SKILLS AND CHALLENGES CO-EVOLVE: LEARNABILITY-AWARE AUTOCURRICULA FOR AGENT SKILL EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents use reusable skills to solve multi-step tasks that require reasoning, tool use, and executable procedures. Existing methods typically optimize skills against largely fixed objectives or generate evaluation tasks separately from skill learning. This separation can favor verifier-specific solutions and produce challenges that provide little learning signal or exceed the skill population's capabilities. We introduce SCOPE, Skill and Challenge Optimization through co-Population Evolution, a learnability-aware autocurriculum that jointly evolves candidate skills and executable challenges near the changing capability boundary of the skill population. SCOPE evaluates each challenge by how well it discriminates among skills, provides actionable signals for attainable improvement, and reveals previously uncovered failures. A solvability gate filters invalid or unsupported challenges through executable verification, reference procedures, or independently verified model consensus. The surviving challenges guide skill mutation and selection. Structured mutations vary tools, schemas, information availability, dependencies, and feedback signals. Hall-of-fame replay maintains pressure from previously informative challenges and reduces regressions on acquired capabilities. We compare SCOPE with fixed-task optimization, random domain perturbation, and adversarial challenge generation without learnability control under hidden tool versions, unseen difficulty ranges, and composed perturbations. Controlled analyses examine evolutionary dynamics, post-evolution transfer, failure coverage, and the individual contributions of learnability, solvability validation, replay, and mutation diversity.

1 INTRODUCTION

Large language model (LLM) agents solve multi-step tasks using reasoning, external tools, executable scripts, and persistent procedural knowledge. A *skill* packages reusable instructions and artifacts that guide agent behavior across related tasks, extending beyond an isolated function call. Skills affect agent performance across professional tasks, with outcomes varying by domain and implementation.(Li et al., 2026) Skill construction and refinement are therefore central to developing agents that adapt their procedures as tools, inputs, and failure modes change without simply accumulating solutions to observed instances.

Existing work provides two complementary foundations. Open-ended agents use automatic curricula to propose objectives and store successful behaviors in reusable skill libraries.(Wang et al., 2023) Skill-evolution systems iteratively revise structured skill packages using execution feedback or co-evolving verification, reducing reliance on manually authored procedures.(Zhang et al., 2026)(Liu et al., 2026) Self-evolving benchmarks generate increasingly complex tasks and validate solutions through reproducible trajectories, revealing capabilities that static evaluations no longer distinguish.(Guo et al., 2026) These approaches evolve skills against largely fixed objectives or evolve evaluation tasks independently of skill learning.

This separation creates a gap. Repeated optimization on a fixed training set or hidden verifier concentrates selection pressure on a narrow set of observable failures. Candidate skills can exploit regularities in the verifier, input schema, tool version, or task composition without remaining reliable

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.

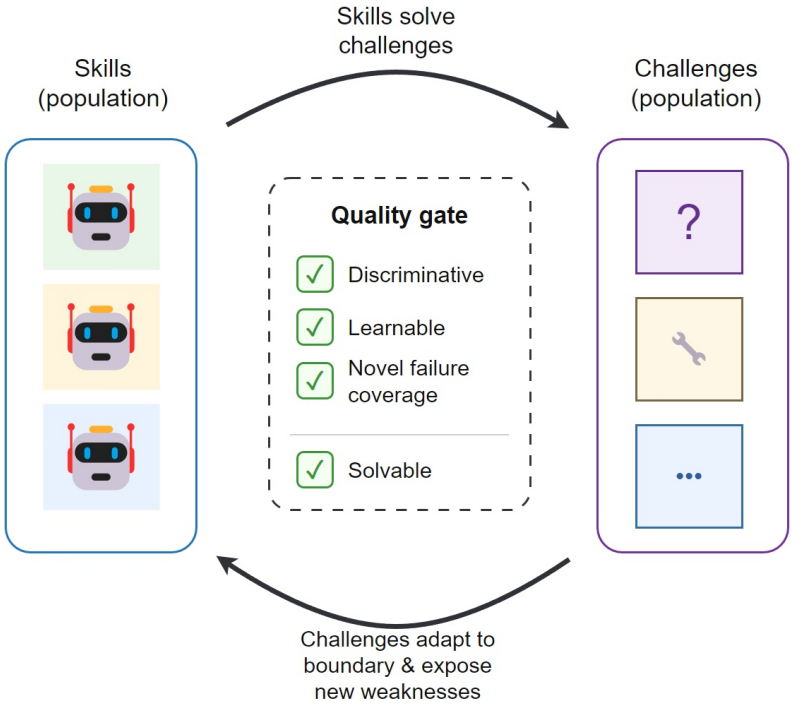

Figure 1: Overview of the SCOPE co-evolution process.

under unseen combinations. Random domain perturbations broaden coverage but ignore current competence, allocating executions to mastered tasks or tasks that provide no actionable learning signal. Adversarial generators that maximize failure can drift toward ambiguous, invalid, or practically unsolvable instances. Such instances may distinguish candidates while providing little evidence of attainable improvement. Skill evolution therefore requires a curriculum that tracks the population's changing capability boundary while remaining discriminative, learnable under the available optimization process, and capable of exposing novel, valid failure modes.

We introduce **SCOPE**, **S**kill and **C**hallenge **O**ptimization through co-**P**opulation **E**volution, a learnability-aware autocurriculum for agent skill evolution. Given seed tasks and an agent environment, SCOPE jointly maintains populations of candidate skills and executable challenges, as illustrated in Figure 1. The challenges explore boundary-valued parameters, tool noise, schema renaming, missing information, conflicting evidence, long-range dependencies, and misleading success signals. SCOPE evaluates each challenge by *discriminativeness*, whether it separates stronger and weaker skills; *learnability*, whether its feedback supports attainable improvement; and *novel failure coverage*, whether it exposes failures absent from the curriculum. A solvability gate filters invalid challenges and penalizes challenges that lack evidence of solvability, using executable verifiers, reference procedures, or independently checked model consensus. The surviving challenges provide targeted feedback for skill mutation and selection, enabling the curriculum to track changes in the skill population.

The two populations form a coupled learning process. As skills resolve existing failures, challenges search nearby boundary cases and compositional perturbations that expose residual weaknesses. As tasks become harder, the learnability and solvability criteria keep fitness gains within the feasible boundary. SCOPE also replays a hall of fame containing previously informative challenges. This replay preserves earlier selection pressure and discourages revisions that sacrifice established robustness for performance on the newest distribution. SCOPE differs from verifier co-evolution, in which verifiers primarily estimate skill quality on a given task,(Zhang et al., 2026) and benchmark evolution, in which generated tasks serve as evaluation artifacts outside the adaptive training process for persistent skill revision.(Guo et al., 2026)

We compare SCOPE with fixed training tasks, random domain perturbation, and adversarial task generation without learnability control. We evaluate evolved skills on hidden tool versions, unseen difficulty ranges, and composed perturbations while tracking task difficulty, skill capability, and failure coverage across generations. We also measure transfer after task evolution stops and isolate the effects of learnability, solvability validation, hall-of-fame replay, and mutation diversity. These experiments assess whether co-evolving attainable challenges improves generalization beyond increasing task difficulty or allocating additional executions to fixed-task search.

Our contributions are:

- We formulate skill evolution as a coupled-population process in which the training distribution adapts to the changing capability boundary of candidate skills, addressing the mismatch between persistent optimization and fixed evaluation pressure.
- We propose SCOPE, which jointly evolves skills and executable challenges according to discriminativeness, learnability, novel failure coverage, and explicit solvability validation.
- We introduce structured challenge mutations and hall-of-fame replay to explore perturbations in tools, schemas, information, dependencies, and feedback while retaining pressure on previously acquired capabilities.
- We design controlled studies of held-out and compositional generalization, evolutionary dynamics, post-evolution transfer, and component ablations to distinguish adaptive, learnable curricula from random perturbation and failure-maximizing generation.

2 Related Work

2.1 Procedural Knowledge and Skill Evolution for LLM Agents

LLM agents interleave reasoning, observations, and external actions. ReAct prompts alternating reasoning and tool-mediated interaction, while Toolformer learns tool invocation through self-supervised API annotations.(Yao et al., 2022)(Schick et al., 2023) Persistent-experience systems extend this loop: Reflexion stores feedback-derived reflections, Self-Refine iteratively critiques current outputs, and ExpeL extracts reusable rules from trajectories.(Shinn et al., 2023)(Madaan et al., 2023)(Zhao et al., 2023) These methods optimize language-based context rather than model parameters.

Structured approaches encode experience as procedural artifacts. Voyager maintains executable skills and retrieves them compositionally.(Wang et al., 2023) ProcMEM evolves procedural units defined by activation, execution, and termination conditions.(Mi et al., 2026) Procedural-memory evaluations distinguish local adaptation from cross-task, cross-role, and cross-model transfer.(Belikova et al., 2026) SkillsBench further reveals substantial variation across domains, implementations, and agent backbones.(Li et al., 2026)

Skill construction can be optimized iteratively. CoEvoSkills jointly revises skills and verification using execution evidence,(Zhang et al., 2026) while Rethinking Skill Evolution studies how update and evaluation choices affect reliability.(Liu et al., 2026) Narrow optimization may cause catastrophic forgetting,(Kirkpatrick et al., 2017) motivating replay, archival selection, and regression testing despite their storage and execution costs.

2.2 Automatic Curricula and Unsupervised Environment Design

Curriculum learning orders examples by difficulty or estimated learning progress.(Bengio et al., 2009)(Graves et al., 2017) Teacher-student methods reward competence changes,(Matiisen et al., 2017) ALP-GMM samples regions with high absolute learning progress,(Portelas et al., 2019) and reverse curricula expand backward from successful goal states.(Florensa et al., 2017)

Open-ended methods remove fixed difficulty orderings. Asymmetric self-play derives curricula from competition,(Sukhbaatar et al., 2017) while POET evolves environment-agent pairs and transfers policies across environments.(Wang et al., 2019) Enhanced POET improves transfer and search over harder distributions.(Wang et al., 2020) These systems combine minimal criteria with novelty search(Lehman & Stanley, 2011) and quality-diversity archives such as MAP-Elites.(Mouret & Clune, 2015)

Unsupervised environment design uses teacher-generated environments without a target distribution. PAIRED approximates regret through protagonist-antagonist return gaps.(Dennis et al., 2020) Prioritized Level Replay resamples levels by learning potential,(Jiang et al., 2020) replay-guided design stabilizes adversarial generation,(Jiang et al., 2021) and ACCEL edits replayed levels to increase complexity near the capability frontier.(Parker-Holder et al., 2022) However, regret proxies may not track intermediate success probability, and direct learnability sampling can be more robust.(Rutherford et al., 2024)

LLMs extend curricula to natural-language and executable-code task spaces. CurricuLLM converts sequenced robot subtasks into reward and goal specifications,(Ryu et al., 2024) while Eurekaverse generates progressively challenging simulation environments.(Liang et al., 2024) Such generation requires execution, solution, and outcome checks to filter invalid, unsafe, or behaviorally redundant tasks.

2.3 Self-Generated Agent Training and Evolving Evaluation

WebRL generates tasks from failed web trajectories and combines online reinforcement learning with replay.(Qi et al., 2024) Self-Evolving Curriculum treats category selection as a non-stationary bandit using policy-gradient advantage magnitude,(Chen et al., 2025) while Absolute Zero jointly trains task proposal and solution behavior using code execution for validation.(Zhao et al., 2025)

Executable benchmarks measure multi-turn tool use. AgentBench covers heterogeneous interactive environments,(Liu et al., 2023) WebArena provides reproducible websites,(Zhou et al., 2023) OS-World evaluates direct operating-system interaction,(Xie et al., 2024) and SWE-bench uses repository issues and regression tests.(Jimenez et al., 2023) TRACE evolves benchmark tasks and validates solutions through reproducible execution trajectories.(Guo et al., 2026) Adaptive generation must track non-stationary learners, verify validity and solvability, and balance novelty with retention. Hall-of-fame coevolution preserves historically strong opponents to reduce cycling,(Rosin & Belew, 1997) paralleling replay mechanisms that retain informative levels, skills, and failure-derived rules.

3 Method

We introduce SCOPE, Skill and Challenge Optimization through co-Population Evolution, a learnability-aware autocurriculum that jointly evolves agent skills and executable challenges. SCOPE generates valid challenges near the current capability boundary, selects them based on learning progress and failure novelty, and uses them to refine skills. Solvability checks filter invalid tasks, and a hall-of-fame archive preserves competence on previously discovered failures.

3.1 Problem Formulation and Framework

Let $\mathcal{X}$ denote a task domain. Each executable task $x = (q, e, v)$ consists of an instruction q, an environment e, and a verifier v. A skill $s \in \mathcal{S}$ is a package of procedural artifacts available to policy π. Executing π produces $\tau \sim \pi(\cdot \mid q, e, s)$ and reward $r(s, x) = v(\tau) \in [0, 1]$. For stochastic executions, the mean reward is $\bar{r}(s, x) = K^{-1} \sum_{k=1}^{K} r_k(s, x)$.

At generation g, SCOPE maintains a skill population $\mathcal{S}_g = \{s_i^g\}_{i=1}^{N_s}$, a task population $\mathcal{T}_g = \{x_j^g\}_{j=1}^{N_t}$, and a hall of fame $\mathcal{H}_g \subset \mathcal{X}$. It selects challenges that are discriminative, learnable, and behaviorally novel, while filtering unreliable tasks. The evolving tasks define the training curriculum. Held-out evaluation tasks remain inaccessible during training.

Each generation mutates and validates challenges, evaluates them against the current skills, selects a curriculum, refines and replaces skills, and updates the archive, as illustrated in 2. After training, SCOPE disables evolution and attaches a final skill or ensemble directly to the agent. The complete procedure is provided in the Appendix.

3.2 Structured Challenge Evolution

Each task has a descriptor $\mathbf{z}(x) = (z^{\text{sem}}, z^{\text{env}}, z^{\text{dep}})$ that represents its semantic objective, environment and tool conditions, and information dependencies. SCOPE generates a candidate as

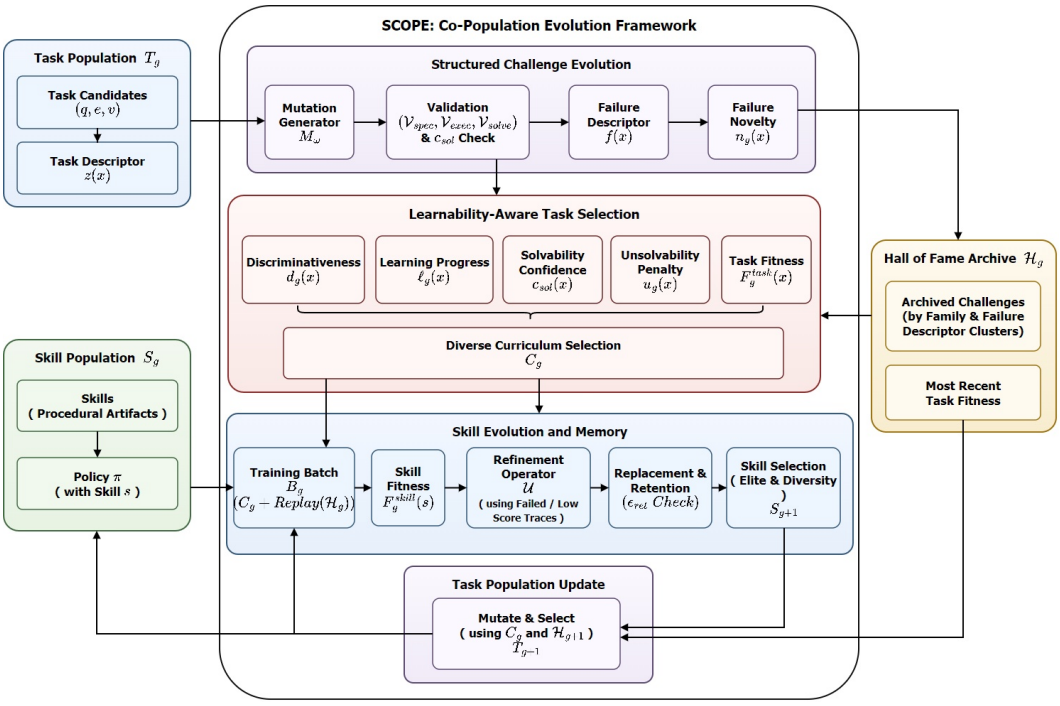

Figure 2: Overview of the SCOPE co-population evolution framework.

$\tilde{x} = M_\omega(x; \xi)$, where $\omega \sim p_g(\omega)$ identifies a perturbation family and $\xi \sim p(\xi \mid \omega)$ specifies its parameters. Mutations can alter boundaries, tool responses, schemas, available information, evidence consistency, temporal dependencies, or misleading intermediate outcomes. Each mutation jointly updates q, e, and v. The distribution $p_g(\omega)$ adapts to favor perturbation families that produce distinct failures.

Candidates must satisfy $V(\tilde{x}) = V_{\text{spec}}(\tilde{x})V_{\text{exec}}(\tilde{x})V_{\text{solve}}(\tilde{x})$, where the three factors verify specification consistency, executable infrastructure, and the existence of a supported solution, respectively. A reference program, a reproducible trajectory, or strong-model consensus can establish solvability. Validation solutions remain hidden during skill refinement. Candidates must also satisfy $c_{\text{sol}}(\tilde{x}) \geq \delta_{\text{sol}}$, where the solvability confidence c_{sol} is measured through reproducible runs or agreement among accepted solutions.

Executions of the current skills produce a failure descriptor $\mathbf{f}(x) = \Phi(\{\tau_i(x)\}_{i=1}^{N_s})$, which can encode verifier failures, tool errors, violated dependencies, or incorrect artifacts. Failure novelty is defined as

$$n_g(x) = \min_{x' \in \mathcal{T}_g \cup \mathcal{H}_g} d_f(\mathbf{f}(x), \mathbf{f}(x')), \tag{1}$$

where the normalized distance d_f rewards distinct behavioral weaknesses and discounts superficial task variations.

3.3 Learnability-Aware Task Selection

Let $p_g(x) = N_s^{-1} \sum_i \bar{r}(s_i^g, x)$. SCOPE measures task discriminativeness using the population variance

$$d_g(x) = \frac{1}{N_s} \sum_{i=1}^{N_s} \left(\bar{r}(s_i^g, x) - p_g(x)\right)^2, \tag{2}$$

which favors challenges that separate skills and assigns low scores to tasks that all skills solve or fail.

SCOPE estimates short-horizon learning progress by applying the refinement operator U to a probe subset:

$$s_i^{g,+} = U(s_i^g, x, \tau_i(x), v(x)), \qquad \ell_g(x) = \frac{1}{|\mathcal{S}_g^{\text{probe}}|} \sum_{s_i^g \in \mathcal{S}_g^{\text{probe}}} \left[\bar{r}(s_i^{g,+}, x) - \bar{r}(s_i^g, x)\right]. \tag{3}$$

The operator U receives the task, execution trace, and verifier feedback without access to the protected solution. The probe skills are discarded after estimating learning progress. SCOPE penalizes tasks that are currently unlearnable or have low solvability confidence:

$$u_g(x) = \mathbb{I}[p_g(x) < \delta_{\text{low}}]\mathbb{I}[\ell_g(x) < \delta_{\text{learn}}] + 1 - c_{\text{sol}}(x). \tag{4}$$

The task fitness is

$$F_g^{\text{task}}(x) = V(x)\left[\lambda_d d_g(x) + \lambda_\ell [\ell_g(x)]_+ + \lambda_n n_g(x) - \lambda_u u_g(x)\right], \tag{5}$$

where $[a]_+ = \max(a, 0)$ and all λ coefficients are nonnegative. Fitness-ranked selection subject to a minimum separation under d_f produces a diverse curriculum $\mathcal{C}_g \subseteq \mathcal{T}_g$.

3.4 Skill Evolution and Memory

SCOPE evaluates skills on the batch $\mathcal{B}_g = \mathcal{C}_g \cup \text{Replay}(\mathcal{H}_g, B_h)$, where replay samples at most B_h archived tasks stratified by perturbation family and failure descriptor. The skill fitness is

$$F_g^{\text{skill}}(s) = \frac{1}{|\mathcal{B}_g|} \sum_{x \in \mathcal{B}_g} w_g(x)\bar{r}(s, x), \qquad w_g(x) = \frac{\exp(\beta F_g^{\text{task}}(x))}{\sum_{x' \in \mathcal{B}_g} \exp(\beta F_g^{\text{task}}(x'))}, \tag{6}$$

where archived challenges use their most recent task fitness.

SCOPE selects parents by rank and refines them using failed or low-scoring traces:

$$\tilde{s} = U\left(s, \{x, \tau_s(x), v(x, \tau_s(x))\}_{x \in \mathcal{B}_g}\right). \tag{7}$$

The operator U can revise artifacts within the skill but cannot modify tasks, verifiers, or the archive. SCOPE retains an offspring only if it improves fitness without reducing archived performance by more than ϵ_{ret}. Elite retention and behavioral diversity then produce

$$\mathcal{S}_{g+1} = \text{SelectSkill}\Big(\mathcal{S}_g \cup \widetilde{\mathcal{S}}_g; F_g^{\text{skill}}, \mathcal{H}_g, N_s\Big). \tag{8}$$

A selected task enters the hall of fame when its discriminativeness or novelty exceeds the corresponding threshold. The bounded archive retains the highest-fitness representative from each perturbation-family and failure-descriptor cluster:

$$\mathcal{H}_{g+1} = \text{Compress}(\mathcal{H}_g \cup \{x \in \mathcal{C}_g : d_g(x) \geq \delta_d \vee n_g(x) \geq \delta_n\}; N_h). \tag{9}$$

Re-evaluating the archive identifies recurring failures. The coupled optimization is

$$\begin{aligned} \mathcal{C}_g &= \underset{x \in \mathcal{T}_g}{\text{Select}}\, F_g^{\text{task}}(x; \mathcal{S}_g, \mathcal{H}_g), \\ \mathcal{S}_{g+1} &= \underset{s \in \mathcal{S}_g \cup \widetilde{\mathcal{S}}_g}{\text{Select}}\, F_g^{\text{skill}}(s; \mathcal{C}_g, \mathcal{H}_g), \\ \mathcal{T}_{g+1} &= \text{MutateSelect}(\mathcal{T}_g, \mathcal{C}_g, \mathcal{H}_{g+1}). \end{aligned} \tag{10}$$

Training terminates after a fixed execution budget or when both the best archived skill fitness and novel-failure coverage remain unchanged for a predefined number of generations.

4 Experiments

4.1 Evaluation Protocol

We evaluate whether SCOPE skills generalize beyond combinations observed during evolution. The benchmark covers domains: spreadsheet manipulation, document-grounded question answering,

Method	ID Test	Hidden-Tool	Unseen-Difficulty	Compositional	Average
FIXED	64.8±1.1	52.6±1.4	49.7±1.7	43.9±1.5	52.8
RANDOM-DR	67.1±0.9	56.8±1.2	53.4±1.5	48.2±1.6	56.4
ADVERSARIAL	63.5±1.8	55.9±2.0	51.6±2.2	47.5±2.4	54.6
VALIDATED-ADV	<u>68.9±1.0</u>	<u>58.7±1.3</u>	**57.6±1.4**	<u>50.9±1.7</u>	<u>59.0</u>
SCOPE (ours)	**70.2±0.8**	**62.1±1.1**	<u>57.2±1.2</u>	**59.0±1.4**	**62.1**

Table 1: Task pass rate (%) under matched execution budgets. Bold and underline denote the best and second-best results, respectively.

Variant	Average Pass Rate	Failure Coverage	Solvable Tasks	GPU Hours	Relative Cost
SCOPE (full)	**62.1±0.7**	**78.6**	**91.3**	46.2	1.00×
w/o learnability	57.4±1.3	73.9	76.5	43.7	0.95×
w/o hall of fame	59.0±1.1	68.2	90.7	44.8	0.97×
failure-rate objective only	55.8±1.7	70.1	68.9	41.9	0.91×
w/o solvability verification	56.6±1.5	75.0	72.4	**38.5**	**0.83**×
single perturbation family	58.1±1.0	59.6	92.0	40.4	0.87×

Table 2: Ablation results and computational cost under a common maximum generation budget. Pass rate, coverage, and solvability are percentages.

scientific data analysis, and repository-level software maintenance. Each domain is partitioned into evolution, validation, and test splits by task templates, tool schemas, and source artifacts. Adaptive methods access evolution. Validation tasks select checkpoints, and the test split remains hidden until evaluation.

We evaluate three suites. *Hidden-Tool* modifies tool versions, argument conventions, or output schemas. *Unseen-Difficulty* extends parameter ranges and dependency lengths beyond evolution. *Compositional* combines two or three perturbations. The population and solvability verifier cannot access test templates or perturbation seeds.

We compare SCOPE with four strategies under matched budgets. FIXED improves skills on the evolution tasks. RANDOM-DR uniformly samples perturbation operators. ADVERSARIAL evolves challenges to minimize the success rate of the skills. VALIDATED-ADV applies SCOPE's solvability check without optimizing learnability or new-failure coverage. All methods use the population, executor, optimizer, population size, interaction budget, and context limit.

The primary metric is pass rate. We report failure coverage, the proportion of audited failure categories exposed during evolution, and challenges judged solvable by references, a strong-model committee, and adjudication. We run each condition with three seeds and report pass rates as mean ± standard deviation. Statistical comparisons use paired bootstrap intervals over test instances.

4.2 GENERALIZATION PERFORMANCE

Table 1 and Figure 3 report test performance. SCOPE leads on in-distribution, Hidden-Tool, and Compositional sets and ranks second on Unseen-Difficulty. Its average pass rate is 62.1%, exceeding FIXED, RANDOM-DR, and VALIDATED-ADV by 9.3, 5.7, and 3.1 percentage points. For the gains over VALIDATED-ADV on Hidden-Tool and Compositional, the two-sided 95% paired-bootstrap intervals exclude zero. Solvability filtering is insufficient. Selecting failures that support attainable improvement yields generalization gains.

The largest improvement occurs under composed shifts. SCOPE achieves 59.0% on Compositional, compared with 50.9% for VALIDATED-ADV, an 8.1-point gain. The challenge population discovers weaknesses, while hall-of-fame replay maintains pressure from tool, schema, and boundary perturba-

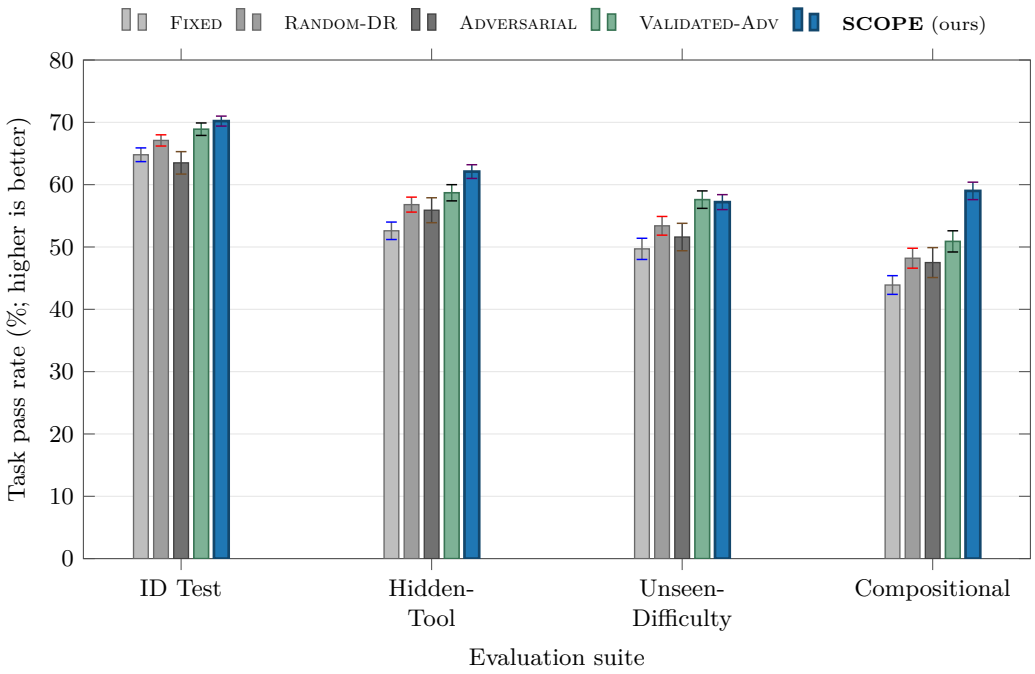

Figure 3: Task pass rates under budgets, averaged over three seeds with one-standard-deviation error bars.

tions. Adversarial generation can replace previously informative challenges with failures that target only the latest skill population.

Results on Unseen-Difficulty differ. VALIDATED-ADV achieves 57.6%, compared with 57.2% for SCOPE, and the bootstrap interval for the difference includes zero. Many instances require longer execution horizons without introducing new recovery procedures. Here, solvable adversarial boundary search remains competitive without learning-progress estimation. SCOPE provides the largest gains when distribution shifts require reusable adaptations or combinations of previously distinct behaviors. It provides no advantage for an increase in execution length.

We isolate persistent skill improvement from deployment-time search by freezing the validation-best checkpoint after evolution and instantiating unseen files, parameter values, and perturbation combinations. The generator, selector, and verifier provide no feedback during evaluation. This protocol ensures that improvements reflect durable procedural changes rather than opportunistic responses to evaluator feedback, repeated artifacts, or perturbations encountered during the evolutionary search. Compared with FIXED, SCOPE improves Hidden-Tool and Compositional performance by 9.5 and 15.1 points, respectively. Compared with RANDOM-DR, the corresponding gains are 5.3 and 10.8 points. These gains persist without test-time task adaptation and are therefore encoded in the evolved skill.

The gain also varies with initial task difficulty. Compared with VALIDATED-ADV, SCOPE gains 0.8 points on tasks where the initial population succeeds more than 80% of the time, 3.6 points on medium-difficulty tasks, and 5.1 points near the capability boundary. The difference decreases on tasks with initial success below 20%, where technically solvable challenges often yield weak or unreliable update signals. This pattern is consistent with a learnability-aware curriculum that prioritizes tasks between mastery and effective impossibility.

4.3 EVOLUTIONARY BEHAVIOR

The population trajectories provide context for the final generalization results. Early generations expose parameter-boundary and tool-interface failures, causing challenge difficulty to increase before

skill capability improves. After the skills acquire basic recovery procedures, challenge mutations increasingly introduce schema changes, conflicting evidence, and long-range dependencies. Progress is stepwise: only 31% of proposed skill updates become validation-best checkpoints, and several generations produce no accepted revision.

Pure adversarial evolution rapidly increases the immediate failure rate, while its solvable-challenge rate decreases from 94.1% to 71.8%. SCOPE maintains solvability above 89% while increasing normalized challenge difficulty from 0.34 to 0.72. It achieves 78.6% cumulative failure coverage, compared with 61.4% for RANDOM-DR and 69.7% for VALIDATED-ADV. The additional coverage is concentrated in misleading-success, conflicting-evidence, and multi-step recovery failures. Identifying these categories requires determining whether feedback supports a productive revision, beyond detecting failure of the current skill.

Hall-of-fame replay reduces regressions as the active population shifts toward new perturbation families. Historical reevaluation rejects most updates that trade robustness to earlier tool or schema perturbations for performance on recent challenges, although replay does not eliminate forgetting. The resulting plateaus arise from conservative checkpoint selection and do not indicate stalled task evolution.

4.4 COMPONENT ANALYSIS

Table 2 separates the contributions of learnability, solvability validation, replay, and mutation diversity. Removing learnability decreases the average pass rate from 62.1% to 57.4% and the solvable-challenge rate from 91.3% to 76.5%. Optimizing only the failure rate yields an average pass rate of 55.8% and a solvability rate of 68.9%, because ambiguous challenges and mutations that remove required resources receive high fitness. Removing explicit verification reduces the pass rate to 56.6% and solvability to 72.4%. Without an independent feasibility constraint, learning-progress estimates do not reliably distinguish productive failures from malformed tasks.

Removing hall-of-fame replay decreases the average pass rate by 3.1 points and failure coverage by 10.4 points, showing that historical challenges help preserve acquired robustness. Restricting evolution to a single perturbation family maintains high solvability (92.0%) but reduces coverage to 59.6% and the pass rate to 58.1%. No individual mutation family accounts for the full gain. The ablations attribute SCOPE's gains to the joint use of attainable challenge selection, explicit solvability validation, diverse mutations, and replay. Increasing task difficulty alone does not produce the same results.

5 CONCLUSION

We presented SCOPE, a learnability-aware autocurriculum that co-evolves candidate skills and executable challenges along the changing capability boundary of LLM agents. By combining discriminativeness, learnability, novel failure coverage, solvability validation, structured mutations, and hall-of-fame replay, SCOPE enables persistent skill refinement while preserving established robustness. Experimental comparisons, held-out and compositional evaluations, evolutionary analyses, transfer assessments, and component ablations confirm that SCOPE produces more generalizable skill evolution than fixed-task optimization, random perturbation, and unconstrained adversarial generation. The results establish adaptive, attainable challenge evolution as an effective mechanism for improving reusable agent skills.

LIMITATIONS

Our evaluation covers four professional agent domains, so the observed gains may not fully extend to embodied, multimodal, or safety-critical settings. The experiments use three random seeds and a fixed set of perturbation operators, which may underrepresent variability across model backbones and challenge-generation strategies. In addition, learnability estimation requires temporary probe updates and repeated executions, introducing computational overhead that could become more substantial for larger skill populations or environments with expensive tool interactions.

AI USAGE DISCLOSURE

This paper made use of AI language tools in a strictly limited manner. We used these tools only for editing and polishing the written language of the manuscript—tasks such as smoothing transitions between paragraphs, improving word choice, and correcting minor grammatical errors. The scientific contributions of this work, including the autocurriculum coevolution framework, theoretical insights, and all experimental results, were produced entirely by the human authors. AI tools played no role in the research design, algorithm development, data processing, or result interpretation. All AI-assisted edits were reviewed by the authors to ensure fidelity to the original intent. The authors are solely responsible for this work.

REFERENCES

Julia Belikova, Rauf Parchiev, Evgeny Egorov, Grigorii Davydenko, Gleb Gusev, Andrey Savchenko, and Maksim Makarenko. Managing procedural memory in llm agents: Control, adaptation, and evaluation. *arXiv preprint arXiv:2606.23127*, 2026. URL https://arxiv.org/abs/2606.23127.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009. URL https://doi.org/10.1145/1553374.1553380.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*, 2025. URL https://arxiv.org/abs/2505.14970.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *arXiv preprint arXiv:2012.02096*, 2020. URL https://arxiv.org/abs/2012.02096.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017. URL https://arxiv.org/abs/1707.05300.

Alex Graves, Marc G. Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017. URL https://arxiv.org/abs/1704.03003.

Dadi Guo, Tianyi Zhou, Dongrui Liu, Chen Qian, Qihan Ren, Shuai Shao, Zhiyuan Fan, Yi R. Fung, Kun Wang, Linfeng Zhang, and Jing Shao. Towards self-evolving agent benchmarks: Validatable agent trajectory via test-time exploration. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2H03gm4Rq6.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *arXiv preprint arXiv:2010.03934*, 2020. URL https://arxiv.org/abs/2010.03934.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *arXiv preprint arXiv:2110.02439*, 2021. URL https://arxiv.org/abs/2110.02439.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017. URL https://doi.org/10.1073/pnas.1611835114.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 2011. URL https://doi.org/10.1162/EVCO_a_00025.

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.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. *arXiv preprint arXiv:2411.01775*, 2024. URL https://arxiv.org/abs/2411.01775.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Yuxuan Liu, Zhaochen Su, Yuhao Zhang, Jiahe Guo, Zhongwei Xie, Huihao Jing, Lingyun Xie, Qing Zong, Yauwai Yim, Zhixiong Zhang, Haoran Li, and Yangqiu Song. Rethinking self-evolving agent skills: Feedback dynamics over multiple rounds. *arXiv preprint arXiv:2608.02636*, 2026. URL https://arxiv.org/abs/2608.02636.

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. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017. URL https://arxiv.org/abs/1707.00183.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026. URL https://arxiv.org/abs/2602.01869.

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.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. *arXiv preprint arXiv:2203.01302*, 2022. URL https://arxiv.org/abs/2203.01302.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. *arXiv preprint arXiv:1910.07224*, 2019. URL https://arxiv.org/abs/1910.07224.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024. URL https://arxiv.org/abs/2411.02337.

Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 1997. URL https://doi.org/10.1162/evco.1997.5.1.1.

Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. *arXiv preprint arXiv:2408.15099*, 2024. URL https://arxiv.org/abs/2408.15099.

Kanghyun Ryu, Qiayuan Liao, Zhongyu Li, Payam Delgosha, Koushil Sreenath, and Negar Mehr. Curricullm: Automatic task curricula design for learning complex robot skills using large language models. *arXiv preprint arXiv:2409.18382*, 2024. URL https://arxiv.org/abs/2409.18382.

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.

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.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017. URL https://arxiv.org/abs/1703.05407.

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.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. *arXiv preprint arXiv:2003.08536*, 2020. URL https://arxiv.org/abs/2003.08536.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

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.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue (Steve) Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026. URL https://arxiv.org/abs/2604.01687.

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.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025. URL https://arxiv.org/abs/2505.03335.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

APPENDIX

A GENERATION-WISE CO-POPULATION UPDATE

This section specifies the dependency order among challenge evolution, learnability-aware curriculum construction, skill refinement, archive maintenance, and task-population evolution. At generation g, task fitness depends on the current skill population $\mathcal{S}_g$ and archive $\mathcal{H}_g$. The next skill, archive, and task populations are constructed after selecting the curriculum $\mathcal{C}_g$.

A.1 VALIDATED CANDIDATE EVALUATION

Let $\widehat{\mathcal{T}}_g$ denote the candidate set obtained by mutating source tasks available at generation g. For each source task x, the procedure samples a perturbation family and its parameters:

$$\omega \sim p_g(\omega), \qquad \xi \sim p(\xi \mid \omega), \qquad \tilde{x} = M_\omega(x; \xi). \tag{11}$$

Because M_ω jointly modifies the instruction, environment, and verifier, validation operates on the complete mutated task $\tilde{x}$. The candidates eligible for population evaluation are

$$\widehat{\mathcal{T}}_g^{\text{valid}} = \left\{ \tilde{x} \in \widehat{\mathcal{T}}_g : V_{\text{spec}}(\tilde{x}) V_{\text{exec}}(\tilde{x}) V_{\text{solve}}(\tilde{x}) = 1,\ c_{\text{sol}}(\tilde{x}) \geq \delta_{\text{sol}} \right\}. \tag{12}$$

Only tasks in $\widehat{\mathcal{T}}_g^{\text{valid}}$ can enter the evaluated task population. Solvability-validation artifacts are withheld from both the policy and the refinement operator.

For each x in the resulting task population $\mathcal{T}_g$, every current skill s_i^g, $i \in \{1, \ldots, N_s\}$, is executed to obtain the traces and mean rewards used by the task-selection statistics:

$$\{\tau_{i,k}(x), r_{i,k}(x)\}_{k=1}^{K}, \qquad \bar{r}(s_i^g, x) = \frac{1}{K} \sum_{k=1}^{K} r_{i,k}(x), \qquad r_{i,k}(x) = v(\tau_{i,k}(x)). \tag{13}$$

Here, $\tau_{i,k}(x) \sim \pi(\cdot \mid q, e, s_i^g)$. These evaluations determine $p_g(x)$, $d_g(x)$, and the failure descriptor $\mathbf{f}(x) = \Phi(\{\tau_i(x)\}_{i=1}^{N_s})$. Distances between this descriptor and those in $\mathcal{T}_g \cup \mathcal{H}_g$ determine $n_g(x)$.

Learning progress is estimated while keeping the persistent skill population unchanged. For each $s_i^g \in \mathcal{S}_g^{\text{probe}}$, the procedure constructs a temporary probe skill:

$$s_i^{g,+} = U(s_i^g, x, \tau_i(x), v(x)). \tag{14}$$

The probe is re-evaluated on the same task to obtain $\bar{r}(s_i^{g,+}, x)$. After computing $\ell_g(x)$ from the before-and-after rewards, the procedure discards $s_i^{g,+}$. Probe refinement therefore affects task fitness without modifying $\mathcal{S}_{g+1}$ directly.

A.2 CURRICULUM, SKILL, ARCHIVE, AND TASK UPDATES

After computing all task statistics, the procedure evaluates $F_g^{\text{task}}(x)$ for every $x \in \mathcal{T}_g$. Fitness-ranked selection under the prescribed minimum separation d_f yields $\mathcal{C}_g$. Hall-of-fame replay is sampled after fixing this curriculum:

$$\mathcal{B}_g = \mathcal{C}_g \cup \text{Replay}(\mathcal{H}_g, B_h). \tag{15}$$

A task that belongs to both operands of the union occurs once in $\mathcal{B}_g$. The normalized task weights used to compute skill fitness satisfy

$$w_g(x) \geq 0, \qquad \sum_{x \in \mathcal{B}_g} w_g(x) = 1. \tag{16}$$

For archived tasks, this normalization uses the most recently available task fitness.

Each selected parent s is evaluated on $\mathcal{B}_g$. The refinement operator U then receives its failed or low-scoring traces and verifier feedback and produces an offspring $\tilde{s}$. When $\mathcal{H}_g \neq \varnothing$, define archived performance as

$$A_g(s) = \frac{1}{|\mathcal{H}_g|} \sum_{x \in \mathcal{H}_g} \bar{r}(s, x). \tag{17}$$

The offspring-retention rule is

$$F_g^{\mathrm{skill}}(\tilde{s}) > F_g^{\mathrm{skill}}(s), \qquad A_g(\tilde{s}) \geq A_g(s) - \epsilon_{\mathrm{ret}}. \tag{18}$$

When $\mathcal{H}_g = \varnothing$, the archive constraint is vacuous. Accepted offspring form $\widetilde{\mathcal{S}}_g$. Rejected offspring are excluded before elite and diversity-based skill selection.

The archive is updated from the selected curriculum, excluding generated candidates that were not selected. A task becomes eligible for compression into $\mathcal{H}_{g+1}$ only if it is validated, selected, and satisfies $d_g(x) \geq \delta_d$ or $n_g(x) \geq \delta_n$. The resulting archive is used by $\mathrm{MutateSelect}$ to construct $\mathcal{T}_{g+1}$. The generation-level dependency chain is

$$(\mathcal{S}_g, \mathcal{T}_g, \mathcal{H}_g) \longrightarrow \mathcal{C}_g \longrightarrow \mathcal{S}_{g+1} \longrightarrow \mathcal{H}_{g+1} \longrightarrow \mathcal{T}_{g+1}. \tag{19}$$

At inference time, the procedure executes no task-generation, validation, probing, refinement, selection, replay, or archive-update steps. A final selected skill, or the final ensemble when used, is attached to the policy and held fixed while the policy generates a trajectory for the input task.

B Environment Details

B.1 Datasets and Task Settings

We construct an executable benchmark spanning four professional agent domains: spreadsheet manipulation, document-grounded question answering, scientific data analysis, and repository-level software maintenance. Each domain contains 800 evolution instances, 100 validation instances, and 200 held-out instances for each evaluation suite. Splits are disjoint with respect to task templates, source artifacts, tool schemas, and perturbation seeds. Adaptive methods access only the evolution split, while the validation split is used for checkpoint selection and hyperparameter tuning. Test instances remain inaccessible until the final evaluation.

Spreadsheet manipulation. Tasks require reading and modifying `.xlsx` workbooks using Python and `openpyxl`. The benchmark covers formula construction, aggregation, filtering, cross-sheet references, formatting, and consistency checks over workbooks containing 2–12 sheets and up to 50,000 populated cells. A task is completed by producing a modified workbook that satisfies an executable cell-level and formula-level verifier.

Document-grounded question answering. Tasks require extracting and reconciling evidence from collections of PDF, DOCX, and plain-text documents. Instances contain 2–20 documents and require short-form answers, structured summaries, or JSON records supported by the provided files. Harder instances involve conflicting evidence, renamed fields, missing metadata, and dependencies spanning multiple documents.

Scientific data analysis. Tasks operate on CSV, JSON, and NumPy-formatted data and require data cleaning, statistical computation, model fitting, or figure generation. Each instance specifies an output artifact and a set of numerical constraints evaluated by an executable reference implementation. Difficulty is controlled through dataset size, missingness, heterogeneous schemas, and the number of dependent analysis steps.

Repository-level software maintenance. Tasks consist of isolated source-code repositories paired with natural-language issue descriptions and regression tests. The agent may inspect and edit repository files and execute tests inside a sandboxed container. Instances include bug repair, API migration, configuration changes, and multi-file modifications, with completion determined by the held-out test suite.

For each domain, we additionally construct three distribution-shift suites. HIDDEN-TOOL changes tool versions, argument conventions, or output schemas while preserving the semantic objective. UNSEEN-DIFFICULTY extrapolates beyond the evolution range in file size, dependency length, execution horizon, or repository scope. COMPOSITIONAL combines two or three independently sampled perturbations from distinct families. The in-distribution test set uses held-out artifacts and templates under the evolution-time tool configuration.

Algorithm 1 Generation-wise training procedure for SCOPE

Require: Initial skill population $\mathcal{S}_0$, initial task population $\mathcal{T}_0$, initial archive $\mathcal{H}_0$, task and skill population sizes N_t, N_s, replay bound B_h, archive capacity N_h, refinement operator U, mutation operator M_ω, policy π, validation functions, and the thresholds and coefficients appearing in the task, retention, and archive rules

Ensure: Final skill population and hall-of-fame archive

Initialize generation index $g \leftarrow 0$
while the fixed execution budget is not exhausted and the prescribed stagnation condition is not met **do**
  Generate candidate tasks by sampling $\omega \sim p_g(\omega)$, sampling $\xi \sim p(\xi \mid \omega)$, and applying $\tilde{x} = M_\omega(x; \xi)$ to source tasks
  Discard each $\tilde{x}$ for which $V(\tilde{x}) = 0$ or $c_{\mathrm{sol}}(\tilde{x}) < \delta_{\mathrm{sol}}$
  Form the evaluated task population $\mathcal{T}_g$ from the surviving candidates according to the task-population update rule
  for each $x \in \mathcal{T}_g$ **do**
    for each $s_i^g \in \mathcal{S}_g$ **do**
      Execute $\pi(\cdot \mid q, e, s_i^g)$ and compute $\bar{r}(s_i^g, x)$
    end for
    Compute $p_g(x)$, $d_g(x)$, and $\mathbf{f}(x)$
    Compute $n_g(x)$ from failure-descriptor distances to $\mathcal{T}_g \cup \mathcal{H}_g$
    for each $s_i^g \in \mathcal{S}_g^{\mathrm{probe}}$ **do**
      Form the temporary probe $s_i^{g,+} = U(s_i^g, x, \tau_i(x), v(x))$
      Re-evaluate $s_i^{g,+}$ on x and record $\bar{r}(s_i^{g,+}, x)$
    end for
    Compute $\ell_g(x)$, discard all temporary probes, and compute $u_g(x)$
    Compute $F_g^{\mathrm{task}}(x)$
  end for
  Select $\mathcal{C}_g \subseteq \mathcal{T}_g$ by task-fitness rank subject to the minimum separation under d_f
  Set $\mathcal{B}_g \leftarrow \mathcal{C}_g \cup \mathrm{Replay}(\mathcal{H}_g, B_h)$
  Compute $w_g(x)$ for all $x \in \mathcal{B}_g$
  for each $s \in \mathcal{S}_g$ **do**
    Evaluate s on $\mathcal{B}_g$ and compute $F_g^{\mathrm{skill}}(s)$
  end for
  Select parent skills by rank
  for each selected parent s **do**
    Construct $\tilde{s} = U(s, \{x, \tau_s(x), v(x, \tau_s(x))\}_{x \in \mathcal{B}_g})$
    Evaluate $\tilde{s}$ on $\mathcal{B}_g$ and, when nonempty, on $\mathcal{H}_g$
    if $\tilde{s}$ improves skill fitness and satisfies the archive-retention constraint **then**
      Add $\tilde{s}$ to $\widetilde{\mathcal{S}}_g$
    end if
  end for
  Set $\mathcal{S}_{g+1} \leftarrow \mathrm{SelectSkill}(\mathcal{S}_g \cup \widetilde{\mathcal{S}}_g; F_g^{\mathrm{skill}}, \mathcal{H}_g, N_s)$
  Set $\mathcal{H}_{g+1} \leftarrow \mathrm{Compress}(\mathcal{H}_g \cup \{x \in \mathcal{C}_g : d_g(x) \geq \delta_d \vee n_g(x) \geq \delta_n\}; N_h)$
  Set $\mathcal{T}_{g+1} \leftarrow \mathrm{MutateSelect}(\mathcal{T}_g, \mathcal{C}_g, \mathcal{H}_{g+1})$
  Update $p_{g+1}(\omega)$ using the observed distinct failures produced by each perturbation family
  $g \leftarrow g + 1$
end while
Disable task and skill evolution
return $\mathcal{S}_g, \mathcal{H}_g$

All agents receive a common system prompt defining the available tools, sandbox restrictions, output directory, and termination protocol. Each episode begins with the task instruction followed by a concise planning prefix, *"Analyze the task, execute the required operations, verify the resulting artifact, and return the final answer."* Skills are inserted as reusable procedural context after the system prompt and before the task instruction. The agent may interleave natural-language reasoning with tool calls, but only the final answer and generated artifacts are exposed to the evaluator. Reference

solutions, hidden tests, validation trajectories, and verifier internals are never included in the agent context.

To limit contamination and template leakage, source artifacts are deduplicated using SHA-256 hashes and near-duplicate filtering over MinHash signatures. Templates derived from the same underlying operation graph are assigned to a single split. Generated challenges are rejected if their normalized instruction or artifact signature has a Jaccard similarity above 0.85 with any validation or test instance. Test perturbation seeds and tool configurations are generated independently and remain unavailable to the challenge generator, skill population, and solvability committee.

B.2 Preprocessing and Evaluation

Inputs are converted to deterministic task bundles containing the instruction, source artifacts, environment specification, allowed tools, resource limits, and verifier entry point. Text files are decoded as UTF-8 and normalized to Unicode NFC. Document extraction preserves page boundaries, table structure, and source identifiers. Spreadsheet formulas are retained rather than replaced by cached values. Repositories are checked out at fixed commits, and generated files, dependency caches, and version-control metadata are removed before execution. Each episode runs in a fresh container with no network access, a 20-minute wall-clock limit, 32 GB of host memory, and at most 20 GB of writable storage.

Evaluation is based on task-specific executable verification. Spreadsheet outputs are checked for required sheets, cell values, formulas, formats, and invariants, with floating-point values compared using $|\hat{y} - y| \leq 10^{-6} \max(1, |y|)$. Document answers are lowercased, Unicode-normalized, stripped of surrounding whitespace and punctuation, and compared after collapsing repeated whitespace. Numeric answers are parsed from plain text or LaTeX, including values enclosed by `\boxed{}`, and canonicalized as integers, reduced fractions, or floating-point values. Dates are converted to ISO-8601 format, and measurement units are converted to the units specified by the verifier. For questions with multiple valid answers, a prediction is accepted if its normalized form matches any reference answer or satisfies the corresponding structured constraint.

Scientific-analysis outputs are validated against reference statistics, schemas, and file-level checks. Scalar results use the tolerance specified above unless the task defines a domain-specific tolerance; arrays additionally require matching shapes and element-wise agreement. Generated figures are evaluated through the underlying plotted data and required labels rather than pixel similarity. Repository tasks pass only when all required public and hidden tests succeed and no protected test or configuration file has been modified.

The primary metric is binary task pass rate,

$$\mathrm{PassRate} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{I}[v(x, \tau_x) = 1] \,.$$

Partial verifier scores are used during evolution when available, but final results count an instance as successful only if all mandatory constraints pass. We additionally report failure coverage, defined as the fraction of manually audited failure categories observed during evolution, and challenge solvability, defined as the fraction of generated challenges accepted by the independent solvability gate. Failure descriptors are assigned from verifier outcomes, tool exceptions, violated dependencies, invalid artifacts, and trajectory-level recovery behavior; semantically equivalent descriptors are merged by agglomerative clustering with cosine distance threshold 0.15.

Each stochastic condition is run with seeds $\{13, 42, 2026\}$. Tables report the mean and sample standard deviation across the three runs. Statistical comparisons use paired bootstrap resampling over test instances with 10,000 replicates and two-sided 95% confidence intervals. Model sampling randomness, mutation choices, task selection, replay sampling, and skill-population initialization are seeded independently from a run-level seed. The compared methods are FIXED, RANDOM-DR, ADVERSARIAL, VALIDATED-ADV, and SCOPE. All methods share the same initial skills, executor, refinement model, context limit, population size, and maximum number of environment executions.

B.3 Implementation and Hardware

The agent executor and skill-refinement operator use `Qwen2.5-72B-Instruct` in bfloat16 precision, deployed locally with `vLLM`. The same backbone is used across all compared methods to isolate the effect of curriculum construction. During task execution, the model produces plans, tool calls, recovery actions, and final responses. During skill refinement, it receives the current skill package, failed or low-scoring trajectories, and verifier feedback, and outputs a revised procedural package. Model parameters remain frozen throughout; evolution modifies only skill text, executable helper scripts, templates, and metadata. Challenge mutation uses the same backbone in a separate prompting mode and has no access to protected solutions or held-out evaluation data.

The solvability gate first applies deterministic specification, infrastructure, and verifier-consistency checks. When no executable reference procedure is available, solvability is estimated by an isolated three-model committee comprising `Qwen2.5-72B-Instruct`, `Llama-3.3-70B-Instruct`, and `DeepSeek-V3`. Committee models receive the task and environment but no candidate-skill trajectories. A challenge is accepted when at least two models independently produce verifier-accepted solutions across three attempts, corresponding to a solvability-confidence threshold of $\delta_{\mathrm{sol}} = 2/3$. Committee trajectories and solutions are discarded after validation and are never provided to the evolving skills.

All executor calls use temperature 0.2, top-$p = 0.95$, and a maximum of 8,192 generated tokens. Challenge generation and skill mutation use temperature 0.7, top-$p = 0.95$, and a maximum of 6,144 tokens to encourage controlled diversity. Solvability attempts use temperature 0.4. The context window is limited to 32,768 tokens for every method; long documents and repository contents are accessed through tools rather than inserted wholesale into the prompt. An episode permits at most 64 tool calls, and repeated identical failing calls are terminated after three occurrences.

SCOPE maintains $N_s = 8$ skills and $N_t = 32$ active challenges. At each generation, every selected parent challenge produces four mutations, yielding a branching factor of 4 and at most 128 unvalidated candidates. Candidates are filtered by static checks and the solvability gate, after which the top 16 challenges are selected subject to a minimum failure-descriptor distance of 0.20. Learning progress is estimated using a probe subset of two skills, with two stochastic executions before and after a temporary refinement. Probe refinements are discarded after scoring. Mean task reward is estimated from $K = 3$ rollouts per skill–challenge pair.

We use $\delta_{\mathrm{low}} = 0.20$, $\delta_{\mathrm{learn}} = 0.05$, and $\delta_{\mathrm{sol}} = 0.67$. Task fitness weights are set to $\lambda_d = 1.0$, $\lambda_\ell = 1.0$, $\lambda_n = 0.5$, and $\lambda_u = 1.0$. Skill-fitness weights use a softmax inverse temperature of $\beta = 2.0$. The hall-of-fame archive stores at most $N_h = 64$ challenges, retaining one high-fitness representative per perturbation-family and failure-descriptor cluster. Each generation replays $B_h = 8$ archived challenges, stratified equally across perturbation families whenever possible. The retention tolerance is $\epsilon_{\mathrm{ret}} = 0.02$, so an offspring is rejected when its mean archived pass rate decreases by more than two percentage points. Two elite skills are copied unchanged to the next generation, and the remaining six slots are filled by rank-based parent selection followed by refinement and behavioral-diversity filtering.

Evolution runs for at most 30 generations or until both validation pass rate and cumulative novel-failure coverage fail to improve for five consecutive generations. Checkpoints are evaluated every generation, and the validation-best skill is selected using average pass rate across the four domains, with archived pass rate used as the tie breaker. Final evaluation freezes this checkpoint and disables challenge generation, refinement, replay updates, and verifier feedback. No test-time self-consistency or skill adaptation is used.

Experiments are executed on a single node with eight NVIDIA H100 SXM 80 GB GPUs, two AMD EPYC 9654 CPUs, and 1 TB of RAM. Tensor parallelism of eight is used for each 70B-scale model, while environment execution and verifier processes run asynchronously on CPU workers. The software stack consists of Ubuntu 22.04, Python 3.11, PyTorch 2.5, CUDA 12.4, `vLLM` 0.6.4, Docker 27, `openpyxl` 3.1, `pandas` 2.2, and `pytest` 8.3. Reported GPU hours include agent inference, challenge generation, solvability checking, probe refinement, and validation, but exclude deterministic CPU-only verifiers and final test evaluation.

Method	Spreadsheet	Document QA	Scientific Analysis	Repo. Maintenance	Macro Average
FIXED	58.7±1.4	61.9±1.2	49.8±1.6	40.8±1.9	52.8
RANDOM-DR	62.0±1.1	64.1±1.0	54.2±1.7	45.3±1.8	56.4
ADVERSARIAL	59.8±2.0	61.7±1.7	52.0±2.3	44.9±2.5	54.6
VALIDATED-ADV	64.8±1.3	66.1±1.1	57.2±1.5	47.9±1.8	59.0
SCOPE (ours)	**67.5±0.9**	**68.0±0.8**	**59.6±1.3**	**53.3±1.6**	**62.1**

Table 3: Task pass rate (%) by domain, averaged across ID, Hidden-Tool, Unseen-Difficulty, and Compositional evaluation suites. Results are mean and standard deviation over three seeds. Abbreviations: QA = Question Answering; Repo. = Repository.

Method	Easy ($p_0 > 0.8$)	Medium ($0.5 < p_0 \leq 0.8$)	Boundary ($0.2 \leq p_0 \leq 0.5$)	Hard ($p_0 < 0.2$)
FIXED	86.4±0.6	64.9±1.2	41.8±1.8	20.7±1.4
RANDOM-DR	87.0±0.7	68.1±1.0	46.7±1.6	23.0±1.7
ADVERSARIAL	84.6±1.2	65.5±1.8	45.9±2.4	22.5±2.1
VALIDATED-ADV	88.1±0.6	70.7±1.1	49.8±1.5	24.1±1.8
SCOPE (ours)	**88.9±0.5**	**74.3±0.9**	**54.9±1.3**	**25.0±1.6**

Table 4: Task pass rate (%) by initial population success rate p_0. Bins are constructed before evolution and remain fixed across methods.

C Additional Experimental Results

C.1 Domain-Level Generalization

Table 3 reports the average pass rate across the four evaluation suites for each domain. SCOPE achieves higher pass rates than FIXED and RANDOM-DR in all four domains. On repository-level software maintenance, SCOPE achieves a pass rate of 53.3%, compared with 40.8% for FIXED and 45.3% for RANDOM-DR. Tasks in this domain frequently require combining tool recovery, dependency tracking, and verifier-aware artifact inspection.

C.2 Performance by Initial Difficulty

Test instances are partitioned according to the success rate of the initial skill population. As shown in Table 4, SCOPE achieves the highest pass rate in each difficulty bin. Relative to VALIDATED-ADV, the differences are 0.8, 3.6, 5.1, and 0.9 percentage points for the Easy, Medium, Boundary, and Hard bins, respectively.

C.3 Execution-Budget Scaling

Table 5 reports performance under fractions of the full interaction budget. SCOPE achieves the highest average pass rate at each evaluated budget. Its average pass rate increases from 52.1% at 25% of the budget to 62.1% at the full budget, with successive increases of 5.5, 3.1, and 1.4 percentage points.

C.4 Hall-of-Fame Replay Sensitivity

Table 6 reports the results for different replay sizes. A replay size of $B_h = 8$ yields an average pass rate of 62.1%, a Compositional pass rate of 59.0%, and failure coverage of 78.6%. The archive regression rate decreases from 14.7% at $B_h = 0$ to 4.8% at $B_h = 12$. For $B_h = 16$, the average pass rate, Compositional pass rate, and failure coverage are 60.8%, 57.1%, and 74.8%, respectively.

Method	25% Budget	50% Budget	75% Budget	100% Budget
FIXED	47.9±1.5	50.4±1.3	51.7±1.2	52.8±1.0
RANDOM-DR	49.5±1.6	53.0±1.2	55.1±1.0	56.4±0.9
VALIDATED-ADV	50.8±1.4	55.7±1.4	58.2±1.1	59.0±1.0
SCOPE (ours)	**52.1±1.3**	**57.6±1.0**	**60.7±0.8**	**62.1±0.7**

Table 5: Average pass rate (%) as a function of the available execution budget. Each fraction proportionally reduces the number of generations while retaining the same population and batch sizes.

Replay Size B_h	Average Pass Rate	Compositional	Failure Coverage	Archive Regression Rate
0	59.0±1.1	54.1±1.8	68.2±1.9	14.7±1.8
4	61.2±0.8	57.8±1.5	75.4±1.4	8.1±1.2
8	**62.1±0.7**	**59.0±1.4**	**78.6±1.2**	5.3±0.9
12	61.7±0.9	58.4±1.5	77.9±1.5	**4.8±0.8**
16	60.8±1.0	57.1±1.7	74.8±1.7	4.9±1.0

Table 6: Sensitivity to the number of hall-of-fame challenges replayed per generation. All metrics are percentages. Archive regression denotes the fraction of previously solved archived challenges that fail after an accepted skill update.

C.5 Contribution of Challenge Mutation Families

Table 7 reports leave-one-family-out results. Removing dependency mutations yields a Compositional pass rate of 53.9%, compared with 59.0% for the full method. Removing tool and interface mutations yields a Hidden-Tool pass rate of 57.4%, compared with 62.1% for the full method. Each leave-one-family-out variant has lower Average and Failure Coverage values than the full method.

C.6 Solvability-Gate Audit

A stratified sample of generated challenges is manually audited to evaluate the precision of the solvability gate. Table 8 reports acceptance rate, solvability precision, solvability recall, and inter-annotator agreement for each validation strategy. Executable-reference validation achieves a precision of 96.2% and a recall of 74.5%. The reference-or-verified-consensus strategy achieves an acceptance rate of 79.6%, a precision of 93.7%, a recall of 90.4%, and an inter-annotator agreement of 0.89.

C.7 Seed-Level Results and Statistical Robustness

Table 9 reports the individual seed results underlying the generalization comparison. The ordering of methods is consistent across the three seeds. SCOPE achieves the highest average pass rate for each seed. Among the evaluated methods, ADVERSARIAL has the highest standard deviation across seeds at 1.7 percentage points.

D Prompt Templates

D.1 Overview

This appendix specifies the prompt templates used for agent execution, structured challenge evolution, solvability validation, skill refinement, failure characterization, and model-based evaluation. Unless otherwise stated, all templates are instantiated independently for each task domain. Protected reference solutions and validation trajectories are never included in prompts used to execute or refine candidate skills.

D.2 Notation

We use the following placeholders throughout the templates:

Variant	Hidden-Tool	Unseen-Difficulty	Compositional	Average	Failure Coverage
SCOPE (full)	**62.1±1.1**	**57.2±1.2**	**59.0±1.4**	**62.1±0.7**	**78.6±1.2**
Without tool and interface mutations	57.4±1.4	56.5±1.3	56.1±1.7	59.1±1.0	71.5±1.6
Without schema mutations	59.0±1.3	56.8±1.2	55.7±1.6	59.4±0.9	70.7±1.8
Without information mutations	60.2±1.2	55.9±1.5	55.1±1.9	59.0±1.1	69.8±2.0
Without dependency mutations	60.5±1.1	54.8±1.6	53.9±1.8	58.6±1.2	68.7±1.7
Without feedback-signal mutations	59.6±1.4	56.3±1.3	55.4±1.5	59.2±0.9	70.2±1.5

Table 7: Leave-one-family-out analysis of structured challenge mutations. Average includes the ID suite in addition to the three shifted-distribution suites.

Validation Strategy	Acceptance Rate	Solvability Precision	Solvability Recall	Inter-Annotator Agreement
Single-model judgment	86.7±1.5	78.4±2.1	91.6±1.7	0.71
Model consensus only	74.9±1.7	89.1±1.6	86.8±1.9	0.79
Executable reference only	61.8±1.4	**96.2±1.0**	74.5±2.2	0.87
Reference or verified consensus	**79.6±1.3**	93.7±1.2	**90.4±1.5**	**0.89**

Table 8: Audit of solvability-validation strategies on a stratified sample of generated challenges. Acceptance, precision, and recall are percentages; agreement is measured using Fleiss' κ.

- {TASK_INSTRUCTION}: the natural-language task instruction q.
- {ENVIRONMENT_SPEC}: the environment e, including available tools, interfaces, files, and execution constraints.
- {TOOL_SCHEMAS}: the names, arguments, and return schemas of available tools.
- {SKILL}: the procedural artifacts associated with a candidate skill s.
- {EXECUTION_TRACE}: the trajectory τ, including actions, tool calls, observations, and produced artifacts.
- {VERIFIER_FEEDBACK}: verifier output $v(x, \tau)$, excluding protected solutions.
- {PARENT_TASK}: an executable task $x = (q, e, v)$ selected for mutation.
- {TASK_DESCRIPTOR}: the semantic, environmental, and dependency descriptor $z(x)$.
- {PERTURBATION_FAMILY}: the selected mutation family ω.
- {PERTURBATION_PARAMETERS}: mutation parameters ξ.
- {REFERENCE_PROCEDURE}: an executable reference program or reproducible solution trajectory used only for validation.
- {CANDIDATE_SOLUTIONS}: independently generated solutions used to estimate solvability by consensus.
- {FAILURE_TAXONOMY}: the set of verifier failures, tool errors, dependency violations, and artifact-level failure categories.
- {CURRICULUM_BATCH}: the selected active and replayed challenges B^g.
- {OUTPUT_SCHEMA}: the task-specific required output format.

D.3 Main Inference Prompt

Purpose. Execute a task using the procedural artifacts of a candidate skill and the tools exposed by the current environment.

Inputs. {TASK_INSTRUCTION}, {ENVIRONMENT_SPEC}, {TOOL_SCHEMAS}, {SKILL}, and {OUTPUT_SCHEMA}.

Method	Seed 17	Seed 42	Seed 73	Mean ± Std.
FIXED	51.8	53.9	52.7	52.8±1.1
RANDOM-DR	55.6	56.2	57.4	56.4±0.9
ADVERSARIAL	52.7	55.1	56.0	54.6±1.7
VALIDATED-ADV	58.0	59.3	59.7	59.0±0.9
SCOPE (ours)	**61.4**	**62.0**	**62.9**	**62.1±0.8**

Table 9: Average pass rate (%) for each random seed. The mean is computed over the four evaluation suites. Abbreviation: Std. = Standard Deviation.

Comparison	Mean Difference	95% Paired-Bootstrap CI	Two-Sided p-value
SCOPE vs. FIXED	+9.3	[7.8, 10.9]	< 0.001
SCOPE vs. RANDOM-DR	+5.7	[4.2, 7.1]	< 0.001
SCOPE vs. ADVERSARIAL	+7.5	[5.6, 9.4]	< 0.001
SCOPE vs. VALIDATED-ADV	+3.1	[1.7, 4.5]	0.002

Table 10: Paired-bootstrap comparisons of the average pass rate. Confidence intervals are computed over test instances using 10,000 bootstrap resamples. Differences are reported in percentage points. Abbreviation: CI = Confidence Interval.

Expected Output. A sequence of valid environment actions followed by a final task artifact conforming to {OUTPUT_SCHEMA}.

```
You are an agent operating in an executable environment. Complete the
task by following the provided skill and using only the available tools.

Task:
{TASK_INSTRUCTION}

Environment:
{ENVIRONMENT_SPEC}

Available tools:
{TOOL_SCHEMAS}

Skill:
{SKILL}

Requirements:
1. Follow the task instruction and all environment constraints.
2. Use the provided skill as reusable procedural guidance.
3. Inspect tool outputs before selecting subsequent actions.
4. Do not assume that a tool call succeeded solely because it returned
   without an explicit error.
5. Check intermediate and final artifacts against the task requirements.
6. If a tool response, schema, or expected resource differs from the
   skill, adapt the procedure using observable environment evidence.
7. Do not access unavailable resources or protected reference solutions.
8. Return the final answer or artifact in the following format:

{OUTPUT_SCHEMA}
```

D.4 Baseline Prompts

D.4.1 Fixed-Task Skill Refinement

Purpose. Refine a candidate skill using failures observed on the original evolution tasks without generating task perturbations.

Inputs. {SKILL}, {TASK_INSTRUCTION}, {ENVIRONMENT_SPEC}, {EXECUTION_TRACE}, and {VERIFIER_FEEDBACK}.

Expected Output. A revised skill package in the same representation as the input skill.

```
Revise the skill using the execution evidence from the fixed training
task.

Current skill:
{SKILL}

Task:
{TASK_INSTRUCTION}

Environment:
{ENVIRONMENT_SPEC}

Execution trace:
{EXECUTION_TRACE}

Verifier feedback:
{VERIFIER_FEEDBACK}

Produce a revised skill that:
1. Corrects the failure indicated by the trace and verifier feedback.
2. Preserves procedures that remain valid.
3. States tool assumptions and observable success conditions explicitly.
4. Avoids encoding task-specific answers, filenames, parameter values,
   or verifier-specific shortcuts.
5. Does not modify the task, environment, verifier, or evaluation data.
6. Uses the same skill representation as the current skill.

Return only the revised skill.
```

D.4.2 RANDOM DOMAIN PERTURBATION

Purpose. Generate an executable challenge by uniformly sampled domain perturbation, without conditioning mutation choice on learnability or failure novelty.

Inputs. {PARENT_TASK}, {TASK_DESCRIPTOR}, {PERTURBATION_FAMILY}, and {PERTURBATION_PARAMETERS}.

Expected Output. A JSON object containing a mutated instruction, environment specification, verifier specification, and task descriptor.

```
Generate one executable variant of the parent task using the specified
perturbation. Apply the mutation consistently to the instruction,
environment, and verifier.

Parent task:
{PARENT_TASK}

Parent descriptor:
{TASK_DESCRIPTOR}

Perturbation family:
{PERTURBATION_FAMILY}

Perturbation parameters:
{PERTURBATION_PARAMETERS}
```

```
Requirements:
1. Preserve the parent task's semantic objective unless the perturbation
   explicitly changes it.
2. Apply only the supplied perturbation family and parameters.
3. Keep the task self-contained and executable.
4. Ensure that the verifier evaluates the mutated requirements.
5. Do not optimize for skill failure, learnability, or failure novelty.

Return only:
{
  "instruction": "...",
  "environment": {...},
  "verifier": {...},
  "descriptor": {
    "semantic": {...},
    "environment": {...},
    "dependencies": {...}
  },
  "mutation_record": {
    "family": "...",
    "parameters": {...}
  }
}
```

D.4.3 ADVERSARIAL CHALLENGE GENERATION

Purpose. Generate a challenge that minimizes the current population's expected success rate, without learnability or solvability control.

Inputs. {PARENT_TASK}, {TASK_DESCRIPTOR}, summarized population failures, and available perturbation families.

Expected Output. A JSON object containing the adversarially mutated task.

```
Generate an adversarial variant of the parent task that is likely to
cause the current skill population to fail.

Parent task:
{PARENT_TASK}

Parent descriptor:
{TASK_DESCRIPTOR}

Observed population failures:
{POPULATION_FAILURE_SUMMARY}

Available perturbation families:
{PERTURBATION_FAMILIES}

Requirements:
1. Select and instantiate perturbations that minimize expected success
   of the current skills.
2. Update the instruction, environment, and verifier jointly.
3. Preserve enough structure for execution in the supplied environment.
4. Do not use learning progress or failure-coverage criteria.
5. Do not reveal a solution in the task instruction.

Return only:
{
  "instruction": "...",
  "environment": {...},
```

```
  "verifier": {...},
  "descriptor": {
    "semantic": {...},
    "environment": {...},
    "dependencies": {...}
  },
  "mutation_record": {
    "families": ["..."],
    "parameters": {...},
    "targeted_failures": ["..."]
  }
}
```

D.4.4 VALIDATED ADVERSARIAL CHALLENGE GENERATION

Purpose. Generate failure-maximizing challenges and retain only those that pass the independent solvability check.

Inputs. The inputs of the adversarial generation prompt and the candidate task produced by that prompt.

Expected Output. An adversarial task in the preceding JSON format, followed by validation with the solvability prompt in Section D.8.3. Learnability and failure novelty are not used for selection.

```
Generate an adversarial task using the same procedure as adversarial
challenge generation. The task will subsequently be checked by an
independent solvability validator.

Parent task:
{PARENT_TASK}

Parent descriptor:
{TASK_DESCRIPTOR}

Observed population failures:
{POPULATION_FAILURE_SUMMARY}

Available perturbation families:
{PERTURBATION_FAMILIES}

Requirements:
1. Maximize the expected failure rate of the current skill population.
2. Update the instruction, environment, and verifier consistently.
3. Include all resources required by at least one supported solution.
4. Do not optimize for learning progress or novel failure coverage.
5. Do not include the solution in the instruction.

Return only:
{
  "instruction": "...",
  "environment": {...},
  "verifier": {...},
  "descriptor": {
    "semantic": {...},
    "environment": {...},
    "dependencies": {...}
  },
  "mutation_record": {
    "families": ["..."],
    "parameters": {...},
    "targeted_failures": ["..."]
```

```
  }
}
```

D.5 Few-Shot and Demonstration Prompts

No few-shot demonstrations are included in the default execution, challenge-generation, validation, refinement, or evaluation prompts. All reported methods use the templates above in zero-shot form. When domain-specific tool documentation is required, it is supplied as part of {ENVIRONMENT_SPEC} or {TOOL_SCHEMAS}, not as a solved demonstration.

D.6 Reasoning and Rationale Prompts

D.6.1 Failure-Grounded Skill Refinement

Purpose. Apply the refinement operator U or U_s to a probe or selected skill using failed or low-scoring traces and verifier feedback.

Inputs. {SKILL}, one or more tasks, execution traces, verifier feedback records, and, for retained updates, archived regression summaries.

Expected Output. A revised skill and a structured change record. The change record is used for audit and does not expose hidden reasoning to the executor.

```
Improve the reusable skill using only the supplied tasks, execution
traces, and verifier feedback. Protected solutions are unavailable.

Current skill:
{SKILL}

Training evidence:
{TRAINING_EVIDENCE}

Archived regression summary:
{ARCHIVED_PERFORMANCE_SUMMARY}

Task:
1. Identify procedural weaknesses directly supported by the evidence.
2. Revise the skill so that it handles the observed failure conditions
   and nearby variations.
3. Prefer general checks, recovery procedures, and explicit termination
   conditions over instance-specific corrections.
4. Preserve behavior that remains correct, especially behavior required
   by archived challenges.
5. Do not copy task answers, reference outputs, hidden verifier logic,
   concrete filenames, or isolated parameter values into the skill.
6. Do not modify tasks, verifiers, tool implementations, or the archive.
7. Keep the revised skill within the original representation and context
   budget.

Return only:
{
  "revised_skill": "...",
  "change_record": {
    "evidence_used": ["..."],
    "failure_modes_addressed": ["..."],
    "procedures_added_or_changed": ["..."],
    "assumptions_made_explicit": ["..."],
    "preserved_capabilities": ["..."]
```

```
  }
}
```

D.6.2 Failure Descriptor Extraction

Purpose. Map population execution traces to a normalized behavioral failure descriptor $f(x)$ for novelty computation and archive clustering.

Inputs. The task specification, population execution traces, verifier feedback, tool errors, and {FAILURE_TAXONOMY}.

Expected Output. A normalized JSON failure descriptor.

```
Characterize the behavioral failures exposed by the task. Use only
observable execution evidence and verifier feedback. Distinguish
procedural failures from superficial differences in task wording or
artifact identity.

Task:
{TASK_SPECIFICATION}

Population execution records:
{POPULATION_EXECUTION_RECORDS}

Failure taxonomy:
{FAILURE_TAXONOMY}

Return only:
{
  "task_outcome_summary": {
    "num_successes": 0,
    "num_failures": 0,
    "num_invalid_executions": 0
  },
  "failure_modes": [
    {
      "category": "...",
      "trigger_condition": "...",
      "failed_procedure": "...",
      "observable_evidence": ["..."],
      "verifier_signal": "...",
      "affected_dependency": "...",
      "affected_tool_or_artifact": "...",
      "recoverability": "recoverable|partially_recoverable|unsupported"
    }
  ],
  "normalized_descriptor": {
    "tool_interface": [],
    "schema_handling": [],
    "information_availability": [],
    "evidence_consistency": [],
    "temporal_or_long_range_dependency": [],
    "misleading_success_signal": [],
    "artifact_correctness": [],
    "recovery_behavior": []
  }
}
```

D.7 Data Generation and Pseudo-Labeling Prompts

D.7.1 Structured Challenge Mutation

Purpose. Generate candidate challenges by mutating semantic, environmental, or dependency descriptors while jointly updating the instruction, environment, and verifier.

Inputs. {PARENT_TASK}, {TASK_DESCRIPTOR}, {PERTURBATION_FAMILY}, {PERTURBATION_PARAMETERS}, and summarized uncovered failures.

Expected Output. A complete candidate task encoded as JSON.

```
Create one executable challenge by applying the specified structured
mutation to the parent task.

Parent task:
{PARENT_TASK}

Task descriptor:
{TASK_DESCRIPTOR}

Perturbation family:
{PERTURBATION_FAMILY}

Perturbation parameters:
{PERTURBATION_PARAMETERS}

Previously uncovered or underrepresented failures:
{FAILURE_COVERAGE_SUMMARY}

The perturbation family may modify one or more of the following:
- boundary values or difficulty parameters;
- tool versions, responses, argument conventions, or output schemas;
- field names, nesting, typing, or schema organization;
- available, missing, delayed, or indirectly recoverable information;
- agreement or conflict among evidence sources;
- temporal, cross-step, or long-range dependencies;
- intermediate outputs that appear successful but violate final
  requirements.

Requirements:
1. Apply the mutation consistently to the instruction, environment,
   resources, and verifier.
2. Preserve a well-defined semantic objective.
3. Include all information and capabilities needed by at least one
   supported solution.
4. Avoid changes that differ only lexically from archived tasks.
5. Do not reveal a reference solution or protected validation artifact.
6. Make every verifier condition traceable to an explicit task
   requirement.
7. Record all changed fields and dependencies.

Return only:
{
  "instruction": "...",
  "environment": {
    "tools": [...],
    "resources": [...],
    "constraints": [...]
  },
  "verifier": {
    "type": "...",
    "checks": [...],
```

```
    "pass_condition": "..."
  },
  "descriptor": {
    "semantic": {...},
    "environment": {...},
    "dependencies": {...}
  },
  "mutation_record": {
    "family": "...",
    "parameters": {...},
    "changed_components": ["instruction", "environment", "verifier"],
    "targeted_failure_modes": ["..."]
  }
}
```

D.7.2 Reference Procedure Generation

Purpose. Construct a candidate reference procedure for solvability validation. The generated procedure is isolated from skill execution and refinement.

Inputs. The candidate instruction, environment, tools, resources, constraints, and verifier specification.

Expected Output. An executable procedure and its predicted verifier outcome.

```
Determine whether the candidate task admits a supported solution in the
specified environment. Construct a concrete procedure that can be
executed and checked independently.

Candidate task:
{CANDIDATE_TASK}

Available tools and resources:
{ENVIRONMENT_SPEC}

Verifier:
{VERIFIER_SPEC}

Requirements:
1. Use only the supplied tools, resources, and observable information.
2. Satisfy every explicit task requirement and verifier check.
3. Do not assume undocumented tool behavior or unavailable data.
4. Identify any nondeterministic step and state how it will be checked.
5. If no supported procedure exists, identify the missing capability,
   resource, or contradictory requirement.

Return only:
{
  "status": "candidate_solution|unsupported",
  "procedure": [
    {
      "step": 1,
      "action": "...",
      "inputs": {...},
      "expected_observation": "...",
      "success_check": "..."
    }
  ],
  "final_artifact_specification": "...",
  "predicted_verifier_result": "pass|fail|unknown",
```

```
  "unsupported_reason": null
}
```

D.7.3 CONSENSUS PSEUDO-LABELING

Purpose. Aggregate independently generated and verified candidate solutions into a solvability label and confidence estimate $c_{\text{sol}}(x)$.

Inputs. The candidate task, isolated candidate solutions, execution outcomes, and verifier results.

Expected Output. A JSON solvability decision with an agreement-based confidence score.

```
Assess task solvability from independently produced candidate solutions
and their execution results. Do not infer solvability from unsupported
claims or shared wording alone.

Candidate task:
{CANDIDATE_TASK}

Independent candidate solutions and checks:
{CANDIDATE_SOLUTIONS}

Return only:
{
  "solvability": "solvable|unsolvable|uncertain",
  "confidence": 0.0,
  "accepted_solution_ids": ["..."],
  "rejected_solution_ids": ["..."],
  "agreement_basis": [
    "executable_pass",
    "reproducible_trajectory",
    "independent_solution_agreement"
  ],
  "specification_issues": ["..."],
  "infrastructure_issues": ["..."],
  "missing_capabilities_or_resources": ["..."]
}
```

D.8 EVALUATION PROMPTS

D.8.1 SPECIFICATION CONSISTENCY CHECK

Purpose. Evaluate $V_{\text{spec}}(x)$ by checking whether the instruction, environment, and verifier define the same task.

Inputs. The complete candidate task.

Expected Output. A binary specification-validity decision and an issue list.

```
Check whether the candidate task is internally consistent.

Candidate task:
{CANDIDATE_TASK}

Verify that:
1. Every required output is defined by the instruction.
2. Every verifier check corresponds to an explicit requirement.
3. Tool names, arguments, schemas, files, and resources referenced by the
```

```
   instruction exist in the environment.
4. Mutated field names and dependencies are used consistently.
5. Requirements do not contradict one another.
6. The task has an unambiguous pass condition.

Return only:
{
  "specification_valid": true,
  "issues": [
    {
      "component": "instruction|environment|verifier|dependency",
      "description": "...",
      "severity": "error|warning"
    }
  ]
}
```

D.8.2 EXECUTABLE INFRASTRUCTURE CHECK

Purpose. Evaluate $V_{\text{exec}}(x)$ by checking that the environment, tools, and verifier can be instantiated and executed.

Inputs. The candidate environment and verifier, together with infrastructure test results.

Expected Output. A binary executability decision and structured diagnostics.

```
Assess whether the candidate task can be instantiated and evaluated in
the supplied execution environment.

Candidate environment and verifier:
{CANDIDATE_ENVIRONMENT_AND_VERIFIER}

Infrastructure test results:
{INFRASTRUCTURE_TEST_RESULTS}

Check:
1. Tool availability and schema validity.
2. Resource and file availability.
3. Argument, type, and return-schema compatibility.
4. Verifier execution and determinism within declared tolerances.
5. Isolation from protected evaluation data.
6. Reproducibility under the declared random seed and environment state.

Return only:
{
  "executable": true,
  "failed_checks": [],
  "nondeterministic_components": [],
  "required_repairs": []
}
```

D.8.3 SOLVABILITY VALIDATION

Purpose. Evaluate $V_{\text{solve}}(x)$ and estimate $c_{\text{sol}}(x)$ using an executable reference, reproducible trajectory, or independently verified model consensus.

Inputs. The candidate task, reference procedure or candidate solutions, execution records, and verifier results.

Expected Output. A binary solvability decision, a confidence score in $[0, 1]$, and evidence identifiers.

```
Validate whether the candidate task has at least one supported solution.
The decision must be grounded in execution or independently checked
agreement.

Candidate task:
{CANDIDATE_TASK}

Validation evidence:
{VALIDATION_EVIDENCE}

Decision rules:
1. Accept executable verifier passes produced by a reference procedure
   or reproducible trajectory.
2. When no reference program is available, require agreement among
   independently generated solutions and independently verified
   outcomes.
3. Reject solutions that rely on unavailable information, undocumented
   tool behavior, verifier modification, or leaked protected artifacts.
4. Mark the task uncertain if execution is irreproducible or the
   available evidence is insufficient.
5. Do not expose accepted solutions outside the validation process.

Return only:
{
  "solvable": true,
  "confidence": 0.0,
  "evidence_type":
    "reference_program|reproducible_trajectory|verified_consensus|none",
  "supporting_evidence_ids": ["..."],
  "reproducibility_checks": {
    "runs": 0,
    "passes": 0,
    "agreement_rate": 0.0
  },
  "invalidating_issues": []
}
```

D.8.4 TASK-OUTCOME AUDIT

Purpose. Support the stratified manual or model-assisted audit of challenge solvability and observed failure categories.

Inputs. A task, environment specification, execution trace, produced artifact, verifier result, and proposed failure labels.

Expected Output. A structured audit record. The task pass/fail decision remains determined by the executable verifier whenever one is available.

```
Audit the supplied execution record. Base every judgment on explicit task
requirements, environment evidence, and verifier output.

Task:
{TASK_SPECIFICATION}

Environment:
{ENVIRONMENT_SPEC}

Execution trace:
```

```
{EXECUTION_TRACE}

Produced artifact:
{PRODUCED_ARTIFACT}

Verifier result:
{VERIFIER_FEEDBACK}

Proposed failure labels:
{PROPOSED_FAILURE_LABELS}

Return only:
{
  "verifier_outcome_supported": true,
  "task_specification_valid": true,
  "task_appears_solvable": true,
  "failure_labels": [
    {
      "label": "...",
      "supported": true,
      "evidence": ["..."]
    }
  ],
  "audit_notes": ["..."]
}
```

D.9 Ablation Prompt Variants

D.9.1 Without Learnability

Purpose. Generate and validate challenges without using probe-skill refinement or short-horizon learning progress $\ell^g(x)$.

Inputs. The inputs to structured challenge mutation, excluding probe-update evidence.

Expected Output. The same candidate-task JSON schema as the main mutation prompt.

```
Use the Structured Challenge Mutation prompt, but do not estimate,
describe, or optimize whether verifier feedback supports an attainable
skill update. Generate candidates using the selected perturbation family
and retain them according to the remaining discriminativeness, novelty,
and configured validation criteria.
```

D.9.2 Without Hall-of-Fame Replay

Purpose. Refine skills using only the current curriculum C^g, without archived challenges.

Inputs. The current skill and active-curriculum execution evidence.

Expected Output. A revised skill in the same format as the main refinement prompt.

```
Use the Failure-Grounded Skill Refinement prompt with:
{TRAINING_EVIDENCE} = evidence from the current curriculum only
{ARCHIVED_PERFORMANCE_SUMMARY} = unavailable

Do not reference historical challenges, archived failure descriptors, or
regression results.
```

D.9.3 FAILURE-RATE OBJECTIVE ONLY

Purpose. Generate challenges solely to reduce the current population's success rate.

Inputs. The parent task, available perturbations, and current population outcomes.

Expected Output. The adversarial candidate-task JSON schema.

```
Generate a task variant that maximizes the observed or predicted failure
rate of the current skill population.

Parent task:
{PARENT_TASK}

Current population outcomes:
{POPULATION_EXECUTION_RECORDS}

Available perturbations:
{PERTURBATION_FAMILIES}

Do not optimize or report:
- discriminativeness among skills;
- short-horizon learning progress;
- failure novelty or failure coverage;
- solvability confidence.

Return only the candidate-task JSON object defined for adversarial
challenge generation.
```

D.9.4 WITHOUT SOLVABILITY VERIFICATION

Purpose. Run challenge evolution without the independent V_{solve} gate or solvability-confidence threshold.

Inputs. The structured mutation inputs and specification/infrastructure checks.

Expected Output. The standard candidate-task JSON schema.

```
Use the Structured Challenge Mutation prompt and apply specification and
infrastructure checks only. Do not generate a reference procedure, run
the Solvability Validation prompt, request verified model consensus, or
filter candidates using solvability confidence.
```

D.9.5 SINGLE PERTURBATION FAMILY

Purpose. Restrict challenge evolution to one designated perturbation family.

Inputs. The standard mutation inputs and one fixed perturbation family.

Expected Output. The standard candidate-task JSON schema with exactly one mutation family.

```
Use the Structured Challenge Mutation prompt with the following fixed
family:

{FIXED_PERTURBATION_FAMILY}

Do not introduce mutations from any other family. All changes to the
```

```
instruction, environment, and verifier must instantiate this family
only. Set "mutation_record.family" to the fixed family and record no
additional families.
```

D.10 Output Formatting

All machine-consumed generation, validation, descriptor-extraction, and audit prompts require a single valid JSON object with double-quoted keys and strings. The response must not contain Markdown fences, comments, trailing commas, or text preceding or following the JSON object. Floating-point confidence values must lie in $[0, 1]$, and categorical values must use exactly the alternatives listed in the corresponding template.

Agent execution prompts are exempt from the JSON requirement when the task specifies a domain-specific artifact. In that case, the final output must conform exactly to {OUTPUT_SCHEMA}. Tool calls and intermediate observations remain in the execution trajectory and are not copied into the final artifact unless explicitly required by the task. Skill-refinement prompts must preserve the input skill representation and must not include protected solutions, hidden verifier implementations, or task-specific answer content.

E Case Study

We present qualitative cases illustrating how the challenge population changes the procedural behavior encoded in an evolved skill. The examples are drawn from held-out task instances and use previously unseen artifacts, tool responses, and perturbation combinations. To isolate persistent skill behavior, challenge evolution, skill refinement, and verifier feedback are disabled during evaluation. We show representative excerpts rather than aggregate outcomes. File names, identifiers, and cell values are lightly normalized for readability while preserving the operational dependencies that determine correctness.

E.1 Recovering from a Renamed Spreadsheet Schema

The first case combines a tool-interface change with a spreadsheet dependency. The agent must update a quarterly summary workbook using a newly supplied transaction sheet. The workbook contains a protected formula column, a hidden mapping sheet, and a visible summary table. In the evolution environment, the spreadsheet inspection tool returns fields named `sheet_name`, `used_range`, and `formula`. In the held-out environment, the corresponding fields are renamed to `title`, `region`, and `expression`. In addition, the source sheet replaces `Customer ID` with `Account Ref`. The task instruction does not mention either schema change.

A skill evolved on fixed tasks treats the familiar response schema as part of the procedure. After failing to find `sheet_name`, it assumes that workbook inspection returned no worksheets and falls back to editing the first visible grid. Its trace contains the following reasoning:

```
Inspection returned no `sheet_name` entries.
Assume the workbook contains a single active sheet.
Write the recalculated totals into columns B:E of the active sheet.
```

This behavior is locally coherent but incorrect. The active sheet is the transaction sheet, not the quarterly summary. More importantly, the skill never checks whether the destination cells contain formulas or whether an identifier mapping is required. It therefore overwrites source records while leaving the requested summary unchanged.

The SCOPE-evolved skill interprets the missing key as an interface mismatch rather than evidence that the workbook is empty. Its tool-use procedure first discovers semantically equivalent fields, then validates the workbook structure before issuing any write:

```
1. Normalize the inspection response by role:
   worksheet identifier := first available of
       {sheet_name, title, worksheet, tab}
   occupied area := first available of
       {used_range, region, bounds}
   cell formula := first available of
       {formula, expression, fx}

2. Enumerate worksheets and classify each from headers and formulas.
   - `Transactions_New`: row-level records
   - `Q3 Summary`: grouped output with formula cells
   - `_AccountMap`: identifier crosswalk; hidden

3. Resolve `Account Ref` through `_AccountMap` before aggregation.

4. Write only literal input cells in `Q3 Summary`.
   Preserve cells whose normalized metadata contains an expression.

5. Re-read the edited range and verify:
   - summary keys remain unique;
   - protected expressions are unchanged;
   - each written account is present in the crosswalk;
   - displayed totals are consistent with the source rows.
```

The important change is not the addition of a synonym list alone. The evolved skill introduces a recovery branch that treats tool schemas as observations to be interpreted, separates discovery from mutation, and verifies semantic invariants after writing. This procedure transfers to a response format that was not explicitly encoded during refinement because it searches for field roles and workbook structure rather than matching one serialized layout.

E.2 Rejecting a Misleading Success Signal

The second case concerns repository-level maintenance. The task asks the agent to update a configuration loader so that environment variables override file-based defaults without changing the behavior of explicitly supplied command-line arguments. The repository wrapper reports `status: success` whenever the test process exits normally. In the held-out tool version, the same response also contains `failed_cases`, which may be nonempty even when `status` is `success`. This occurs because the wrapper process catches test exceptions and emits a structured report before exiting.

A skill optimized against routine repository tasks stops after observing the top-level status:

```
Patch applied successfully.
Test tool returned `status: success`.
The configuration precedence change is complete.
```

Inspection of the generated patch reveals a second error. The skill moves environment loading after command-line parsing and then unconditionally assigns environment values, causing environment variables to overwrite explicit command-line arguments. The misleading success field prevents the skill from examining the failed precedence test embedded in the structured response.

The SCOPE-evolved skill contains an explicit evidence hierarchy for executable feedback:

```
Execution outcome policy:
- Do not equate transport success or normal process exit with test success.
- Parse case-level outcomes when they are available.
- Treat any reported failed case, uncaught exception, or missing required
  test group as failure, regardless of the wrapper's top-level status.
```

```
- If summary fields conflict, inspect the raw test log and rerun the
  smallest affected test group.
```

It then reads the detailed result:

```
status: success
failed_cases:
  - test_cli_value_has_priority
diagnostic:
  expected: "cli-region"
  observed: "env-region"
```

Rather than applying an unrelated patch, the skill reconstructs the intended precedence from existing call sites and changes the loader to fill only unresolved options. Its final explanation states that command-line values remain authoritative, environment variables fill absent command-line values, and the configuration file supplies the remaining defaults. It reruns both the focused precedence tests and the existing loader tests, checks the case-level results, and only then declares completion.

This case shows why failure-maximizing challenges are insufficient by themselves. A malformed challenge could make the status fields arbitrarily contradictory without admitting a principled response. The selected challenge remains solvable because the detailed case report and raw log provide enough evidence to identify the true execution outcome. Its feedback is also actionable: the failed assertion points directly to a recoverable mistake in precedence handling and motivates a reusable rule for interpreting future tool responses.

E.3 Separating Conflicting Evidence in Document-Grounded QA

The third case requires answering a policy question from several document versions. The provided folder contains a current policy, an obsolete implementation guide, and meeting notes discussing a proposed exception. The user asks whether a request submitted after the policy effective date requires director approval. The obsolete guide contains a direct sentence saying that manager approval is sufficient, while the current policy requires director approval. The meeting notes mention relaxing the requirement but record the change as a proposal pending legal review.

A brittle document-grounded skill retrieves the most lexically similar passage and answers:

```
Manager approval is sufficient because the implementation guide states
that requests may be approved by the requester's manager.
```

The answer quotes a relevant-looking sentence but ignores document authority, effective dates, and proposal status. A purely adversarially generated task can expose this weakness by adding more contradictions, yet such additions may become ambiguous if the materials do not indicate which source governs.

The SCOPE-evolved skill uses a source-resolution procedure before composing the answer:

```
Claim: approval level for a request submitted after the effective date

Candidate evidence:
A. Current policy
   Status: approved
   Scope: all requests submitted on or after the effective date
   Requirement: director approval

B. Implementation guide
   Status: superseded
   Scope: workflow under the previous policy
```

```
   Requirement: manager approval

C. Meeting notes
   Status: proposal only
   Scope: possible future exception
   Requirement: not operative

Resolution:
Use A as governing evidence.
Use B only to explain why older instructions conflict.
Do not treat C as an enacted exception.
```

The resulting answer is specific about both the conclusion and its boundary:

> The request requires director approval because it was submitted after the current policy became effective. The manager-approval statement appears in a superseded implementation guide and does not govern new submissions. The meeting notes discuss a possible exception, but the supplied record identifies it as pending rather than adopted.

The qualitative improvement lies in evidence organization rather than verbosity. The evolved skill attaches status, temporal scope, and authority to each passage before resolving the claim. It also avoids merging mutually incompatible statements into a hedged answer. Because the challenge has a supported reference resolution, verifier feedback can identify which dependency was violated, such as using a superseded source or treating a proposal as operative, instead of merely marking the final answer incorrect.

E.4 Composing Missing-Data and Long-Range Dependency Handling

The fourth case is a scientific data-analysis task. The agent receives measurement files for several experimental runs and must produce a cleaned summary. A manifest states that sensor readings are valid only after calibration, and a later appendix specifies that one instrument changed units midway through collection. One run lacks its calibration record. The execution tool can still parse every file and produce a plausible summary, so syntactic success does not guarantee a valid analysis.

A task-specific skill converts all numeric columns and silently includes the uncalibrated run. It also applies one conversion factor to the entire instrument history because the unit change is described far from the file schema. The output is well formed, but two semantic dependencies are violated: calibration is a prerequisite for inclusion, and conversion depends on collection time.

The SCOPE-evolved skill externalizes these dependencies as preconditions:

```
For each run:
  require manifest entry
  require calibration record valid at acquisition time
  determine instrument unit regime from acquisition timestamp
  apply the conversion associated with that regime
  preserve the raw value and conversion provenance
  exclude rather than impute when required calibration is absent

Before summarization:
  verify that every included row has
    - a resolved run identifier,
    - a valid calibration reference,
    - a resolved unit regime, and
    - a recorded transformation.

If a required dependency is missing:
  continue processing unaffected runs;
```

```
  report the excluded run and the missing dependency;
  do not fabricate a calibration value from neighboring runs.
```

Its final output excludes the affected run and includes a concise limitation:

> The summary includes only runs with calibration records valid at acquisition time. One run was excluded because its required calibration record is absent. Measurements collected before and after the instrument change were normalized using their respective unit regimes; raw values and transformation provenance were retained in the output artifact.

This response is preferable to either unconditional completion or total refusal. The skill preserves usable results, makes the exclusion auditable, and does not infer unsupported scientific values. The case also illustrates compositional generalization. Missing information and temporal unit changes can each be handled independently, but their combination requires the agent to maintain per-run provenance while applying a dependency discovered elsewhere in the document set.

E.5 How the Curriculum Changes the Skill Artifact

Across these cases, the accepted revisions do not memorize isolated answers. They introduce procedural guards that operate at distinct stages of agent execution:

```
OBSERVE
  Normalize tool responses by semantic role rather than fixed key name.
  Preserve raw observations needed to audit later decisions.

RESOLVE
  Identify prerequisites, authority relations, temporal scope, and
  conflicts before acting.
  Distinguish missing evidence from negative evidence.

ACT
  Minimize irreversible edits.
  Preserve formulas, protected regions, explicit user choices, and
  unaffected records.
  Apply transformations only after their preconditions are resolved.

VERIFY
  Inspect granular outcomes rather than relying on top-level success.
  Re-read edited artifacts and check task-specific invariants.
  Treat conflicting status fields as a reason for targeted diagnosis.

REPORT
  State exclusions and unresolved dependencies explicitly.
  Do not convert proposals, guesses, or neighboring observations into facts.
  Separate completed work from residual limitations.
```

The challenge population is essential to obtaining this artifact. Boundary cases expose a concrete weakness while retaining enough execution evidence for a targeted revision. Structured mutations then move the same weakness across tool schemas, document layouts, dependency lengths, and feedback channels, discouraging surface-level patches. Finally, hall-of-fame replay tests whether a new recovery rule damages previously acquired behavior. For example, a revision that handles renamed spreadsheet fields but begins rewriting formula cells is rejected when replayed against the earlier protected-formula challenge. Similarly, a repository revision that distrusts every success response and repeatedly reruns completed tests is not retained merely because it handles one contradictory wrapper response. The resulting skill therefore encodes conditional recovery procedures rather than universal caution or challenge-specific exceptions.