SKILL-MAP-ELITES: OPEN-ENDED EVOLUTION OF AGENT SKILLS THROUGH BEHAVIORAL NICHES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reusable skills adapt frozen large language model agents by encoding task procedures, tool-use strategies, and recovery policies as inspectable multi-file artifacts. Existing skill-evolution approaches select candidates primarily based on aggregate validation performance or predefined objectives, discarding behaviorally distinct stepping stones whose value emerges on unseen tasks. We formulate skill evolution as a quality-diversity problem and introduce *Skill-MAP-Elites*, which preserves high-performing skills across execution-grounded behavioral niches. The framework executes each candidate and characterizes its behavior by failure-type coverage, tool-call structure, task-stage coverage, execution cost, robustness, and cross-model gains. It assigns candidates to behavioral niches and selects local elites, separating archive coverage from within-niche optimization. Behavior-level mutations modify procedural structure, recovery strategies, postconditions, parameterization, and abstention policies. Uncertainty-aware admission and periodic reevaluation reduce incorrect elite assignments under sparse and variable rollouts. We evaluate the framework under matched execution budgets on expertise-intensive agent tasks, including transfer to unseen domains, execution models, and agent harnesses, and robustness to schema changes, tool noise, and token-budget perturbations. We compare Skill-MAP-Elites with iterative refinement, sampling-based search, Pareto selection, random archives, and novelty-only search. Descriptor and mutation ablations evaluate whether execution-grounded niches preserve complementary procedural capabilities beyond textual variation and unrestricted candidate retention.

1 INTRODUCTION

Large language model (LLM) agents increasingly solve multi-step tasks by interacting with files, tools, execution environments, and external verifiers. Their performance depends on the underlying model and the procedures used to gather evidence, invoke tools, handle failures, and validate outputs. Agent skills encode these procedures as reusable packages of instructions, scripts, templates, and auxiliary resources. They adapt frozen agents without parameter updates and remain inspectable and portable across tasks and harnesses. SkillsBench shows that curated skills improve performance on diverse expertise-intensive tasks, with gains varying substantially across models and harnesses (Li et al., 2026).

Effective skill use requires methods for discovering and improving skills. Existing approaches typically apply feedback-driven edits and retain candidates based on validation performance. EvoSkill analyzes failed executions, revises structured skill folders, and uses a Pareto frontier to select agent programs that improve held-out performance (Alzubi et al., 2026). The Darwin Gödel Machine maintains an expanding archive of coding agents and branches from previous variants, preserving multiple self-modification paths (Zhang et al., 2025). At the parameter level, CycleQD alternates task-specific quality measures and applies model merging and parameter-space mutation to produce capable LLM agents across domains (Kuroki et al., 2025). These studies establish evolutionary search as an approach to agent improvement, but they do not directly preserve meaningful diversity among reusable, multi-file skill artifacts.

This limitation creates selection and representation bottlenecks. Single refinement chains and globally ranked populations favor immediate validation gains, while compact Pareto frontiers preserve only

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

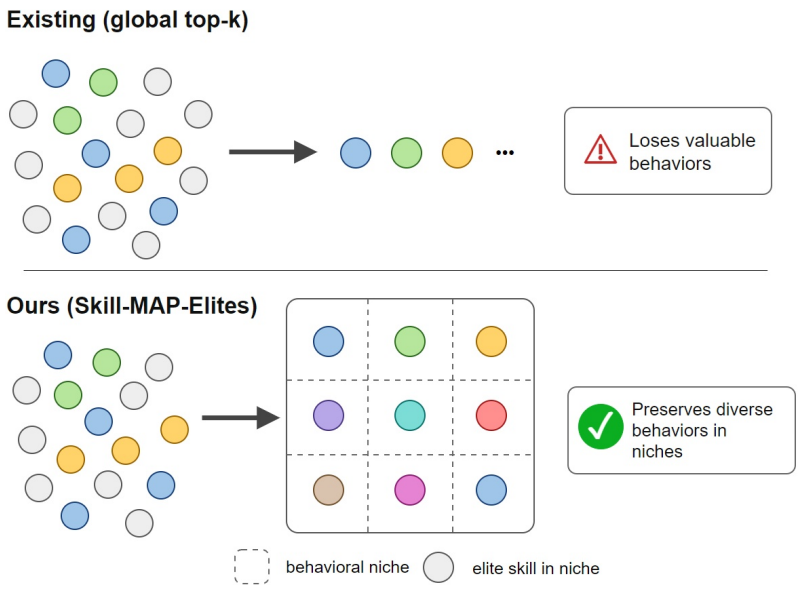

Figure 1: Overview of the Skill-MAP-Elites framework.

the variation represented by predefined objectives. Under limited execution budgets, candidates with modest aggregate scores can encode recovery policies, tool-use patterns, task-stage specializations, or robustness properties that support subsequent mutations or combinations. Eliminating these candidates can remove behavioral stepping stones before their value appears on unseen tasks. Textual or structural differences provide an inadequate measure of behavioral diversity: dissimilar skills can induce nearly identical executions, while small procedural edits can substantially change recovery behavior or tool use. An effective archive must define diversity through measurable agent behavior and perform selection locally within behavioral niches.

Quality-diversity (QD) search seeks high-quality solutions that exhibit distinct behaviors while avoiding selection based solely on a global optimum (Pugh et al., 2016). MAP-Elites partitions a user-defined behavior space into niches and retains the highest-quality solution in each niche, separating behavioral coverage from within-niche optimization (Mouret & Clune, 2015). Applying this principle to agent skills presents three challenges. Skills are structured, composable artifacts whose behavior emerges when a model executes them within a harness. Behavior spaces must capture execution-relevant properties rather than textual differences, mutations must produce valid procedural modifications across files, and archive updates must account for uncertainty from sparse or variable rollouts.

We introduce *Skill-MAP-Elites*, a framework for open-ended skill evolution through behavioral niches. Given initial skills and a task distribution, the framework executes each candidate and maps it to a descriptor that captures failure-type coverage, tool-call structure, task-stage coverage, execution cost, robustness, and cross-model gains. It assigns candidates to niches defined by MAP-Elites or centroidal Voronoi tessellation MAP-Elites and retains the highest-quality candidate in each occupied niche. The search applies behavior-level mutations that add postconditions, replace recovery strategies, split or merge modules, parameterize hard-coded procedures, or revise abstention policies. Each candidate returns to the archive according to its observed behavior and quality. Uncertainty-aware comparisons and periodic reevaluation reduce false elite assignments under limited rollouts. These mechanisms preserve complementary behaviors while improving quality within each niche, distinguishing the framework from selection based solely on aggregate validation scores or textual novelty. Figure 1 illustrates this behavior-grounded evolution and archive-update process.

Under matched execution budgets, we evolve skills on a subset of SkillsBench domains and evaluate them on unseen domains. We compare Skill-MAP-Elites with iterative refinement, best-of-N search, EvoSkill-style Pareto selection, random archives, and novelty-only search. In addition to in-domain pass rate, we measure QD score, archive coverage, effective behavioral mutation rate, worst-model

performance, and execution cost. Transfer experiments vary the execution models and agent harnesses, while robustness tests introduce schema renaming, tool noise, and token-budget perturbations. The ablations replace behavioral descriptors with text embeddings, remove novelty preservation, retain only globally top-k candidates, disable module merging, and vary descriptor compositions. This protocol evaluates whether behavioral niches preserve useful stepping stones and improve transfer beyond the gains produced by additional sampling or unconstrained candidate retention.

Our contributions are:

- We formulate self-evolving agent skills as a QD problem that jointly optimizes task performance and preserves distinct execution behaviors, addressing the selection bottleneck in single-chain, top-k, and Pareto-only refinement.
- We introduce Skill-MAP-Elites, which maps multi-file skills into execution-grounded behavioral niches and selects elites locally to retain complementary procedural capabilities.
- We design behavior-level mutations, uncertainty-aware archive admission, and periodic reevaluation for compositional skill evolution under noisy, budget-limited evaluation.
- We establish a budget-matched evaluation protocol covering unseen tasks, models, harnesses, and environment perturbations, with ablations that separate behavioral diversity from textual variation and unrestricted archive growth.

2 Related Work

2.1 Procedural Adaptation and Skill Acquisition for LLM Agents

LLM agents combine model inference with tools, external state, and iterative interaction. ReAct interleaves reasoning, actions, and observations(Yao et al., 2022); MRKL routes requests among language, symbolic, and retrieval modules(Karpas et al., 2022); and Toolformer learns when to invoke fixed APIs(Schick et al., 2023). Gorilla retrieves API documentation to reduce hallucination(Patil et al., 2023), whereas ToolLLM learns multi-tool trajectories over real-world APIs(Qin et al., 2023).

Experience-based methods adapt without updating model weights. Reflexion reuses verbal feedback(Shinn et al., 2023), Self-Refine iteratively revises outputs(Madaan et al., 2023), and CRITIC grounds correction in external tools(Gou et al., 2023). ExpeL retrieves insights and demonstrations distilled from trajectories(Zhao et al., 2023), while Voyager maintains and repairs an executable skill library(Wang et al., 2023).

Recent work modifies persistent interfaces and programs. AgentOptimizer revises callable functions and descriptions from execution feedback(Zhang et al., 2024), and automated tool construction synthesizes and tests interfaces from papers and repositories(Wölflein et al., 2025). SkillsBench evaluates reusable multi-file skill packages(Li et al., 2026), while EvoSkill diagnoses failures, modifies skill folders, and selects candidates through held-out evaluation and Pareto criteria(Alzubi et al., 2026).

2.2 Automated Optimization of Prompts and Agent Programs

Automatic Prompt Engineer generates and ranks instruction candidates(Zhou et al., 2022), whereas Optimization by PROmpting proposes improvements from optimization histories and objective values(Yang et al., 2023). Population methods include EvoPrompt's mutation, crossover, and selection(Guo et al., 2023) and Promptbreeder's co-evolution of task and mutation prompts(Fernando et al., 2023). TextGrad uses natural-language critiques as backward signals for textual variables(Yuksekgonul et al., 2024).

DSPy compiles modular pipelines by optimizing instructions and demonstrations(Khattab et al., 2023). Automated Design of Agentic Systems generates agent architectures as code from archived designs(Hu et al., 2024), and the Darwin Gödel Machine evaluates self-modifying coding-agent branches(Zhang et al., 2025). CycleQD instead evolves model parameters through task-specific quality criteria, merging, mutation, and quality-diversity selection(Kuroki et al., 2025).

2.3 Quality-Diversity and Open-Ended Search

Novelty Search selects behaviorally distinct solutions rather than optimizing only task objectives(Lehman & Stanley, 2011a). Novelty Search with Local Competition additionally rewards quality among similar behaviors(Lehman & Stanley, 2011b), motivating quality-diversity algorithms that seek high-quality behavioral variation(Pugh et al., 2016).

MAP-Elites discretizes behavior space and stores the best solution per cell(Mouret & Clune, 2015), while CVT-MAP-Elites uses Voronoi niches for higher-dimensional spaces(Vassiliades et al., 2016). CMA-ME adapts covariance-based emitters(Fontaine et al., 2019), MAP-Elites via Evolution Strategies estimates quality and novelty gradients(Schlechta, 2019), and PGA-MAP-Elites combines policy gradients with genetic variation(Kuo et al., 2020).

AURORA learns behavioral descriptors from trajectories(Vassiliades & Mouret, 2018), and data-driven encodings alternate representation learning with archive reconstruction(Cully, 2019). QDax supplies parallel implementations for large-scale optimization(Chalumeau et al., 2023). Learned descriptors reduce manual design but may drift, whereas handcrafted descriptors remain interpretable yet can omit unanticipated behaviors.

Under stochastic evaluation, adaptive-sampling and drifting-elite methods repeat evaluations and correct optimistic replacements(Justesen et al., 2019). Uncertain quality-diversity distinguishes expected performance from reproducibility and studies noisy archive updates(Flageat & Cully, 2023). POET diversifies paired environments and agents through transfer and minimal criteria(Wang et al., 2019).

3 Method

We introduce Skill-MAP-Elites, a quality-diversity framework that evolves reusable agent skills and preserves candidates with distinct behavioral strengths. A skill $s \in \mathcal{S}$ is a versioned, multi-file artifact containing reusable instructions, executable utilities, recovery procedures, and interface metadata. Given a task $x \sim \mathcal{D}$, execution model $m \in \mathcal{M}$, and agent harness $h \in \mathcal{H}$, the skill induces a stochastic trajectory $\tau \sim p(\tau \mid x, s, m, h)$, which contains agent states, tool invocations, observations, resource usage, termination status, and the final answer. An evaluator assigns a score $r(x, \tau) \in [0, 1]$.

3.1 Problem Formulation and Framework

We evolve skills on $\mathcal{D}_{\text{evo}}$ under fixed model and harness distributions. The expected quality of a skill is

$$q(s) = \mathbb{E}_{\substack{x\sim\mathcal{D}_{\text{evo}},m\sim\pi_{\mathcal{M}},h\sim\pi_{\mathcal{H}} \\ \tau\sim p(\cdot|x,s,m,h)}} \left[r(x,\tau) - \lambda_c \widetilde{c}(\tau) \right], \tag{1}$$

where $\widetilde{c}(\tau) \in [0, 1]$ is the normalized execution cost and $\lambda_c \geq 0$ is its weight. Skill-MAP-Elites maintains an archive $\mathcal{A}$ that stores a high-quality skill for each reachable behavioral niche, beyond the single maximizer $\arg\max_s q(s)$.

The framework starts from validated seed skills. At each iteration, it selects parents from occupied cells, applies a behavior-level edit or merge, validates the resulting artifact, evaluates it on a sampled batch, and computes its behavioral descriptor and confidence-adjusted quality. A candidate fills an empty niche or replaces its incumbent when it achieves sufficient local improvement. Periodic reevaluation corrects estimates affected by stochastic execution and small evaluation batches. The resulting repertoire supports behavior-conditioned deployment. A separate validation objective can also reduce the repertoire to a single skill. Figure 2 summarizes this evolution, evaluation, archive-update, and deployment workflow.

3.2 Behavioral Descriptor and Archive

For an evaluation batch B, let $\mathcal{T}_B(s)$ denote the trajectories generated by s. Textual differences do not necessarily correspond to behavioral differences, so we compute descriptors from execution trajectories:

$$\mathbf{b}(s; B) = \text{Norm}\big([\mathbf{b}^{\text{fail}}; \mathbf{b}^{\text{tool}}; \mathbf{b}^{\text{stage}}; b^{\text{cost}}; \mathbf{b}^{\text{rob}}; \mathbf{b}^{\text{xmodel}}]\big) \in [0, 1]^d. \tag{2}$$

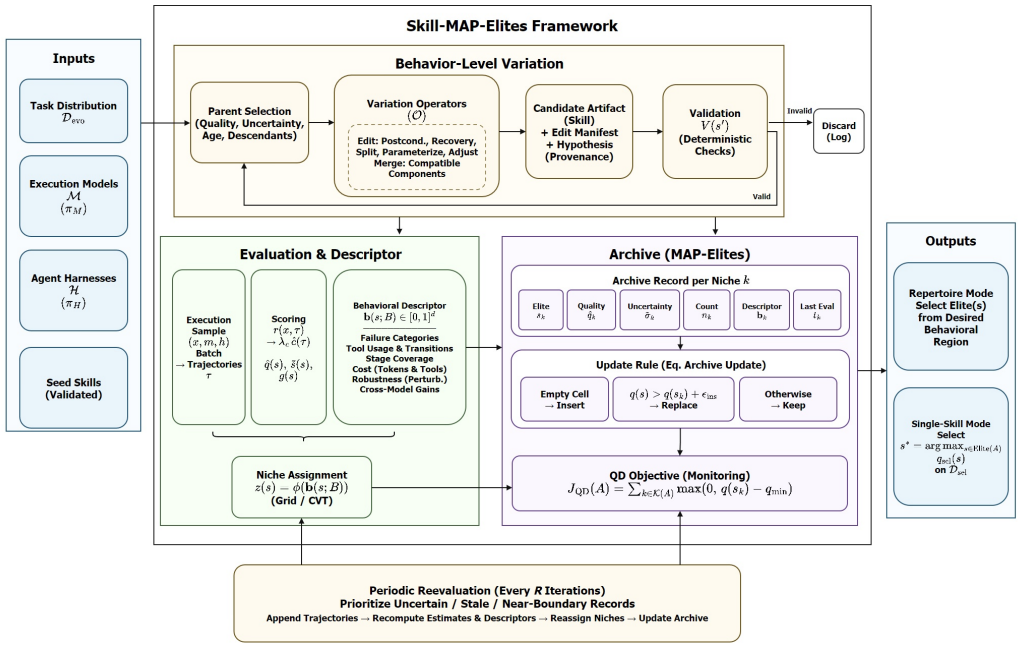

Figure 2: Overview of the Skill-MAP-Elites framework.

These components measure resolved failure categories, tool-family frequencies and transitions, task-stage coverage, normalized token and tool costs, stability under controlled perturbations, and relative gains across execution models. All candidates use a fixed feature schema and coordinate-wise normalization derived from seed runs or predefined ranges. A fixed, candidate-independent rule imputes missing components. The Appendix provides the component definitions and perturbation settings.

A fixed niche map $\phi : [0,1]^d \to \{1, \dots, K\}$ uses a regular grid for low-dimensional descriptors and centroidal Voronoi tessellation for higher-dimensional descriptors:

$$z(s) = \phi(\mathbf{b}(s;B)) = \arg\min_{k \in \{1,\dots,K\}} \|\mathbf{b}(s;B) - \boldsymbol{\mu}_k\|_2, \tag{3}$$

where the centroids $\boldsymbol{\mu}_k$ and normalization ranges remain fixed throughout each run.

Each occupied cell stores $\mathcal{A}[k] = (s_k, \widehat{q}_k, \widehat{\sigma}_k, n_k, \mathbf{b}_k, t_k)$, where the entries represent the elite, empirical quality, uncertainty, trajectory count, descriptor, and most recent reevaluation iteration. Using the confidence-adjusted score $\underline{q}$ defined in Section 3.4, the archive update is

$$\mathcal{A}[k] \leftarrow \begin{cases} \text{Record}(s), & \mathcal{A}[k] = \varnothing, \\ \text{Record}(s), & \underline{q}(s) > \underline{q}(s_k) + \epsilon_{\text{ins}}, \\ \mathcal{A}[k], & \text{otherwise}, \end{cases} \tag{4}$$

where $\epsilon_{\text{ins}} \geq 0$ is the insertion margin. We summarize the archive using

$$J_{\text{QD}}(\mathcal{A}) = \sum_{k \in \mathcal{K}(\mathcal{A})} \max(0, \underline{q}(s_k) - q_{\min}), \tag{5}$$

where $\mathcal{K}(\mathcal{A})$ is the set of occupied cells and $q_{\min}$ is a fixed reference. This metric combines archive coverage with within-niche quality and is not used as a differentiable training loss.

3.3 Behavior-Level Variation

At iteration t, the primary parent is sampled according to

$$p_t(k) = (1-\rho)\frac{1}{|\mathcal{K}(\mathcal{A}_t)|} + \rho\frac{\exp(u_{k,t}/T)}{\sum_{j \in \mathcal{K}(\mathcal{A}_t)} \exp(u_{j,t}/T)}, \tag{6}$$

Method	Doc.	Web	Sheet	Code	Analysis	Workflow	Avg.
Iterative Refinement	58.2±1.3	41.6±1.0	53.7±1.4	46.9±1.2	38.8±1.1	44.1±1.0	47.2
Best-of-N	59.5±1.1	42.8±1.2	55.1±1.0	48.0±1.3	39.4±1.2	44.7±1.1	48.3
Top-k	61.4±0.9	44.0±1.0	56.6±1.2	49.1±1.1	40.2±1.0	45.3±0.9	49.4
Pareto-Only	60.8±1.0	44.9±0.8	57.2±0.9	50.1±1.0	41.0±0.8	46.4±0.7	50.1
Random Archive	59.9±1.4	43.1±1.3	55.8±1.5	48.6±1.4	39.7±1.1	45.0±1.3	48.7
Novelty-Only	57.6±1.5	42.2±1.4	54.0±1.6	47.4±1.5	39.1±1.3	44.5±1.4	47.5
Text-MAP-Elites	60.7±1.1	44.3±1.0	56.4±1.2	49.5±1.1	40.5±0.9	45.9±1.0	49.6
Skill-MAP-Elites	**63.0±0.8**	**46.2±0.7**	**58.9±0.8**	**52.0±0.9**	**42.4±0.7**	**47.5±0.8**	**53.0**

Table 1: Held-out pass rate under an equal candidate-evaluation budget. Results are mean ± standard deviation over three seeds. Abbreviations: Doc. = Document; Avg. = Average.

where $\rho \in [0, 1]$, $T > 0$, and $u_{k,t}$ combines quality, uncertainty, selection age, and successful descendants using fixed coefficients when these terms are included. The uniform component preserves exploration. For a merge operation, the framework samples a secondary parent from another niche, with preference given to descriptor separation.

Given a parent s and its bounded failure traces, an operator $o \in \mathcal{O}$ generates $s' \sim M_o(\cdot \mid s, \mathcal{T}_{\text{fail}}(s))$. The operators insert postconditions, substitute recovery policies, split overloaded modules, merge compatible components, parameterize hard-coded steps, or adjust retry, evidence-request, and abstention conditions. Each proposal contains the modified files, an edit manifest, and a behavioral hypothesis used exclusively for provenance.

Each candidate must satisfy a deterministic predicate $V(s')$ that checks parsing, interface compatibility, dependencies, file-access constraints, and a lightweight execution test. Mutations are restricted to the sampled behavioral concern. Merge operations resolve module-interface conflicts without concatenating artifacts. Invalid candidates are recorded but excluded from the archive. The Appendix provides additional operator details.

3.4 Uncertainty-Aware Evaluation and Output

For $E(s) = \{(x_i, m_i, h_i, \tau_i)\}_{i=1}^{n_s}$, define $y_i(s) = r(x_i, \tau_i) - \lambda_c \widetilde{c}(\tau_i)$. We estimate quality and its standard error as $\widehat{q}(s) = \frac{1}{n_s} \sum_{i=1}^{n_s} y_i(s)$ and

$$\widehat{\mathrm{se}}(s) = \sqrt{\frac{1}{n_s(n_s-1)} \sum_{i=1}^{n_s} (y_i(s) - \widehat{q}(s))^2}. \tag{7}$$

When $n_s = 1$, a prespecified conservative upper bound replaces the standard error. Candidates are ranked using the confidence-adjusted score $\underline{q}(s) = \widehat{q}(s) - \beta_{n_s} \widehat{\mathrm{se}}(s)$, where $\beta_{n_s} \geq 0$ controls the uncertainty penalty. This bounded-budget correction guides archive insertion and parent prioritization; it provides no statistical guarantee.

Every R iterations, the framework prioritizes uncertain, stale, or near-boundary records for reevaluation. It appends new trajectories, recomputes all estimates and descriptors, and removes and reinserts any elite whose niche changes according to Equation (4). The Appendix presents the complete procedure.

In repertoire mode, deployment selects an elite from the desired behavioral region. In single-skill mode, a validation distribution $\mathcal{D}_{\text{sel}}$, disjoint from the evolution tasks, selects $s^\star = \arg\max_{s \in \text{Elite}(\mathcal{A})} q_{\text{sel}}(s)$, where q_{sel} represents the target task, model, and harness. The framework performs no additional evolution on $\mathcal{D}_{\text{sel}}$.

4 Experiments

We evaluate whether execution-grounded quality-diversity search improves skill evolution under a fixed budget, preserves useful behavioral stepping stones, and transfers beyond the evolution setting.

Method	QD-score	Coverage (%)	Effective mutation (%)
Top-k	32.8±1.0	38.6±2.2	31.4±1.8
Pareto-Only	36.9±0.8	49.5±1.9	39.8±1.5
Random Archive	31.7±1.2	63.2±2.5	38.1±2.0
Novelty-Only	29.9±1.4	**74.6±2.1**	**48.7±1.9**
Text-MAP-Elites	35.8±0.9	66.1±1.7	41.6±1.6
Skill-MAP-Elites	**42.7±0.7**	71.8±1.6	47.2±1.3

Table 2: Archive statistics after 512 candidate evaluations.

Method	Unseen domains	New model	New harness	Perturbed environment	Worst model
Iterative Refinement	39.6±1.2	43.0±1.1	40.8±1.3	35.9±1.4	38.4±1.2
Top-k	41.1±1.0	44.7±1.0	42.2±1.1	37.1±1.2	40.0±1.0
Pareto-Only	42.3±0.9	45.6±0.8	43.4±1.0	38.5±1.1	41.2±0.9
Text-MAP-Elites	41.8±1.1	45.0±1.0	43.0±1.2	38.0±1.3	40.7±1.1
Skill-MAP-Elites	**44.6±0.8**	**47.1±0.7**	**45.0±0.9**	**41.3±0.9**	**43.5±0.8**

Table 3: Zero-shot transfer and robustness pass rates without target-test feedback.

We also test whether the observed gains result from behavioral niching, as opposed to archive size, textual diversity, or additional computation.

4.1 PROTOCOL

Tasks and evaluation. We conduct experiments on the six SKILLSBENCH domains: document reasoning, web research, spreadsheet manipulation, repository-level coding, data analysis, and workflow automation. We partition each domain into evolution, validation, and test sets using a 30/20/50 split. Evolution traces and ground-truth feedback are available only for the evolution partition. Validation performance determines archive replacement, and the test partition is evaluated only after evolution terminates. This evaluation protocol further separates optimization feedback from final assessment, reducing the risk that repeated archive updates implicitly overfit reported test outcomes across domains, models, and execution conditions overall. For cross-domain evaluation, we evolve skills on the first four domains and use data analysis and workflow automation as unseen target domains. All methods receive the same task instances, candidate-evaluation budgets, token limits, and timeouts. This controlled design isolates differences in search and retention strategies, while repeated seeds quantify variability arising from stochastic candidate generation. The shared constraints also ensure that improvements cannot be attributed to longer contexts, more executor calls, or preferential access to evaluation feedback.

The primary metric is verifier-based test pass rate. We also report QD-score, defined as the sum of normalized non-negative validation qualities across occupied niches; archive coverage, measured over reachable niches; and effective behavioral mutation rate, defined as the fraction of mutations that change at least one descriptor dimension on at least two probe tasks. We summarize transfer using mean and worst-model pass rates. Search cost includes generated tokens, environment interactions, and wall-clock time.

Unless otherwise specified, each run uses 64 CVT-MAP-Elites niches, eight candidates per iteration, and 512 candidate evaluations. The descriptors capture failure-type coverage, task-stage coverage, tool-call graph structure, execution cost, perturbation robustness, and cross-model benefit. We reevaluate elites after every 32 insertions and use a lower-confidence estimate of validation quality for admission. We report the mean ± standard deviation over three seeds.

Baselines. All baselines use the same skill format, mutation model, executor, initialization, and evaluation budget. ITERATIVE REFINEMENT maintains a single lineage and accepts candidates that improve validation performance. BEST-OF-N samples independent candidates and returns the highest-performing library. TOP-k retains the globally highest-scoring candidates. PARETO-ONLY maintains a quality–cost frontier that approximates EvoSkill-style selection. RANDOM ARCHIVE uses

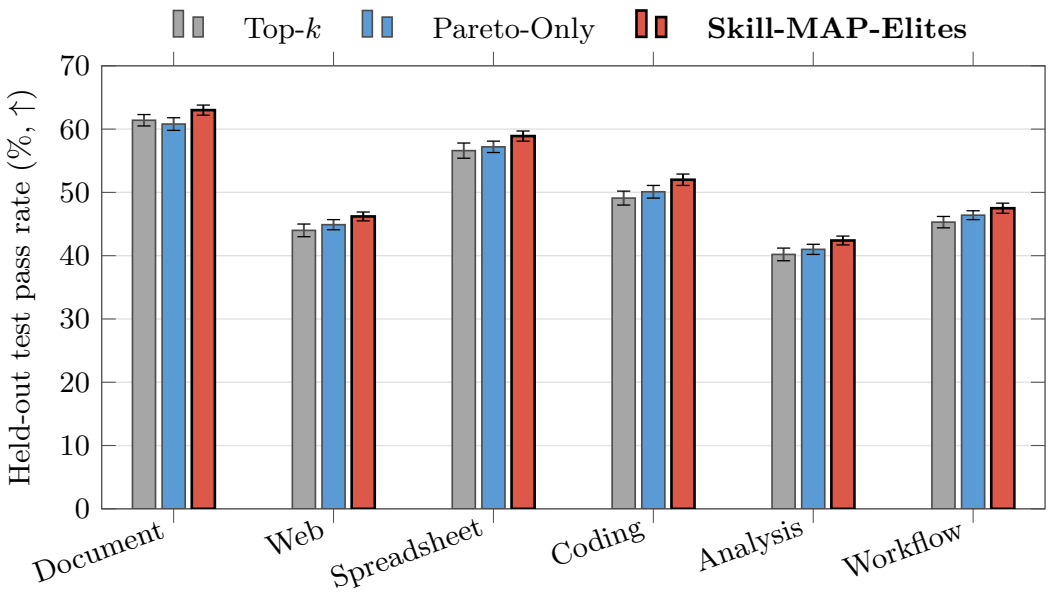

Figure 3: Held-out test pass rates across six SKILLSBENCH domains under a budget of 512 candidate evaluations. Bars and error bars show the mean and standard deviation over three seeds, respectively.

the same archive capacity as our method but assigns candidates to cells uniformly. NOVELTY-ONLY selects candidates based on behavioral novelty without considering quality. TEXT-MAP-ELITES defines niches by clustering skill-text embeddings. Failed executions count toward the evaluation budget. We select hyperparameters from a shared validation grid without test feedback.

4.2 TASK PERFORMANCE AND ARCHIVE QUALITY

Table 1 reports held-out performance under a budget of 512 candidate evaluations, and Figure 3 shows the corresponding comparison across the six domains. SKILL-MAP-ELITES obtains the highest score in all six domains and achieves an average pass rate of 53.0%. The corresponding averages are 50.1% for PARETO-ONLY, 49.4% for TOP-k, and 47.2% for ITERATIVE REFINEMENT, yielding improvements of 2.9, 3.6, and 5.8 percentage points, respectively. Compared with PARETO-ONLY, the per-domain improvements range from 1.1 points on workflow automation to 2.2 points on document reasoning. In coding, separate niches retain skills that address compilation failures, failing tests, and dependency recovery. In document reasoning, the archive preserves distinct procedures for extraction, verification, and numerical reasoning.

Archive capacity and novelty alone do not account for these results. The random archive, which has the same capacity as our archive, achieves a pass rate of 48.7%, and the archive based on textual niches achieves 49.6%. Effective retention therefore depends on niches that capture execution-relevant behavioral distinctions. NOVELTY-ONLY obtains the highest coverage but trails SKILL-MAP-ELITES by 5.5 points because it retains unusual traces without accounting for task success. Global TOP-k selection repeatedly retains variants that repair common failures and removes candidates that could seed less frequent capabilities.

The archive statistics in Table 2 show the same pattern. SKILL-MAP-ELITES achieves the highest QD-score, 42.7, while covering 71.8% of reachable niches. NOVELTY-ONLY covers 74.6% of reachable niches but obtains a QD-score of 29.9, whereas TOP-k reaches 38.6% coverage. Execution-derived niches exceed text-derived niches by 6.9 QD-score points. Skills with different text can produce equivalent tool trajectories, while small procedural changes can alter verification and recovery behavior.

We test the role of low-scoring candidates as stepping stones by removing ancestors whose validation quality was below the archive median at the time of generation. This removal reduces the final pass rate from 53.0% to 50.8%. Moreover, 27% of skills that solve test tasks contain at least one such ancestor. The reductions are largest in coding and document reasoning, where initially weak post-condition checks are later combined with specialized recovery procedures. These results support local niche preservation: candidates rejected by global score selection can contribute to stronger descendants.

4.3 Transfer and Robustness

We evaluate the finalized skill repositories without target-specific modification. For cross-domain transfer, we apply the archive evolved on four source domains to the two withheld domains. Cross-model and cross-harness evaluations replace the executor or runtime while preserving the skill files. In perturbed environments, we rename tool arguments, inject transient failures into 10% of tool calls, and reduce the token budget by 25%.

Skill-MAP-Elites achieves the highest pass rate in every transfer condition. Compared with Pareto-Only, it improves performance by 2.3 points on unseen domains, 1.5 points with a new execution model, 1.6 points with a new harness, 2.8 points under environmental perturbations, and 2.3 points in the worst-model evaluation. Skills in failure and recovery niches often contain checks for absent outputs, schema mismatches, and retriable errors that remain applicable after runtime changes. The gain is smaller under severe token limits because multi-stage verification policies can exceed the available budget even when their recovery logic remains valid.

4.4 Design Factors

We use ablations to identify the contributions of individual design components. Removing novelty-guided parent selection reduces the pass rate from 53.0% to 50.3% and coverage from 71.8% to 52.4%. Replacing behavioral descriptors with text embeddings yields a pass rate of 49.6%. Global top-k retention yields 49.4% pass rate and 38.6% coverage. Disabling module merging reduces the pass rate to 51.1%, indicating that complementary modules provide greater value when mutation can assemble them into a single executable library. Removing uncertainty correction lowers performance to 52.0% and increases seed variation from 0.7 to 1.3 points. Removing the cross-model descriptor yields 51.7% on the source evaluation and reduces worst-model transfer from 43.5% to 41.4%. These ablations attribute the performance gains to execution-grounded diversity, compositional mutation, and robust archive admission.

5 Conclusion

We presented *Skill-MAP-Elites*, a quality-diversity framework that evolves reusable agent skills by mapping execution-grounded behaviors into niches and optimizing candidates locally within each niche. Behavior-level mutations, uncertainty-aware archive admission, and periodic reevaluation preserve complementary procedural capabilities under constrained evaluation budgets. Budget-matched experiments across domains, models, harnesses, and environment perturbations confirm that behavioral niche preservation strengthens skill quality, diversity, transfer, and robustness. Ablation results further establish that execution-grounded descriptors and local elite selection are central to these gains, validating quality-diversity search as an effective foundation for open-ended agent skill evolution.

Limitations

Our behavioral descriptors rely on a fixed, manually specified feature schema and may not capture every execution pattern that is relevant to downstream tasks. The evaluation covers six task domains and a limited set of execution models and agent harnesses, so broader applicability remains to be examined. In addition, periodic reevaluation mitigates stochasticity but introduces extra execution cost, and descriptor estimates may remain noisy when only a small number of rollouts are available for each candidate.

GENERATIVE AI DISCLOSURE

Generative AI tools were used in this work exclusively for the purpose of language polishing and manuscript editing. We leveraged these tools to improve the fluency and clarity of the written text, fix grammatical errors, and ensure consistent use of terminology. The scientific content of this paper—including the skill map-elites algorithm, theoretical analysis, experimental design, and result interpretation—was developed entirely by the human authors without AI involvement. We did not use generative AI to produce any research artifacts, generate data, or assist with technical implementation. All AI-assisted edits were reviewed by the authors. We are fully responsible for the content of this submission.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026. URL https://arxiv.org/abs/2603.02766.

Felix Chalumeau, Bryan Lim, Raphael Boige, Maxime Allard, Luca Grillotti, Manon Flageat, Valentin Macé, Arthur Flajolet, Thomas Pierrot, and Antoine Cully. Qdax: A library for quality-diversity and population-based algorithms with hardware acceleration. *arXiv preprint arXiv:2308.03665*, 2023. URL https://arxiv.org/abs/2308.03665.

Antoine Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. *arXiv preprint arXiv:1905.11874*, 2019. URL https://arxiv.org/abs/1905.11874.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023. URL https://arxiv.org/abs/2309.16797.

Manon Flageat and Antoine Cully. Uncertain quality-diversity: Evaluation methodology and new methods for quality-diversity in uncertain domains. *arXiv preprint arXiv:2302.00463*, 2023. URL https://arxiv.org/abs/2302.00463.

Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. Covariance matrix adaptation for the rapid illumination of behavior space. *arXiv preprint arXiv:1912.02400*, 2019. URL https://arxiv.org/abs/1912.02400.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. URL https://arxiv.org/abs/2305.11738.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023. URL https://arxiv.org/abs/2309.08532.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

Niels Justesen, Sebastian Risi, and Jean-Baptiste Mouret. Map-elites for noisy domains by adaptive sampling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019. URL https://doi.org/10.1145/3319619.3321904.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. URL https://arxiv.org/abs/2205.00445.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023. URL https://arxiv.org/abs/2310.03714.

Chia-Wen Kuo, Chih-Yao Ma, Jia-Bin Huang, and Zsolt Kira. Featmatch: Feature-based augmentation for semi-supervised learning. *arXiv preprint arXiv:2007.08505*, 2020. URL https://arxiv.org/abs/2007.08505.

So Kuroki, Taishi Nakamura, Takuya Akiba, and Yujin Tang. Agent skill acquisition for large language models via cycleqd. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Kvdh12wGC0.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 2011a. URL https://dblp.org/rec/journals/ec/LehmanS11.

Joel Lehman and Kenneth O. Stanley. Evolving a diversity of creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011b. URL https://doi.org/10.1145/2001576.2001606.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026. URL https://arxiv.org/abs/2602.12670.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. URL https://arxiv.org/abs/1504.04909.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. URL https://arxiv.org/abs/2305.15334.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 2016. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2016.00040/full.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Karl Schlechta. A short remark on analogical reasoning. *arXiv preprint arXiv:1910.02453*, 2019. URL https://arxiv.org/abs/1910.02453.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Vassilis Vassiliades and Jean-Baptiste Mouret. Discovering the elite hypervolume by leveraging interspecies correlation. *arXiv preprint arXiv:1804.03906*, 2018. URL https://arxiv.org/abs/1804.03906.

Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm. *arXiv preprint arXiv:1610.05729*, 2016. URL https://arxiv.org/abs/1610.05729.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019. URL https://arxiv.org/abs/1901.01753.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. Llm agents making agent tools. *arXiv preprint arXiv:2502.11705*, 2025. URL https://arxiv.org/abs/2502.11705.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023. URL https://arxiv.org/abs/2309.03409.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. URL https://arxiv.org/abs/2505.22954.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. *arXiv preprint arXiv:2402.11359*, 2024. URL https://arxiv.org/abs/2402.11359.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022. URL https://arxiv.org/abs/2211.01910.

APPENDIX

A COMPLETE ARCHIVE EVOLUTION AND REEVALUATION PROCEDURE

This section defines the state transitions linking behavior-level variation, uncertainty-aware evaluation, niche assignment, archive insertion, and periodic reevaluation. The procedure operates on validated

skill artifacts and applies the same trajectory-derived estimates to candidate insertion and elite reevaluation.

Let

$$E(s) = \{(x_i, m_i, h_i, \tau_i)\}_{i=1}^{n_s}$$

denote the accumulated evaluation records for skill s, where $x_i \sim \mathcal{D}_{\text{evo}}$, $m_i \sim \pi_{\mathcal{M}}$, $h_i \sim \pi_{\mathcal{H}}$, and $\tau_i \sim p(\cdot \mid x_i, s, m_i, h_i)$. For each record, define

$$y_i(s) = r(x_i, \tau_i) - \lambda_c \widetilde{c}(\tau_i).$$

The empirical quality, uncertainty estimate, and confidence-adjusted quality are

$$\widehat{q}(s) = \frac{1}{n_s} \sum_{i=1}^{n_s} y_i(s), \tag{8}$$

$$\widehat{\sigma}(s) := \widehat{\mathrm{se}}(s) = \sqrt{\frac{1}{n_s(n_s-1)} \sum_{i=1}^{n_s} \left(y_i(s) - \widehat{q}(s)\right)^2}, \quad n_s > 1, \tag{9}$$

$$\underline{q}(s) = \widehat{q}(s) - \beta_{n_s} \widehat{\sigma}(s). \tag{10}$$

For $n_s = 1$, the method replaces $\widehat{\sigma}(s)$ with the prespecified conservative upper bound. Each candidate therefore has a defined confidence-adjusted score before comparison with an incumbent.

A.1 Evaluation Record and Candidate State

The behavioral descriptor is computed from the accumulated trajectories in $E(s)$ using the fixed feature schema, coordinate-wise normalization, and missing-component imputation rule:

$$\mathbf{b}(s; E(s)) = \mathrm{Norm}\left([\mathbf{b}^{\text{fail}}; \mathbf{b}^{\text{tool}}; \mathbf{b}^{\text{stage}}; b^{\text{cost}}; \mathbf{b}^{\text{rob}}; \mathbf{b}^{\text{xmodel}}]\right). \tag{11}$$

The corresponding archive index is

$$z(s) = \phi(\mathbf{b}(s; E(s))) = \arg\min_{k \in \{1,\ldots,K\}} \|\mathbf{b}(s; E(s)) - \boldsymbol{\mu}_k\|_2. \tag{12}$$

For a regular-grid archive, ϕ denotes the fixed grid assignment in place of the centroid-based assignment above. The normalization ranges and niche map remain fixed throughout evolution.

At iteration t, define the complete record as

$$\mathrm{Record}_t(s) = (s, \widehat{q}(s), \widehat{\sigma}(s), n_s, \mathbf{b}(s; E(s)), t). \tag{13}$$

A skill competes only within $z(s)$. An empty cell accepts the record. An occupied cell replaces its incumbent $s_{z(s)}$ exactly when

$$\underline{q}(s) > \underline{q}(s_{z(s)}) + \epsilon_{\text{ins}}. \tag{14}$$

The descriptor determines the competition niche, and the confidence-adjusted quality determines survival within that niche.

A.2 Periodic Reevaluation as Removal and Reinsertion

Reevaluation appends observations to the existing evaluation set. Given newly collected records $E_{\text{new}}(s)$, the accumulated state becomes

$$E^+(s) = E(s) \cup E_{\text{new}}(s). \tag{15}$$

The procedure then recomputes all quantities in Equations (8)–(12) from $E^+(s)$. Because the descriptor depends on the observed trajectories, reevaluation can change the assigned niche.

A selected elite is first removed from its current cell. The procedure then treats its updated record as a candidate and inserts it into the niche assigned by the recomputed descriptor. If another elite occupies the destination cell, the procedure applies the margin-based comparison in Equation (14). Removing the previous record before reinsertion ensures that a stochastic descriptor update cannot leave an outdated copy in the original niche.

The procedure excludes invalid proposals before evaluation and archive updates. It applies the same descriptor construction, uncertainty adjustment, niche assignment, and local replacement criterion to valid proposals and reevaluated elites. The resulting archive supports behavior-conditioned deployment and can also serve as input to the separate single-skill selection objective without further evolution.

Algorithm 1 Skill-MAP-Elites Evolution with Periodic Reevaluation

Require: Validated seed skills $\mathcal{S}_0$; evolution distribution $\mathcal{D}_{\text{evo}}$; model and harness distributions $\pi_{\mathcal{M}}$ and $\pi_{\mathcal{H}}$; fixed niche map ϕ; operator set $\mathcal{O}$; validation predicate V; insertion margin ϵ_{ins}; reevaluation period R; prespecified evolution termination condition

Ensure: Archive $\mathcal{A}$ of behaviorally distinct elite skills

Initialize $\mathcal{A}[k] \leftarrow \varnothing$ for all $k \in \{1, \dots, K\}$
for all $s \in \mathcal{S}_0$ **do**
    Collect $E(s)$ by sampling tasks, models, harnesses, and trajectories from their prescribed distributions
    Compute $\widehat{q}(s)$, $\widehat{\sigma}(s)$, $\underline{q}(s)$, $\mathbf{b}(s; E(s))$, and $z(s)$
    if $\mathcal{A}[z(s)] = \varnothing$ **or** $\underline{q}(s) > \underline{q}(s_{z(s)}) + \epsilon_{\text{ins}}$ **then**
        $\mathcal{A}[z(s)] \leftarrow \text{Record}_0(s)$
    end if
end for
Set $t \leftarrow 1$
while the prespecified evolution termination condition is not satisfied **do**
    Let $\mathcal{K}(\mathcal{A})$ be the currently occupied cells
    Sample a primary cell $k \in \mathcal{K}(\mathcal{A})$ according to $p_t(k)$, and set $s \leftarrow s_k$
    Sample an operator $o \in \mathcal{O}$
    if o is a merge operator **then**
        Sample a secondary parent from a different occupied niche, with preference for descriptor separation
        Generate a proposal s' by applying o to the selected parents and their available bounded failure traces
    else
        Generate $s' \sim M_o(\cdot \mid s, \mathcal{T}_{\text{fail}}(s))$
    end if
    if $V(s')$ is true **then**
        Collect an evaluation set $E(s')$ under $\mathcal{D}_{\text{evo}}$, $\pi_{\mathcal{M}}$, and $\pi_{\mathcal{H}}$
        Compute $\widehat{q}(s')$, $\widehat{\sigma}(s')$, $\underline{q}(s')$, $\mathbf{b}(s'; E(s'))$, and $z(s')$
        if $\mathcal{A}[z(s')] = \varnothing$ **or** $\underline{q}(s') > \underline{q}(s_{z(s')}) + \epsilon_{\text{ins}}$ **then**
            $\mathcal{A}[z(s')] \leftarrow \text{Record}_t(s')$
        end if
    end if
    if t is a multiple of R **then**
        Select the prespecified subset of occupied records prioritized by uncertainty, staleness, or proximity to an insertion boundary
        for all selected records containing elite s_k **do**
            Save s_k and its accumulated evaluation set $E(s_k)$
            Remove its current record: $\mathcal{A}[k] \leftarrow \varnothing$
            Collect additional records $E_{\text{new}}(s_k)$ and set $E(s_k) \leftarrow E(s_k) \cup E_{\text{new}}(s_k)$
            Recompute $\widehat{q}(s_k)$, $\widehat{\sigma}(s_k)$, $\underline{q}(s_k)$, $\mathbf{b}(s_k; E(s_k))$, and $k' \leftarrow z(s_k)$
            if $\mathcal{A}[k'] = \varnothing$ **or** $\underline{q}(s_k) > \underline{q}(s_{k'}) + \epsilon_{\text{ins}}$ **then**
                $\mathcal{A}[k'] \leftarrow \text{Record}_t(s_k)$
            end if
        end for
    end if
    $t \leftarrow t + 1$
end whilereturn $\mathcal{A}$

B Environment Details

B.1 Datasets and Task Settings

We evaluate on the six domains of SkillsBench: document reasoning, web research, spreadsheet manipulation, repository-level coding, data analysis, and workflow automation. Document reasoning requires extracting, integrating, and verifying information from heterogeneous files; web research

requires multi-hop evidence collection with source attribution; spreadsheet tasks require formula construction, table editing, and consistency checks; repository-level coding requires modifying existing codebases and passing project tests; data-analysis tasks require transforming data and producing numerically correct outputs; and workflow-automation tasks require coordinating multiple tools under explicit state and dependency constraints.

Each domain is partitioned into evolution, validation, and test subsets using a fixed $30/20/50$ split, stratified by task template and difficulty whenever metadata are available. Evolution tasks expose verifier feedback and bounded failure traces to the mutation procedure. The validation subset is used for archive admission, periodic elite reevaluation, hyperparameter selection, and final single-skill selection. The test subset remains inaccessible until evolution terminates and is evaluated exactly once per completed run. For cross-domain transfer, document reasoning, web research, spreadsheet manipulation, and repository-level coding constitute the source domains, while data analysis and workflow automation are held out as unseen target domains.

Tasks are executed in their native interactive environments rather than converted to a uniform text-only format. Each instance provides a natural-language request, an isolated working directory, the tools permitted by the benchmark, and any required input artifacts. The executor receives a system prompt defining the agent interface, the task request, and the selected multi-file skill package. The skill is presented as reusable procedural guidance and may contain instructions, scripts, templates, interface metadata, and recovery policies. We use a fixed ReAct-style interaction protocol in which the model alternates between internal reasoning, tool calls, and environment observations. No task-specific demonstrations or test-derived examples are included in the prompt.

A task terminates when the agent submits a final response, produces the required artifact, exceeds its token or tool-call budget, reaches the wall-clock timeout, or triggers an unrecoverable execution error. The default limits are $32{,}768$ generated tokens, 64 tool calls, and 20 minutes per trajectory. Repository-level coding tasks receive a 30-minute timeout to accommodate dependency installation and test execution. Skills may request retries or abstain when required evidence is unavailable, but all retries count toward the same per-task budget.

To limit contamination and task leakage, splits are constructed at the task-family level when multiple instances share templates, repositories, source documents, spreadsheets, or workflow schemas. Near-duplicate instructions are removed across splits using normalized character 5-gram overlap and embedding similarity, with pairs above a cosine similarity of 0.92 manually or programmatically grouped into the same partition. Test prompts, verifier outputs, hidden tests, and target artifacts are unavailable to the mutation model and archive-selection procedure. Web-research tasks use frozen query specifications and timestamped verifier targets; retrieved pages are treated only as execution-time observations and are not added to subsequent prompts or skill files.

B.2 Preprocessing and Evaluation

All textual outputs are normalized by Unicode NFKC normalization, conversion of non-breaking spaces, removal of leading and trailing whitespace, and collapsing consecutive whitespace. For short-answer fields, evaluation additionally removes surrounding quotation marks, terminal punctuation, and optional LaTeX wrappers such as `\boxed{}`, `\text{}`, and math-mode delimiters. Numeric answers are parsed after removing thousands separators and currency or percentage symbols when permitted by the task schema. Integers are compared exactly; floating-point values are accepted when

$$|\hat{y} - y| \leq \max(10^{-6}, 10^{-4}|y|).$$

Equivalent fractions, decimal representations, scientific notation, and percentages are converted to a common numeric representation before comparison. Unordered multi-answer outputs are canonicalized as sets after element-wise normalization, while ordered outputs retain their original sequence. When the reference specifies multiple valid answers, a prediction is counted as correct if it matches any accepted canonical form.

Artifact-producing tasks are evaluated by domain-specific executable verifiers. Document tasks check required content, extracted values, structural constraints, and file readability. Spreadsheet tasks load the generated workbook and verify target cells, formulas, data types, sheet structure, and recalculated values. Coding tasks are evaluated using isolated builds, lint or compilation checks when specified, and hidden unit and integration tests. Data-analysis tasks verify the requested output files, numerical

results, schemas, and plot metadata when applicable. Workflow tasks replay the resulting state and check required actions, dependencies, and terminal conditions. Web-research tasks require both a correct answer and supporting evidence that resolves to the cited source. A trajectory receives $r(x, \tau) = 1$ only when all mandatory verifier checks pass; otherwise it receives zero. Timeouts, malformed artifacts, invalid tool calls, and executor failures are scored as failures.

The primary metric is verifier-based pass rate,

$$\text{PassRate} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{I}[\text{verify}(x, \hat{y}) = 1].$$

We additionally report QD-score, reachable-niche coverage, effective behavioral mutation rate, mean and worst-model transfer pass rate, generated tokens, environment interactions, and wall-clock time. QD-score is the sum of non-negative confidence-adjusted validation qualities over occupied niches using the fixed reference quality $q_{\min} = 0$. Coverage is the fraction of the 64 reachable CVT niches occupied after the evaluation budget is exhausted. A mutation is behaviorally effective if it changes at least one discretized descriptor coordinate on at least two fixed probe tasks.

Stochastic executions use deterministic sampling where supported, with temperature 0.2, nucleus parameter $p = 0.95$, and a maximum of one trajectory per task-candidate pair during routine candidate evaluation. Elites selected for reevaluation receive additional trajectories on independently sampled validation tasks. All complete search procedures are repeated with seeds 0, 1, and 2; tables report the arithmetic mean and sample standard deviation across the three runs. Task partitions, CVT centroids, initial skills, candidate budgets, model parameters, and perturbation schedules are held fixed across methods for each seed.

Robustness evaluation applies three controlled perturbations independently and jointly: consistent renaming of tool arguments, transient failure injection into 10% of eligible tool calls, and a 25% reduction in the generation-token budget. Transient failures are sampled from timeout, empty-result, and retriable server-error categories using a fixed per-task random seed. No robustness trajectory or target-domain result is used for archive updates.

We compare against Iterative Refinement, Best-of-N, global Top-k retention, Pareto-Only selection, Random Archive, Novelty-Only search, and Text-MAP-Elites. Every baseline uses the same seed skills, candidate generator, validation tasks, executor, failure traces, and budget of 512 candidate evaluations. Invalid candidates and failed executions consume the budget. Hyperparameters are selected from a shared validation grid without access to test performance.

B.3 Implementation and Hardware

Skills are represented as versioned directories containing a required `SKILL.md` entry point, optional executable utilities, templates, schemas, and auxiliary resources. Before execution, each candidate undergoes deterministic validation comprising UTF-8 decoding, manifest parsing, path-safety checks, dependency resolution, interface-schema validation, static syntax checks for Python and shell utilities, and a lightweight sandboxed smoke test. Candidate processes have no access to other candidates, hidden verifier files, archive metadata, or host-environment credentials.

We use `Qwen3-32B` as the primary frozen execution model and mutation model. The executor invokes the model with task context, tool schemas, and the selected skill package; the mutation process invokes the same checkpoint with bounded failure traces, the parent skill, the sampled variation operator, and a structured edit specification. Mutation output contains the complete modified files, an edit manifest, and a behavioral hypothesis. Model weights remain frozen throughout all experiments, and no gradients, fine-tuning, or parameter merging are used. Using a shared backbone controls for model capability across search methods. Cross-model descriptors and transfer experiments additionally execute candidates with `Llama-3.3-70B-Instruct`; this model is never used to generate mutations. Cross-harness experiments replace the default ReAct executor with an event-driven tool-calling harness while preserving model checkpoints, skill files, task inputs, and verifier logic.

Both checkpoints are hosted locally using `vLLM` with bfloat16 weights, tensor parallelism, prefix caching, and paged attention. Tool execution occurs in isolated Docker containers with network access disabled except for benchmark web tools. The default harness records model messages, tool names and arguments, observations, exit conditions, token counts, latency, and artifact hashes. These traces are the sole inputs to the execution-grounded behavioral descriptors.

The archive contains $K = 64$ fixed CVT-MAP-Elites niches. CVT centroids are generated once from 100,000 uniformly sampled points in the normalized descriptor space and remain unchanged across methods and seeds. Descriptor coordinates summarize failure-type coverage, task-stage coverage, tool-family frequencies and transitions, normalized execution cost, perturbation robustness, and relative cross-model gain. Coordinate-wise normalization ranges are estimated from the validated seed skills and then frozen. Missing descriptor components are imputed with the corresponding seed-set median and accompanied by a missingness indicator.

Each run begins by evaluating the same validated seed skills and proceeds for 64 iterations with eight candidate proposals per iteration, yielding 512 candidate evaluations. Each proposal has a single primary parent. Merge mutations additionally sample one secondary parent from a different niche, weighted by descriptor distance. Parent selection mixes uniform archive sampling with utility-based sampling using $\rho = 0.5$ and softmax temperature $T = 0.2$. Variation operators are sampled uniformly from postcondition insertion, recovery-policy substitution, module splitting, compatible-module merging, hard-coded-step parameterization, and retry or abstention adjustment. A candidate may modify at most three files and introduce at most 200 non-comment lines per proposal.

Candidate quality is computed as

$$\hat{q}(s) = \frac{1}{n_s} \sum_{i=1}^{n_s} \left[r(x_i, \tau_i) - \lambda_c e_c(\tau_i) \right], \qquad \lambda_c = 0.05,$$

where execution cost $e_c \in [0, 1]$ is the mean of normalized generated-token and tool-call costs. Archive admission uses the lower-confidence score

$$q(s) = \hat{q}(s) - \beta_{n_s} \widehat{\mathrm{se}}(s), \qquad \beta_{n_s} = 1.0,$$

and an insertion margin $\epsilon_{\mathrm{ins}} = 0.01$. A candidate fills an empty niche immediately or replaces its incumbent only when its confidence-adjusted score exceeds the incumbent by the insertion margin. Elites are reevaluated after every 32 successful insertions. Reevaluation prioritizes records with the largest uncertainty, records not evaluated during the previous 64 candidate evaluations, and descriptors within 0.05 Euclidean distance of a CVT boundary. New trajectories are appended to the existing record before recomputing quality, uncertainty, and niche assignment.

No separately trained value or reward model is used. Task verifiers provide the success signal, while normalized execution cost supplies the only auxiliary penalty. The mutation model does not score its own proposals. For deployment in single-skill mode, every final elite is evaluated on the disjoint validation subset, and the skill with the highest confidence-adjusted validation pass rate is selected without further mutation.

Experiments are executed on a cluster of nodes equipped with eight NVIDIA H100 80 GB GPUs, two 64-core AMD EPYC processors, and 1 TB of system memory per node. `Qwen3-32B` uses four-way tensor parallelism, while `Llama-3.3-70B-Instruct` uses eight-way tensor parallelism. Candidate and task evaluations are parallelized asynchronously, with at most eight active candidates per search iteration; archive updates are serialized in candidate-index order to ensure deterministic outcomes. The software stack uses Ubuntu 22.04, Python 3.11, PyTorch 2.6, CUDA 12.4, and vLLM 0.8.5. Random seeds control task sampling, CVT initialization, parent and operator selection, decoding, and environment perturbations.

C Additional Experimental Results

C.1 Scaling with Candidate-Evaluation Budget

Table 4 reports performance at different search budgets. The pass rate of Skill-MAP-Elites increases from 47.7 at 128 evaluations to 54.7 at 1,024 evaluations and exceeds the other methods at each evaluated budget. Its pass-rate difference relative to Pareto-Only increases from 1.4 points at 128 evaluations to 3.3 points at 1,024 evaluations. Archive coverage also increases from 39.6% to 80.2% over the same budget range.

Table 4: Held-out pass rate and archive statistics as a function of the candidate-evaluation budget. The upper and lower panels report the lower and higher budgets, respectively. Results are mean ± standard deviation over three seeds. Abbreviations: Eval. = Evaluations; PR = Pass rate; Cov. = Coverage.

	128 Eval.		256 Eval.	
Method	PR	Cov.	PR	Cov.
Iterative Refinement	44.1±1.4	–	45.8±1.2	–
Top-k	45.7±1.2	21.9±1.8	47.8±1.1	30.7±2.0
Pareto-Only	46.3±1.1	28.1±2.1	48.4±0.9	39.6±1.7
Text-MAP-Elites	46.0±1.3	35.4±2.5	48.1±1.1	51.0±2.1
Skill-MAP-Elites	**47.7±1.0**	**39.6±2.2**	**50.4±0.8**	**57.8±1.9**
	512 Eval.		1,024 Eval.	
Method	PR	Cov.	PR	Cov.
Iterative Refinement	47.2±1.0	–	48.0±1.1	–
Top-k	49.4±0.9	38.6±2.2	50.5±1.0	43.8±2.4
Pareto-Only	50.1±0.8	49.5±1.9	51.4±0.8	55.7±1.8
Text-MAP-Elites	49.6±0.9	66.1±1.7	50.8±0.9	72.3±1.9
Skill-MAP-Elites	**53.0±0.7**	**71.8±1.6**	**54.7±0.6**	**80.2±1.5**

Table 5: Effect of archive resolution after 512 candidate evaluations. Coverage is computed with respect to reachable niches at each resolution. Results are mean ± standard deviation over three seeds.

Number of niches	Pass rate	QD-score	Coverage (%)	Evaluations per occupied niche
16	51.2±0.9	11.8±0.4	89.6±2.1	35.8±1.0
32	52.3±0.8	23.7±0.6	81.4±1.8	19.7±0.5
64	**53.0±0.7**	**42.7±0.7**	71.8±1.6	11.2±0.3
128	52.5±0.9	65.9±1.3	57.1±1.9	7.0±0.2
256	51.6±1.1	84.8±2.4	38.4±2.2	5.2±0.3

C.2 Archive Resolution

The number of CVT niches was varied while the evaluation budget remained fixed. As shown in Table 5, the configuration with 64 niches achieves a pass rate of 53.0, a QD-score of 42.7, and 71.8% coverage. Configurations with fewer niches have higher coverage and lower QD-scores, whereas configurations with more niches have lower coverage and fewer evaluations per occupied niche under the fixed budget.

C.3 Descriptor Component Ablations

Table 6 reports ablations of the execution-grounded descriptor. Removing any component reduces the source-domain pass rate, QD-score, or worst-model pass rate relative to the full descriptor. Removing failure coverage or task-stage coverage yields source-domain pass rates of 51.5 and 51.4, respectively, compared with 53.0 for the full descriptor. Removing robustness or cross-model gain yields worst-model pass rates of 41.8 and 41.4, respectively, compared with 43.5 for the full descriptor. Using only tool-call frequencies yields 73.0% coverage, a source-domain pass rate of 49.8, and a worst-model pass rate of 39.7.

C.4 Mutation Operator Analysis

Table 7 reports operator-level validity and archive insertion rates. Procedure parameterization has a validity rate of 96.1%, while module merging has a validity rate of 78.9%. Recovery substitution and postcondition insertion have archive insertion rates of 33.9% and 31.7%, respectively. Among candidates that replace an incumbent, module merging has an average local quality improvement of 2.31.

Table 6: Ablation of behavioral descriptor components. "Source" denotes the six-domain held-out pass rate, and "Worst model" denotes the minimum pass rate across execution models. Results are mean ± standard deviation over three seeds.

Descriptor	Source	QD-score	Coverage (%)	Worst model
Full descriptor	**53.0±0.7**	**42.7±0.7**	71.8±1.6	**43.5±0.8**
Without failure coverage	51.5±0.9	39.1±0.9	66.4±1.9	42.4±0.9
Without tool-call structure	52.1±0.8	40.0±0.8	67.8±1.7	42.7±0.8
Without task-stage coverage	51.4±1.0	38.8±1.0	65.9±2.0	42.2±1.0
Without execution cost	52.4±0.8	40.6±0.7	69.3±1.6	42.8±0.9
Without robustness	52.0±0.9	39.8±0.9	67.1±1.8	41.8±1.0
Without cross-model gain	51.7±0.9	39.5±0.8	68.0±1.7	41.4±0.9
Tool-call frequencies only	49.8±1.2	35.4±1.1	**73.0±2.1**	39.7±1.2

Table 7: Statistics of behavior-level mutation operators accumulated over three runs. Quality improvement is measured among candidates that replace an incumbent. Abbreviations: Prop. = Proposal; Valid. = Validity; Insert. = Archive insertion; Local QI = Local quality improvement.

Operator	Prop. (%)	Valid. (%)	Insert. (%)	Local QI
Postcondition insertion	21.6±1.4	94.8±1.1	31.7±1.8	1.38±0.16
Recovery substitution	20.4±1.2	91.5±1.3	33.9±1.6	1.62±0.18
Module split	13.1±0.9	86.7±1.8	27.4±2.0	1.47±0.21
Module merge	14.8±1.0	78.9±2.3	29.8±1.9	**2.31±0.27**
Procedure parameterization	17.3±1.1	**96.1±0.8**	26.6±1.7	1.19±0.14
Retry or abstention adjustment	12.8±0.8	93.4±1.2	24.9±1.8	1.11±0.17

Table 8 reports the results obtained after removing individual mutation operators. Removing module merging yields a pass rate of 51.1 and a QD-score of 39.2, while removing recovery substitution yields a pass rate of 51.4 and a QD-score of 39.6. Module merging combines procedures preserved in different niches, whereas recovery substitution modifies behavior within failure-specific niches.

C.5 Perturbation Breakdown

Table 9 separates the environmental perturbations used in the aggregate robustness evaluation. Skill-MAP-Elites achieves pass rates of 46.1 under schema renaming, 45.3 under transient tool failures, and 43.8 under token-budget reduction. Under the combined perturbation setting, its pass rate is 41.3 and its relative degradation is 22.1%. All methods achieve lower pass rates under the combined setting than under each individual perturbation.

C.6 Uncertainty-Aware Admission

Table 10 reports results for different elite reevaluation intervals and uncertainty coefficients. Without periodic reevaluation, the false-replacement rate is 21.4%, compared with 8.1% for $\beta = 1.0$ and an interval of 32 insertions. An interval of 16 insertions yields 68.9% final coverage, while an interval of 64 insertions yields a false-replacement rate of 12.9%. The configuration with $\beta = 1.0$ and an interval of 32 insertions achieves a pass rate of 53.0, a QD-score of 42.7, and 71.8% final coverage.

C.7 Search Cost

Table 11 compares resource consumption under the matched candidate-evaluation budget. Skill-MAP-Elites uses 34.2 ± 0.8 million tokens, 45.1 ± 1.0 thousand tool calls, and 17.6 ± 0.6 hours of wall-clock time. Its pass rate per million tokens is 1.55 ± 0.04, compared with 1.51 ± 0.04 for Pareto-Only and 1.48 ± 0.04 for Text-MAP-Elites. Novelty-Only uses 32.4 ± 0.9 million tokens and achieves a pass rate per million tokens of 1.47 ± 0.06.

Table 8: Mutation-operator ablations under 512 candidate evaluations. Results are mean ± standard deviation over three seeds.

Variant	Pass rate	QD-score	Effective mutation (%)
Full operator set	**53.0±0.7**	**42.7±0.7**	**47.2±1.3**
Without postcondition insertion	51.8±0.9	40.2±0.8	44.0±1.5
Without recovery substitution	51.4±0.9	39.6±0.9	42.8±1.6
Without module split	52.2±0.8	40.9±0.8	45.1±1.4
Without module merge	51.1±1.0	39.2±1.0	43.5±1.7
Without parameterization	52.3±0.8	41.0±0.7	44.7±1.4
Without retry or abstention adjustment	52.1±0.9	40.6±0.9	44.3±1.5
Edit-only mutations	50.7±1.1	38.3±1.1	41.9±1.8

Table 9: Pass rate under individual and combined environment perturbations. The combined setting applies schema renaming, a 10% transient tool-failure rate, and a 25% token-budget reduction. Results are mean ± standard deviation over three seeds. Abbreviations: Schema = Schema renaming; Degrad. = Relative degradation.

Method	Schema	Tool noise	Token budget −25%	Combined	Degrad. (%)
Iterative Refinement	40.1±1.3	38.7±1.5	39.4±1.4	35.9±1.4	23.9±2.1
Top-k	41.8±1.2	40.2±1.3	40.7±1.2	37.1±1.2	24.9±1.9
Pareto-Only	43.0±1.0	42.1±1.2	41.5±1.1	38.5±1.1	23.2±1.8
Text-MAP-Elites	42.5±1.2	41.2±1.3	41.0±1.3	38.0±1.3	23.4±2.0
Skill-MAP-Elites	**46.1±0.8**	**45.3±0.9**	**43.8±1.0**	**41.3±0.9**	**22.1±1.5**

C.8 Seed-Level Results

Table 12 reports the individual seed results underlying the Skill-MAP-Elites averages. The average pass rates are 50.9, 51.8, and 52.4 for seeds 1, 2, and 3, respectively. Across the three seeds, the domain-level standard deviations range from 0.6 to 1.0.

D Prompt Templates

D.1 Overview

This appendix records the prompt templates used to execute, evolve, and validate reusable agent skills. Unless otherwise stated, all search methods use the same execution and mutation prompts; they differ only in candidate retention, parent selection, and archive organization. Template fields enclosed in angle brackets are instantiated at runtime. Deterministic verifiers, archive updates, behavioral descriptor computation, and metric aggregation are implemented outside the language model and therefore require no prompts.

D.2 Notation

We use the following variables throughout the templates:

- <TASK>: the natural-language task specification.
- <INPUTS>: task-specific inputs, including files, URLs, repository paths, tables, or structured records.
- <AVAILABLE_TOOLS>: tool names, signatures, and usage constraints exposed by the agent harness.
- <SKILL_ARTIFACT>: the complete versioned skill package, including instructions, utilities, recovery procedures, templates, and interface metadata.
- <PARENT_SKILL> and <SECONDARY_SKILL>: primary and secondary parent artifacts selected from the current population or archive.

Table 10: Sensitivity to uncertainty-aware admission and elite reevaluation. False replacements are candidates that initially replace an incumbent but lose the comparison after reevaluation. Results are mean ± standard deviation over three seeds.

Setting	Pass rate	QD-score	False replacements (%)	Final coverage (%)
No uncertainty correction	52.0±1.3	40.5±1.2	18.7±2.4	70.9±2.0
$\beta = 0.5$, interval 32	52.6±0.9	41.8±0.8	12.5±1.8	71.5±1.7
$\beta = 1.0$, interval 16	52.7±0.8	42.1±0.7	7.4±1.3	68.9±1.6
$\beta = 1.0$, interval 32	**53.0±0.7**	**42.7±0.7**	8.1±1.4	**71.8±1.6**
$\beta = 1.0$, interval 64	52.4±0.9	41.6±0.9	12.9±1.9	72.1±1.8
$\beta = 1.5$, interval 32	52.5±0.8	41.9±0.8	**6.8±1.1**	68.1±1.9
No periodic reevaluation	51.6±1.2	39.9±1.1	21.4±2.7	72.6±2.2

Table 11: Search cost for 512 candidate evaluations. Token counts include candidate generation and task execution. Results are mean ± standard deviation over three seeds. Abbreviations: Calls = Tool calls; Time = Wall-clock time; PR/M token = Pass rate per million tokens.

Method	Tokens (10^6)	Calls (10^3)	Time (h)	PR/M token
Iterative Refinement	31.8±0.7	41.2±1.1	15.7±0.5	1.48±0.04
Top-k	32.6±0.8	42.8±1.0	16.1±0.4	1.52±0.05
Pareto-Only	33.1±0.6	43.6±0.9	16.5±0.5	1.51±0.04
Novelty-Only	32.4±0.9	42.2±1.2	16.3±0.6	1.47±0.06
Text-MAP-Elites	33.5±0.7	43.9±1.1	16.9±0.5	1.48±0.04
Skill-MAP-Elites	34.2±0.8	45.1±1.0	17.6±0.6	**1.55±0.04**

- <FAILURE_TRACES>: bounded execution traces associated with unsuccessful or partially successful tasks.
- <EVALUATION_SUMMARY>: aggregate task outcomes, verifier feedback, execution costs, and observed failure categories.
- <TARGET_CONCERN>: the sampled behavioral concern to which a mutation is restricted.
- <OPERATOR>: the selected behavior-level variation operator.
- <PERTURBATION>: a controlled environment modification, such as schema renaming, transient tool failure, or token-budget reduction.
- <OUTPUT_CONSTRAINTS>: task-specific requirements on the final deliverable.
- <EDITED_FILES>: the complete contents of all files modified or created by a variation operator.

D.3 Main Inference Prompt

Purpose. Execute a task using a fixed skill artifact, the tools exposed by the harness, and the skill's prescribed validation and recovery procedures.

Inputs. <TASK>, <INPUTS>, <AVAILABLE_TOOLS>, <SKILL_ARTIFACT>, and <OUTPUT_CONSTRAINTS>.

Expected Output. A task-compliant final answer or artifact. Tool calls and intermediate observations are captured by the harness as an execution trajectory. If the task cannot be completed safely and reliably, the agent returns a concise abstention containing the blocking condition.

```
You are an agent operating in a tool-enabled execution environment.

Task:
<TASK>

Task inputs:
```

Table 12: Seed-level held-out pass rates for Skill-MAP-Elites under 512 candidate evaluations.

Seed	Document	Web	Spreadsheet	Coding	Analysis	Workflow	Average
1	62.2	45.5	58.1	51.0	41.7	46.7	50.9
2	63.1	46.4	59.0	52.1	42.3	47.6	51.8
3	63.7	46.7	59.6	52.9	43.2	48.2	52.4
Mean $\pm$ std.	63.0 ± 0.8	46.2 ± 0.6	58.9 ± 0.8	52.0 ± 1.0	42.4 ± 0.8	47.5 ± 0.8	51.7 ± 0.8

```
<INPUTS>

Available tools:
<AVAILABLE_TOOLS>

Reusable skill:
<SKILL_ARTIFACT>

Output requirements:
<OUTPUT_CONSTRAINTS>

Follow the reusable skill as the governing procedure for this task. Use only
the tools and resources available in the current environment.

Requirements:
1. Inspect the task inputs before acting.
2. Apply the relevant procedures, utilities, and interface conventions from
   the skill.
3. Execute tool calls only when they contribute to task completion or
   verification.
4. Check tool outputs and required postconditions after each consequential
   operation.
5. If an operation fails, apply the recovery policy specified by the skill.
   Do not repeat an unchanged failing action beyond the permitted retry limit.
6. Preserve task constraints, file-access restrictions, and output formats.
7. Verify the final result using the strongest available task-specific check.
8. If a required input, tool, or verifiable condition is unavailable, abstain
   according to the skill's abstention policy and identify the blocking
   condition.

Return only the final task answer or the requested artifact.
```

D.4 Baseline Prompts

All baselines use the main inference prompt above and the same mutation model. The following templates define the proposal procedures used by baselines that maintain a single refinement chain or independently sample candidates.

D.4.1 Iterative Refinement

Purpose. Revise the current skill using execution failures and validation feedback while maintaining a single candidate lineage.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <EVALUATION_SUMMARY>.

Expected Output. A revised skill artifact and a structured edit manifest. The external search procedure accepts the revision only when it improves validation performance.

```
Revise the reusable agent skill below using the supplied execution evidence.
```

```
Current skill:
<PARENT_SKILL>

Failure traces:
<FAILURE_TRACES>

Evaluation summary:
<EVALUATION_SUMMARY>

Produce one coherent revision that addresses the most consequential recurring
failure. Preserve behavior that is already supported by successful executions.

The revision must:
- remain a valid multi-file skill artifact;
- preserve existing public interfaces unless a compatible migration is
  included;
- use only declared dependencies and permitted file accesses;
- add explicit checks for any newly assumed postconditions;
- define bounded recovery behavior for anticipated failures; and
- avoid unrelated stylistic or textual changes.

Return the result using the structured format in
Appendix~\ref{app:prompt}, ``Candidate Output Schema.''
```

D.4.2 BEST-OF-N INDEPENDENT PROPOSAL

Purpose. Generate an independent skill candidate from a validated seed without using previous sampled candidates.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <EVALUATION_SUMMARY>.

Expected Output. One independently revised skill artifact and its edit manifest. Repeated calls produce the candidate set from which the external procedure selects the highest-validation candidate.

```
Generate one independent improvement to the validated reusable skill below.

Validated seed skill:
<PARENT_SKILL>

Observed failure traces:
<FAILURE_TRACES>

Evaluation summary:
<EVALUATION_SUMMARY>

Select one concrete procedural weakness supported by the evidence and revise
the skill to address it. Do not use or assume access to other sampled
candidates. Preserve successful procedures and public interfaces whenever
possible.

The proposed artifact must be executable, self-contained, compatible with the
declared harness, and limited to changes justified by the supplied evidence.

Return the result using the structured format in
Appendix~\ref{app:prompt}, ``Candidate Output Schema.''
```

D.4.3 SHARED POPULATION-BASED PROPOSAL

Purpose. Generate candidates for Top-k, Pareto-Only, Random Archive, Novelty-Only, Text-MAP-Elites, and Skill-MAP-Elites under a shared mutation model.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, <TARGET_CONCERN>, and <OPERATOR>.

Expected Output. A candidate skill artifact, edit manifest, and behavioral hypothesis. Candidate retention is performed externally according to the corresponding baseline.

```
Apply one behavior-level variation to the reusable agent skill.

Primary parent:
<PARENT_SKILL>

Bounded failure traces:
<FAILURE_TRACES>

Selected behavioral concern:
<TARGET_CONCERN>

Selected variation operator:
<OPERATOR>

Modify only the files and procedures required by the selected concern and
operator. Preserve unrelated behavior, interfaces, and dependencies.

The candidate must:
1. remain parseable and internally consistent;
2. preserve or explicitly reconcile all public interfaces;
3. obey dependency and file-access constraints;
4. include explicit postconditions for modified operations;
5. define bounded failure handling where applicable; and
6. be suitable for deterministic validation and lightweight execution.

The behavioral hypothesis is provenance metadata only. Do not optimize the
wording of the hypothesis and do not treat it as evaluation evidence.

Return the result using the structured format in
Appendix~\ref{app:prompt}, ``Candidate Output Schema.''
```

D.5 Few-Shot and Demonstration Prompts

No few-shot demonstrations are used. Candidate generation and task execution are zero-shot with respect to target-test examples. Evolution traces and ground-truth feedback are available only for the evolution partition and are supplied through the fields defined above.

D.6 Reasoning and Rationale Prompts

D.6.1 Failure-Grounded Mutation Analysis

Purpose. Identify an execution-grounded behavioral concern before applying a mutation. The analysis is restricted to observed traces and does not expose hidden chain-of-thought in the stored artifact.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <EVALUATION_SUMMARY>.

Expected Output. A concise JSON object containing the selected failure category, supporting trace identifiers, affected procedure, proposed operator, and expected observable change.

```
Analyze the supplied execution evidence and select one mutation target.

Skill:
<PARENT_SKILL>
```

```
Failure traces:
<FAILURE_TRACES>

Evaluation summary:
<EVALUATION_SUMMARY>

Use only observable evidence from the traces. Identify one behavioral concern
that can be addressed by a bounded procedural modification. Do not propose
changes based solely on wording, style, or speculative capabilities.

Return exactly one JSON object:
{
  "failure_category": "<observed failure category>",
  "supporting_trace_ids": ["<trace id>", "..."],
  "affected_procedure": "<file, module, or procedure>",
  "target_concern": "<specific behavioral concern>",
  "recommended_operator":
  ↪  "<postcondition|recovery|split|merge|parameterize|policy_adjustment>",
  "expected_observable_change": "<change in execution behavior>"
}
```

D.6.2 Behavioral Hypothesis

Purpose. Record the intended observable effect of a candidate for provenance without using the statement as a quality estimate or archive descriptor.

Inputs. <TARGET_CONCERN>, <OPERATOR>, and <EDITED_FILES>.

Expected Output. A short JSON record describing the targeted behavior and the trace-level evidence that would support or contradict the hypothesis.

```
Summarize the intended behavioral effect of the proposed edit.

Target concern:
<TARGET_CONCERN>

Variation operator:
<OPERATOR>

Edited files:
<EDITED_FILES>

Describe only effects that can be observed in execution trajectories.
Do not predict validation scores, archive admission, or test performance.

Return exactly one JSON object:
{
  "target_behavior": "<behavior being modified>",
  "mechanism": "<procedural change producing the effect>",
  "supporting_observation": "<trajectory pattern that would support the hypothesis>",
  "contradicting_observation": "<trajectory pattern that would contradict the
  ↪  hypothesis>"
}
```

D.7 Data Generation and Pseudo-Labeling Prompts

No synthetic task generation or pseudo-labeling is used. Evolution, validation, and test instances are drawn from the benchmark partitions, and task outcomes are assigned by external verifiers.

D.8 Evaluation Prompts

D.8.1 Task Execution under Controlled Perturbation

Purpose. Evaluate a finalized skill under schema changes, transient tool failures, or a reduced token budget without modifying the skill for the target condition.

Inputs. <TASK>, <INPUTS>, <AVAILABLE_TOOLS>, <SKILL_ARTIFACT>, <PERTURBATION>, and <OUTPUT_CONSTRAINTS>.

Expected Output. A task-compliant final answer or artifact produced under the perturbed environment. Correctness is determined by the same external verifier used in the unperturbed condition.

```
You are an agent operating in a tool-enabled execution environment.

Task:
<TASK>

Task inputs:
<INPUTS>

Available tools:
<AVAILABLE_TOOLS>

Reusable skill:
<SKILL_ARTIFACT>

Active environment condition:
<PERTURBATION>

Output requirements:
<OUTPUT_CONSTRAINTS>

Complete the task using the skill without modifying the skill files. Treat the
active environment condition as part of the runtime rather than as feedback
for skill evolution.

Requirements:
1. Follow the skill's procedures and interface conventions.
2. Detect schema or interface mismatches from observable tool responses.
3. Apply only the recovery and compatibility procedures available in the
   supplied skill.
4. Keep retries bounded and avoid unchanged repetition after transient
   failures.
5. Respect the active token, tool, and time budgets.
6. Verify the final result whenever the remaining budget permits.
7. If completion cannot be verified, apply the skill's abstention policy.

Return only the final task answer or the requested artifact.
```

D.8.2 Execution Record Normalization

Purpose. Normalize harness-native logs into a common execution record when the harness does not directly emit the required schema. This template is used only as a structured transcription step; behavioral descriptors and scores are computed deterministically from the resulting record.

Inputs. A task identifier, harness-native execution log, tool registry, stage schema, and failure-category schema.

Expected Output. A JSON execution record containing only events explicitly supported by the log.

```
Convert the execution log into the required structured record.

Task identifier:
<TASK_ID>

Harness-native execution log:
<EXECUTION_LOG>

Tool-family registry:
<TOOL_FAMILY_REGISTRY>

Task-stage schema:
<TASK_STAGE_SCHEMA>

Failure-category schema:
<FAILURE_CATEGORY_SCHEMA>

Transcribe only information supported by the log. Do not infer successful
completion, failure resolution, or unavailable resource measurements. Use
null for unavailable scalar fields and an empty list for unsupported event
collections.

Return exactly one JSON object:
{
  "task_id": "<task identifier>",
  "termination_status": "<success|failure|abstention|timeout|unknown>",
  "tool_calls": [
    {
      "index": 0,
      "tool_name": "<name>",
      "tool_family": "<registered family or unknown>",
      "outcome": "<success|failure|unknown>",
      "failure_category": "<registered category or null>"
    }
  ],
  "tool_transitions": [
    ["<source tool family>", "<destination tool family>"]
  ],
  "stages_attempted": ["<stage>", "..."],
  "stages_completed": ["<stage>", "..."],
  "failure_categories_observed": ["<category>", "..."],
  "failure_categories_resolved": ["<category>", "..."],
  "token_count": <integer or null>,
  "tool_call_count": <integer or null>,
  "wall_clock_seconds": <number or null>,
  "final_output_present": <true|false>,
  "evidence_trace_ids": ["<event identifier>", "..."]
}
```

The verifier score, normalized execution cost, perturbation robustness, cross-model gain, confidence-adjusted quality, niche assignment, and archive update are computed programmatically and do not use an LLM-as-a-judge prompt.

D.9 Ablation Prompt Variants

D.9.1 No Novelty-Guided Parent Selection

Purpose. Evaluate the effect of removing novelty-guided parent selection.

Inputs. The same inputs as the shared population-based proposal.

Expected Output. The same candidate schema as the main method.

```
Use the Shared Population-Based Proposal prompt without modification.
Parent selection is performed externally without a novelty-guided term.
```

D.9.2 TEXT-MAP-ELITES

Purpose. Replace execution-grounded behavioral niches with niches derived from skill-text embeddings.

Inputs. The same inputs as the shared population-based proposal.

Expected Output. The same candidate schema as the main method.

```
Use the Shared Population-Based Proposal prompt without modification.
Niche assignment is performed externally from embeddings of the candidate
skill text rather than from execution-derived behavioral descriptors.
```

D.9.3 GLOBAL TOP-k RETENTION

Purpose. Replace local within-niche elite selection with retention of the globally highest-scoring candidates.

Inputs. The same inputs as the shared population-based proposal.

Expected Output. The same candidate schema as the main method.

```
Use the Shared Population-Based Proposal prompt without modification.
Candidate retention is performed externally using global confidence-adjusted
quality rather than local niche competition.
```

D.9.4 NO MODULE MERGING

Purpose. Disable compositional mutations that merge compatible components from distinct parents.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and a non-merge <OPERATOR>.

Expected Output. A single-parent candidate using postcondition insertion, recovery substitution, module splitting, parameterization, or policy adjustment.

```
Apply one behavior-level variation to the primary skill.

Primary parent:
<PARENT_SKILL>

Bounded failure traces:
<FAILURE_TRACES>

Selected behavioral concern:
<TARGET_CONCERN>

Selected operator:
<OPERATOR>
```

```
The merge operator is disabled. Do not copy, combine, or reconstruct modules
from any secondary skill. Modify only the primary parent using the selected
non-merge operator.

Return the result using the structured format in
Appendix~\ref{app:prompt}, ``Candidate Output Schema.''
```

D.9.5 No Uncertainty Correction

Purpose. Evaluate archive admission without the lower-confidence adjustment.

Inputs. The same execution outputs and candidate proposals as the main method.

Expected Output. The same candidate and execution formats as the main method.

```
Use all generation and execution prompts without modification.
Archive admission is performed externally using empirical mean quality with
the uncertainty penalty set to zero.
```

D.9.6 Descriptor-Component Ablations

Purpose. Remove one or more descriptor components, including the cross-model descriptor, while preserving the candidate-generation process.

Inputs. The same inputs as the shared population-based proposal.

Expected Output. The same candidate schema as the main method.

```
Use the Shared Population-Based Proposal prompt without modification.
The specified descriptor component is omitted only during external descriptor
construction and niche assignment.
```

D.9.7 Stepping-Stone Removal

Purpose. Remove descendants of candidates whose validation quality was below the archive median at generation time.

Inputs. The same generated candidates, provenance records, and validation outcomes as the main method.

Expected Output. No modified model output is required; filtering is applied externally through recorded ancestry.

```
No additional prompt is used. Apply the standard proposal and execution
templates, then remove the designated ancestors and their dependent lineage
using the recorded provenance graph.
```

D.10 Behavior-Level Variation Templates

D.10.1 Postcondition Insertion

Purpose. Add an explicit check after a consequential operation whose output was previously assumed valid.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A candidate skill containing a verifiable postcondition and a bounded response to postcondition failure.

```
Apply a postcondition-insertion mutation.

Parent skill:
<PARENT_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Identify the consequential operation implicated by the traces. Add the
smallest explicit postcondition that can verify its required result. Specify
the observable success criterion and a bounded response when the criterion is
not met. Do not alter unrelated procedures.

Return the result using the Candidate Output Schema.
```

D.10.2 Recovery-Policy Substitution

Purpose. Replace an ineffective or missing recovery procedure for an observed failure category.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A candidate skill with a trigger, bounded recovery sequence, termination condition, and fallback behavior.

```
Apply a recovery-policy substitution mutation.

Parent skill:
<PARENT_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Replace only the recovery policy implicated by the traces. Define:
- the observable trigger;
- the bounded recovery actions;
- the success condition;
- the retry or termination limit; and
- the fallback or abstention behavior.

Do not conceal failures, weaken verification, or introduce undeclared tools.

Return the result using the Candidate Output Schema.
```

D.10.3 Module Splitting

Purpose. Separate an overloaded module into components with explicit responsibilities and interfaces.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A candidate artifact with separated modules, reconciled imports, and preserved external behavior.

```
Apply a module-splitting mutation.

Parent skill:
<PARENT_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Split only the overloaded module implicated by the evidence. Assign each new
component one coherent responsibility, define explicit interfaces between
components, update all affected references, and preserve the original public
entry points unless a compatible adapter is included.

Return the result using the Candidate Output Schema.
```

D.10.4 COMPATIBLE MODULE MERGING

Purpose. Combine complementary components from behaviorally distinct parent skills without concatenating entire artifacts.

Inputs. <PARENT_SKILL>, <SECONDARY_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A single coherent skill artifact with resolved interfaces, dependencies, and control flow.

```
Apply a compatible-component merge.

Primary parent:
<PARENT_SKILL>

Secondary parent:
<SECONDARY_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Select only the secondary component that addresses the target concern and is
compatible with the primary skill. Integrate the component into the primary
artifact.

Requirements:
1. Do not concatenate the complete parent artifacts.
2. Reconcile names, interfaces, dependencies, state assumptions, and error
   handling.
3. Preserve one authoritative control flow and one public interface.
4. Remove duplicated or conflicting procedures.
5. Add compatibility checks where assumptions differ.
6. Keep all changes restricted to the target concern.
```

```
Return the result using the Candidate Output Schema.
```

D.10.5 PROCEDURE PARAMETERIZATION

Purpose. Replace a hard-coded procedural choice with an explicit bounded parameter.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A candidate artifact with a declared parameter, default value, valid range, and deterministic fallback.

```
Apply a procedure-parameterization mutation.

Parent skill:
<PARENT_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Replace only the hard-coded choice implicated by the traces. Introduce an
explicit parameter with:
- a descriptive name;
- a default preserving current successful behavior;
- a bounded set or range of valid values;
- a rule for selecting the value from observable task conditions; and
- a deterministic fallback for missing or invalid values.

Do not expose unrestricted code execution or undeclared resources.

Return the result using the Candidate Output Schema.
```

D.10.6 RETRY, EVIDENCE-REQUEST, OR ABSTENTION ADJUSTMENT

Purpose. Revise decision conditions governing retries, requests for additional evidence, or abstention.

Inputs. <PARENT_SKILL>, <FAILURE_TRACES>, and <TARGET_CONCERN>.

Expected Output. A candidate skill with explicit observable triggers and bounded decision rules.

```
Apply a policy-adjustment mutation.

Parent skill:
<PARENT_SKILL>

Supporting failure traces:
<FAILURE_TRACES>

Target concern:
<TARGET_CONCERN>

Revise exactly one of the following policies: retry, evidence request, or
abstention. Base the policy on observable execution conditions. Specify the
trigger, permitted action, maximum resource use, stopping condition, and
fallback. Avoid unconditional retries and unsupported claims of completion.
```

```
Return the result using the Candidate Output Schema.
```

D.11 Output Formatting

D.11.1 Candidate Output Schema

Purpose. Provide a machine-readable proposal that can be materialized as a multi-file artifact and checked by the deterministic validator.

Inputs. The outputs of any behavior-level variation or baseline proposal prompt.

Expected Output. Exactly one JSON object with complete file contents, an edit manifest, and a provenance-only behavioral hypothesis.

```
{
  "candidate": {
    "files": [
      {
        "path": "<relative path>",
        "operation": "<create|modify|delete>",
        "content": "<complete file content; empty only for delete>"
      }
    ],
    "entry_points": ["<relative path or callable>", "..."],
    "dependencies": ["<declared dependency>", "..."],
    "interface_version": "<version identifier>"
  },
  "edit_manifest": {
    "operator":
    ↪ "<postcondition|recovery|split|merge|parameterize|policy_adjustment>",
    "target_concern": "<behavioral concern>",
    "modified_components": ["<component>", "..."],
    "preserved_interfaces": ["<interface>", "..."],
    "resolved_conflicts": ["<conflict and resolution>", "..."],
    "validation_checks": ["<required deterministic check>", "..."]
  },
  "behavioral_hypothesis": {
    "target_behavior": "<observable behavior>",
    "mechanism": "<procedural mechanism>",
    "supporting_observation": "<supporting trajectory pattern>",
    "contradicting_observation": "<contradicting trajectory pattern>"
  }
}
```

All file paths are relative to the skill root. For each created or modified file, `content` contains the complete materialized file rather than a diff. Generated artifacts that fail parsing, interface compatibility, dependency checks, file-access constraints, or lightweight execution are logged and excluded from the archive.

D.11.2 Abstention Format

Purpose. Standardize outputs when a task cannot be completed or verified under the available inputs, tools, and budgets.

Inputs. The blocking condition and any completed verifiable work.

Expected Output. A concise structured abstention.

```
{
  "status": "abstain",
  "blocking_condition": "<missing input, unavailable tool, unresolved failure, or
  ↪  exhausted budget>",
  "completed_work": "<brief description of verifiably completed work>",
  "required_next_step": "<minimal action required to continue>"
}
```

E CASE STUDY

We qualitatively inspect representative evolution traces to understand how execution-grounded niches preserve procedural behaviors that would be removed by global candidate selection. The cases below focus on the contents of the evolved skill artifacts, the resulting tool trajectories, and the role of behaviorally distinct ancestors. To avoid conflating qualitative evidence with aggregate evaluation, we omit scores and report only the relevant task context, execution behavior, and generated outputs.

E.1 CASE I: PRESERVING A DEPENDENCY-RECOVERY STEPPING STONE

Task. The agent is asked to modify a Python repository so that a configuration loader accepts environment-variable references of the form `${NAME}`, preserves ordinary strings, and passes the repository tests. The repository uses a package layout in which the test runner imports the project through an editable installation. In the initial environment, invoking the test command fails before any task-specific test is collected because an optional parsing dependency is unavailable.

Behavior of a globally selected skill. A candidate favored by global selection contains a direct code-editing procedure: locate the loader, patch the string-resolution branch, and run the narrowest relevant test. Its execution begins by searching for configuration-related files and correctly identifies `src/config/loader.py`. It then inserts a regular-expression substitution and invokes the test runner. After the import failure, however, the skill treats the failure as evidence that the modified source is invalid. It repeatedly changes import statements in the loader, even though the traceback originates from the test environment rather than the edited function. The resulting patch introduces an unnecessary fallback import and never reaches the tests that exercise environment-variable expansion.

A condensed version of the resulting trajectory is:

```
Action: search repository for "load_config" and "${"
Observation: implementation found in src/config/loader.py

Action: edit loader.py
Patch: add regex replacement for ${NAME}

Action: run targeted test
Observation:
  ModuleNotFoundError: No module named 'yaml'

Action: edit loader.py
Patch: wrap parser import in try/except and use a local fallback

Action: run targeted test
Observation:
  ImportError raised while importing test configuration

Final answer:
  Implemented environment-variable interpolation and added a parser fallback.
  Tests could not be completed because the repository has an import issue.
```

The central problem is not the initial interpolation edit. It is the absence of a failure-localization policy. The skill does not distinguish a source-level regression from a missing execution dependency, so it responds to an environment failure by modifying task code.

A behaviorally distinct candidate. Skill-MAP-Elites retains another candidate in a dependency-recovery niche. This candidate is not initially attractive as a general repository skill because it performs additional environment inspection before editing source files. Its artifact contains a small recovery module with three explicit postconditions:

```
recovery/dependency_failure.md

Trigger:
  A test or build command fails before task-relevant code is executed.

Procedure:
  1. Read the first complete traceback and identify the earliest external
     module that cannot be imported.
  2. Check the repository metadata before changing application source:
     pyproject.toml, requirements files, lock files, and test configuration.
  3. If the dependency is declared, repair the environment using the
     repository's existing installation mechanism.
  4. If installation is unavailable, report the blocked verification step.
     Do not add an application-level fallback unless the task requires one.

Postconditions:
  - The original import succeeds, or the environment limitation is recorded.
  - No production file is changed solely to compensate for a test-environment
    dependency.
  - The task-specific test is rerun only after collection succeeds.
```

When executed on the same task, the candidate reads the traceback, checks the project metadata, and finds that the missing parser is already declared as a development dependency. It repairs the editable installation and reruns test collection before touching the loader. Only after the environment is valid does it implement interpolation.

The candidate also introduces a verification rule that distinguishes missing variables from empty variables. Instead of replacing an unresolved reference with an empty string, it preserves the original token and emits a controlled validation error at the configuration boundary. Its final implementation follows the repository's existing error type and leaves unrelated parser imports unchanged.

```
Action: run targeted test
Observation:
  Test collection fails: ModuleNotFoundError for declared dependency "yaml"

Action: inspect pyproject.toml and test configuration
Observation:
  Parser dependency is declared in the development dependency group

Action: repair editable development installation
Observation:
  Test collection succeeds

Action: inspect loader.py and existing configuration error classes
Observation:
  String values are normalized in resolve_value()
  Missing configuration values use ConfigResolutionError

Action: edit loader.py
Patch behavior:
  - expand each ${NAME} from the provided environment mapping
  - preserve strings without placeholders
```

```
  - raise ConfigResolutionError for an unresolved name
  - avoid recursive expansion of substituted values

Action: run focused tests, then repository tests
Observation:
  Placeholder, ordinary-string, and unresolved-name paths complete successfully

Final answer:
  Added environment-variable reference resolution in resolve_value().
  Missing names now follow the repository's existing configuration-error path.
  No parser fallback or unrelated import change was introduced.
```

Subsequent composition. The dependency-recovery candidate becomes more useful after a later merge combines it with a postcondition-focused coding skill. The merged artifact does not concatenate both instruction files verbatim. Instead, it assigns them to separate execution stages: the recovery module governs failures before test collection, while the postcondition module governs failures after task-relevant tests begin executing.

This separation prevents a common category error. Environment repair is permitted when the traceback terminates in dependency loading, whereas source revision is permitted only after the relevant code path has been exercised. The descendant additionally records the command that established a valid environment, allowing another harness to substitute its own installation tool without changing the underlying recovery decision.

This case illustrates why a candidate with cautious and locally specialized behavior can serve as a stepping stone. A global selector may discard it because its preliminary inspection adds actions on straightforward tasks. An execution-grounded archive instead preserves it for the distinct behavior it contributes: localizing pre-execution failures and refusing to "repair" them through unrelated source edits.

E.2 Case II: Verification-Oriented Document Extraction Under Conflicting Evidence

Task. The agent receives a collection of business documents containing a policy memorandum, an implementation guide, and meeting notes. It must identify the effective reimbursement rule, explain whether the rule applies to submissions already in progress, and provide supporting passages. The files contain superficially conflicting statements. The memorandum announces the policy, the guide describes an earlier operational workflow, and the meeting notes discuss a proposed exception that was not adopted.

Single-pass extraction behavior. A direct extraction skill searches all files for the terms `reimbursement`, `effective`, and `in progress`. It retrieves a sentence from the meeting notes stating that ongoing submissions "should continue under the previous rule." Because this sentence closely matches the question, the agent promotes it into the final answer without checking whether the note records a decision, a proposal, or a rejected alternative.

Its output is fluent but unsupported:

```
The previous reimbursement rule remains applicable to submissions already in
progress. The meeting notes explicitly state that ongoing submissions should
continue under the existing process. Therefore, the new policy applies only to
new submissions.
```

The omitted context is decisive. The quoted sentence occurs under an agenda item labeled "Option discussed," followed later by a decision entry stating that no grandfathering exception will be introduced. The skill's lexical retrieval behavior finds a locally relevant span but fails to establish document authority, temporal status, or decision status.

A stage-coverage niche. Skill-MAP-Elites preserves a candidate whose procedure explicitly separates evidence discovery, status resolution, and answer construction. The skill artifact defines an evidence ledger rather than allowing retrieved spans to flow directly into the final response:

```
templates/evidence_ledger.txt

Claim:
Source file:
Document type:
Document date:
Section heading:
Exact supporting span:
Status:
  [binding | operational guidance | proposal | superseded | unclear]
Scope:
Conflict check:
Final-use decision:
```

The associated instruction module requires the agent to classify each passage before using it. A passage from meeting notes is not treated as binding merely because it contains directive language. The agent must inspect nearby headings and decision markers such as `approved`, `rejected`, `deferred`, or `for discussion`. It must also resolve conflicts using document role and chronology rather than retrieval rank.

On the representative task, the execution produces the following internal evidence ledger:

```
Claim:
  The revised reimbursement rule applies from the stated effective date.

Source file:
  policy_memorandum.pdf

Document type:
  Issued policy memorandum

Section heading:
  Effective Date and Scope

Status:
  binding

Scope:
  Claims submitted or pending review on and after the effective date

Conflict check:
  Operational guide predates the memorandum.
  Meeting-note exception was discussed but not adopted.

Final-use decision:
  Use as primary support.


Claim:
  In-progress submissions keep the previous rule.

Source file:
  implementation_meeting_notes.docx

Document type:
  Meeting notes

Section heading:
```

```
  Option discussed: transition treatment

Status:
  proposal

Conflict check:
  Later decision entry states that the transition exception was rejected.

Final-use decision:
  Do not use as support; mention only to explain the apparent conflict.


Claim:
  Staff should follow the legacy approval sequence.

Source file:
  reimbursement_guide.pdf

Document type:
  Operational guide

Status:
  superseded

Conflict check:
  Guide predates the policy memorandum and lacks the revised scope language.

Final-use decision:
  Exclude from the current-rule answer.
```

The final answer generated from this ledger is materially different:

```
The revised reimbursement rule applies to submissions that are still pending
when the policy takes effect, not only to submissions created afterward. The
controlling source is the issued policy memorandum, whose scope provision
covers both newly submitted claims and claims pending review on the effective
date.

The implementation guide does not establish a different transition rule
because it predates the memorandum. The meeting notes record a proposal to
continue the previous rule for in-progress submissions, but the later decision
entry in the same notes states that this exception was not adopted. The
proposal therefore should not be cited as the governing policy.
```

Why textual novelty is insufficient. A text-based archive can retain multiple skills that use different terminology, such as "cross-check evidence," "verify citations," or "inspect context," while all of them still execute a single retrieval-first trajectory. The retained candidate above is behaviorally distinct because it creates an intermediate ledger, assigns authority status before synthesis, and revisits apparent conflicts. Conversely, a small edit that adds the `Status` and `Final-use decision` fields changes the execution substantially even though most of the skill text remains unchanged.

The niche assignment captures this distinction through task-stage coverage and tool-transition structure. The candidate performs a repeated sequence of retrieval, local-context inspection, metadata inspection, and conflict resolution before drafting. This behavior is qualitatively different from retrieving several passages and immediately composing an answer.

E.3 Case III: Abstention and Schema Recovery in Spreadsheet Manipulation

Task. The agent must update a workbook by adding a derived `Review Status` column to a transactions sheet. The intended rule depends on an approval field and a due-date field. In the original

workbook, the headers are `Approval State` and `Due Date`. In the perturbed workbook, the first header is renamed to `Decision`, and a separate notes sheet contains the phrase "approval state" in free text.

Brittle parameter binding. A hard-coded spreadsheet skill searches for the exact header `Approval State`. When the header is absent, it scans the workbook for matching text and incorrectly binds the rule to a cell in the notes sheet. The skill then inserts formulas into the transactions sheet using an invalid column reference. Since the formulas are syntactically accepted by the workbook library, the agent reports success without reopening the saved workbook or inspecting computed cells.

```
Detected approval field:
  Notes!B7 = "approval state discussed with regional leads"

Inserted formula:
  =IF(AND(Notes!B7="Approved",F2<TODAY()),"Overdue","Open")

Final answer:
  Added Review Status formulas to the transaction records and saved the
  updated workbook.
```

This failure combines three procedural weaknesses: exact-name dependence, unrestricted fallback search, and the absence of output postconditions.

Parameterization and abstention behavior. A candidate retained in a schema-robustness niche replaces hard-coded header lookup with a bounded schema-resolution procedure. It restricts candidate fields to the header row of the target table, samples values from each candidate column, and checks whether the observed value vocabulary is compatible with the required semantic role. Free text elsewhere in the workbook is never eligible as a column binding.

```
schema_resolution.md

For each required logical field:
  1. Search only headers in the identified target table.
  2. Prefer an exact normalized name.
  3. Otherwise consider aliases declared for the logical field.
  4. Validate each alias using sampled cell values and neighboring headers.
  5. If multiple candidates remain plausible, do not write formulas.
     Return the candidate headers and the evidence needed to disambiguate them.

Logical field: approval_state
Accepted aliases:
  - approval state
  - approval status
  - decision

Value-shape check:
  Values should be categorical workflow states, not dates, identifiers,
  or unrestricted prose.

Logical field: due_date
Accepted aliases:
  - due date
  - deadline
  - review by

Value-shape check:
  Non-empty values should be parseable as dates.
```

The candidate binds `Decision` to the logical approval field because it appears in the target table and contains workflow-state values. It rejects the notes-sheet phrase because it is not a table header and its neighboring cells contain unrestricted prose. Before saving, it verifies that the new column is inside the target table, formulas reference row-local cells, untouched sheets remain unchanged, and the workbook can be reopened.

Its execution summary is:

```
Target table:
  Transactions

Resolved fields:
  approval_state -> Transactions[Decision]
  due_date       -> Transactions[Due Date]

Rejected match:
  Notes!B7
  Reason: not a header in the target table; free-text context

Write:
  Add "Review Status" as the final table column
  Populate row-relative formulas using resolved field positions

Post-save verification:
  Reopen workbook
  Confirm header and formula range
  Confirm formulas do not reference Notes
  Confirm original sheet names and non-target cells are preserved

Final answer:
  Added Review Status to the Transactions table using Decision as the
  approval-state field. The similarly worded text in Notes was excluded
  because it is not part of the transaction schema. The saved workbook was
  reopened and the inserted formula references were checked.
```

The same skill also defines a conservative alternative. If both `Decision` and `Approval Status` are present with compatible categorical values, it abstains from editing rather than selecting whichever header appears first:

```
Unable to bind the required approval_state field unambiguously.

Plausible headers:
  - Decision
  - Approval Status

No workbook cells were modified. To continue, specify which field controls the
review rule or provide an example transaction with its expected Review Status.
```

This abstention policy is useful under schema changes because a syntactically valid workbook can still encode the wrong business rule. The candidate therefore treats semantic ambiguity as a write-blocking condition, while treating a unique, value-compatible alias as a recoverable rename.

E.4 Qualitative Synthesis

Across the three cases, the useful diversity is procedural rather than stylistic. The coding case separates environment failures from source regressions. The document case separates retrieval from authority and conflict resolution. The spreadsheet case separates recoverable schema variation from ambiguity that should block modification. Each distinction is visible in execution traces, intermediate artifacts, and postconditions, but may be weakly reflected in the surface text of the skill.

The cases also clarify the role of local elite preservation. A specialized candidate can appear unnecessarily cautious on common tasks while encoding a recovery or verification behavior that becomes decisive after a distribution shift. Retaining such candidates in execution-grounded niches allows later mutations to combine their modules with stronger task-solving procedures. The resulting descendants inherit concrete behavioral components, such as dependency localization, evidence-status ledgers, schema validation, and safe abstention, rather than merely blending instruction wording.

Finally, postconditions consistently separate plausible completion from verified completion. The successful skills do not stop after producing a patch, extracting a matching sentence, or saving a workbook. They verify that task-relevant execution was reached, that evidence has the correct authority, or that written formulas bind to the intended schema. These checks explain qualitatively how behavioral niches preserve complementary capabilities that global ranking, textual diversity, or unrestricted candidate retention can overlook.