGS	Discovered candidate	**51.20**	**34.72**	**48.88**	**44.93**
	w/o orthogonalization	41.30	26.80	34.94	34.35
	w/o standardization and orthogonalization	29.40	23.68	23.47	25.52
	Row-ℓ_2 normalization instead of standardization	25.80	27.96	23.08	25.61
Leverage	Discovered candidate	**48.10**	**36.12**	**48.10**	**44.11**
	w/o sequential matrix update	30.80	23.44	23.53	25.92
FPS	Discovered candidate	**48.70**	**32.96**	**48.13**	**43.26**
	w/o variance scaling	31.70	23.92	27.49	27.70
	Cosine distance	12.40	25.24	16.49	18.04

Table 3 reports component ablations for the three discovered selectors. ORGS removes each selected direction from the remaining residuals, guiding subsequent selections toward unrepresented feature directions. Removing this orthogonalization lowers the three-task mean from 44.93% to 34.35%. Removing both orthogonalization and coordinate standardization lowers it further to 25.52%, while replacing standardization with row-ℓ_2 normalization gives 25.61%. These results support combining residual updates with coordinate standardization for effective reconstruction. Leverage updates its inverse information matrix after each selection to discount represented directions and favor complementary tokens. Removing this update lowers the three-task mean to 25.92%, supporting sequential information updates. FPS uses variance scaling to balance feature coordinates in the coverage distance. Removing this scaling lowers the mean from 43.26% to 27.70%, while replacing the scaled squared Euclidean distance with cosine distance gives 18.04%. These comparisons identify variance-scaled squared Euclidean distance as a key component of effective coverage selection.

Appendix C.2 reports the smaller component effects used to finalize the selectors. The component results support searching over complete formulation–solution pairs: the objective defines what to preserve, while the update rules and feature geometry determine whether it is preserved effectively.

6.4 Generalization Across Tasks, Token Budgets, and Backbones

Generalization beyond development. At 5% token retention, ORGS, Leverage, and FPS retain 73.3%, 73.2%, and 72.3% of full-token performance across all 14 Qwen benchmarks, compared with 63.1% for MMTok. Table 4(a) reports the shared-benchmark results. On the 11 benchmarks outside development, their means are 79.2%, 79.3%, and 78.4%, compared with 72.1% for MMTok. Each selector outperforms MMTok on all 11 out-of-development benchmarks at all three budgets, demonstrating consistent gains beyond the development tasks.

At 10% retention, the selectors preserve 86.0%–87.2% on the 11 out-of-development benchmarks, compared with 81.1% for MMTok; at 20%, they retain 92.7%–93.2%, compared with 89.4%. Appendix Tables 22 and 23 provide the complete task-level results.

Larger gains under tighter budgets. Across seven shared benchmarks, the best discovered selector retains 96.3%, 91.4%, and 82.7% of full-token performance at 20%, 10%, and 5% token retention, versus the strongest baseline's 90.5%, 82.3%, and 71.6%. Gains rise from 5.8 to 9.1 to 11.1 points as

Table 4: Evaluation across tasks, token budgets, and backbones. (a) Qwen2.5-VL-7B task scores (%, except MME); Avg.† denotes mean performance retention. (b) Mean performance retention (%) on seven additional QwenVL tasks and seven tasks per transfer backbone. Bold marks the best compressed result at each budget.

(a) QwenVL task-level results

Method	GQA	MMB	MME	POPE	TextVQA	SQA	OCRBench	Avg.†
Full tokens	60.48	83.25	2327	86.16	82.95	87.46	83.80	100%
10% tokens								
VisionZip (Yang et al., 2025a)	52.47	75.60	2003	78.90	46.17	82.30	36.90	78.4%
DivPrune (Alvar et al., 2025)	53.43	72.85	1957	74.99	53.77	79.57	37.30	78.2%
CDPruner (Zhang et al., 2025)	51.76	73.88	1946	73.59	63.47	82.85	42.50	80.8%
MMTok (Dong et al., 2026)	55.09	74.74	2051	78.75	59.33	80.47	43.60	82.3%
ORGS	**55.92**	77.06	**2218**	83.42	74.86	**83.19**	**64.60**	**91.4%**
Leverage	**55.92**	**77.23**	2180	**83.50**	**74.95**	82.80	63.20	90.9%
FPS	55.47	76.46	2137	81.81	74.06	82.00	60.60	89.4%
5% tokens								
VisionZip (Yang et al., 2025a)	46.28	67.53	1677	66.38	35.24	79.57	19.70	66.2%
DivPrune (Alvar et al., 2025)	49.01	65.89	1739	68.45	39.03	77.05	24.90	68.5%
CDPruner (Zhang et al., 2025)	47.33	68.13	1739	64.75	50.62	80.17	28.10	70.9%
MMTok (Dong et al., 2026)	50.66	65.89	1796	71.35	44.15	77.19	30.70	71.6%
ORGS	50.85	71.22	**2019**	75.48	**67.62**	**80.52**	**51.20**	**82.7%**
Leverage	**51.72**	**72.59**	2010	**76.61**	67.42	80.47	48.10	**82.7%**
FPS	50.89	71.74	2011	74.65	66.29	80.22	48.10	81.8%

(b) Additional tasks and backbone transfer

	Qwen2.5-VL-7B			SAIL-VL-2B			InternVL3.5-4B		
Method	20%	10%	5%	20%	10%	5%	20%	10%	5%
MMTok (Dong et al., 2026)	81.5	66.6	54.6	78.8	75.2	69.7	73.4	68.7	65.2
ORGS	86.2	**75.6**	**63.9**	**91.9**	85.0	**77.9**	**88.0**	**80.5**	71.9
Leverage	**86.3**	75.4	63.6	91.4	85.1	77.5	87.7	80.1	**72.0**
FPS	85.9	75.0	62.8	91.3	**85.3**	77.4	86.7	78.5	70.3

budgets tighten, consistent with a greater need to avoid redundant directions. At 5% token retention, our visual-only selectors generate the first token 1.37–1.44× faster, including selection overhead (Appendix D.4), while allowing selected tokens to be reused across questions.

Gains span benchmark types but peak on detail-sensitive tasks: at 5% retention, ORGS scores 51.20% on OCRBench and 67.62% on TextVQA, versus MMTok's 30.70% and 44.15%. Reconstruction, information gain, and coverage yield competitive selectors with similar averages but different rankings across tasks and budgets, supporting exploration across formulations.

Transfer across backbones. With objectives, updates, and parameters unchanged, all three selectors outperform MMTok in mean performance retention at every budget on SAIL-VL-2B and InternVL3.5-4B (Table 4(b)). At 5% retention, they retain 77.4%–77.9% and 70.3%–72.0%, versus MMTok's 69.7% and 65.2%, respectively. Gains peak on TextVQA and OCRBench, while MMTok remains competitive on some GQA and POPE settings (full results in Appendix D.3). Together, cross-task and cross-model gains support the generality of the discovered algorithms.

7 Conclusion and Discussion

Our AI–Math–AI framework connects mathematical knowledge to executable algorithms through formulation–solution pairs and a growing toolbox. It improves discovery scores under a shared attempt budget and, through component analysis, yields ORGS, Leverage, and FPS with strong generalization. To complement breadth-oriented discovery, we also explored depth by selecting previously discovered algorithms as parents and deriving, implementing, validating, and evaluating their variants. This refinement yielded local improvements within existing algorithm families but did not surpass the best breadth-discovered candidate (Appendix B.9).

AI USE STATEMENT

We use generative AI to derive proposals, generate code, and repair programs in the discovery pipeline. We also used generative AI to review mathematical reasoning and methodological consistency, provide feedback on experimental methodology, interpret experimental results, and assist with language editing and LaTeX drafting. We reviewed the resulting text, mathematical claims, citations, and experimental materials. Generated programs undergo the source and execution checks in Section 4.7. We take responsibility for the paper and its accompanying materials.

ETHICS STATEMENT

This work studies automated algorithm discovery on existing benchmark data. Generated code is limited to a specified region, and evaluation code stays fixed. Source review and runtime checks enforce the interface and token budget. Reported discovery scores exclude known invalid results identified by audits and reflect the checks active during evaluation. Post-hoc probe failures are reported separately (Appendices E.8 and A.5).

REPRODUCIBILITY STATEMENT

Section 3 defines the task interface and terminology, and Section 4 describes the discovery process and evaluation feedback. Section 6.1 defines the metrics and evaluation protocols. Appendices A–F provide implementation details, discovery evidence, complete results, and prompt templates. The anonymized supplementary material will include candidate code, run configurations, software and application programming interface (API) versions, and random seeds.

REFERENCES

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. DivPrune: Diversity-based visual token pruning for large multimodal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9392–9401, 2025.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2403.20330.

Sixun Dong, Juhua Hu, Mian Zhang, Ming Yin, Yanjie Fu, and Qi Qian. MMTok: Multimodal coverage maximization for efficient inference of VLMs. In *International Conference on Learning Representations*, 2026.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. MME: A comprehensive evaluation benchmark for multimodal large language models, 2023. URL https://arxiv.org/abs/2306.13394.

Drew A Hudson and Christopher D Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6700–6709, 2019.

Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch. Accessed 2026-08-04.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV*, pp. 235–251. Springer, 2016. URL https://arxiv.org/abs/1603.07396.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/7886b9bafe76c52fd568db10ff9772df-Paper-Conference.pdf.

Bohao Li, Yuying Ge, Yi Chen, Yixiao Ge, Ruimao Zhang, and Ying Shan. SEED-Bench-2-Plus: Benchmarking multimodal large language models with text-rich visual comprehension, 2024. URL https://arxiv.org/abs/2404.16790.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models, 2023. URL https://arxiv.org/abs/2305.10355.

Yu Li, Chenyang Shao, Xinyang Liu, Ruotong Zhao, Peijie Liu, Hongyuan Su, Zhibin Chen, Qinglong Yang, Anjie Xu, Yi Fang, Qingbin Zeng, Tianxing Li, Jingbo Xu, Fengli Xu, Yong Li, and Tie-Yan Liu. AutoSOTA: An end-to-end automated research system for state-of-the-art AI model discovery. *arXiv preprint arXiv:2604.05550*, 2026. doi: 10.48550/arXiv.2604.05550. URL https://arxiv.org/abs/2604.05550.

Fei Liu, Tong Xialiang, 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. In *Proceedings of the 41st International Conference on Machine Learning*, 2024a.

Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. Scientific algorithm discovery by augmenting AlphaEvolve with deep research. *arXiv preprint arXiv:2510.06056*, 2025a. URL https://arxiv.org/abs/2510.06056.

Wanhao Liu, Zonglin Yang, Jue Wang, Lidong Bing, Di Zhang, Dongzhan Zhou, Yuqiang Li, Houqiang Li, Erik Cambria, and Wanli Ouyang. MOOSE-Chem3: Toward experiment-guided hypothesis ranking via simulated experimental feedback. *arXiv preprint arXiv:2505.17873*, 2025b. doi: 10.48550/arXiv.2505.17873. URL https://arxiv.org/abs/2505.17873.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. MMBench: Is your multi-modal model an all-around player? In *European Conference on Computer Vision*, pp. 216–233. Springer, 2024b.

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. OCRBench: On the hidden mystery of OCR in large multimodal models. *Science China Information Sciences*, 67(12):220102, December 2024c. doi: 10.1007/s11432-024-4235-6. URL https://doi.org/10.1007/s11432-024-4235-6.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. doi: 10.48550/arXiv.2408.06292. URL https://arxiv.org/abs/2408.06292.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026. doi: 10.48550/arXiv.2603.08127. URL https://arxiv.org/abs/2603.08127.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minesh Mathew, Dimosthenis Karatzas, and C.V. Jawahar. DocVQA: A dataset for VQA on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2200–2209, January 2021.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. InfographicVQA. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 1697–1706, 2022. URL https://arxiv.org/abs/2104.12756.

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

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

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025. doi: 10.48550/arXiv.2501.04227. URL https://arxiv.org/abs/2501.04227.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326, 2019.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous scientific innovation. *arXiv preprint arXiv:2505.18705*, 2025. URL https://arxiv.org/abs/2505.18705.

xAI. Grok-1.5 Vision Preview. https://x.ai/blog/grok-1.5v, 2024. Introduces the RealWorldQA benchmark for real-world spatial understanding.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025. doi: 10.48550/arXiv.2504.08066. URL https://arxiv.org/abs/2504.08066.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. VisionZip: Longer is better but not necessary in vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19792–19802, 2025a.

Zonglin Yang, Wanhao Liu, Ben Gao, Yujie Liu, Wei Li, Tong Xie, Lidong Bing, Wanli Ouyang, Erik Cambria, and Dongzhan Zhou. MOOSE-Chem2: Exploring LLM limits in fine-grained scientific hypothesis discovery via hierarchical search. In *Advances in Neural Information Processing Systems 38*, 2025b. URL https://arxiv.org/abs/2505.19209.

Zonglin Yang, Wanhao Liu, Ben Gao, Tong Xie, Yuqiang Li, Wanli Ouyang, Soujanya Poria, Erik Cambria, and Dongzhan Zhou. MOOSE-Chem: Large language models for rediscovering unseen chemistry scientific hypotheses. In *International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=X9OfMNNepI.

Qizhe Zhang, Mengzhen Liu, Lichen Li, Ming Lu, Yuan Zhang, Junwen Pan, Qi She, and Shanghang Zhang. Beyond attention or similarity: Maximizing conditional diversity for token pruning in MLLMs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2025.

Organization of the supplementary material. The supplementary material is organized into six appendices. Appendix A defines the experimental protocols, counting units, and evaluation metrics. Appendix B reports matched-budget discovery results, ablation studies, and extended exploration. Appendix C traces candidate origins, validates the core mechanisms produced by discovery through diagnostic component perturbations, and specifies the selector implementations used for final evaluation. Appendix D presents complete benchmark and inference-efficiency results for these selectors. Appendix E details the discovery workflow, implementation settings, and record accounting needed for reproduction. Finally, Appendix F provides prompt templates, historical discovery records, and a worked formulation-to-code example.

A Experimental Protocols

Appendix A.1 defines attempts, completion records, valid evaluations, and the two probe protocols. Appendix A.2 specifies the inputs, outputs, and editable region of the generated selection block. Appendix A.3 explains dataset roles and the definitions of discovery accuracy and performance retention. Appendix A.4 summarizes the discovery and final-evaluation settings. Appendix A.5 defines the matched-search comparison and post-hoc behavioral clustering protocols.

A.1 Counting Units and Validation Protocols

An *attempt* is one outer search cycle, covering proposal generation, implementation, validation, and downstream evaluation if reached. Every attempt counts toward the matched search budget, including failures and rejections; local repairs remain within the originating attempt. The matched comparisons use attempt IDs 1–150 for each configuration.

A *completion record* is a timestamped outcome entry in the saved search state. Records and attempt IDs describe distinct counting units: an ID can recur after a restart, and an interrupted attempt can lack a completion record. Extended-run source and score statistics count completion records; Appendix E.12.1 reconciles repeated and missing IDs.

A *valid evaluation* completes successfully and records a finite numeric score without an execution-failure flag. Numeric harness values attached to failed executions are excluded. Validity here concerns the recorded downstream evaluation. Post-hoc probe outcomes are recorded separately, with the original evaluation score retained.

We distinguish two uses of synthetic probes. *Discovery-time probes* check generated programs before downstream evaluation, with a 180 s process timeout and failures routed to repair. *Post-hoc behavioral probes* compare the evaluated code versions on a common cached suite under empty-question and fixed-question conditions, with a 60 s process limit. A failure in a post-hoc condition excludes a candidate from clustering under that condition. The matched-comparison protocol below specifies the cached inputs and cluster definition; discovery-time implementation details appear in Appendix E.7.

A.2 Interface of the Generated Selection Block

We call X the *selection features* and V the *model-input embeddings*. The implementation interface calls these CLIP-space features and projector-space embeddings, respectively. These are interface labels; the vision encoder depends on the backbone. Table 5 gives their shapes for discovery.

Code generation edits `select_vision_tokens`. The permitted region spans `# --- ALGORITHM CORE START ---` and `# --- ALGORITHM CORE END ---`. It replaces this block in a fixed class template. A legacy marker pair is also supported.

The generated block assigns `selected_indices`, an ascending `torch.long` tensor with exactly K indices on the input device. These indices select model-input embedding rows. Visual features and question embeddings occupy different feature spaces. A question-dependent procedure must use compatible model-input embeddings or specify a mapping between spaces.

Fixed surrounding code validates the budget and supplies the function's return statement. It removes unselected tokens, pads sequences, and updates `position_ids` and `attention_mask`. The

Table 5: Inputs to the generated block in the discovery configuration. Rows of the two visual tensors refer to the same token positions.

Input	Shape or type	Role
`image_features`	$[N, 1280]$	Selection features X, used by the geometric selectors
`image_embeds`	$[N, 3584]$	Model-input embeddings V
`text_token_embedding`	$[m, 3584]$	Embeddings of the m question tokens in the model space
`question`	String	Question text available to question-dependent candidates
`image_grid_thw`	Optional grid metadata	Temporal and spatial grid shape; passed as `None` on the discovery evaluation path
`target_vision_tokens`	Integer K	Validated selection budget, $0 < K < N$ for calls to the editable block

evaluator enforces the token count derived from the prescribed token-retention ratio. Figure 7 lists the numerical and resource requirements.

Inserting the generated code. The merging step extracts the code block and locates the selection markers in the fixed template. It adjusts indentation, supplies missing markers, and inserts the block. The surrounding class and model pipeline remain fixed. The merged file must parse and assign `selected_indices`. Further checks cover types, prohibited operations, and changes outside the permitted region. Error reports enter the repair process in Section 4.7.

A.3 Dataset Roles and Score Definitions

Discovery evaluates candidates on 1000 OCRBench samples. Component analysis uses OCRBench, ChartQA (Masry et al., 2022), and DocVQA (Mathew et al., 2021). These three benchmarks serve as development tasks throughout the paper. The appendix uses them to assess the contributions of components in discovered algorithm families through controlled removals and replacements, taking each discovered procedure as the reference. Final evaluation adds 11 benchmarks to these tasks; their mean reports performance outside development.

The seven-benchmark reference comparison includes GQA for visual reasoning (Hudson & Manning, 2019), MMBench for multimodal understanding (Liu et al., 2024b), MME for perception and cognition (Fu et al., 2023), POPE for object hallucination (Li et al., 2023), TextVQA (validation split) and OCRBench for text-based questions (Singh et al., 2019; Liu et al., 2024c), and ScienceQA-IMG for science questions (Lu et al., 2022). OCRBench serves development; the other six contribute to the mean outside development. VisionZip (Yang et al., 2025a), DivPrune (Alvar et al., 2025), CDPruner (Zhang et al., 2025), and MMTok (Dong et al., 2026) provide the references.

The additional QwenVL comparison includes InfoVQA (Mathew et al., 2022), RealWorldQA (xAI, 2024), AI2D (Kembhavi et al., 2016), ChartQA, DocVQA, SEED-Bench-2-Plus (Li et al., 2024), and MMStar (Chen et al., 2024), with MMTok as the reference method. Its five tasks outside development combine with the six above to give the 11-benchmark mean. Backbone transfer uses the same seven tasks as the first comparison and MMTok as the reference.

The discovery score U is fixed-resolution OCRBench accuracy reported as a percentage. Throughout this appendix, discovery accuracy is reported directly on the percentage scale; accuracy entries in tables omit the percentage sign. Score thresholds use the same percentage scale, while accuracy differences are expressed in percentage points. Component diagnostics at 5% token retention use dynamic resolution and average the three development-task scores expressed as percentages. For final evaluation, performance retention is $R_{a,b} = 100\, s_{a,b}/s_{\mathrm{full},b}$, using the full-token score for the same benchmark, backbone, and input protocol. The mean over a benchmark set $\mathcal{B}$ is $\overline{R}_a = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} R_{a,b}$. Each benchmark has equal weight, and rounding follows averaging. The Introduction reports the mean over all 14 benchmarks, combining the seven-benchmark reference comparison and the seven additional tasks before rounding. The 11-benchmark mean excludes OCRBench, ChartQA, and DocVQA. The seven-benchmark means in the individual result tables use the tasks listed in those tables.

Table 6: Reported evaluation settings. Each column lists the settings for its evaluation stage. The token budget K is defined in Section 3; OCRBench uses `echo840/OCRBench`. Pixel bounds refer to Qwen; SAIL-VL-2B and InternVL3.5-4B use native dynamic tiling. Component diagnostics use the QwenVL final-evaluation protocol at 5% token retention. Inference timing uses the fixed-resolution protocol in Appendix D.4.

Setting	**Discovery**	**Final evaluation**
Downstream model	Frozen Qwen2.5-VL-7B	Qwen2.5-VL-7B-Instruct; transfer to SAIL-VL-2B and InternVL3.5-4B
Evaluation data	OCRBench, 1000 samples	14 benchmarks; 7 benchmarks per transfer backbone
Score reporting	OCRBench accuracy $U(a)$ on a percentage scale	Native benchmark scores and mean performance retention (percentage)
Token-retention ratio	$\rho = 0.05$	$\rho \in \{0.20, 0.10, 0.05\}$ in the token-budget comparison
QwenVL pixel bounds	$\text{MinPix} = \text{MaxPix} = 2048 \times 28 \times 28$	$\text{MinPix} = 256 \times 28 \times 28$; $\text{MaxPix} = 2048 \times 28 \times 28$
Batch size	16	1
Selector used	Generated candidate, including its recorded initialization	Final ORGS, Leverage, and FPS
Search budget	2011 primary-run completion records; 150 per matched search comparison	Evaluations of the selected algorithms
Numerical computation	Scoring, similarity, and matrix operations in `float32`	Scoring, similarity, and matrix operations in `float32`
Output constraint	Exactly K indices	Exactly K indices
Downstream decoding	Greedy; temperature 0; single beam; at most 128 new tokens	Greedy; temperature 0; single beam; at most 128 new tokens
Framework seeds	`random` 0, NumPy 1234, PyTorch 1234	`random` 0, NumPy 1234, PyTorch 1234
Evaluation timeout	3600 s per attempt, with 2 retry attempts	harness defaults
Software	Python 3.12; in-tree `lmms-eval` 0.7.1	Python 3.12; in-tree `lmms-eval` 0.7.1
Hardware	Multiple NVIDIA A800 GPUs, each with 80 GB memory	Single 80 GB GPU (NVIDIA A800)

A.4 Reported Evaluation Settings

Table 6 separates discovery from final evaluation; component diagnostics follow the final-evaluation input protocol.

Discovery-time probes use reduced feature dimensions to check execution before the full downstream evaluation.

The proposal and program generators both use `deepseek-v4-pro`. Each request has a 600 s timeout and up to 3 API retries. Discovery uses multiple NVIDIA A800 80 GB GPUs, allowing up to three candidate evaluations to run concurrently across the available devices. Before starting, each evaluation requires at least 30 GB of free memory on its assigned GPU. Appendix E.9 gives generation temperatures, and Appendix E.3 gives search constants.

A.5 Matched-Search Comparison Protocol

The matched-search protocol governs the evolutionary-baseline and framework-ablation comparisons in Tables 1 and 2. Ours (direct mutation) replaces mathematical tool selection, retrieval, and explicit formulation–solution derivation with direct proposal generation from the current best program and recent outcomes. Ours w/o retrieval retains formulation and derivation using model-

internal knowledge. Ours (full) uses the complete discovery workflow. Each configuration has one 150-attempt run and shares the task interface, initial MMTok (Dong et al., 2026) coverage selector, generator, evaluator, and token budget. All configurations can revise both objectives and solution procedures. The comparison matches the number of search attempts; total computation cost varies with the generation, repair, and evaluation work performed within those attempts. The full configuration uses the first 150 attempts of the primary run.

Valid evaluations in the matched runs. Using the counting rules defined above, the FunSearch-style (Romera-Paredes et al., 2024) run produces 108 programs of which 103 receive valid scores, the EoH-style (Liu et al., 2024a) run produces 49 programs of which 41 receive valid scores, and the ShinkaEvolve-style (Lange et al., 2026) run produces 87 programs of which 84 receive valid scores. One retrieval-ablation attempt records a numeric score together with a failure flag; the strict definition excludes it, giving 139 valid evaluations for that configuration. Candidates within a run share search history and are statistically dependent. The counts summarize the outcomes of one recorded run per configuration.

All eleven candidates with $U \geq 40\%$ return exactly $K = 7$ unique, in-range indices on all five cached trials under each question condition, supporting the high-score and algorithm-diversity comparisons. Table 7 reports outcomes for all evaluated candidates; Appendix E.12 verifies score-to-code attribution.

Table 7: Evaluation and probe counts for the matched 150-attempt runs. Probe outcomes and output clusters are given under the two input conditions of the probe suite (empty question / fixed question). Probe execution failures leave the recorded discovery scores unchanged and exclude the candidate from clustering under the failed input condition.

Search configuration	Attempts	Valid evaluations	Probe ok (empty / question)	Probe failed (empty / question)	Output clusters (empty / question)
EoH-style	150	41	41 / 41	0 / 0	36 / 41
FunSearch-style	150	103	103 / 100	0 / 3	103 / 99
ShinkaEvolve-style	150	84	84 / 84	0 / 0	84 / 84
Ours (direct mutation)	150	109	109 / 107	0 / 2	107 / 105
Ours w/o retrieval	150	139	139 / 139	0 / 0	137 / 139
Ours (full)	150	119	110 / 118	9 / 1	109 / 117

Behavioral probe protocol. Output clusters compare executed selections. Each valid candidate's selection code runs on a fixed probe suite: synthetic inputs of 144 tokens on a 12×12 grid with a budget of 7 tokens, over five trials. Four trials use fixed-seed random features, and two of these replace the features with clustered low-rank inputs built from four base directions plus small noise; the fifth trial removes the grid information. All trial inputs are generated in advance, and each candidate receives an independent copy. The suite runs under two input conditions: an *empty-question* condition without question text or embeddings, and a *fixed-question* condition that supplies one fixed non-empty question with fixed-seed embeddings, identical for every candidate. Two candidates belong to the same cluster when their selected token sets are identical on all five trials within one condition. Candidates execute in isolated subprocesses with a 60-second limit; a candidate whose code raises an exception under a condition is recorded as a probe execution failure for that condition, and its discovery score remains valid.

Changing scoring weights changes selections, so most candidates occupy their own cluster and cluster counts in Table 7 are high for every configuration. The fixed-question condition raises the EoH-style cluster count from 36 to 41 and reduces the full framework's probe failures from 9 to 1; under this condition, the FunSearch-style and direct-mutation runs record 3 and 2 probe failures, respectively. All eleven high-scoring candidates pass both conditions with identical clustering.

With the comparison units and score conventions fixed, we next report discovery outcomes.

B Algorithm Discovery Results

Appendix B.1 compares high-score yield and conditional hit rates under the matched 150-attempt budget. Appendix B.2 classifies high-scoring candidates by their selection procedures and probe outputs. Appendix B.3 examines the contribution of knowledge retrieval to discovery quality and algorithm diversity. Appendix B.4 reports the joint effect of removing history-based guidance and randomizing strategy selection. Appendix B.5 analyzes the outcomes and selection procedures of the direct-mutation ablation. Appendix B.6 examines the search statistics and leading programs of the evolutionary controls. Appendix B.7 presents representative solution procedures from the extended primary and companion runs. Appendix B.8 reports toolbox expansion and discoveries associated with added tools. Appendix B.9 summarizes the outcomes of parent-based refinement during extended exploration. Appendix B.10 describes search-label coverage, tool reuse, and the progression and distribution of discovery scores.

B.1 Discovery Quality under Matched Budgets

High-score yield and conditional hit rate. Let $T = 150$ be the attempt budget and τ the high-score threshold. The high-score yield is defined as follows:

$$\text{High-score yield}_\tau = \frac{\#\{\text{valid } a : U(a) \geq \tau\}}{150}. \tag{2}$$

The corresponding conditional hit rate is defined as follows:

$$\text{Conditional hit rate}_\tau = \frac{\#\{\text{valid } a : U(a) \geq \tau\}}{\#\{\text{valid } a\}}. \tag{3}$$

Yield measures the output of the whole discovery process under the fixed budget; the conditional hit rate measures quality among the successful evaluations. The main text reports yield at $\tau = 40\%$, which requires improving over the 32% initial selector by at least eight percentage points. The threshold is chosen after the runs; Table 8 reports both measures at $\tau \in \{38\%, 40\%, 45\%\}$.

Table 8: High-score statistics of the matched 150-attempt runs under three thresholds. Yield divides the high-score count by the fixed 150-attempt budget; the conditional hit rate divides it by the number of valid evaluations. *Attempts to* $U \geq 40\%$ gives the first attempt reaching the shared threshold; *First attempt at best* gives the first attempt at each configuration's own best score.

Search configuration	Yield τ=38%	Yield τ=40%	Yield τ=45%	Hit rate τ=40%	Attempts to $U \geq 40\%$	First attempt at best
EoH-style	3 (2.0%)	0 (0.0%)	0 (0.0%)	0.0%	not reached	39
FunSearch-style	8 (5.3%)	0 (0.0%)	0 (0.0%)	0.0%	not reached	42
ShinkaEvolve-style	8 (5.3%)	2 (1.3%)	0 (0.0%)	2.4%	43	61
Ours (direct mutation)	12 (8.0%)	0 (0.0%)	0 (0.0%)	0.0%	not reached	22
Ours w/o retrieval	6 (4.0%)	3 (2.0%)	0 (0.0%)	2.2%	38	38
Ours (full)	12 (8.0%)	6 (4.0%)	3 (2.0%)	5.0%	36	44

At $\tau = 38\%$, direct mutation matches the complete framework in yield, with 12 candidates each. The separation appears at higher thresholds: the full framework produces six candidates at 40%, compared with three without retrieval and two for ShinkaEvolve-style (Lange et al., 2026) evolution; only the complete framework reaches 45%. The metric *attempts to reach* $U \geq 40\%$ gives every configuration the same target; configurations without such a candidate are marked as not reached. This statistic complements the per-configuration first-attempt-at-best column, since reaching a configuration's own best compares different targets across rows.

B.2 Diversity of High-Scoring Algorithms

Table 9 lists all eleven candidates with $U \geq 40\%$ over the first 150 attempts of each matched run. We inspect the evaluated code and group candidates by their core selection score and state-update

procedure. Preprocessing, feature or coverage-row construction, numerical constants, tie handling, and equivalent bookkeeping define variants within a cluster. Distinct selection criteria or an additional search stage define separate clusters. This procedure groups the implemented algorithms at a common level of detail across all configurations.

The full framework produces five algorithm clusters. F1 selects the largest residual norm and expands an orthogonal basis (attempts 44 and 67). F2 computes SVD leverage scores and selects the top-K once (50). F3 combines kernel determinantal initialization with particle-sampling refinement (36). F4 uses greedy kernel power scores with incremental Cholesky updates (92). F5 selects by residual Gram row energy and expands an orthogonal basis (141). F1 and F5 use different selection scores; F3 includes a sampling stage that changes the greedy kernel subset.

The retrieval ablation produces three clusters: normalized residual Gram-energy selection with deflation (N1, attempt 38), directional-extrema selection with iterative direction replacement (N2, 74), and two-pass innovation scoring followed by top-K selection (N3, 124). ShinkaEvolve-style (Lange et al., 2026) evolution produces one cluster (S1, 43 and 61). Both S1 candidates maximize summed coverage gain and update coverage by an elementwise maximum; their coverage rows and weights differ. EoH (Liu et al., 2024a), FunSearch (Romera-Paredes et al., 2024), and direct mutation have zero candidates reaching the shared threshold in this window. The resulting algorithm-cluster counts are $5/3/1$ for the full framework, retrieval ablation, and ShinkaEvolve-style baseline.

The fixed probes provide a complementary comparison of selected token sets. They give $5/3/2$ high-scoring output clusters for these same configurations. The two S1 candidates select different token sets while sharing the greedy-coverage procedure. Full-run attempts 44 and 67 share both an algorithm cluster and an output fingerprint. Both are counted as evaluated candidates in the matched comparison. Table 7 gives output-cluster statistics across all valid candidates.

Table 9: All candidates at or above $\tau = 40\%$ in the matched 150-attempt runs. Selection procedures are confirmed by reading the evaluated code versions. Cluster IDs group core selection scores and state-update procedures. Variants of preprocessing, coverage rows, and weights share a cluster. ShinkaEvolve attempts 43 and 61 both belong to S1; their probe outputs form two output clusters.

Configuration	Attempt	Score (%)	Code-confirmed selection procedure	Algorithm cluster
Ours (full)	44	49.30	greedy maximum residual norm with Gram–Schmidt orthogonalization; incremental norm updates on raw features	F1
Ours (full)	67	49.10	same procedure as attempt 44 with different bookkeeping (identical fingerprint)	F1
Ours (full)	50	43.70	random orthogonal projection, SVD leverage scores, one-shot top-K	F2
Ours (full)	36	44.30	RBF-kernel determinantal greedy log-determinant initialization with sampling-based refinement	F3
Ours (full)	92	41.40	RBF-kernel greedy power function with incremental Cholesky updates	F4
Ours (full)	141	48.30	residual Gram row-energy greedy selection with spatial-smoothing preprocessing	F5
Ours w/o retrieval	38	44.90	centered Gram energy scores with deflation and reorthogonalization	N1
Ours w/o retrieval	74	41.80	extreme-projection selection along iterative refinement directions	N2
Ours w/o retrieval	124	40.90	Kalman-style two-pass innovation scoring over the spatial grid	N3
ShinkaEvolve-style	43	40.10	question-weighted greedy coverage with off-diagonal visual similarity and optional local context	S1
ShinkaEvolve-style	61	40.20	greedy coverage combining question-conditioned, visual, and spatial-distance rows	S1

Mechanism classification. Behavior clusters compare outputs; the keyword tally compares code features. We assign each valid candidate a primary class by applying uniform keyword rules to its generated code body, with the fixed priority *orthogonal residual update* > *sequential inverse-matrix*

update > *kernel approximation* > *coverage and representative-point selection* > *other*; composite candidates take their highest-priority class. Keyword matches can fall outside the final selection path, so Table 10 reports the distribution of code features among valid candidates. Under this reading, the controls concentrate on the coverage and representative-point class, while the two formulation-centric configurations produce larger shares of valid candidates with residual-update, inverse-matrix, and kernel code features (27 of 119 valid candidates for the complete framework, 19 of 139 for the retrieval ablation). The high-scoring candidates' mechanism interpretation comes from code review in Table 9, which confirms the direct role of residual updates, one-shot leverage scoring, and two kernel-based selection procedures in their final token selection. The per-candidate assignments accompany the supplementary material. The search-label entropy in Table 11 measures the distribution of declared search labels; the analyses here describe probe outputs and generated code features.

Table 10: Primary-mechanism distribution of the valid candidates in each matched run, assigned by uniform keyword rules over the generated code body with fixed priority. Counts sum to each configuration's valid evaluations in Table 7.

Search configuration	Coverage / representative	Orthogonal residual	Sequential inverse-matrix	Kernel approximation	Other
EoH-style	23	1	3	0	14
FunSearch-style	59	3	5	1	35
ShinkaEvolve-style	65	0	0	0	19
Ours (direct mutation)	57	1	7	0	44
Ours w/o retrieval	23	7	4	8	97
Ours (full)	28	14	8	5	64

B.3 Framework Ablation: Knowledge Retrieval

We test the contribution of retrieved knowledge within the complete framework. The ablation uses the same 150-attempt budget, initial selector, generator, and evaluator. The retrieval ablation, labeled Ours w/o retrieval, uses model-internal knowledge. It retains the required mathematical objective, input mapping, and derivation before code generation. Table 11 reports the comparison. Its full-configuration column uses the first 150 attempts of the primary run.

Table 11: Retrieval ablation under the matched 150-attempt budget. The full configuration is the primary run truncated at attempt 150. Improvement events count strict increases of the running best score. Search-label entropy is computed over the distribution of declared search labels of all attempts. Component occurrences are code-reviewed and include unscored attempts; scored uses require valid evaluation. Numeric-literal counts use the top ten valid programs and the common rule below.

	Ours (full)	Ours w/o retrieval
Best score (%; first attempt at best)	**49.30** (44)	44.90 (38)
Candidates at $U \geq 40\%$	6	3
High-scoring algorithm types	5	3
Best-score improvement events	7	5
Search-label entropy (nats)	2.05	2.17
FPS component first appears (implementations)	attempt 8 (7)	attempt 120 (2)
First scored FPS implementation (scored uses)	attempt 25 (3)	attempt 120 (1)
Coordinate scaling first appears	attempt 46	attempt 28
Sherman–Morrison update first appears	attempt 113	attempt 6
Numeric literals, top-10 programs (median)	4	10

Algorithms from model-internal knowledge. The retrieval ablation retains mathematical formulation and derivation and generates varied procedures from model-internal knowledge. Its three

candidates at $U \geq 40\%$ implement centered Gram-energy selection with deflation and reorthogonalization (attempt 38, 44.9%), extreme-projection selection along refined directions (attempt 74, 41.8%), and Kalman-style two-pass innovation scoring on the spatial grid (attempt 124, 40.9%). They form three output clusters on the fixed probes. An inverse-matrix leverage update also appears early, at attempt 6. Adding retrieval raises the best score from 44.9% to 49.3% and doubles the number of candidates reaching $U \geq 40\%$ from three to six in these runs. The component records below distinguish these outcome differences from when individual mathematical operations first appear.

Code review identifies FPS in full-run attempts $8, 25, 55, 82, 83, 97, 99$ and no-retrieval attempts $120, 143$. An occurrence requires farthest-from-nearest selection with a nearest-distance update, including uses in initialization, crossover, and conditional fill. Static isolation scores and facility-location sum-gain rules are classified separately. Three full-run occurrences and one no-retrieval occurrence receive valid scores. These counts measure code component presence; conditional execution depends on benchmark inputs. The first full-run occurrence is crossover at attempt 8; this attempt fails, and attempt 25 is the first scored occurrence. Code inspection classifies attempt 7 as one-shot nearest-neighbor isolation ranking, correcting the earlier "farthest" keyword assignment.

Coordinate scaling means dividing feature coordinates by empirical standard deviation across tokens, including projected coordinates. It first occurs at $46/28$ (full/no retrieval). Full-run attempt 2 scores tokens by variance; coordinate-wise variance scaling first appears at attempt 46. Sherman–Morrison updates also occur earlier without retrieval (6 versus 113). The ablation's higher search-label entropy measures diversity in declared search labels. Component-level records show earlier FPS implementations with retrieval and earlier scaling and inverse updates in the ablation.

Implementation size. We rank valid candidates from the first 150 attempts by descending score, breaking ties by attempt ID, and inspect the ten highest-scoring programs per configuration. We count numeric literal occurrences in the marked selection block using the Python abstract syntax tree, excluding booleans and magnitudes 0, 1, and 2; repeated occurrences count separately. The median is 4 with retrieval and 10 without retrieval. This descriptive metric counts literal occurrences used for weights, stabilizers, and loop limits. Each occurrence contributes to the count regardless of its role or whether it can be tuned independently. The common rule also underlies Table 15.

First-115 comparison: diversity of high-performing solutions. To supplement the matched 150-attempt retrieval comparison in Table 2, we also examine attempt IDs 1–115 under the same scoring and clustering protocol. This window contains 90 valid evaluations for the full framework and 105 for the internal-knowledge ablation. The primary state contains 116 completion records because ID 41 has two failed records; it still covers 115 distinct attempt IDs. We reuse the evaluated code versions and fixed probes defined in Appendix A.5.

Table 12: Retrieval comparison restricted to the first 115 attempt IDs of each run. High-scoring means $U \geq 40\%$; yield divides candidate count by 115. Procedure counts come from reading the high-scoring selection code; output clusters come from cached fixed probes.

First 115 attempt IDs	With retrieval	Internal knowledge only
Valid evaluations	90	105
Best score (%)	49.30	44.90
Candidates at $U \geq 40\%$	5	2
High-score yield	4.35%	1.74%
Code-reviewed high-scoring procedures	4	2
High-scoring output clusters (empty / question)	4/4	2/2
All output clusters (empty / question)	84/89	105/105
Probe failures (empty / question)	5/0	0/0

The full framework's high-scoring programs are attempts $36, 44, 50, 67, 92$; attempts 44 and 67 implement the same residual-norm procedure and share one output fingerprint. Its four procedures cover residual-norm selection, one-shot SVD leverage scoring, and two kernel-based procedures: determinantal selection with sampling refinement and greedy power-function selection with incremental Cholesky updates. The two kernel-based procedures share a mathematical foundation and use different selection and update rules, so each contributes one distinct procedure to this compari-

son. The internal-knowledge run's two high-scoring programs are attempts 38 and 74, implementing Gram-energy selection with deflation and extreme-projection selection along refined directions. Its Kalman-style candidate at attempt 124 falls outside this window.

Retrieval therefore broadens the set of high-performing solutions in the matched first-115 window: four distinct procedures reach the shared threshold, compared with two using internal knowledge alone. The successful procedures span residual selection, leverage scoring, and kernel-based selection. This advantage also appears at thresholds 38% and 45%, where the fixed-question high-scoring cluster counts are 10 versus 5 and 1 versus 0, respectively. Across all valid candidates, the internal-knowledge run produces more output clusters, as shown in Table 12. The two runs have similar search-label entropy (1.99 versus 2.00 nats, using the latest completion per attempt ID), consistent with broad use of the recorded labels in both runs. Code review and output clustering connect the benefit of retrieval to the range of solutions that achieve high scores, matching the discovery-quality comparison in Section 6.2.

B.4 Framework Ablation: Guided Strategy Selection

The ablation, labeled w/o guided strategy selection, removes history-feedback text and randomizes strategy selection; Appendix E.11 specifies the switches and scheduling behavior.

Recorded outcomes. The saved state contains exactly one completion record for each attempt ID 1–150. Of these, 142 are successful evaluations with finite accuracy, and all 142 scores match their saved execution-log results. The best score is 39.2% at attempt 145. The full framework's first 150 attempts yield 119 valid evaluations and reach 49.3% at attempt 44. At thresholds $U \geq 38\%$, 40%, and 45%, the ablation yields 4, 0, and 0 candidates, respectively, compared with 12, 6, and 3 for the full framework. The corresponding yields per attempt are 2.7%, 0%, and 0%, versus 8.0%, 4.0%, and 2.0%. Both configurations produce 21 candidates at or above the initial selector's score of 32%, and the additional high-scoring candidates demonstrate the full framework's stronger upper-end discovery quality.

Evidence for feedback-guided discovery. The higher best accuracy and greater high-score yield support the use of history feedback to guide exploration toward promising directions. Each configuration has one recorded run; Appendix E.8 reports validation and probe analyses. The accompanying audit script reproduces the counts and verifies the ablation's scores against the saved evaluation logs.

B.5 Framework Ablation: Direct Mutation

The direct-mutation ablation, labeled Ours (direct mutation), starts from the shared MMTok (Dong et al., 2026) coverage selector. Its prompt receives the current best program and recent outcomes and requests an objective or scoring rule together with an implementation. This replaces the mathematical tool selection, retrieval, and explicit formulation–solution derivation stages of the full framework. It produces evaluated candidates in 109 of 150 attempts. Repeated-idea screening directs later attempts toward distinct proposals. These statistics use the corrected protocol of Section 6.2. An earlier run evaluates 124 candidates and reaches 39.6% before protocol corrections. The matched comparison uses the corrected run specified in Appendix A.5.

Figure 9 gives the generation prompt. The following analysis concerns the evaluated programs.

Solution procedures. Table 13 classifies all 109 evaluated programs. They use question-dependent scores followed by greedy selection. Main forms include maximal marginal relevance (MMR), which balances relevance and novelty; determinantal point processes (DPP), which favor diverse subsets; and coverage objectives. Coverage and facility-location variants account for 38.5% of candidates. Facility location (FL) rewards similarity to selected representatives. The best score is 39.9%, and 12 candidates reach at least 38%. The run mainly refines this set of solution procedures.

Highest-scoring candidates. Table 14 lists the five highest-scoring distinct candidates. Their programs accompany the supplementary material. Each uses question-weighted coverage, which rewards representation of important tokens. They vary coverage weights and similarity feature spaces. Weights may use text similarity, visual saliency, spatial information, or coverage of question tokens. Similarities combine model-input embeddings, selection features, or grid positions.

Table 13: Solution-procedure distribution of the evaluated candidates in the direct-mutation ablation. All candidates use question-dependent scoring rules; columns report the greedy selection template.

Evaluated	MMR	Question-weighted DPP	Coverage / FL	Other	$\geq$38%
109	38 (34.9%)	24 (22.0%)	42 (38.5%)	5 (4.6%)	12

Table 14: Top-5 distinct candidates of the direct-mutation ablation (OCRBench, ρ=0.05). All are question-weighted coverage/facility-location variants.

Candidate	Acc. (%)	Distinguishing component
Q-ViCo (attempt 22)	39.90	saliency-weighted facility location, text coverage
DS-QC (40)	39.90	row-normalized dual-space coverage graph
QW-DSFL (143)	39.50	sharpened dual-space kernels, spatial prior
QAMC (85)	39.40	query-critical anchors, fused coverage
QA-MFL (123)	39.30	dual visual coverage + question-text coverage

A worked instance. Q-ViCo reaches 39.9% at attempt 22. It illustrates the control's main selection pattern. A question-relevance score combines the maximum token–text cosine similarity and a top-k mean, with $k = \min(8, m)$. Here, m is the number of question tokens. A visual-saliency score combines global centrality with local 3×3 contrast when a patch grid is available. Both scores are converted to ranks. The coverage weights are $w = 0.35 + 0.45\, q_{\text{rank}} + 0.20\, s_{\text{rank}}$.

The similarity matrix combines selection-feature and model-input cosine similarities with weights 0.55/0.45. At each step, the procedure maximizes $\sum_j w_j \max(A_{ij} - \text{cov}_j, 0)$. Here, A_{ij} is similarity and cov_j is the coverage already provided to token j. The selected token updates these coverage values. Additional bonuses reward question relevance, question-text coverage, and spatial spread, with weights 0.10/0.10/0.05. The other leading candidates vary these weights and combined feature spaces. Appendix F.4 gives sampled attempts.

Direct mutation improves the initial 32% score to 39.9% in 150 attempts, while the complete framework reaches 49.3%. Continuing the primary run produces the 52.1% coverage candidate at attempt 441.

B.6 Evolutionary Search Outcomes

We examine the evaluated programs behind the evolutionary controls' matched-budget results; their search configurations are specified in Appendix E.10.

B.6.1 Search Statistics of the Evolutionary Controls

Under the same 150-attempt budget, the complete framework reaches 49.3%. ShinkaEvolve-style (Lange et al., 2026) evolution reaches 40.2%, direct mutation and FunSearch-style (Romera-Paredes et al., 2024) evolution each reach 39.9%, and EoH-style (Liu et al., 2024a) evolution reaches 39.2%. ShinkaEvolve, FunSearch, and EoH first reach their best scores at attempts 61, 42, and 39, respectively. The following analysis connects these outcomes to the generated programs.

ShinkaEvolve outcomes. Of the 150 attempts, 63 fail during proposal generation, while the remaining 87 produce candidate programs. Among these 87 programs, 84 receive valid OCRBench scores and three fail during implementation or execution. Its best score is 40.2% at attempt 61, and two candidates reach $U \geq 40\%$: attempts 43 (40.1%) and 61 (40.2%). Thus, high-score yield is $2/150 = 1.3\%$ and the conditional hit rate is $2/84 = 2.4\%$.

Both high-scoring candidates use greedy coverage: each step chooses the token with the largest sum of improvements over the coverage already provided by selected tokens. Attempt 43 combines question relevance, spatial context when grid metadata is available, and visual similarities with self-similarity removed. Attempt 61 combines rows, similarities from selection features and model-input embeddings, and a spatial-distance term; its local-context rows also depend on grid availability.

Discovery supplies no grid metadata, as specified in Appendix A.2. The two candidates share the greedy-coverage algorithm cluster S1 and form two output clusters under each fixed-probe input condition. Their results extend the successful coverage-based baselines, while the full framework reaches 49.3% through a broader set of high-scoring procedures (Table 9).

B.6.2 Highest-Scoring Selectors of the Evolutionary Controls

Leading FunSearch-style candidate (attempt 42, score 39.9%). The leading candidate is *Query-Weighted Multi-View Coverage* (QW-MVC). It selects tokens to improve coverage of question content, visual features, and spatial positions:

$$\max_{S,\,|S|=K} \underbrace{\textstyle\sum_t w_t \max_{i\in S} P_{ti}}_{\text{text groundedness}} + \underbrace{\textstyle\sum_j w_j \max_{i\in S} K^{\text{sem}}_{ji}}_{\text{semantic diversity}} + \underbrace{\textstyle\sum_u w_u \max_{i\in S} K^{\text{spa}}_{ui}}_{\text{spatial coverage}}. \tag{4}$$

The matrix P applies a temperature-0.05 softmax to text–vision cosine similarities. Its text weights reduce the influence of generic question tokens. The matrix K^{sem} combines selection-feature and model-input similarities. Its rows use question-relevance weights with relative weight 0.50. The matrix K^{spa} uses a row-normalized Gaussian similarity over grid positions. It has an adaptive distance scale and relative weight 0.20. Selection repeatedly adds the token with the largest coverage gain until K tokens are retained.

Leading EoH-style candidate (attempt 39, score 39.2%). The leading candidate is *Question-Weighted Multi-Modal K-Medoids Facility Location* (QW-KMFL). It chooses representatives that minimize a question-weighted distance to the token set. Its combined distance is $D = 0.4\,(1 - \cos_{\text{proj}}) + 0.6\,(1 - \cos_{\text{CLIP}})$. The weighted selection objective takes the following form:

$$\min_{S,\,|S|=K} \textstyle\sum_i w_i \min_{j\in S} D(i,j), \qquad w_i = 0.7\, r_i + 0.3/N. \tag{5}$$

The relevance value r_i combines text-to-vision attention across question tokens. Text aggregation is weighted by saliency. The uniform term distributes some weight across all visual regions. Selection starts from the most relevant token and repeatedly adds the token giving the largest objective decrease. It then performs up to 5 refinement rounds. Each round assigns tokens to their nearest representative and recomputes the weighted medoid, minimizing weighted within-cluster distance.

Shared pattern. The best evolutionary candidates use question-dependent weights with greedy coverage or reconstruction across combined feature spaces. They mainly vary the weight construction and the similarity combination used by the leading direct-mutation candidates. Our complete framework also explores residual updates, sequential matrix updates, and scaled-distance coverage. These program-level differences accompany the reported discovery-score differences.

B.6.3 Program-Level Analysis of the Evolutionary Controls

We inspect the programs generated within the 150-attempt comparison. The analysis identifies changes to objectives, update rules, and numerical settings.

Program ancestry and selection weights. All 108 FunSearch-style (Romera-Paredes et al., 2024) programs descend from the seed. Most retain its greedy-coverage structure, and several share identical weighting expressions. Others introduce procedures such as log-determinant selection at attempt 94. The leading programs have seven declared search labels. Their selection uses weighted coverage with different weights.

Across the EoH-style (Liu et al., 2024a) run, nine distinct programs enter the population. Their selection rules form three groups: five weighted coverage, three log-determinant selection, and one weighted k-medoids program. The parameter-change operator also produces closely related variants. One changes seven constants and scores 35.2%, compared with its parent's 35.9%. Table 15 summarizes the records.

From the same seed and within 150 attempts, the complete framework proposes additional computations: pivoted matrix factorization, orthogonal residual pursuit, inducing-point approximations, and radial-basis kernel criteria. The records show each configuration explores different procedures.

Table 15: Lineage and best-score progression of the FunSearch-style and EoH-style controls over the matched 150-attempt budget. Valid scores count programs whose evaluation completed with a numeric score without an execution-failure flag (Appendix A.5). Procedure groups capture shared selection logic; fingerprint clusters capture identical selections on every synthetic probe. Numeric-literal counts follow Appendix B.3; the full framework has a median of 4 and reaches 49.3% at attempt 44.

	FunSearch-style	EoH-style
Programs produced, valid scores (of 150 attempts)	108, 103	49 (9 distinct in population), 41
Procedure groups among top candidates	1 macro-cluster (100% seed descendants)	3 clusters; largest 5/9
Best score (%; attempt)	39.90 (42)	39.20 (39)
Numeric literals in top-10 programs (median)	22.5	9.5

Procedures within the same objective family. The FunSearch-style and EoH-style controls generate log-determinant selectors with incremental inverse updates. The FunSearch-style program at attempt 94 scores 35.1%. The EoH-style program at attempt 122 scores 35.9%. The longer primary run reaches 49.2% with an information-based candidate at attempt 1198. These examples show that related objectives can yield differently scoring procedures. The longer run uses a larger search budget. Table 15 reports matched-attempt results.

Numerical settings. Under the same top-ten selection and literal-counting rule, FunSearch-style and EoH-style programs have medians of 22.5 and 9.5, compared with 4 for the complete framework. This comparison describes the numerical expressions used by the generated programs; the discovery scores and component ablations separately establish their effectiveness.

Selection rules and token overlap. We group programs by their selection steps and ancestry. Fixed synthetic probes then compare their selected token sets. Leading evolutionary candidates and leading primary-run candidates both have mean pairwise Jaccard similarity 0.28. This similarity is the intersection size divided by the union size. Related procedures can therefore select different subsets. The comparison describes the tested programs under the shared interface.

B.7 Extended Exploration: Representative Solution Procedures

The remaining discovery results concern extended exploration and are separate from the matched 150-attempt comparison. Tables 16 and 17 list representative candidates from the primary and companion runs, with attempt IDs local to each run. Principal component analysis (PCA), used by one listed candidate, projects features onto high-variance directions.

Table 16: Top-scoring valid solution procedures discovered on OCRBench at ρ=0.05. "Attempt" identifies the first attempt reaching the reported score within a run. "Run" identifies the primary run or a companion run (*v2*, *full*). Attempt numbers are local to each run.

Solution procedure	Objective family	Acc. (%)	Attempt	Run
Variance scaling + medoid initialization + FPS	coverage	**52.10**	441	primary
Genetic search for max–min dispersion	metric dispersion	52.00	597	companion (*v2*)
Greedy orthogonal residual pursuit	reconstruction	50.70	1582	primary
Appearance×spatial product FPS	metric dispersion	50.60	978	companion (*full*)
PCA subspace + per-cluster medoid	coverage	50.50	927	companion (*v2*)

Other solution procedures. Table 17 illustrates additional selection procedures. All use visual inputs only. The highest-scoring candidate uses variance-scaled distances and farthest-point selection for coverage. Its score is 52.1%. A genetic subset search scores 52%. It maximizes the smallest

pairwise distance among selected vectors using unwhitened features. Its settings are population 50, tournament selection, single-point crossover, 10% mutation, duplicate repair, top-2 retention, and 30 generations. Their scores differ by 0.1 percentage point. They are complete algorithms with different distances, objectives, and solution procedures.

The remaining candidates combine projection, coverage, reconstruction, and distance-based selection. They vary feature transformations and token-budget allocation. Supplied formulations guide the properties to preserve; each proposal specifies a concrete solution procedure.

Table 17: Representative visual-only selection procedures from extended exploration, ordered by recorded accuracy. Attempt numbers are local to each run. Each row describes a representative selection procedure through its computations. Behavioral output clusters are defined by selected token sets on the fixed probes.

Solution procedure	Acc. (%)	Attempt	Run	Objective family
Variance-scaled FPS, medoid initialization	**52.10**	441	primary	coverage
Genetic search over token subsets	52.00	597	*v2*	max–min dispersion
Appearance×spatial product-metric FPS	50.60	978	*full*	dispersion, product metric
PCA + ε-ball coverage + medoids	50.50	927	*v2*	coverage
Principal-direction projection pursuit	49.50	280	primary	subspace alignment
Streaming threshold coverage, learned τ	49.30	1472	primary	facility location
Multi-scale density-weighted coverage	49.30	1475	primary	weighted facility location
Risk-averse matching pursuit	49.20	1130	primary	weighted reconstruction
Whitened feature + coordinate FPS	49.20	891	*full*	dispersion, joint metric
Per-image quota FPS	48.70	1667	primary	per-image dispersion

Variations in residual and information-based procedures. Residual-based selection and sequential matrix updates also admit varied implementations. The residual procedure chooses the largest unrepresented component, adds its direction to the selected span, and removes it from remaining residuals. Before exhausting the span, the residual length also measures the added feature volume. Ridge-regularized determinant objectives produce related leverage scores and inverse-matrix updates. Appendix C.3 specifies them. Regularization changes scores and updates relative to unregularized residual removal.

The search varies these procedures. Input transformations include standardization, mean centering, robust rescaling, distribution-based transformations, dimensionality reduction, and random projection. Changes also add spatial coordinates or distance penalties. Scoring rules include residual length, logarithmic gain, trace reduction, and maximum-coordinate gain. Initialization uses a maximum-norm vector, medoid, or centroid. Per-image quotas and question-dependent terms also vary.

Component diagnostics separate the discovered core mechanisms from auxiliary choices. Residual orthogonalization gives a large gain, whereas standardization adds only a small gain once that update is present. The sequential matrix update gives a large gain in the information-based procedure. For FPS, variance scaling is essential, while alternative initializations have similar three-task means. Table 3 reports the perturbations with large performance drops; Appendix C.2 reports the smaller-effect variants.

B.8 Discoveries during Extended Exploration

The primary run started with 158 seed tools; the runtime log records this value. The complete runtime log records the first toolbox expansion at attempt 60, triggered by the periodic schedule, adding ten tools. Attempts 80 and 100 each add ten more. Within the first 150 attempts, eight successful expansion events each add ten tools after deduplication, yielding 80 expansion-added tools in total, including three events at attempt 120. Matching recorded tool names against the added-tool bank identifies attempt 211 as the first attempt selecting an added tool. During the first 150 attempts, tool selection draws exclusively from the seed toolbox, alongside the scheduled expansion of the available tool bank. The matched comparison therefore evaluates discovery using seed tools; selection from the expanded bank begins at attempt 211 in the extended run. The final expansion-

added tool bank contains 2,806 tools, excluding the 158 initialization seed tools. Of these additions, 30 were already present and retained at the restart where the preserved console begins; that console records 282 subsequent expansion events adding another 2,776 tools, so $30 + 2{,}776 = 2{,}806$. Each expansion requests ten tool names, but deduplication can reduce the number actually added. The 80 tools added within the first 150 attempts are a subset of the final 2,806 expansion-added tools.

Extended exploration with added tools produces 24 candidates with $U \geq 45\%$, a best discovery score of 52.1%, and nine behavioral clusters. The best candidate, at attempt 441, combines variance scaling, medoid initialization, and farthest-point selection. Appendix F.5.1 gives its proposal and generated selection block. These results characterize exploration using added tools in the extended run. The matched comparison characterizes the first 150 attempts, whose selected tools all come from the seed toolbox.

The high-performing procedures observed during extended exploration include subspace selection, residual-based selection, density-based selection, inverse-matrix updates, and coverage. Table 18 summarizes representative computations. It organizes the observed computations into a mechanism inventory. Individual candidate scores and discovery examples provide the corresponding performance results and implementation details, while behavioral cluster counts summarize output diversity. Behavioral clusters compare selected token sets on the fixed probes, whereas discovery scores measure OCRBench performance.

Table 18: Representative selection computations observed during extended exploration. The entries describe mechanisms observed in the extended run. The 24 high-scoring added-tool candidates are grouped into behavioral clusters using their selected token sets on the fixed probes.

Mechanism	Representative computation
Coverage	variance-scaled distances with medoid initialization and farthest-point selection
Residual norm	greedy maximum-residual selection with Gram–Schmidt orthogonalization
Residual correlation	OMP-style residual-correlation greedy selection
Subspace selection	SVD-subspace projection with greedy selection or column-pivoted Gram–Schmidt
Risk-averse residual selection	risk-weighted residual greedy selection
Density-based selection	density-peak selection
Information updates	random projection with Sherman–Morrison log-determinant updates
Hybrid selection	diffusion-graph and residual computations

B.9 Outcomes of Depth Exploration

Parent-based refinement yields an ORGS variant scoring 50.7% at attempts 1582 and 1599, while the earlier coverage selector remains best at 52.1%. Saved `depth_pipeline/pipeline_summary.json` files identify 185 distinct attempts between attempts 1472 and 1769, of which 143 have successful OCRBench evaluations verified against execution logs; 45 summaries additionally record diagnosis and mathematical recall. These extended-run results complement the breadth-oriented matched comparison.

B.10 Search Coverage and Score Distribution

The extended primary run contains 2011 completion records and 1562 valid scores. Appendix E.12.1 defines the counting units.

B.10.1 Distribution of Explored Approaches

Table 19 summarizes the single search label stored with each of the 2011 completion records. These labels support scheduling and run accounting, as defined in Appendix E.4. A candidate can combine tools from several mathematical areas: attempt 441, for example, carries the label `geometry` and combines a variance-scaled distance with farthest-point selection. All 15 named labels and the `general` category occur, each in at least five records. The 1562 scored candidates use 1393 recorded mathematical tools. The tool-reuse analysis below examines the procedures produced on repeated visits.

Table 19: Recorded search-label frequencies in the primary run (2011 completion records). Each record contributes once through its stored `PRIMARY_DIRECTION` label. A method may draw on multiple mathematical areas; these counts describe labels assigned during search.

Recorded label	Records	Recorded label	Records
geometry	666	svd	47
ranking	348	qr	46
general	151	eigenvalue	40
graph	146	cross_modal	40
projection	134	spatial_coherence	22
entropy	114	clustering	19
sampling	113	attention	19
gradient	101	multimodal_fusion	5

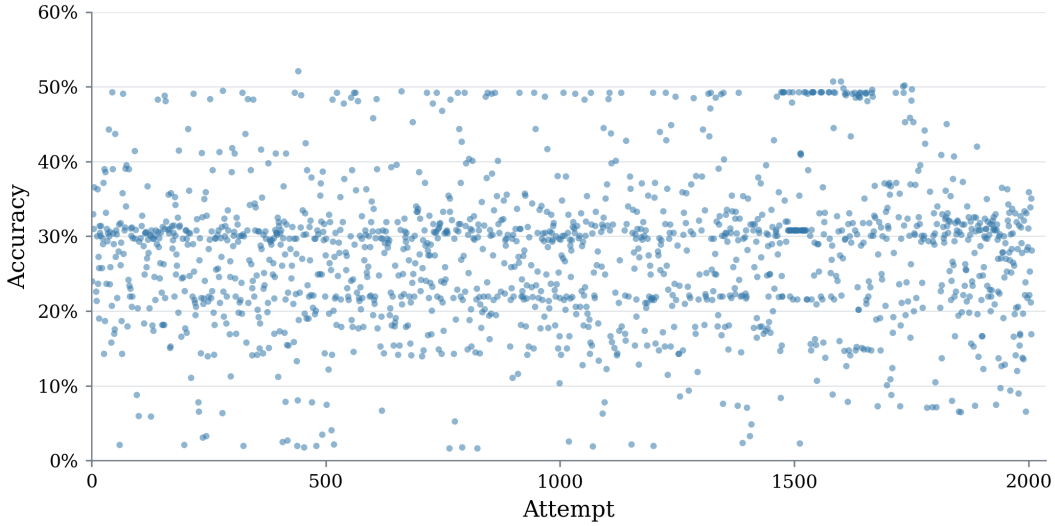


Figure 1: AI–Math–AI loop discovery accuracy (%) across search attempts. Each point is a valid numeric result; failed and unscored attempts are omitted. The horizontal axis retains attempt IDs (the recorded iteration IDs), including reused IDs, with sparse ticks.

The records show within-tool variation: twenty mathematical tools have at least three scored candidates, each tool with at least three distinct recorded operation sets. The median count is 4, and the maximum is 22. These sets describe the computations listed in the operation signatures. The result shows that repeated mathematical tool visits can yield different proposed procedures. Appendix B.2 separately compares executed selections.

B.10.2 Search Progress and Score Distribution

Figure 1 shows the 1562 valid scores across the primary run. Guided search finds a strong candidate early, reaching 49.3% by attempt 44. Continued exploration with toolbox expansion raises the best score to 52.1% at attempt 441, whose candidate uses an on-demand tool composition (Appendix B.8). This progression supplies a stronger coverage selector, while subsequent exploration broadens the pool of high-scoring procedures.

Continued exploration broadens the pool of strong candidates: 98 later results reach $U \geq 45\%$ after attempt 441. The added-tool analysis identifies nine high-scoring output clusters; Table 18 summarizes representative computations observed during extended exploration. Later examples include risk-averse residual selection at attempt 447, density-peak selection at 538, and inverse-matrix updates at 1320. Together, these results show that the framework finds strong candidates

early and continues to discover distinct high-performing selection behaviors after the best score plateaus.

Among the valid scores, the median is 29.1%, with lower and upper quartiles of 21.7% and 31.7%. Overall, 153 results reach 40%, and 111 reach 45%.

These discovery records provide the candidate pool for the component validation that follows. The purpose of this analysis is to test which parts of the discovered procedures carry their performance and which auxiliary choices can be removed or simplified without materially changing accuracy.

C Component Validation and Final Algorithm Specifications

Appendix C.1 traces candidate origins and distinguishes initialization inputs, components generated by the discovery loop, and post-discovery diagnostic variants. Appendix C.2 tests the discovered procedures by removing or replacing individual components on the development tasks, separating core mechanisms from auxiliary choices. Appendix C.3 defines the objectives, preprocessing, selection rules, and state updates of the selector implementations used for final evaluation. Appendix C.4 collects their numerical constants, initialization settings, and safeguards.

C.1 Candidate Origins and Component Provenance

We distinguish the initialization inputs, components generated during search, and diagnostic perturbations applied after discovery. The component study takes discovered candidates as references and evaluates controlled perturbations on the development tasks. It tests which generated computations contribute to performance, which replacements reduce accuracy, and which auxiliary implementation choices admit simplification with comparable accuracy. The provenance records connect each reference mechanism to its discovery attempt and identify the components examined in the subsequent comparisons.

C.1.1 Initialization Inputs

The iterative loop receives four types of input: the task interface, an 18×14 search-space coverage table, and the 23 formulations and 158 seed tools generated during initialization and stored in the task configuration. Of these formulations, 17 have no solution-procedure hint and 6 include a short hint. The tools use 15 named search labels; run accounting additionally includes a catch-all `general` label. These inputs guide mathematical exploration. Proposal generation uses these tools and retrieved mathematical knowledge to instantiate the selected formulation, then derives a solution procedure specifying initialization, scores, selection, updates, and stopping. Code generation implements the solution procedure.

C.1.2 Generated Components

The system selects formulations and tools, expands tools into retrieval queries, retrieves mathematical knowledge, instantiates the formulations, derives solution procedures, generates and repairs code, and records evaluation feedback. Its generated computations include variance scaling, medoid initialization, sequential matrix updates, orthogonal residual updates, question-dependent terms, product distances, and genetic subset search. Appendix B.10 gives the primary-run coverage and score statistics, Appendix B.7 presents representative high-scoring procedures, and Appendix E.8 documents validation checks.

C.1.3 Discovery Provenance of the Three Evaluated Selectors

Table 20 traces the components of the three evaluated selectors through the discovery records. The mathematical foundations include reconstruction by projection, log-determinant design, Sherman–Morrison inverse updates, and farthest-point selection. The discovery loop turns these foundations into task-specific transformations and executable solution procedures.

The reference candidates are FPS at attempt 441, Leverage at attempt 1198, and ORGS at attempt 1316. These candidates supply the core mechanisms evaluated in the main paper: variance-scaled

Table 20: **Discovery provenance of the three evaluated selectors.** "System" marks a component proposed, implemented, and evaluated by the discovery loop within the candidate. The table records the discovered objectives and core update rules alongside the auxiliary implementation choices tested in the diagnostic comparisons. Together, these entries trace the components from the reference candidates to the final evaluation implementations.

Component	FPS (attempt 441)	Leverage (attempt 1198)	ORGS (attempt 1316)
Recorded objective	minimax coverage	information gain: preserve reconstructive information and limit redundancy	span reconstruction (reconstruction formulation)
Preprocessing	variance scaling in the attempt 441 candidate (system)	none retained	coordinate standardization (system)
Initialization	medoid initialization (system); fixed-seed initialization tested as an auxiliary variant and used in final evaluation	$P_0 = \lambda^{-1} I_d$ (system)	empty orthonormal basis (system)
Update rule	farthest-point selection (retrieved classic)	Sherman–Morrison update (system)	orthogonal residual updates (system; traced lineage at attempt 674)
Question-dependent term	none	none	none

farthest-point updates for FPS, sequential ridge-leverage updates for Leverage, and orthogonal-residual updates for ORGS; the ORGS reference candidate additionally uses coordinate standardization. The post-discovery comparisons use these objectives and core update rules as references. The diagnostic variants remove or substitute specified components to assess their contributions and the sensitivity of the procedure to those choices. Leverage and ORGS retain their discovered core procedures in final evaluation. For FPS, the final evaluation adopts fixed-seed initialization following a comparison with the discovered medoid initialization and a max-norm variant. These choices yield similar development means, and fixed-seed initialization reduces the initialization cost by avoiding the $O(N^2)$ medoid computation. An explicit exclusion rule restricts each selection to the remaining indices, ensuring K distinct outputs while retaining the variance-scaled distance and farthest-point update. Historical excerpts are preserved in Appendix F.5.

C.1.4 Representative Families and Diagnostic Scope

We analyze three representative high-performing families surfaced by discovery in order to test three characteristic mechanisms: orthogonal residual updates, sequential inverse-matrix updates, and variance-scaled coverage. The corresponding ORGS, Leverage, and FPS reference candidates are attempts 1316, 1198, and 441; related discovered variants provide additional context. The later ORGS variant at attempt 1582 reaches a higher discovery score, illustrating that the same residual-based family admits multiple implementations while preserving its core mechanism.

The following comparisons measure component contributions through controlled perturbations of the discovered procedures. They remove or replace individual components and evaluate each variant against its reference candidate. Large performance drops indicate a substantial contribution from the affected mechanism under the tested settings, while small differences indicate robustness across the tested auxiliary choices. The subsequent specifications state the implementations used in final evaluation.

C.2 Component Diagnostics: Core Mechanisms and Auxiliary Choices

We compare perturbations of the reference candidates on OCRBench, ChartQA, and DocVQA at 5% token retention under dynamic resolution, averaging the three task scores expressed as percentages. These scores use a different input protocol from fixed-resolution discovery. The reference candidate in each family remains the anchor. Table 3 reports removals or replacements that substantially degrade performance, while Table 21 reports variants with smaller effects. This organization separates validation of the discovered core mechanisms from auxiliary implementation choices.

Table 21: Diagnostic component variants at $\rho = 0.05$ under dynamic resolution, complementing the large-effect ablations in Table 3. Reference candidates are repeated. Scores and their three-task mean are reported in percent. The ORGS and Leverage evaluation implementations retain their discovered reference procedures; FPS uses the fixed-seed auxiliary initialization while preserving the discovered variance scaling and farthest-point update. Bold indicates the best score within each family and column.

Family	Variant	OCRBench	ChartQA	DocVQA	Mean
ORGS	reference candidate (final evaluation)	**51.20**	34.72	**48.88**	**44.93**
	without standardization	48.10	**36.16**	48.08	44.11
Leverage	reference candidate (final evaluation)	48.10	36.12	48.10	44.11
FPS	discovered candidate	**48.70**	32.96	48.13	43.26
	max-norm initialization	47.50	33.28	**48.61**	43.13
	fixed-seed initialization (final)	48.10	**34.08**	**48.61**	**43.60**

ORGS: the orthogonal residual update is the core mechanism. The diagnostic variants isolate the role of the residual update from feature preprocessing. With orthogonalization retained, removing coordinate standardization changes the three-task mean only from 44.93% to 44.11%, a difference of 0.82 percentage points. In contrast, removing orthogonalization while keeping standardization lowers the mean to 34.35%, and removing both components lowers it to 25.52%. Thus the large effect is tied to the discovered orthogonal residual update, while coordinate standardization is a smaller auxiliary gain once that update is present. Replacing coordinate standardization with row-ℓ_2 normalization gives 25.61%, showing the sensitivity of ORGS to the feature geometry induced by these two normalization schemes.

Leverage: the sequential inverse-matrix update is the core mechanism. The discovered Leverage procedure obtains a three-task mean of 44.11% in Tables 3 and 21. Removing the Sherman–Morrison state update lowers the mean to 25.92%. This large drop supports the contribution of the sequential information update generated by discovery. Repeatedly discounting directions already represented by the selected set adapts the scores after each selection; removing this update leaves a one-shot quadratic scoring rule.

FPS: variance scaling is core; initialization is auxiliary. The discovered FPS candidate combines variance scaling with farthest-point selection. Removing variance scaling lowers the mean from 43.26% to 27.70%, and replacing the scaled squared Euclidean distance with cosine distance gives 18.04%. By contrast, changing only the initialization has a small effect: the discovered medoid initialization gives 43.26%, max-norm initialization gives 43.13%, and fixed-seed initialization gives 43.60%. The spread among these initialization choices is at most 0.47 percentage points. These results show that the tested FPS variants achieve similar accuracy across initialization choices, with substantially greater sensitivity to the distance definition and variance scaling. Final evaluation uses fixed-seed initialization because it preserves the discovered variance-scaled farthest-point mechanism while avoiding the pairwise $O(N^2)$ medoid computation.

C.3 Mathematical Formulations and Final Selection Procedures

Following the component analysis, we specify the FPS, ORGS, and Leverage implementations used for final evaluation through their objectives, preprocessing, selection rules, and state updates. These specifications state the discovered objectives and update rules together with the numerical safeguards used in implementation. For FPS, they incorporate the fixed-seed initialization evaluated above, connecting the component results to the final selection procedure. Figure 2 summarizes the mechanisms.

The matrix $X \in \mathbb{R}^{N \times d}$ contains row-wise selection features, with $x_i = X_i^\top$. A selected set S indexes the aligned model-input embeddings V. All procedures choose K indices and sort them into their original order before returning. Numerical computation uses float32; Appendix C.4 collects the constants and safeguards used in the implementation.

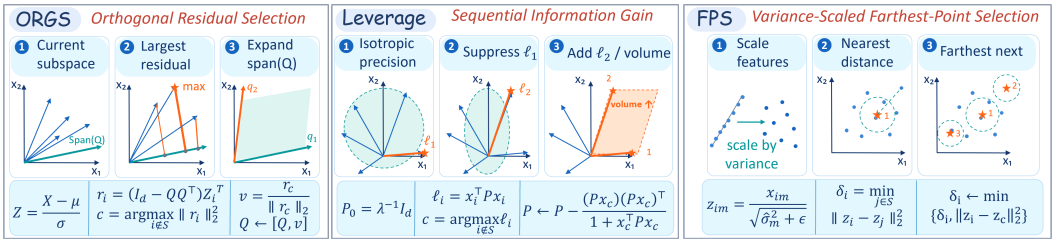


Figure 2: **Three selection principles and their updates.** *Orthogonal Residual Greedy Selection* (ORGS) expands the represented span by adding the largest residual direction. *Sequential Information Gain* (Leverage) scores a token's contribution and reduces weights along each newly selected direction. *Variance-Scaled Farthest-Point Selection* (FPS) rescales feature coordinates, tracks the nearest selected representative, and adds the farthest token.

C.3.1 FPS: Variance-Scaled Farthest-Point Selection

FPS seeks representatives that minimize the largest distance from any token to its nearest selected representative. Variance scaling defines distances for both the objective and selection rule.

For N row-wise feature vectors, coordinate m has empirical mean μ_m and variance σ_m^2. The mean, variance, and scaled coordinate z_{im} are defined as follows:

$$\mu_m = \frac{1}{N}\sum_{i=1}^{N} x_{im}, \qquad \sigma_m^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{im}-\mu_m)^2, \qquad z_{im} = \frac{x_{im}}{\sqrt{\sigma_m^2+\epsilon}}. \tag{6}$$

The generated code uses $\epsilon = 10^{-8}$. The denominator N corresponds to `unbiased=False`. Selection uses the scaled matrix Z, and the returned indices gather the model-input embeddings V. The squared distance between scaled vectors takes the following form:

$$\|z_i - z_j\|_2^2 = \sum_{m=1}^{d} \frac{(x_{im}-x_{jm})^2}{\sigma_m^2+\epsilon}. \tag{7}$$

This is a squared diagonal Mahalanobis distance in the original features: each coordinate difference is weighted by its inverse variance. Subtracting μ_m from both vectors leaves their difference unchanged. The implementation therefore rescales features directly.

The coverage objective takes the following form:

$$\min_{S\subseteq[N],\,|S|=K} F_{\text{cover}}(S;X), \qquad F_{\text{cover}}(S;X) = \max_{i\in[N]}\min_{j\in S}\|z_i - z_j\|_2^2. \tag{8}$$

The inner minimum gives each token's distance to the selected set; the outer maximum gives the largest. Farthest-point selection reduces this gap by repeatedly adding the least well-covered token.

The discovered candidate computes pairwise squared distances $D_{ij} = \|z_i - z_j\|_2^2$. It chooses the initial representative with the smallest total distance:

$$c_1 = \arg\min_{j\in[N]} \sum_{i=1}^{N} D_{ij}, \qquad S = \{c_1\}, \qquad \delta_i = D_{ic_1}. \tag{9}$$

This starting representative is the *medoid* under the summed squared-distance criterion. The initialization diagnostic in Appendix C.2 shows that medoid, max-norm, and fixed-seed starts have similar three-task means, whereas removing variance scaling causes a large drop. We therefore keep the discovered variance-scaled farthest-point mechanism and use the cheaper fixed-seed initialization for final evaluation. Specifically, the evaluation implementation chooses c_1 using a fixed random seed of 42, then initializes $S = \{c_1\}$ and $\delta_i = \|z_i - z_{c_1}\|_2^2$. Both versions use the following nearest-representative update; the evaluation implementation explicitly restricts each choice to unselected indices as a correctness safeguard:

$$c = \arg\max_{i\in[N]\setminus S} \delta_i, \qquad S \leftarrow S\cup\{c\}, \qquad \delta_i \leftarrow \min\{\delta_i, \|z_i - z_c\|_2^2\}. \tag{10}$$

The maintained δ_i is token i's squared distance to its nearest selected representative. Selection ends at K indices, even when features coincide. Squared Euclidean and Euclidean distances give the same farthest-point ranking; the recorded medoid uses the sum of squared distances.

The final selector stores scaled features, selected indices, and one length-N distance vector. Each selection and distance update costs $O(Nd)$, giving $O(KNd)$ total time and $O(Nd)$ memory. Appendix F.6 maps these steps to code and explains the initialization cost.

C.3.2 ORGS: Orthogonal Residual Greedy Selection

ORGS seeks a set of vectors whose span reconstructs the full feature set. For coordinate m, let μ_m and s_m be its empirical mean and sample standard deviation (denominator $N-1$). Coordinate standardization is defined as follows:

$$z_{im} = \frac{x_{im} - \mu_m}{\max(s_m, \epsilon)}, \qquad \epsilon = 10^{-8}. \tag{11}$$

Let Z contain these standardized rows. For subset S, the matrix $\Pi_S \in \mathbb{R}^{d\times d}$ projects onto the span of the selected feature vectors. This span is the set of their linear combinations. The reconstruction objective takes the following form:

$$\min_{S\subseteq[N],\,|S|=K} \|Z - Z\Pi_S\|_F^2, \qquad \Pi_S = Z_S^\dagger Z_S, \tag{12}$$

where † denotes the Moore–Penrose pseudoinverse. The procedure maintains an orthonormal basis Q, a set of mutually perpendicular unit vectors for the span. Its residual matrix is $R = Z - ZQQ^\top$. It starts with an empty basis, $R = Z$, and $S = \varnothing$. At each nondegenerate step, the procedure computes residuals and selects a token as follows:

$$r_i = (I_d - QQ^\top)Z_i^\top, \qquad c = \arg\max_{i\in[N]\setminus S} \|r_i\|_2^2, \tag{13}$$

where I_d is the identity matrix. The vector r_i is the transpose of row i of R, and its length measures the unrepresented component. Gram–Schmidt orthogonalization removes components of r_c along the existing basis. Normalization then gives a unit vector v. The state updates are performed as follows:

$$Q \leftarrow [Q, v], \qquad R \leftarrow R - (Zv)v^\top, \qquad S \leftarrow S \cup \{c\}. \tag{14}$$

Under exact orthogonality, $Zv = Rv$, so this implementation equals residual projection. The update costs $O(Nd)$ per direction.

The pivot score $\|r_i\|_2^2$ measures the remaining component of an individual token. Under exact orthogonality, the selected unit direction v decreases reconstruction loss by

$$\|R\|_F^2 - \|R - (Rv)v^\top\|_F^2 = \|Rv\|_2^2.$$

The pivot therefore supplies an efficient direction-selection heuristic, and the projection update reduces reconstruction loss. The total cost is $O(KNd + K^2d)$, including basis orthogonalization. If the largest squared residual falls below 10^{-12}, remaining slots are filled by random sampling without replacement from unselected indices. Appendix C.4 specifies numerical safeguards.

In a two-dimensional standardized feature space with a horizontal selected span, $(2, 0.1)$ and $(0, 1)$ have residual lengths 0.1 and 1, respectively. ORGS selects $(0, 1)$ and adds its vertical direction to the basis. The selection follows the unrepresented component of each vector.

C.3.3 Leverage: Sequential Information Gain

Let $\lambda > 0$ be a ridge parameter, a positive constant that keeps the information matrix invertible. For a size-K subset S, the objective is to maximize $J(S)$, defined together with its associated inverse information matrix as follows:

$$J(S) = \log\det\left(I_d + \lambda^{-1}\sum_{i\in S} x_i x_i^\top\right), \qquad P = \left(\lambda I_d + \sum_{i\in S} x_i x_i^\top\right)^{-1}. \tag{15}$$

Here, x_i is the transpose of row i of X. The matrix P is the inverse regularized information matrix for the current set S. It gives larger weights to weakly represented directions. The matrix determinant lemma yields $J(S \cup \{i\}) - J(S) = \log(1 + x_i^\top P x_i)$. This identity motivates the sequential selection score. The final selector initializes $S = \varnothing$ and $P = \lambda^{-1} I_d$. It selects the largest score and applies the Sherman–Morrison formula, an incremental update of the inverse matrix:

$$c = \arg\max_{i \in [N] \setminus S} x_i^\top P x_i, \qquad P \leftarrow P - \frac{(Px_c)(Px_c)^\top}{1 + x_c^\top P x_c}, \qquad S \leftarrow S \cup \{c\}. \tag{16}$$

The update discounts represented directions, making subsequent scores reflect the selected set. Selection continues to K indices. For the visual-only score, $\log(1 + s)$ and s give the same ranking. The dense implementation takes $O(KNd^2)$ time and maintains an $O(d^2)$ matrix.

The final selector uses raw selection features. Appendix C.4 gives the ridge parameter, precision, and numerical safeguards.

C.4 Final Selector Settings

The settings below are those used in final evaluation. ORGS and Leverage retain the discovered reference procedures; FPS retains the discovered variance scaling and farthest-point update while using the auxiliary fixed-seed initialization described above. All selectors compute scores and matrix operations in float32 and return K indices in ascending order. ORGS uses sample standard deviations (denominator $N - 1$), clamped below at 10^{-8}. If the largest squared residual is below 10^{-12}, it randomly fills remaining slots without replacement.

Leverage uses raw selection features and $\lambda = 10^{-3}$. Its numerical safeguards add 10^{-8} to the update denominator, restore matrix symmetry, and add 10^{-8} to the diagonal. These settings follow attempt 1198.

FPS computes population variances with denominator N, adds $\epsilon = 10^{-8}$, and multiplies each coordinate by the inverse square root. The final selector initializes using `torch.Generator` with seed 42. Subsequent steps exclude selected indices, including when features coincide. The discovered candidate uses medoid initialization, choosing the token with the smallest total squared distance to all tokens.

The finalized selectors above are used in the complete benchmark and timing comparisons below.

D Complete Benchmark and Efficiency Results

All results below use the finalized selectors specified in Appendix C.3. Appendix D.1 completes the QwenVL reference-benchmark comparison at 20% token retention. Appendix D.2 reports results on seven additional QwenVL benchmarks across three token budgets. Appendix D.3 reports transfer results on SAIL-VL-2B and InternVL3.5-4B across the same three token budgets. Appendix D.4 specifies the fixed-resolution timing protocol and reports selection cost, inference latency, and peak memory.

D.1 QwenVL Reference-Benchmark Results at 20% Token Retention

Table 22 completes the seven-reference-benchmark comparison at 20% token retention; the 10% and 5% results appear in Table 4(a). The following subsections report the additional tasks and backbone-transfer results at all three budgets.

D.2 Complete Additional-Benchmark Results

The additional evaluation includes InfoVQA (Mathew et al., 2022), RealWorldQA (xAI, 2024), AI2D (Kembhavi et al., 2016), ChartQA (Masry et al., 2022), DocVQA (Mathew et al., 2021), SEED-Bench-2-Plus (Li et al., 2024), and MMStar (Chen et al., 2024). ChartQA and DocVQA are also used for the component diagnostics described in Appendix C.2. The other five tasks are outside development. Table 23 gives the task-level scores underlying the additional-Qwen-task means in Table 4(b). It compares the final selectors with MMTok (Dong et al., 2026) at 20%, 10%, and 5% token retention using the same Qwen2.5-VL-7B pipeline.

Table 22: **QwenVL reference-benchmark results at 20% token retention.** Task scores (percent, except MME) and mean performance retention (Avg.†, %) for Qwen2.5-VL-7B under native dynamic resolution. Benchmarks, abbreviations, and metrics follow Table 4(a). Bold marks the best compressed result per column.

Method	GQA	MMB	MME	POPE	TextVQA	SQA	OCRBench	Avg.†
Full tokens	60.48	83.25	2327	86.16	82.95	87.46	83.80	100%
20% tokens								
VisionZip	56.80	80.33	2174	83.38	65.16	84.23	59.50	89.5%
DivPrune	56.70	76.98	2163	80.59	64.75	80.91	48.10	85.8%
CDPruner	55.23	78.44	2136	78.85	72.72	83.64	57.90	88.7%
MMTok	58.09	79.30	2217	82.38	71.93	81.61	59.60	90.5%
ORGS	**58.69**	80.58	2289	**85.76**	79.69	**84.83**	73.50	96.1%
Leverage	58.63	**80.93**	**2299**	85.46	**79.85**	84.73	**74.40**	**96.3%**
FPS	58.41	80.50	2289	84.12	79.21	84.78	74.30	95.8%

Table 23: **Additional-benchmark comparison on Qwen2.5-VL-7B under native dynamic resolution.** All task scores are reported in percent. Avg.† is mean performance retention relative to the full-token result of the same pipeline. Average normalized Levenshtein similarity (ANLS) measures text-answer agreement using edit distance.

Method	InfoVQA ANLS ↑	RealWorldQA Acc. ↑	AI2D Acc. ↑	ChartQA Acc. ↑	DocVQA ANLS ↑	SEED-Bench-2-Plus Acc. ↑	MMStar Acc. ↑	Avg.† ↑
Dynamic Resolution (MinPix $= 256 \times 28 \times 28$, MaxPix $= 2048 \times 28 \times 28$)								
Full tokens	79.91	69.28	82.74	83.24	94.47	70.58	62.75	100%
Token-Retention Ratio: 20% ↓ 80%								
MMTok	48.77	63.40	78.47	59.24	73.28	62.01	54.12	81.5%
ORGS	52.50	**67.06**	80.15	**65.60**	80.58	64.08	**56.12**	86.2%
Leverage	52.66	66.93	80.08	65.56	80.74	**64.34**	55.92	**86.3%**
FPS	**55.26**	66.01	**80.63**	62.56	**83.14**	62.85	54.74	85.9%
Token-Retention Ratio: 10% ↓ 90%								
MMTok	35.80	58.30	71.53	39.12	48.41	54.68	47.12	66.6%
ORGS	39.16	63.79	**76.10**	50.84	67.18	57.62	**51.76**	**75.6%**
Leverage	39.06	63.01	75.45	**51.04**	66.82	**58.94**	51.04	75.4%
FPS	**39.83**	**64.31**	74.84	50.08	**68.04**	57.62	48.91	75.0%
Token-Retention Ratio: 5% ↓ 95%								
MMTok	27.09	53.20	66.97	22.28	26.06	50.72	40.59	54.6%
ORGS	**30.78**	**61.05**	**70.47**	34.72	**48.88**	**51.52**	43.12	**63.9%**
Leverage	29.52	60.78	70.14	**36.12**	48.10	51.47	**43.17**	63.6%
FPS	30.72	59.87	68.69	34.08	48.61	50.77	42.32	62.8%

D.3 Complete Cross-Backbone Results

Table 24 gives task-level scores underlying the SAIL-VL-2B and InternVL3.5-4B means in Table 4(b). The final selectors and MMTok (Dong et al., 2026) retain 20%, 10%, or 5% of visual tokens under each backbone's native tiling pipeline. Comparisons share per-image budgets and each backbone's full-token reference.

Mean performance retention equally averages the seven benchmark-wise score ratios, as defined in Appendix A.3. The benchmark-level gain over MMTok is $\Delta_{a,b} = 100(s_{a,b} - s_{\mathrm{MMTok},b})/s_{\mathrm{full},b}$, measured in percentage points. Gains use the displayed scores and are rounded afterwards.

D.4 Measured Inference Speed

We time the final selectors on Qwen2.5-VL-7B using 100 OCRBench images and one NVIDIA A800 80 GB GPU. Fixed resolution produces about 2094 visual tokens per image. A 5% budget keeps about 104. We time selection, vision encoding, and language-model prefill (input processing before generation) separately. We record TTFT and E2E latency through two generated tokens.

Table 24: **Complete cross-backbone results under dynamic tiling.** All methods for a given backbone share the same input visual-token sequences and per-image output budgets. The "Full tokens" row gives that backbone's reference performance. Task scores are reported in percent, except MME. Avg.† is mean performance retention relative to this reference, recomputed from the displayed benchmark scores and rounded to one decimal place. Bold marks the best compressed method in each column within a backbone and token budget.

Method	GQA Acc. ↑	MMB Acc. ↑	MME P+C ↑	POPE F1 ↑	TextVQA Acc. ↑	SQA Acc. ↑	OCRBench Acc. ↑	Avg.† ↑
SAIL-VL-2B (InternViT + MLP projector, dynamic tiling)								
Full tokens	62.05	77.95	1918	87.07	75.03	94.82	81.60	100%
Token-Retention Ratio: 20%								
MMTok	60.08	74.14	1679	87.61	38.62	88.99	21.40	78.8%
ORGS	60.57	73.97	**1813**	**88.46**	67.80	91.87	**54.90**	**91.9%**
Leverage	60.27	**74.23**	1793	88.34	**67.90**	92.17	52.90	91.4%
FPS	**60.79**	74.05	1752	88.23	67.47	**92.22**	54.20	91.3%
Token-Retention Ratio: 10%								
MMTok	**58.51**	70.19	1651	85.77	33.03	85.92	18.80	75.2%
ORGS	58.28	**70.45**	1665	**87.83**	**59.92**	88.65	40.70	85.0%
Leverage	58.22	69.50	1653	87.69	59.85	88.75	**42.70**	85.1%
FPS	58.20	68.56	**1690**	87.78	59.89	**89.79**	42.10	**85.3%**
Token-Retention Ratio: 5%								
MMTok	**55.49**	**65.81**	1509	82.36	26.22	83.09	14.80	69.7%
ORGS	54.96	64.09	**1541**	**84.96**	**49.93**	**86.66**	31.50	**77.9%**
Leverage	54.89	63.83	1537	84.78	49.32	85.57	31.60	77.5%
FPS	54.44	63.66	1517	83.85	49.17	86.47	**33.10**	77.4%
InternVL3.5-4B (InternViT + MLP projector, dynamic tiling)								
Full tokens	59.71	81.79	2286	89.55	75.84	96.48	83.00	100%
Token-Retention Ratio: 20%								
MMTok	**56.68**	74.48	1996	**88.04**	30.67	85.47	11.00	73.4%
ORGS	55.25	77.32	**2148**	86.30	**66.17**	90.18	**47.90**	**88.0%**
Leverage	55.42	**77.58**	2121	86.08	65.98	**90.28**	47.00	87.7%
FPS	54.87	77.15	2129	85.75	65.45	90.23	43.20	86.7%
Token-Retention Ratio: 10%								
MMTok	**54.25**	70.62	1860	**86.74**	25.45	81.21	6.50	68.7%
ORGS	52.19	73.45	**1992**	84.44	56.42	85.82	**34.20**	**80.5%**
Leverage	52.17	**73.54**	1968	83.90	**56.54**	85.67	33.30	80.1%
FPS	50.41	72.77	1965	82.94	55.10	**85.87**	29.90	78.5%
Token-Retention Ratio: 5%								
MMTok	**51.80**	67.96	1782	**84.29**	20.66	77.69	5.60	65.2%
ORGS	48.37	67.44	**1786**	81.42	45.78	82.20	**21.20**	71.9%
Leverage	48.84	**68.47**	1759	81.70	**45.93**	**82.55**	20.00	**72.0%**
FPS	46.70	67.87	1760	79.00	44.00	81.95	18.90	70.3%

Each image has three timed repetitions after eight warm-up runs; the table averages over images and repetitions.

Table 25 gives the timing results. Removing 95% of visual tokens reduces prefill from 167.3 ms to about 24.9 ms, or about $6.7\times$ faster. Selection adds 9.9–29.0 ms, less than the prefill time saved. Time to first token falls from 428.6 ms to 297.9–313.8 ms, giving 1.37–$1.44\times$ speedups. Latency through two generated tokens falls from 451.3 ms to 320.4–335.8 ms, giving 1.34–$1.41\times$ speedups. Per-token decoding remains close to the full-token result. Peak memory is 15.94 GB, compared with 15.96 GB. The main improvement comes from reduced input processing time.

The remaining appendices document the search implementation and historical records underlying these results.

Table 25: **Inference time on Qwen2.5-VL-7B using** 100 **OCRBench images.** The full-token reference processes about 2094 visual tokens per image. A 5% budget keeps about 104. Select measures token selection, Vision measures image encoding, and Prefill measures language-model input processing. TTFT denotes time to first token. E2E is latency through two generated tokens. Decode is per-token generation time, and Mem is peak GPU memory. Entries are averaged over images and three timed repetitions per image.

Method	Select (ms)	Vision (ms)	Prefill (ms)	TTFT (ms)	E2E (ms)	Decode (ms/tok)	Mem (GB)
Full tokens	—	253.1	167.3	428.6	451.3	22.69	15.96
MMTok	10.4	253.2	24.8	296.5	318.9	22.45	15.94
ORGS	17.6	253.3	24.9	303.8	326.0	22.19	15.94
Leverage	29.0	251.9	24.8	313.8	335.8	22.07	15.94
FPS	9.9	254.9	24.9	297.9	320.4	22.50	15.94

E Discovery Framework Implementation

Appendix E.1 describes the discovery stages, saved artifacts, data flow, and feedback paths. Appendix E.2 lists the configured mathematical formulations and explains their use in proposal generation. Appendix E.3 specifies scheduling rules, repeated-exploration controls, and extended-run settings. Appendix E.4 defines toolbox growth and the area–operation coverage accounting used to guide exploration. Appendix E.5 details query expansion, retrieval sources, quality filters, and publication-window rotation. Appendix E.6 explains how retrieved mathematical ingredients become an instantiated formulation and solution procedure. Appendix E.7 specifies code generation, program insertion, validation, and repair. Appendix E.8 reports selection-integrity checks and distinguishes discovery-time validation from post-hoc verification. Appendix E.9 records the generator model, decoding parameters, and request limits. Appendix E.10 specifies the FunSearch-style, EoH-style, and ShinkaEvolve-style controls. Appendix E.11 defines the changes used in the framework ablations. Appendix E.12 explains score-to-code attribution and reconciles primary-run records and repair outcomes.

E.1 Discovery Stages, Data Flow, and Feedback

Each attempt links its formulation and mathematical tool to a proposal, an executable program, and an evaluation outcome or failure record.

E.1.1 Stages and Saved Artifacts

Table 26 lists the stages, model calls, and saved artifacts. Fixed routines schedule attempts, apply checks, run evaluation, and update records; language-model calls synthesize tools, extract mathematical ingredients, derive proposals, and implement or repair programs.

From attempt 1530 onward, cross-domain transfer attempts replace stages 2–3 with a prompt that identifies the parent candidates' shared problem and derives an approach from another mathematical area (Figure 6).

Stage 5 applies the configured checks in the following order: static source checks (interface completeness, prohibited patterns, and hard-coded budgets), screening of the operation combination against saturation and quota limits, static fallback patterns, the synthetic probes, and the behavioral-duplicate comparison. Implementation and execution failures return to implementation with the gate's diagnostic, within the remaining repair allowance. Rejections for saturation, quota limits, or matching probe outputs also return their diagnostic to implementation; when no repair allowance remains, the attempt ends with the recorded reason and skips downstream evaluation. A candidate that passes the applicable checks reaches the frozen evaluation of stage 6.

Table 26: Stages of one discovery attempt. "LLM" marks stages that call the generator model. Artifacts are the files persisted per attempt in the run log.

#	Stage	LLM	Persisted artifact
0	Formulation scheduling and area–operation choice from the coverage table	no	decision record
1	Mathematical tool synthesis and selection	yes	trajectory entry
2	Knowledge retrieval: retrieval query expansion, retrieval, computation extraction	yes	`0a_recalled_knowledge.json`
3	Formulation instantiation and solution derivation (formulation–solution pair)	yes	`1_algorithm_idea.md`
4	Implementation in the selection block + repair	yes	`2_modified_code_*.py`
5	Source checks, operation screening, synthetic probes, and duplicate checks	no	check results
6	Frozen downstream evaluation	no	`3_execution_*.log`
7	Audit and update of search-space coverage records, operation records, and results	no	search record update

Table 27: Information exchanged between stages. The formulation–solution pair guides implementation and repair, and its record links the candidate program to the evaluation outcome.

Stage group	Receives	Produces
Formulation/tool selection and proposal (stages 0–3)	Task interface; chosen mathematical tool, formulation, and question-usage mode; retrieved titles, abstracts, and definition articles	Objective and constraints; mapping to token selection; initialization and update rules; pseudocode; numerical settings; sources
Implementation and repair (stage 4)	The proposal; allowed tensors and their shapes; marked function template; diagnostics from prior attempts	A merged candidate program, together with validation and repair diagnostics
Validation and evaluation (stages 5–7)	Candidate program and proposal; fixed probe inputs; downstream evaluator; previous evaluation records	Probe outputs; execution log; valid score or failure status; updated candidate history

E.1.2 Information Passed Between Stages

All stages share the task interface and execution limits. Table 27 specifies the inputs and outputs passed between stage groups.

Feedback to subsequent attempts. These records support four feedback paths:

- Evaluation outcomes identify mathematical areas associated with stronger proposals and guide the generation of additional tools in those areas.
- Area–operation coverage and tool-usage records identify underexplored areas and tools, balancing performance-guided expansion with broader exploration.
- Retrieval and execution failures guide tool filtering and program repair.
- Operation signatures and selected-index outputs identify repeated computations and discourage redundant exploration.

The following subsections specify the formulation catalog, scheduling rules, and processing stages that produce these records.

E.2 Formulations and Their Use

The search configuration contains 23 formulations generated during initialization. Each expresses a preservation goal through a mathematical objective, variables, and constraints. For each selected formulation, proposal generation uses retrieved mathematical knowledge to specify the task mapping, feature transformations, distances, and parameters, then derives a solution procedure for the instantiated formulation. Section 3 illustrates coverage, reconstruction, and information gain. The search also starts with 158 seed tools, whose role is described in Appendix E.4. Tables 28 and 29 list selected catalog entries and whether each can use the question. A 1:1 schedule alternates visual-only and question-dependent proposals. Seventeen formulations give an objective without a solution-procedure hint, and six include a short hint. Examples include greedy removal for an independence criterion, extreme-direction scans for boundary coverage, and sensitivity sampling for kernel approximation. These hints help turn a formulation into a concrete procedure.

The formulation tables use mathematical abbreviations. Mutual information (MI) measures statistical dependence. Model predictive control (MPC) plans actions ahead. The evidence lower bound (ELBO) is an objective for variational inference. Optimal transport (OT) compares distributions by transport cost.

The Hilbert–Schmidt independence criterion (HSIC) measures dependence using kernels. Kernel density estimation (KDE) approximates a distribution by combining local kernels. The Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) solves structured optimization problems through repeated gradient and thresholding steps. The formulation identifiers retain their configured spelling.

The scheduler alternates between visual-only and question-dependent proposals in a 1:1 ratio. Within each mode, a separate formulation cycle determines the selected objective. The visual-only cycle contains 15 slots, and the question-dependent cycle contains 6 slots. These cycle lengths determine formulation frequencies within each mode; they do not change the 1:1 allocation between modes. The visual-only sequence has 15 slots: coverage, kde_coreset, decision, hsic_independence, control, vector_balancing, game, boundary_hull, sampling_measure, partition_matroid, reconstruction, self_expressive, information gain, kde_coreset, and hsic_independence. The question-dependent sequence has 6 slots: alignment, kde_coreset, information, vector_balancing, decision, and partition_matroid. The scheduler uses the repeated-exploration records described in Appendix E.3 to guide formulation selection. Each analyzed candidate record identifies its formulation.

The final Leverage formulation uses $\lambda > 0$ to keep its information matrix nonsingular (Equation (15)).

E.3 Scheduling Rules and Extended-Run Settings

The matched comparison over the first 150 attempts uses breadth-oriented sibling attempts. The settings below distinguish this phase from later parent-based refinement and cross-domain transfer; Table 30 collects the scheduling, retrieval, and validation constants.

Attempt allocation. The primary run initially uses `breadth_only`. The restart after attempt 1471 disables this restriction and permits later depth and cross-domain attempts. Full-run statistics therefore include these later scheduling phases. Most attempts explore a mathematical tool drawn from the growing mathematical toolbox described in Appendix E.4. From attempt 1530 onward, some attempts transfer a problem to another mathematical area, capped at 10% of attempts. From attempt 1472 to 1773, part of the budget derives variants of the best-scoring candidates recorded so far; Appendix B.9 reports the corresponding refinement outcomes.

Limiting repeated exploration. The scheduler tracks repeated exploration of computation groups and supplies relevant exclusions to the proposal prompt. These records guide later attempts toward other computations.

Comparing proposed computations. An operation signature records a candidate's computations and their order. Comparisons use 11 broad labels and 21 individual operation patterns. Broad labels include singular-value decomposition (SVD), which separates a matrix into singular directions and values, and farthest-point selection. Individual patterns include `cdist` and `topk`. A registry maps

Table 28: Selected entries from the configured formulation catalog. The Text column indicates whether the question can enter the objective directly. Expressions reproduce the configured formulations, with formulation-local symbols; each proposal maps them to the task inputs and specifies their dimensions and constraints. Further entries appear in Table 29.

Formulation	Objective and purpose	Text
coverage	$\max_{\lvert S\rvert=K} f_{\text{cover}}(S;V)$ cover/represent the full set (facility location, dispersion, coresets)	no
alignment	$\max_{\lvert S\rvert=K} f_{\text{align}}(S;q)$ the query itself defines the objective	yes
reconstruction	$\min_{\lvert S\rvert=K} D(V,\text{recon}(S))$ full set reconstructible from S	no
information	$\max I(S;q)-\lambda I(S;V\backslash S)$ bottleneck / conditional-MI forms	yes
decision	$\max \mathbb{E}[U(S)]-\lambda\,\text{Risk}(S)$ utility under uncertainty, risk-averse	yes
control	$\pi^*(s_t)=\arg\max_a Q(s_t,a)$ sequential bandit/MPC-style selection	no
game	equilibrium allocation of K slots auction/matching/voting among tokens	no
sampling_measure	$S\sim P(S)$, $\max \mathbb{E}[\text{quality}]$ DPP/Gibbs/stratified subset measures	no
optimal_transport	$\min_{\pi\in\Pi(S,V)}\sum_{ij}\pi_{ij}c(i,j)$ Wasserstein/Sinkhorn coupling to V	no
variational_inference	$\max\ \text{ELBO}(S)$ free-energy/posterior approximation	no
robust_optimization	$\max_S \min_{w\in\mathcal{U}} f_w(S)$ worst case over an uncertainty set	no

Table 29: Additional catalog entries, continuing Table 28. Formulation names and objective forms follow the configuration.

Formulation	Objective and purpose	Text
hsic_independence	$\min\sum_{i,j\in S}\text{HSIC}(v_i,v_j)$ remove nonlinear dependence; suggested procedure: greedy removal	no
boundary_hull	union of per-direction $\arg\max$ / $\arg\min$ convex-hull boundary preservation; extreme-direction scan	no
self_expressive	$\min_{S,A}\lVert V-AV_S\rVert_F^2+\lambda\lVert A\rVert_1$ global sparse reconstruction; FISTA	no
kde_coreset	min kernel-sum error sensitivity sampling preserving attention sums	yes
vector_balancing	$\min\lVert\sum_{i\in S}\sigma_i u_i-\frac{K}{N}\sum_i u_i\rVert_\infty$ discrepancy/cancellation, sign-walk	yes
partition_matroid	max coverage s.t. per-block quotas constrained greedy, exact quotas	yes
energy_physics	$\min E(S)=\sum_{i,j}U+\sum_i V$ Riesz/Thomson-type repulsion energies	no
sparse_coding	$\min_{D,A}\lVert V-DA\rVert_F^2+\lambda\lVert A\rVert_1$ dictionary/union-of-subspaces	no
conditional_rate_distortion	$\min D_q(V,\text{recon}(S;q))$ query-weighted reconstruction error	yes
cross_modal_transport	$\min_\pi \text{OT}(\pi;S\to\text{sem}(q))$ distribution-level coupling to text	yes
causal_intervention	$\max\ \mathbb{E}[Y\lvert\text{do}(S)]-\mathbb{E}[Y\lvert\text{do}(\varnothing)]$ counterfactual/Shapley-style attribution	yes

recorded operations and operation combinations to the cells of the search-space coverage table. The ordered operation sequence is stored with the operation set.

Operation-set comparisons use Jaccard similarity: intersection size divided by union size. An overlap above 0.72 with recent history rejects a proposal. Sequence comparisons use the normalized longest common subsequence (LCS), measuring shared operation order. A value above 0.75 also rejects a proposal. Message-Digest Algorithm 5 (MD5) produces a compact 12-hexadecimal-digit identifier for each combined record. Different scores for one record are flagged for inspection.

Scheduling formulations. The catalog contains 23 configured formulations; selected entries appear in Tables 28 and 29. Question-usage cycles appear in Appendix E.2. Repeated-exploration records inform the scheduling of these formulations.

The configuration defines the objective schedule. Each ordinary attempt records its formulation and proposed computations. Appendix C.1.3 describes the resulting records.

Search-label coverage enforcement. Each proposal declares one of 15 named search labels, such as `svd`, `clustering`, `geometry`, or `graph`; run accounting additionally includes a catch-all `general` label, giving 16 recorded labels in total. These labels are tracked separately from the area–operation cells of Appendix E.4. After a warm-up of 20 attempts, a search-label coverage rule may steer mathematical tool generation toward under-used named search labels. Never-used labels take priority. Labels used below 10% of the average usage are promoted with probability 0.15, by injecting mathematical building blocks associated with the label into the generated mathematical tools. A promoted label then cools down for five attempts after a never-used intervention and three after an under-used one. All 15 named labels and the `general` category appear in the primary-run records (Table 19).

Table 30: Discovery configuration and validation settings. Appendix E.8 reports validation events and probe analyses.

Group	Quantity	Value
Allocation	cross-domain transfer; variants of best-scoring candidates	$\leq$10%, from attempt 1530; attempts 1472–1773
Admission	duplicate rejection: operation-set Jaccard; operation-sequence LCS	0.72; 0.75
	repeated-exploration screening	computation-group records and proposal exclusions
Scheduling	formulations; visual-only / question-dependent cycle	23; 15 / 6 slots
	usage-mode rotation: visual-only / question-dependent	1:1
	toolbox extension	$\geq$60% used, <30 seeds untried, or every 20 attempts after attempt 50
Retrieval	retrieval queries per tool; sources	3–5; arXiv, OpenAlex, OpenReview, Wikipedia
	quality gate; year-window widening	$\geq$3 papers, $\geq$100 chars; +2 then +5 years
	era rotation (window, step); result cache	modern 1995–2014 (5, 2) / frontier 2015–2026 (3, 1); 7 days
Search-label coverage	recorded labels; warm-up; under-used trigger; forced cooldown	16 (15 named + `general`); 20 attempts; <10% of average use, p=0.15; 5 / 3 attempts
Implementation	repair budget; early stop	$\leq$2 batch rounds; repeated failure fingerprint
	deterministic fixers; issues per repair call; review agents	23 rules; 8; 4 parallel, 30 s each
	execution-log repair: attempts; context window; log summary	$\leq$2; $\pm$1000 lines; 4000 chars
Probes	synthetic input; trials; timeout	144 tokens, 12×12 grid, 5% (K=7); 4+1; 180 s

E.4 Growing Mathematical Toolbox and Search-Space Coverage Accounting

Section 4.3 describes toolbox growth in terms of mathematical areas and tools. This subsection specifies the implementation used to track exploration. An *operation* is a category of computation, such as distance calculation or projection; an *area–operation pair* combines a mathematical area with one such category. These pairs provide search-space coverage counters for scheduling tools. The search records explored computations in a search-space coverage table whose cells pair a mathematical area with an operation, giving $18 \times 14 = 252$ cells. Three levels of description appear in the records: a mathematical area is a broad field, a tool is a mathematical concept, principle, or operation used to instantiate a formulation or construct its solution procedure, and a *search label* summarizes a proposed mathematical approach for counting and reporting. Search labels mix areas (e.g., geometry) with methods or operations (e.g., SVD and projection). Each record has one declared label, while its procedure can combine several mathematical areas and operations. Label counts and entropy describe the distribution of these recorded tags; the formulation and implemented procedure describe the mathematical content. The (geometry, distance metric) cell, for instance, identifies feature-distance methods.

Mathematical areas include geometry, information theory, optimization, probability, algebra, combinatorics, game theory, analysis, cross-modal alignment, spatial topology, signal processing, evolutionary computation, control theory, neuroscience, compression, online learning, meta-learning, and adversarial robustness.

Operations include distance metric, ranking, greedy selection, thresholding, sampling, projection, iterative refinement, filtering, gating, evolution, feedback control, coding, simulation, and ensembling.

Steering attempts toward cells. The scheduler uses the search-space coverage counts to prioritize lightly explored mathematical areas and computations. For example, low coverage of (geometry, distance metric) motivates retrieval about geometric distances. The proposal uses the retrieved knowledge to derive a distance-based solution procedure for the selected formulation.

The proposal records one search label in `PRIMARY_DIRECTION`, while coverage uses a set of credited cells $\mathcal{C}_t$ for attempt t. The history tracker combines cells verified by the proposal judge with cells inferred from code operations. Without judge metadata, it combines proposal-text and code-derived cells. Composite proposals can credit multiple cells. The update is

$$C_{a,o} \leftarrow C_{a,o} + 1, \qquad (a, o) \in \mathcal{C}_t.$$

Attempt counts include unsuccessful outcomes; the registry separately tracks successful attempts and best recorded scores. Thus coverage measures exploration of configured cells, while discovery scores measure algorithm quality.

The registry defines the 252 cells as the Cartesian product of the 18 areas and 14 operations listed above. A cell is uncovered when its attempt count is zero, and full recorded coverage requires a positive count in every cell. The recorded cells support mathematical tool selection and proposal generation. This accounting uses the full area–operation registry, with the legacy search labels tracked separately.

Saved proposal records for the first 150 attempt IDs contain 147 explicit target-cell fields and 47 composite-cell fields; all 150 include unexplored-cell guidance. These artifacts directly document coverage-guided proposal generation. The complete run log first reports 100.0% coverage on August 10, during extended exploration. The supplementary audit records the relevant log entries and implementation rules.

Toolbox growth and mathematical tool selection. The toolbox holds seed tools generated by the LLM during initialization and supplied through the task configuration, together with later additions. Each tool has a descriptive name, and the scheduler tracks how often it is explored. Retrieval queries are generated only after tool selection and are separate from these tool-usage counts. The toolbox grows on demand: extension triggers when at least 60% of its tools have been explored, when fewer than 30 seed tools remain untried, or every 20 attempts after attempt 50. Each extension asks the language model for ten theory-focused tool names and removes duplicate tool names (temperature 0.75; Appendix E.9).

Toolbox expansion balances exploitation and diversity. Evaluation outcomes identify mathematical areas associated with stronger proposals, and the LLM generates additional tools that explore related mechanisms within those areas. Coverage and usage records also steer expansion toward underexplored areas and new combinations of operations. Repetition and failure records discourage redundant tools and repeated unsuccessful approaches. The expansion prompt additionally requires at least 60% of the proposed tools to target underexplored search labels when such labels are supplied. This coverage requirement constrains the expansion batch alongside performance-guided generation; it does not specify a separate fixed quota for high-scoring areas.

Each attempt assembles a shortlist from the existing toolbox and newly generated tools. The shortlist is deduplicated before selection. Tool selection uses performance feedback together with coverage, usage, and failure records, allocating further exploration to promising areas while retaining opportunities for underexplored tools. The selected tool then enters retrieval query expansion and formulation–solution derivation.

The run log records the generated tools and the selected tool, together with the per-attempt retrieval and proposal records of Table 26. A revisit to a tool can generate another solution procedure.

E.5 Retrieval Query Expansion, Retrieval, and Time Rotation

Retrieval supplies the mathematical ingredients for a proposal. Retrieval query expansion explores the theory of a selected tool through several lines of inquiry; the academic channels contribute research summaries, and Wikipedia supplies definitions. The following settings specify the retrieval stage of Section 4.4.

Retrieval settings. Retrieval query expansion produces three to five retrieval queries per tool focused on mathematical theory, at temperature 0.7; the instructions forbid library names and implementation keywords and require theorem-oriented terms. Retrieval queries use round-robin assignment and parallel academic-source searches: arXiv (at most 10 results per retrieval query), OpenAlex (10 records per page), OpenReview (venue-filtered, with per-venue pagination offsets), and Wikipedia definitions. Paper sources return titles and abstracts, while Wikipedia supplies full definition articles through the Wikimedia API.

A separate channel asks the generator to extract supported computations from the material retrieved in the same attempt: for each work it records the objective, the selection or update operator, data structures, and complexity statements that the text supports, marks unresolved attributes as uncertain, and skips works irrelevant to the mathematical tool.

Publication years rotate between a modern era (1995–2014) and a frontier era (2015–2026), scanned with sliding windows of five and three years and step sizes of two years and one year, respectively; the earliest era (1975–1994) is excluded because its records rarely carry abstracts.

Three filters then apply: title deduplication across retrieval queries, removal of papers without a usable abstract (missing or at most 50 characters), and a retrieval quality gate requiring at least three papers and 100 characters of material. A mathematical tool that fails the gate is retried with the year window widened by two and then by five years. Tool selection is described in Appendix E.4.

Turning retrieved text into usable ingredients. The extraction prompt in Figure 4 associates each retained work with its supported objective, selection or update rule, data structures, and cost. The resulting record is written to `0a_recalled_knowledge.json`. Proposal generation receives these extracted ingredients with the retrieved text and source information. For example, the FPS record connects a divergence tool to retrieved Bregman-divergence material and then to a variance-scaled coverage objective (Appendix F.5.1).

E.6 Mathematical Formulation and Solution Derivation

Inputs and output. The proposal model receives the selected formulation, mathematical tool, question-usage mode, retrieved material, and task interface. It writes the proposal to `1_algorithm_idea.md`. The model instantiates the formulation by specifying the objective, variables, constraints, task mapping, feature transformations, distances, and parameters. It then

derives a solution procedure from this instantiated formulation, specifying initialization, scores, selection, updates, and stopping, together with pseudocode and numerical settings. Figure 5 gives the prompt format, including mathematical reasoning and source attribution.

Connecting an objective to a computation. The reasoning explains which property the procedure improves and how its state tracks that property. For FPS, δ_i stores the current nearest-representative distance. Adding representative c changes it to $\min\{\delta_i, D(i, c)\}$, directly yielding the tensor update. For Leverage, the determinant lemma gives the gain from adding a vector, and the inverse-matrix identity gives an incremental update. For ORGS, choosing a large residual supplies a new direction, and projecting out that direction reduces reconstruction error. Appendix C.3 states these relationships, including the distinction between the information-gain score and the heuristic residual pivot score.

E.7 Implementation and Repair Details

Implementation compresses the proposal and inserts generated code into the fixed selection block. Source checks, synthetic execution, and bounded repair determine whether the candidate proceeds to downstream evaluation.

Proposal compression. Before code generation, a rule-based compressor reduces the proposal—about 6 KB of Markdown—to roughly 1 KB, and it does so without a generator call. Sections are classified by heading patterns. Sections describing the objective, the procedure or computation graph, hyperparameters, numerical stability, constraints, and the input–output mapping are kept; sections with first-principles arguments, theoretical sources and citations, background, experiments, and proofs are dropped. A section whose heading matches neither list is kept by default, preserving its content for code generation. The compressed text carries the algorithm name, at most two code blocks, key parameters, stability notes, and constraints, followed by up to five key-difference sentences marked as requirements to preserve. If the result falls below 200 characters for an original above 500, the compressor abandons classification and returns the first 30 non-empty lines together with up to two code blocks.

Marker-based merging. The editable region of the fixed model file is delimited by a marker pair. Merging reads the indentation of the start-marker line in the fixed file, normalizes the generated block by subtracting its minimum indentation so that its relative structure is preserved, shifts the block to the target indentation, and replaces the marker-delimited lines at line boundaries. The merged file must then parse as Python. If an earlier attempt has removed the markers, the merging step re-injects them around the algorithm core using semantic anchors—such as the assignment of the selected embeddings—and indentation analysis, and it falls back to the original file when the re-injected region fails its completeness checks.

Validation battery. Each generated or repaired program passes a battery of checks, with cheap checks first and all blocking issues collected each round. (i) *Syntax*: compilation, plus an abstract-syntax-tree check for the required class and method. (ii) *Safety*: twelve patterns reject shell commands, `eval`/`exec`, dynamic imports, and file deletion. (iii) *API rules*: for example, `pivoting=True` in `torch.linalg.qr` is rejected and automatically removed, and `svd`/`qr`/`eigh` calls require a float32 conversion. (iv) *Fallback detection*: uniform-stride selection, retain-all index vectors, and `arange` calls appearing shortly after an `except` clause are rejected. (v) *Token-retention-ratio checks*: hard-coded budgets are rejected; budget assignment must reference the configured ratio variable. (vi) *Logic checks*: required variables and their types, prohibited runtime method-selection branches, expensive linear-algebra calls such as `cholesky` or `svd` inside loops, tensor assignments inside deeply nested loops, and gradient-related operations. (vii) *Sandbox*: a restricted execution with a whitelist of builtins, a mocked `nn` module, and a read-only `os` interface, catching missing names and malformed definitions without importing the full model stack.

Two-tier repair. Repair first tries deterministic rules that need no generator call. The fixer holds 23 such rules, applied in a fixed order; representative examples convert tabs to spaces, balance brackets, remove the unsupported `pivoting` argument, insert `.float()` before NumPy conversion,

inject device arguments into tensor constructors, add float32 conversion before LAPACK calls, replace list-literal index vectors with typed tensors, and rewrite perfect-square grid assumptions into a divisor search. Any successful deterministic fix triggers re-validation without a generator call. If no rule applies, the highest-priority issues—syntax before API, logic, sandbox, fallback, safety, and retention—enter a single batch-repair call, at most eight issues per call, together with the compressed proposal and the full code. Two consecutive rounds with an identical issue set stop the loop early. The battery allows at most two batch-repair rounds per attempt.

Execution-log repair. A program that passes the battery can still fail at execution time. A further repair tier then receives the proposal, the full program when it fits within a size limit and otherwise the code context around the failing lines (up to ± 1000 lines), and a summary of the execution log that retains the first and last tracebacks within 4000 characters. A playbook of 19 rules matches known failure patterns—half-precision `svd`, dtype mismatches before matrix products, undefined text embeddings, or incorrect grid-shape assumptions, among others—and injects up to three prioritized hints into the repair prompt; one rule, the unsupported `pivoting` argument, rewrites the code directly without a generator call. At most two execution-log repair rounds are allowed per candidate, permitting an initial execution followed by at most two repaired executions. Battery-repair rounds and the additional review-triggered repair pass have separate allowances.

Review agents. After the battery passes, four lightweight review agents re-check syntax, logic, API usage, and safety in parallel, each with a 30 s timeout. Their findings feed one further repair pass before the program proceeds to validation.

Synthetic probes. These are small synthetic inputs used to check execution. Each candidate runs in a separate process with a 180 s timeout. A timeout or execution failure sends the candidate to repair. Each input contains 144 tokens on a 12×12 grid. A 5% token-retention ratio gives $K = 7$. The suite has four seeded trials and one additional trial with `image_grid_thw=None`. Even-numbered trials use independent Gaussian features. Odd-numbered trials use four clusters with effectively low-rank features to test redundant inputs.

The selection features have dimension 32, and the model-input embeddings have dimension 64. Unequal dimensions test whether a program uses each input in the correct feature space. The described probes check termination within the timeout, compatible feature dimensions, the absence of runtime errors, and enforce the return of exactly K unique, in-range indices. A candidate passes the probes only when its selections satisfy these count, uniqueness, and bounds checks. The full task contract additionally requires the indices to be returned in ascending order as a `torch.long` tensor on the input device. A separate audit of the eleven matched high scorers confirms exact count, uniqueness, and bounds on the cached probes; verification of original return order, dtype, and device requires the original outputs, since the cache stores sorted selections.

Each trial records a compact hash (MD5) of the selected index set. When behavioral-duplicate screening is applied, a match with an earlier candidate on every probe skips another downstream evaluation. Appendix E.8 reports the associated validation analysis. These outputs support the program-grouping and validation analyses in Appendices B.2 and E.8.1.

E.8 Result Verification and Selection Integrity

We link evaluated code to recorded scores and compare candidate selections under common probes. All eleven high-scoring candidates in the matched comparison pass both question conditions (Appendix A.5). Token-budget checks and behavioral comparisons provide complementary evidence: the former screen execution requirements, while the latter distinguish repeated outputs. Table 31 counts validation events, which may occur more than once within an attempt.

Discovery-time validation and the common post-hoc probe comparison are reported separately. The post-hoc comparison applies the fixed inputs and accounting rules in Appendix A.5.

E.8.1 Token-Budget Enforcement

Source checks inspect the abstract syntax tree (AST), which represents the program structure. The table groups these issues with other interface violations.

Table 31: Validation events in the retained primary-run console segment (from the restart near attempt 116 onward): 121 admission and 170 execution-integrity events. Counts measure validation events in the retained segment; early console events are outside this census.

Group	Event	Count
Search admission	saturated-operation proposal	115
	matching-output proposal	6
Execution integrity	source/AST interface issue	11
	runtime fault on probes	93
	feature-space dimension mismatch	56
	probe timeout	5
	incorrect token count	1
	fallback fingerprint	4
Total		291

The audit excludes an over-budget Ward-clustering candidate from the companion run. In the primary run, the token-count check intercepts attempt 1725, which returns 144 indices for a budget of 7; repair yields a budget-compliant candidate scoring 35.7%. This example illustrates how execution checks recover an evaluable procedure. Appendix E.12.1 gives the repair accounting.

E.8.2 Different Proposals with Identical Probe Outputs

Three primary-run proposals illustrate repeated selection behavior. The following descriptions summarize their recorded rules:

- attempt 1476 selects the largest residual and breaks ties using the smallest index.
- attempt 1488 selects the largest component outside the span of the selected vectors.
- attempt 1538 uses greedy residual selection without explicitly assigning negative scores to previously selected tokens.

All three return identical selections on the probe suite and score 49.3%. The residual update reduces scores along previously selected directions. On these probes, the versions with and without an exclusion mask therefore select the same tokens. An explicit exclusion rule handles vanishing residuals or ties when exactly K indices are required. The example shows why comparing descriptions and comparing executed outputs provide different information.

E.9 Generator and Decoding Configuration

All language-model roles use the hosted model `deepseek-v4-pro`. The roles cover retrieval, proposal generation, checking, code generation, and repair. Each role has a generation temperature, which controls output variation. The values are 0.7 for retrieval query expansion, 0.3 for extraction, and 0.7 for proposal generation. They are 0.5 for problem abstraction, 0.75 for cross-domain design, and 0.6 for proposal fusion. Tool generation uses 0.8, and toolbox extension uses 0.75. Code generation uses 0.2, with 0.1 for the fallback pass. Repair uses 0.2.

Calls allow up to 131,072 output tokens and use retries with increasing delays. These run-configuration values override code defaults. Code defaults specify a code-generation temperature of 0.3 and an output cap of 16,384 tokens. Reproduction therefore requires the run configuration.

E.10 Evolutionary Search Controls: FunSearch, EoH, and ShinkaEvolve

We adapt FunSearch-style (Romera-Paredes et al., 2024), EoH-style (Liu et al., 2024a), and ShinkaEvolve-style (Lange et al., 2026) search by replacing the proposal-generation step. Each produces a name, an objective or scoring rule, an implementation, and an expected improvement. The controls use the common task interface and evaluator; implementation and validation procedures are described in Appendix E.7. All three controls start from the 32% coverage selector and

run for 150 attempts on OCRBench at $\rho = 0.05$. They share the generator and asynchronous evaluation budget. FunSearch and EoH use proposal temperature 0.7; ShinkaEvolve samples a temperature from $\{0, 0.5, 1.0\}$ for each proposal. The adaptations place all three search strategies in the shared task pipeline.

E.10.1 Implementation of the FunSearch-Style Control

The FunSearch-style control uses a small version of the island-based program database in FunSearch (Romera-Paredes et al., 2024). Each island is a separate group of candidate programs.

Program database. The database has 4 islands, each holding at most 20 programs. Every island starts with the shared coverage selector. An entry stores the selection code and evaluation score. The score is filled in when asynchronous evaluation finishes. Implementation or validation failures receive 0 in the control's database after later evaluations are resolved. When an island exceeds capacity, it removes its lowest-scoring entries.

Prompt construction. Each attempt samples an island using a softmax over island-best scores, with temperature 1.0. The prompt includes that island's 2 highest-scoring programs in ascending score order. Each example carries its score. The prompt also includes the task interface and eight recent outcomes. It asks for a distinct solution procedure likely to improve the score.

Island reset. The island-reset counter uses archive-entry resolution events as its counting unit. Outer attempts and downstream evaluations are tracked through their respective records. Each event assigns a recorded score, or a failure value of 0, to a pending archive entry. Multiple entries associated with the same attempt can therefore contribute separately to the counter. Across the 150 outer attempts, the saved counter records 174 such events: 110 attempt IDs contribute one event each and 32 contribute two each, giving $110 + 32 \times 2 = 174$. The remaining eight attempt IDs contribute no event to this counter in the saved archive: entries for three IDs were removed by island resets before resolution, while entries for five IDs remain pending. Downstream evaluation status is determined from evaluation records; these archive states describe entry removal or pending resolution. Resets occur when synchronization crosses successive multiples of 40, yielding four resets at counter values 40, 81, 120, and 161. At each reset, the weakest island is replaced by a copy of the strongest island. Island sampling is close to uniform, with 33–43 choices per island and 150 choices in total.

E.10.2 Implementation of the EoH-Style Control

The EoH-style control adapts Evolution of Heuristics (Liu et al., 2024a) to a steady-state population. It generates one candidate per outer attempt.

Population. Each individual stores a one-line algorithm description, selection code, and score. The population holds at most 8 individuals and starts from the shared MMTok (Dong et al., 2026) coverage selector. The first 7 variants rotate through modification operators M1, M2, and M3. Later attempts sample uniformly from E1, E2, M1, M2, and M3.

E1 and E2 generate a new algorithm from two sampled parents. E2 can draw motivation from their shared ideas. M1 changes one core step. M2 changes numerical parameters. M3 removes redundant components. The run uses E1 30 times, E2 37 times, M1 28 times, M2 28 times, and M3 27 times.

Parent selection and survival. Parent-sampling weights are proportional to each score minus the population minimum. The prompt includes the chosen operation, parent descriptions, scored code, and eight recent outcomes. It also lists recently discarded attempts to reduce repetition. An evaluated offspring replaces the lowest-scoring individual only if its score is higher. Other offspring enter a 10-entry discard history used in later prompts.

E.10.3 ShinkaEvolve-Style Configuration

ShinkaEvolve introduces parent sampling, code-novelty filtering, and adaptive selection among generator models (Lange et al., 2026). Our adaptation uses four islands, each retaining up to 20 scored programs together with pending candidates. An island is sampled uniformly. Within it, parent i is

sampled with probability proportional to $\sigma(0.1(U(a_i) - \widetilde{U}))/(1 + n_i)$, where $U(a_i)$ is its recorded discovery score in percent, $\widetilde{U}$ is the island's median discovery score in percent, n_i counts its offspring, and $\sigma(u) = 1/(1 + e^{-u})$. Proposal context includes the parent, up to two highest-scoring alternatives, and up to two randomly sampled alternatives from that island. The migration routine copies leading non-best programs from the strongest island to the weakest while retaining each island's best program; its configured interval and rate are 10 archive score updates and 0.1.

For novelty filtering, this run represents code as normalized counts of tokens mapped into 512 bins by hashing, and computes cosine similarity against the island archive. Similarity above 0.95 triggers another proposal, with at most three generation tries within the outer attempt; after the third rejection, the last proposal proceeds to implementation. This is the `token-hash-cosine` backend recorded in the run log. The generator is fixed to `deepseek-v4-pro`, giving a single model choice in the shared comparison. Each proposal samples its temperature from $\{0, 0.5, 1.0\}$; code generation uses 0.2. These settings define the ShinkaEvolve-style adaptation evaluated here.

E.11 Framework Ablation Settings

Direct mutation replaces tool selection, retrieval, and derivation with proposals conditioned on the current best program and recent outcomes (Appendix B.5). The retrieval ablation retains formulation and derivation using model-internal knowledge (Appendix B.3).

Guided strategy selection ablation. The w/o guided strategy selection run uses the same generator (`deepseek-v4-pro`), OCRBench task, frozen Qwen2.5-VL-7B backbone, fixed resolution, 5% token retention, and 150-attempt budget as the discovery comparison. Its configuration enables `disable_score_feedback` and `random_strategy_selection`, removing history-based guidance from proposal generation and randomizing strategy choices, as described in Section 6.2. Mathematical retrieval, formulation–solution derivation, implementation, repair, evaluation, and score recording remain enabled. The comparison measures the combined effect of removing history-based guidance and randomizing strategy selection.

E.12 Evaluation Records and Version Attribution

Score-to-code attribution. Table 7 reconciles the counts behind Tables 1 and 2. Two bookkeeping details affect the raw records: the EoH-style (Liu et al., 2024a) run has 149 state records because one interrupted attempt left code without a record, and the full-framework state contains a duplicate failure record for one attempt. The budget stays fixed at 150 attempts in both cases. For each valid candidate, we identify the code version that produced its score from the evaluation records. The evaluation logs use uniform file names, so attribution also uses system logs, file timestamps, and stored gating fingerprints where available. Stored fingerprints are recomputed over the saved code versions for comparison. A gating fingerprint is an output signature that several code versions can share, and the timestamps resolve such cases. This recovered three direct-mutation attempts whose newest repair file was left over from an earlier interrupted attempt. For full-framework attempts with multiple saved versions, score-to-code attribution was checked against the gate sequences in the system logs. For ShinkaEvolve (Lange et al., 2026), all 84 selected code versions match the gating signatures stored with their valid scores. The score and the probe fingerprint of every clustered candidate therefore refer to the same program. All eleven candidates at or above 40% pass the probes under both input conditions defined in Appendix A.5.

E.12.1 Primary-Run Outcome Accounting

The primary run stores 2011 completion records over 14 calendar days. Within the attempt-ID range 1–2007 (called iteration IDs in the raw logs), exactly two IDs, 1690 and 1816, have no completion record, leaving $2007 - 2 = 2005$ distinct recorded attempt IDs. Six IDs—41, 1473, 1474, 1637, 1813, and 1814—each have two completion records; all other recorded IDs occur once. Thus, the total number of completion records is $2005 + 6 = 2011$. Full-run source and score statistics use timestamped completion records as the counting unit. The matched comparison fixes attempt IDs 1–150 as its budget, with the repeated failure at ID 41 contributing no score.

Repair outcomes. Of 2011 completion records, 1562 have valid scores and 449 lack valid scores. Score validity requires both successful execution and a finite numeric accuracy, verified from the outcome fields. The 70 harness-zero records flagged as unsuccessful are excluded.

Repair counts use the explicit second-inner-round marker, excluding separate in-generation self-correction and deterministic fixes. Logs contain this marker for 141 attempt IDs. One, 1690, has no completion record; the remaining 140 repair episodes can be linked by timestamp to outcomes, with 78 valid scores and 62 failures. The recovery rate is $78/140 = 55.7\%$, counting an episode as recovered when execution succeeds and returns a finite score. Eleven repaired candidates reach 40%, including eight reaching 45%. Joining only by attempt ID would incorrectly count 79 recoveries: at ID 1813, a repaired proposal fails, and a later fresh proposal succeeds. Within IDs 1–150, eight attempts enter this repair tier and five receive valid scores (62.5% recovery); attempt 36 reaches 44.3%. Its scored r1 version changes the axis of an `unsqueeze` in a Cholesky block concatenation, preserving the kernel, scores, and selection rule of r0. This case documents preservation of the selection procedure through shape repair.

Table 31 counts 291 events in the retained console segment: 121 search-admission events and 170 execution-integrity events. A candidate may trigger several events. The counts describe issues detected by the active checks within this retained segment.

The prompt templates and discovery excerpts below provide the corresponding generation records.

F Prompt Templates and Discovery Examples

Appendix F.1 provides the knowledge-retrieval and mathematical proposal-generation templates. Appendix F.2 provides the code-generation and local-repair templates. Appendix F.3 presents the direct-mutation ablation prompt. Appendix F.4 illustrates the direct-mutation run with sampled attempts and their scores. Appendix F.5 preserves historical proposals and code excerpts for the FPS, ORGS, and Leverage candidates. Appendix F.6 derives the FPS distance and incremental update and maps them to tensor operations. These historical records document the generated candidates; the final algorithm specifications appear in Appendix C.3.

Templates retain the recorded instructions with terminology aligned to this paper. Bracketed ellipses mark omissions, {`braces`} identify runtime placeholders, and translated headings are marked. The proposal prompt lists 15 named search labels; run accounting additionally includes the catch-all `general` label, giving 16 labels in total.

F.1 Knowledge Retrieval and Proposal Generation

Generating retrieval queries from a tool. Retrieval begins with the LLM expanding a tool into multiple retrieval queries. The prompt calls these queries "search entries". The prompt focuses on mathematical objectives, solution procedures, and theory, excluding implementation and machine learning (ML) terms. Figure 3 gives the template.

Extracting supported computations. For each retrieved title and abstract, the model extracts supported objectives, update rules, data structures, and complexity statements. These records supply ingredients for the proposal. Figure 4 gives the extraction prompt.

A recorded attempt on kernel methods illustrates this filtering. The Wikipedia channel returned the full *Kernel (algebra)* article; the extraction skipped it as irrelevant to the optimization task, kept three papers with supported computations, and marked their unresolved estimator and complexity details as `[UNCERTAIN]`.

Deriving a proposal. The proposal-generation prompt receives a formulation, a tool, retrieved mathematical knowledge, and the task interface. It requires the objective and task mapping to be stated explicitly, followed by a solution procedure derived from the formulation before code generation. It also requests a search label for search records. Figure 5 lists the output sections.

The prompt alternates between visual-only and question-dependent selection in a 1:1 schedule. The visual-only mode excludes question inputs; its prompt label is *pure diversity*. The question-

Prompt Template for Theory-Oriented Search-Entry Generation

Role. Assume the role of a researcher specializing in mathematical algorithm design.

Task. Expand the supplied mathematical tool into three to five search entries centered on mathematical theory. [...]

Task requirements.

1. Generate search entries with a **pure mathematical focus**; each search entry should address a different perspective:
   - **Core theory:** the central mathematical concept or theorem.
   - **Classical mapping:** connections to established problems or frameworks in the same mathematical domain.
   - **Algorithmic complexity:** time or space bounds, lower bounds, or hardness results.
2. Each search entry must focus on **mathematical principles** [...] and include at least one theoretical term such as "theorem," "proof," "bound," "guarantee," "algorithm," or "complexity." [...]
3. Do not include library names (e.g., `scipy`, `sklearn`, `pytorch`, or `numpy`), implementation terms (e.g., code, tutorial, or example), or vision/ML terms (e.g., image, token, transformer, or neural). [...]

Figure 3: Theory-focused retrieval query expansion prompt used by the retrieval stage. Bracketed ellipses denote fields omitted to keep the presentation compact.

Prompt Template for Supported-Computation Extraction

Task. For **each** paper relevant to the mathematical tool, extract its **supported computations**—the concrete, implementable operations of the method—using only information stated in its title, abstract, or—for Wikipedia entries—the full definition article.

1. **Source paper:** report the exact paper title.
2. **Objective function:** report the mathematical objective, loss, or optimization target in LaTeX notation when stated or implied by the abstract.
3. **Selection or update operator:** identify the concrete operator (e.g., a greedy rule, sampling scheme, projection, or threshold), including its formula when available.
4. **Data structures:** identify any non-trivial data structure required by the method.
5. **Time complexity:** report stated or directly implied computational complexity.

Evidence constraints.

- Ground every statement exclusively in the supplied titles, abstracts, and definition articles; do not introduce unsupported content.
- If an attribute cannot be determined from the supplied text, mark it as `[UNCERTAIN]`. [...]

Figure 4: Computation-extraction prompt that converts retrieved titles, abstracts, and definition articles into evidence-grounded objectives and computational operations.

dependent mode has the prompt label *text-conditioned*. It disallows standalone text–vision similarity ranking as the entire selection rule. This instruction implements the schedule described in Appendix E.3.

Moving a problem to another mathematical area. These attempts identify high-scoring parent candidates' common problem, then seek a solution procedure in another mathematical area. The prompt requires changes to the objective and core operation. Figure 6 gives the instructions.

Prompt Template for Formulation-Guided Algorithm Synthesis

Mandatory output order. Complete the mathematical argument before producing code. *[This section header is translated from Chinese.]*

1. **Algorithm name and mathematical objective.** Express the objective in mathematical notation [...].
2. **Mathematical formalization and mapping argument.** Complete this stage before writing code.
   (a) Formalize visual-token selection as a mathematical optimization problem using the language of set theory, optimization, or probability. Do not refer to tokens, vision, neural networks, or deep learning in this formalization.
   (b) Explain, in at least two rigorous sentences, why the recalled mathematical theory provides a natural solution to the formalized problem.
   (c) For each core theoretical computation, identify the corresponding PyTorch operation or a concrete approximation; for example, map a distance matrix to `torch.cdist`, sorting to `torch.topk`, and a custom iterative procedure to a loop with a heap.
   (d) Do not directly instantiate any of the following saturated procedures. If the proposal resembles one of them, replace at least one core component with a fundamentally different mechanism: [...]
   - `FPS+Norm weighting`: geometry-only selection without a text query;
   - `RandomProjection+TopK`: projection followed by ranking;
   - `KMeans+distance ranking`: centroid-distance ranking;
   - `CosineSimilarity+TopK`: direct text–vision similarity sorting;
   - `CrossAttentionScore+TopK`: attention-weighted ranking;
   - `QueryNormDotProduct+SpatialPenalty`: a weighted sum of dot-product relevance and spatial regularization.
3. **Forward computation graph and PyTorch pseudocode.** Derive these only after completing Item 2. [...]
4. **Key hyperparameters.** Provide recommended settings. [...]
5. **Numerical stability.** State the required precautions. [...]
6. **Theoretical sources.** Report the sources supporting the formulation. [...]

...

7. **Search label.** Return `PRIMARY_DIRECTION: one of 15 named search labels`. Run accounting additionally includes the catch-all `general` label, giving 16 labels in total.

Fixed-budget constraint. Do not modify `retain_ratio`. The experimental environment fixes this value at {`rr`}, so it cannot serve as an adjustable algorithm hyperparameter. [...]

Figure 5: Proposal-generation prompt used to derive a mathematical objective, justify its problem mapping, and specify the corresponding computation graph before program generation.

F.2 Program Implementation and Repair

Code generation within the task interface. The code-generation prompt defines the editable body of `select_vision_tokens`. It supplies the available tensors, their shapes and spaces, and an example file structure. Its constraints specify the interface and execution requirements checked during validation. Figure 7 gives the main constraints.

Local program repair. The repair prompt receives the proposed algorithm, current file, and execution log. It asks for the smallest correction that resolves the failure and preserves the solution procedure. Figure 8 gives the instructions.

Known error patterns add predefined prompt repair hints, including half-precision linear-algebra calls and uniform-sampling fallbacks. The hints identify recurring faults and guide correction.

Prompt Template for Cross-Paradigm Formulation

Explored solution families. The listed families may not be copied or fine-tuned. [...]

Mechanism constraints.

1. Do not reuse the parent algorithms' specific implementation mechanisms. For example, if a parent uses a distance matrix followed by greedy selection, the new algorithm must employ a different core mechanism.
2. Do not construct a hyperparameter variant or a reweighted version of a parent algorithm. [...]

Migration requirements.

1. Select a mathematical domain **not used by any parent algorithm**. [...]
4. The new objective must be **fundamentally different** from all parent objectives; it cannot remain a variant of maximizing a minimum distance or a covering radius. [...]

Figure 6: Cross-domain proposal-generation prompt used to extend the search to a mathematical area distinct from those of the parent algorithms.

Prompt Template for Contract-Constrained Program Compilation

5. **No silent fallback.** Do not wrap the core algorithm in a `try--except Exception` block that falls back to uniform sampling or retaining all tokens. Handle boundary cases with explicit conditions.
6. **No `arange` fallback.** Do not use `torch.arange(0, N, step=...)` or `torch.arange(N)` as a fallback. [...]
7. **Fixed logic.** Implement exactly one fixed algorithm; do not introduce runtime branches that switch among selection strategies. [...]
11. **Dtype discipline.** Perform scoring, similarity, and Gram matrix computations in `float32`. [...]
12. **Grid reality.** The token count N is generally not a perfect square, and `image_grid_thw` may be `None`. Do not assume $HW = N$. [...]
13. **Memory and runtime budget.** Evaluation has a hard wall-clock limit per attempt, and timeouts receive a failure score. Do not materialize three-dimensional intermediates of shape $[N, N, D]$. [...]

Figure 7: Code-generation prompt encoding the task interface, numerical requirements, and resource constraints imposed on generated implementations.

F.3 Direct-Mutation Ablation

Direct-mutation prompt. Figure 9 gives the generation prompt with Markdown rendered in LaTeX. The placeholder {`best_block`} supplies the current best code and score. The placeholder {`recent_block`} supplies recent outcomes. The complete framework also performs the retrieval and mathematical derivation described in Section 4.

F.4 Direct-Mutation Examples

Table 32 samples eight attempts from the direct-mutation run. The displayed names are assigned by the generator. We classify programs by their implemented selection steps. These programs use question-dependent scores followed by greedy selection. Scores fluctuate, and the best result reaches 39.9%. The examples show repeated refinement of this broad pattern.

F.5 Discovery Examples

The following historical records show how FPS, ORGS, and Leverage candidates were proposed and evaluated. Excerpts retain their recorded wording, with bracketed ellipses marking omissions; the commentary uses the paper's shared notation and identifies differences from the final selectors.

Prompt Template for Minimal Program Repair

Repair requirements.

1. Read the execution log and identify the exact failing line and root cause.
2. Modify only the minimum code required to correct the error.
3. Check boundary conditions explicitly.
4. Specify the device for every newly created `torch.Tensor`.
5. Return only the complete corrected program in a fenced `python` code block.
6. Implement exactly one fixed algorithm without runtime branching among alternative strategies.

Figure 8: Repair prompt used to localize execution failures and apply the smallest interface-preserving correction.

Prompt Template for Direct Program Mutation

Role. You are an expert algorithm engineer improving a visual-token selection algorithm for a frozen vision-language model (Qwen2.5-VL).

Task. The selector receives visual-token embeddings and must retain exactly `target_vision_tokens` of the N tokens to maximize downstream VQA accuracy. It operates as a plug-and-play module in the forward pass, without fine-tuning, using pure Python and PyTorch with at most $O(N^2)$ complexity.

Available inputs.

- `image_embeds`: projector output of shape $[N, 3584]$;
- `image_features`: CLIP-space features of shape $[N, 1280]$;
- `question`: a string;
- `text_token_embedding`: tensor of shape $[m, 3584]$;
- `image_grid_thw`: grid metadata, which may be `None`;
- `target_vision_tokens`: the integer target budget.

The selector must return `selected_indices`, a `LongTensor` containing exactly K unique indices.

Current best algorithm. {`best_block`}

Recent results. {`recent_block`}

Generation instruction. Propose a modified or new algorithm that is likely to improve on the current best. The scoring rule, selection procedure, feature space, and initialization may all be changed.

Output format.

1. algorithm name and one-line description;
2. concise objective or scoring rule;
3. complete, self-contained PyTorch implementation of the selection block, including numerical-stability handling;
4. two to three sentences explaining the expected improvement.

Figure 9: Direct-mutation prompt used by the ablation control. This compact interface conditions generation on the current program and recent evaluation outcomes.

F.5.1 Coverage Selection through Variance-Scaled Distances

The attempt-441 record contains the retrieved material, coverage proposal, generated code, and fixed-resolution discovery score of 52.1%. This candidate already contains the final FPS core mechanism: variance scaling followed by farthest-point updates. It uses medoid initialization; the final evaluation implementation uses the cheaper fixed-seed auxiliary initialization and explicitly excludes previously selected indices.

Table 32: Sampled attempts of the direct-mutation ablation initialized from the coverage selector. Names are the generator's own; the selection templates span coverage/facility-location, MMR, DPP, and farthest-point selection variants.

Attempt	Self-assigned name	Acc. (%)
2	QC-MMR: question-coverage weighted greedy MMR with CLIP diversity	14.30
11	Q-MMR: entropy-weighted question-specific MMR with spatial coverage	31.40
22	Q-ViCo: query-weighted visual semantic facility location	39.90
26	SQFPS: spatial query-aware farthest-point selection	32.40
57	QA-DPP: query-aware DPP over a multimodal similarity kernel	29.30
70	DC-QaFL: dual-coverage query-attentive facility location	38.10
85	QAMC: query-anchored multimodal coverage with fused similarity	39.40
143	QW-DSFL: query-weighted dual-space facility location	39.50

Search state. The recorded tool name is *"Grassmannian Lyapunov exponent information geometry divergence minimization"*. The search label is `geometry`. Selection is visual-only (recorded as *pure diversity*). Retrieval returns material on divergences, including Bregman divergences in Kronecker-factored optimization and Bregman bias–variance decompositions.

Formulation. The generator names the proposal *Adversarially Robust Bregman Coverage* (ARBC). Its implemented criterion is coverage under a quadratic divergence. Using the paper's row-wise feature notation, the proposed objective takes the following form:

$$S^* = \operatorname*{arg\,min}_{S\subseteq[N],\,|S|=K} \max_{i\in[N]} \min_{j\in S} D_B(x_i, x_j), \tag{17}$$

where $x_i = X_i^\top$. The proposal defines $D_B(x, y) = \frac{1}{2}(x-y)^\top \Sigma^{-1}(x-y)$ with diagonal empirical covariance $\Sigma = \mathrm{diag}(\hat{\sigma}_1^2, \ldots, \hat{\sigma}_d^2)$. The unregularized expression requires all coordinate variances to be positive. The generated code regularizes this covariance as $\Sigma_\epsilon = \Sigma + \epsilon I_d$ with $\epsilon = 10^{-8}$, yielding a well-defined variance-scaled squared distance even when a coordinate has zero variance. The constant factor $1/2$ preserves the selection ranking. The name records the generator's proposal; the distance specifies the evaluated computation.

The proposal maps this criterion to tensor operations. It computes scaled pairwise distances with `torch.cdist(x*sqrt(prec), y*sqrt(prec)).square()`. The record follows the eight-part format in Figure 5. It includes the objective, input mapping, pseudocode, numerical settings, sources, and `PRIMARY_DIRECTION: geometry`. The retrieved material and code are stored with it.

Solution procedure and outcome. The candidate starts from the medoid, the vector with the smallest total squared distance to the set, and repeatedly adds the farthest vector from its nearest selected representative. The code compiles on the first attempt and passes the checks then in use. Its recorded discovery score is 52.1%. This combination appears once among the $2{,}011$ completion records. Appendix C.2 diagnoses which parts of this discovered procedure are essential and which initialization choices have only small effects.

The generated selection block. The code below reproduces the complete generated selection block from the run record. It occupies the marked region defined in Appendix A.2.

```
eps = 1e-8
N = image_features.shape[0]
X = image_features.float()  # [N, d_feat]

# Mahalanobis scaling: diagonal covariance
var = X.var(dim=0, unbiased=False) + eps      # [d_feat]
inv_std = 1.0 / var.sqrt()                    # [d_feat]
X_scaled = X * inv_std               # Euclidean == Mahalanobis

# Medoid init: smallest sum of squared distances
```

```
dist_full = torch.cdist(X_scaled, X_scaled, p=2).square()
total_dists = dist_full.sum(dim=1)
c0 = total_dists.argmin().item()
min_dist = dist_full[c0]

# Greedy farthest-point selection
k = target_vision_tokens
selected_indices = torch.empty(k, dtype=torch.long,
                               device=image_embeds.device)
selected_indices[0] = c0
for i in range(1, k):
    new = min_dist.argmax()
    selected_indices[i] = new
    diff_new = X_scaled - X_scaled[new]
    d_new = (diff_new * diff_new).sum(dim=1)
    min_dist = torch.min(min_dist, d_new)

selected_indices = selected_indices.sort()[0]
```

This historical block selects each index from the full distance vector, allowing repeated indices when distances vanish. The final evaluation implementation applies the exclusion rule in Equation (10) to guarantee exactly K distinct indices. It also adopts fixed-seed initialization, which yields a development mean comparable to the discovered medoid initialization and reduces initialization cost. The final procedure combines this initialization with the discovered variance scaling and farthest-point update.

F.5.2 Residual and Information-Based Selection

The following excerpts show how the two families develop their selection and update rules. ORGS uses orthogonal directions to represent the feature set. Leverage preserves visual information through ridge-leverage scoring and sequential inverse-matrix updates. We state their objectives in the common row-wise notation and retain selected computational passages from the records.

ORGS. The recorded tool name is *"Markov chain spectral gap simplicial complex centroid selection optimization bound"*. The selected reconstruction formulation asks that the full set remain reconstructible from the selected vectors, and the proposal connects ORGS token selection to feature reconstruction. For standardized features Z, its implemented loss is $\|Z - Z\Pi_S\|_F^2$, where $\Pi_S \in \mathbb{R}^{d\times d}$ projects onto the span of the selected rows. Projection acts on the right for row-wise features. The proposal specifies these computational steps:

```
(c) Computational Steps and PyTorch mapping
1. Data standardisation -- torch.std_mean along feature dimension,
   then standardise to avoid scale biases.
2. Initialise basis -- empty list Q for orthonormal columns; S = [].
3. For t = 1 to k:
   a. Residuals for all points: R = X - (X @ Q[:, :t-1]) @ Q[:, :t-1].T
   b. Squared norms: scores = torch.norm(R, p=2, dim=1) ** 2.
   c. Select index with maximal score: i = torch.argmax(scores).
   d. Update basis: q = R[i] / ||R[i]||; orthogonalise against
      previous basis, normalise; store as new column of Q.
4. Output the selected indices S. [...]

8. Primary Direction Declaration
PRIMARY_DIRECTION: geometry
```

The excerpt's phrase "along feature dimension" is imprecise: the implementation computes the mean and sample standard deviation over the token dimension (`dim=0`) for each feature coordinate. The excerpt uses X for the features processed by the residual loop. In the final notation, these standardized features are Z. The largest-residual pivot is a heuristic for choosing a new direction. Once that unit direction v is selected, projecting it out of the residual matrix decreases the full reconstruction loss by $\|Rv\|_2^2$. Equation (12) defines the objective, and Equation (14) defines the

update. The attempt-1316 candidate scores 49.1%; a later variant that recomputes the projection directly from the selected features reaches 50.7% at attempt 1582.

Leverage (attempt 1198, score 49.2%). The candidate, named *Online Ridge Information Maximization* (ORIM), is motivated by an information gain formulation that preserves reconstructive information while limiting redundancy. It uses raw visual features, initializes $P = \lambda^{-1} I_d$, and repeatedly selects the unselected token maximizing $x_i^\top P x_i$. The following excerpt records its scoring and inverse-matrix update:

```
Px = X_avail @ P  # [n_avail, d]
scores = (Px * X_avail).sum(dim=1)  # [n_avail]
[...]
best_local = torch.argmax(scores)
best_global = avail_idx[best_local].item()
[...]
x_sel = X[best_global:best_global+1]  # [1, d]
Px_sel = P @ x_sel.T  # [d, 1]
denom = 1.0 + (x_sel @ Px_sel).squeeze()
P = P - (Px_sel @ Px_sel.T) / (denom + eps)
```

The update discounts represented directions before the next selection. The candidate uses $\lambda = 10^{-3}$ and $\epsilon = 10^{-8}$, restores matrix symmetry, and adds diagonal jitter after each update. The final selector retains this visual-only procedure and returns the K selected indices in their original order.

F.6 A Worked Formulation-to-Code Example

Using the historical FPS candidate above, we derive the variance-scaled distance and its incremental coverage update. Table 33 maps the medoid-based candidate to tensor operations; the accompanying text identifies the final selector's initialization change.

From a quadratic function to a token distance. For one image, fix $W = \mathrm{diag}((\sigma_1^2 + \epsilon)^{-1}, \ldots, (\sigma_d^2 + \epsilon)^{-1})$ and define the convex function $\phi(x) = \frac{1}{2} x^\top W x$. Its Bregman divergence is

$$\begin{aligned} D_\phi(x, y) &= \phi(x) - \phi(y) - \nabla\phi(y)^\top (x - y) \\ &= \tfrac{1}{2}(x - y)^\top W (x - y) = \tfrac{1}{2}\|W^{1/2}x - W^{1/2}y\|_2^2. \end{aligned} \tag{18}$$

Multiplication by $W^{1/2}$ is coordinate-wise division by the stabilized standard deviation. The factor $1/2$ leaves the medoid and farthest-point choices unchanged in exact arithmetic.

From the selected set to an incremental update. For a current set S, write $\delta_i(S) = \min_{j \in S} D(i, j)$. Adding a new representative c gives

$$\delta_i(S \cup \{c\}) = \min\{\delta_i(S), D(i, c)\}.$$

Each token compares its previous best representative with the new one. The maximum of these distances can only decrease or stay unchanged as the selected set grows. FPS chooses c with the largest current distance, giving a greedy rule that directly targets the current coverage gap.

From the update to tensors. The feature transformation in Equation (6) computes population variances, then multiplies coordinates by inverse standard deviations. The discovered candidate computes a pairwise distance matrix to find the medoid in Equation (9). The final selector initializes from a fixed-seed index and computes one distance vector. Thereafter, Equation (10) becomes a maximum over unselected indices, a vector of distances to the new token, and an elementwise minimum.

For the coverage update on a one-dimensional token set at $0, 1, 4, 5$, starting at 0 gives squared distances $0, 1, 16, 25$. FPS selects 5, and the nearest-representative squared distances become $0, 1, 1, 0$. These values are expressed in the original coordinates. Variance scaling multiplies every squared distance by the same positive factor and preserves both choices and nearest-representative assignments. The example fixes the starting point at 0 to illustrate the coverage update. The evaluated initialization procedures are specified separately in the algorithm description.

Table 33: Mapping the coverage objective to PyTorch operations. The discovered candidate initializes from a pairwise distance matrix. The final selector initializes from an index chosen with a fixed random seed.

Mathematical step	Tensor computation
Compute in the selection feature space	`X = image_features.float()`
Estimate coordinate variances	`var = X.var(dim=0, unbiased=False) + 1e-8`
Apply the diagonal scaling	`Z = X * var.rsqrt()`
Choose the medoid initialization	Compute `torch.cdist(Z, Z).square()`, sum each row, and take `argmin`
Find the least represented unselected point	Maximize the maintained length-N distance vector over unselected indices
Compute distance to the new point c	`((Z - Z[c]) ** 2).sum(dim=1)`
Update distance to the selected set	Apply `torch.minimum` to the old and new distance vectors
Return the selected subset	Sort the K indices as a `torch.long` tensor on the input device

Cost and simplification. The recorded implementation finds its initial medoid by constructing a full pairwise distance matrix, taking $O(N^2d)$ time and $O(N^2)$ matrix storage. The final selector draws the first index with a fixed seed and computes its distances to all tokens in one pass. Subsequent selection costs $O(Nd)$ per token, giving $O(KNd)$ total time and $O(Nd)$ memory, including scaled features. Appendix C.2 compares these implemented initializations.