EVIDENCE OVER PLANS: ONLINE TRAJECTORY VERIFICATION FOR SKILL DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills can remarkably improve task success rates by using human-written procedural documents, but their quality is difficult to assess without environment-grounded verification. Existing skill generation methods heavily rely on preference logs rather than direct environment interaction, often yielding negligible or even degraded gains. We identify that it is a fundamental timing bottleneck: robust skills should be *posterior-based*, distilled from empirical environment interaction rather than prior plans. In this study, we introduce the **Posterior Distillation Index (PDI)**, a trajectory-level metric that quantifies how well a distilled skill is grounded in the task-environment evidence. To operationalize PDI, we present **SPARK** (**S**tructured **P**ipelines for **A**utonomous **R**unnable tas**K**s and s**K**ill generation) for preserving task execution evidence towards full trajectory-level analysis. SPARK generates environment-verified trajectories used to compute PDI; SPARK + PDI applies the index as an online diagnostic and intervention signal to ensure posterior skill formation. Across 86 runnable tasks, SPARK-generated skills consistently surpass no-skill baselines and outperform human-written skills on student models (inference cost up to 1,000$\times$ cheaper than teacher models). These findings show that PDI-guided distillation produces efficient and transferable skills grounded in the task-environment interaction. Our code is available at `https://anonymous.4open.science/r/spark_anonymous-2C82/`.

1 INTRODUCTION

Large Language Model (LLM) agents are increasingly deployed on wide-ranging interactive software tasks (Yao et al., 2023; Shinn et al., 2023; Zhou et al., 2025b;a; Jin et al., 2024). An emerging strategy is to equip agents with *skills* (Li et al., 2026), defined as in-context procedural documents that encode task-specific knowledge such as execution ordering, environment constraints, and verifier contracts. Benchmarks (Li et al., 2026; Chen et al., 2026a) demonstrate that prior-based, human-written skills can remarkably boost task success rates, making such reusable procedural knowledge a promising abstraction for building scalable agent systems.

Yet a pessimistic picture appears that the quality of these skills is nearly impossible to assess without environment-grounded verification (Li et al., 2026). Researchers propose lifelong skill accumulation (Yang et al., 2026), large-scale skill cataloging (Liang et al., 2026), or RL-based skill evolution (Xia et al., 2026). These studies lack trajectory verification and solely rely on preference logs. As a result, they can not produce transferable skills between teacher and student models at different capabilities. This raises a fundamental question:

What makes skill distillation transferable across tasks and models efficiently and reliably?

We identify that a key bottleneck is the timing of skill formation relative to the task-environment interaction. Existing skill generation (Wang et al., 2023; Liang et al., 2026) asks the model to prescribe how a task should be solved before interacting with the environment, yielding skills dominated by generic priors and heuristics. Instead, we propose that effective procedural knowledge should be *posterior-based* since its creation time: it must encode environment-specific constraints, execution dependencies, and failure modes discoverable only through environmental interactions. Once distilled, this posterior experience becomes reusable and verifiable guidance for future agent development.

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.

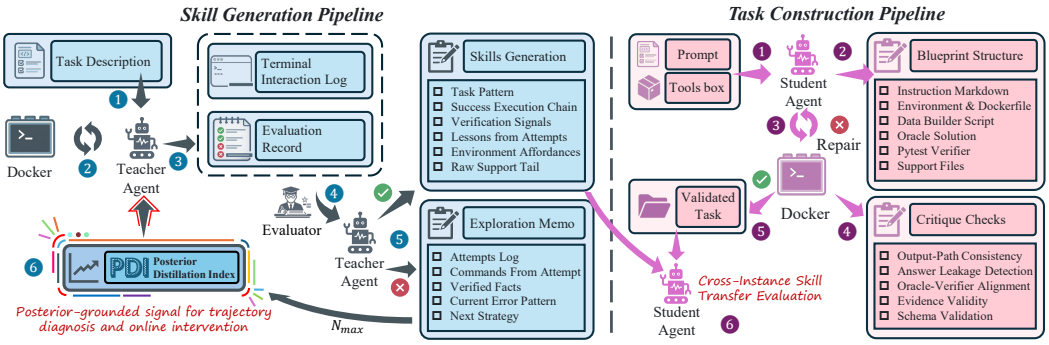


Figure 1: SPARK and PDI overview. **1) Left (Skill Generation)**: Starting from a task description, a teacher agent (e.g., Claude Opus 4.6) interacts with a Dockerized environment (up to $N_{\max}$ attempts) and updates an exploration memo from execution feedback. Upon success, the full trajectory trace is distilled into `SKILL.md`. With SPARK + PDI, a proxy can trigger targeted interventions after failure. **2) Right (Task Construction):** To validate the quality of skill generation, this is a supporting pipeline that converts prompt into oracle-verified benchmark instances via blueprint generation and critique checks. Then, a student agent (e.g., GPT 5.4 mini) evaluates the generated skill on these independently constructed tasks, testing whether PDI-grounded knowledge transfers across tasks rather than overfitting to the teacher's original trajectory.

We introduce the **Posterior Distillation Index (PDI)**[1] as a principled metric that quantifies whether skills are grounded in environment-verified evidence rather than purely prior-based plans. By design, PDI aggregates three interpretable trajectory-level signals, including execution grounding, plan copying, and memo ossification, using an equal-weight linear combination. We avoid nonlinear fitting to ensure benchmark independence. To operationalize PDI, we present **SPARK** (**S**tructured **P**ipelines for **A**utonomous **R**unnable tas**K**s and s**K**ill generation). In Figure 1, SPARK generates the environment-grounded trajectories from which PDI is computed; SPARK + PDI uses the index as an online intervention signal during skill generation. Our main contributions are as follows.

(1) Trajectory-verified Metric: PDI introduces a trajectory-level measure whether generated skills are grounded in task-environment evidence rather than unverified prior plans. PDI also provides a mechanistic assessment of which trajectories yield truly transferable skills (Section 3.2).

(2) Skill Generation: SPARK makes skill generation transparent and analyzable by preserving execution logs, verifier signals, and memo histories towards full trajectory-level analysis (Section 3.1.2).

(3) Online Intervention: A memo-based proxy of PDI enables an online monitoring and intervention during skill exploration. Our primary **with/without PDI** ablation quantifies the performance gain enabled by this online intervention (Section 4.3).

(4) Trajectory Benchmark: We provide a trajectory-level benchmark with full execution traces, verifier signals, memo histories, and distilled skills. This collected environment-verified evidence allows for evaluating skill transfer ability across environment interactions.

2 Related Work

Agent skill acquisition and evaluation. Agent skill acquisition has increasingly shifted from collecting experiences to distilling and deploying skills across tasks. Voyager (Wang et al., 2023) accumulates executable skills in an embodied environment. ExpeL (Zhao et al., 2024) stores natural-language lessons from prior failures. Growing attempts distill the experience into transferable skill libraries, including AutoRefine (Qiu et al., 2026), Trace2Skill (Ni et al., 2026), AutoSkill (Yang et al., 2026), and CUA-Skill (Chen et al., 2026b). SkillRouter (Zheng et al., 2026) further addresses the deployment by selecting highly-relevant skills from large registries. Beyond skill acquisition, skill evaluation is crucial to assess the quality of skills in real-world applications. SkillsBench (Li et al., 2026) evaluates when curated skills help at inference time. SkillWild (Liu et al., 2026) studies more realistic retrieval-and-adaptation settings. Unlike these efforts, SPARK emphasizes interaction-grounded skill documents, cross-instance skill transfer, and trajectory-level analysis.

[1]'Posterior' denotes knowledge distilled from environment interaction. No probabilistic or Bayesian update is assumed.

LLM agents with external feedback. SPARK follows the interact–reflect–retry line of agentic system development, including ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), and ETO (Song et al., 2024). These studies mainly improve decision making or trajectory revision within a task-solving episode (one attempt). By contrast, we emphasize how agent execution traces can be distilled (Hinton et al., 2015; Furlanello et al., 2018; Jin et al., 2024) from the environment interaction that can transfer across tasks, turning trial-and-error records into a reusable knowledge base.

Agent tooling and skill utility. LLM agent tooling and its transfer capability have evolved from learning tool policies to structuring skills. For instance, Toolformer (Schick et al., 2023) and ToolLLM (Qin et al., 2024) focus on learning to invoke tools. Meanwhile, CREATOR (Qian et al., 2023) and LATM (Cai et al., 2024) show that stronger models can synthesize reusable artifacts for weaker ones. Skill-oriented frameworks go further by packaging capabilities into modular, self-contained units with clear invocation protocols. SkillCraft (Chen et al., 2026a) studies tool-usage skills, SkillNet (Liang et al., 2026) organizes skills as reusable units, and SkillX (Wang et al., 2026) automatically constructs multi-level skill knowledge bases. In our study, SPARK highlights that the transferred artifact is a natural-language `SKILL.md` document rather than the executable code.

3 METHOD

We study whether distilled skills are grounded in environment-verified evidence or merely reflect prior plans. To start, a teacher model generates skills from environment-interacted trajectories for use by a student model (here "teacher" denotes the behavior agent that produces exploration trajectories and "student" is the agent that consumes the learnable ones; no capacity asymmetry is implied). Our goal is to measure when these trajectories yield transferable procedural knowledge. We define a trajectory-level score (i.e., Posterior Distillation Index (PDI)) that measures the degree to which the generated skill reflects posterior evidence rather than prior intent. In Figure 1, SPARK produces trajectories for PDI computation. SPARK + PDI uses the index as an online signal during exploration. We describe skill generation (Section 3.1), PDI (Section 3.2), and task construction (Section 3.3).

3.1 SKILL GENERATION PIPELINE

In Figure 1 left, ❶ the teacher agent interacts with a Dockerized environment up to $N_{\max}$ attempts. At each attempt, ❷ teacher agent issues shell commands and observes terminal output. Then ❸ teacher agent produces a *Terminal Interaction Log* and *Evaluation Record*. ❹ When an attempt succeeds from the evaluator, ❺ the full trajectory is distilled into a reusable document `SKILL.md` (Section 3.1.1); when it fails, the structured *exploration memo* (Section 3.1.2) is updated to capture the resulting evidence and revise the next strategy. This closed-loop design makes the generated skill explicitly traceable to the environment interaction. ❻ These records provide the inputs for PDI (Section 3.2) and its online proxy. Below we detail major blocks of evidence-driven skill generation and exploration memo. A full procedure is provided in Appendix H.1.

3.1.1 EVIDENCE-DRIVEN SKILL GENERATION

When a task is accomplished, SPARK assembles six structured evidence blocks to provide the skill-generation model with a comprehensive view of the solution path and obstacle encountered. This multi-source evidence design allows the generated skill toward reusable procedural structure rather than task-instance surface form, while establishing an explicit interface for evaluating predictive validity of individual evidence sources with respect to downstream skill utility.

In Figure 1 left: **1) Task Pattern** is a condensed task instruction capturing the problem class. **2) Execution Chain** offers key commands from the successful attempt, after semantic classification, importance scoring, and low-signal filtering. **3) Verification** records tests passed and the final reward. **4) Lessons** store repeated failure patterns and confirmed cautions distilled from the full attempt history. **5) Environment** offers runtime context from the Dockerfile (base image, packages, SDKs). **6) Raw Support Tail** shows the tail of the agent's stdout from the successful run.

3.1.2 EXPLORATION MEMO

A single agent attempt can produce excessive amounts of tokens of raw stdout (standard output) and shell commands. Carrying this history forward verbatim would quickly exhaust the context window and dilute the model's attention. We therefore introduce the *exploration memo* as a structured summary state that the teacher agent maintains across attempts. This design keeps the exploration process interpretable while preventing low-value repetition from overwhelming the context window. After each failed attempt, the teacher agent *completely rewrites* the memo using recent agent's commands and a structured test summary, rather than appending to the old version. Its *Verified Facts* and *Next Strategy* fields separate environment-confirmed outcomes from planned actions; the full five-section structure is given in Appendix H.1.

Notably, we preserve the full historical memo so that cross-attempt patterns remain available for subsequent analysis. SPARK's exploration-quality techniques are summarized in Appendix H.2.1.

3.2 POSTERIOR DISTILLATION INDEX (PDI)

To formalize PDI, we compare *token-frequency distributions* between trajectory segments and the final skill to capture both token presence and its emphasis.

Token-distribution setup. For any text segment x and a fixed vocabulary $\mathcal{V}$ constructed from the trajectory, let $c_x(w)$ denote the count of token w. We define the additively smoothed empirical distribution $P_x(w) = \frac{c_x(w)+\alpha}{\sum_{v\in\mathcal{V}} c_x(v)+\alpha|\mathcal{V}|}, \quad \alpha > 0$. The Kullback–Leibler (KL) divergence $\mathrm{KL}(P \,\|\, Q) = \sum_w P(w) \log \frac{P(w)}{Q(w)}$ captures token-frequency differences but is asymmetric and unbounded. We adopt the Jensen–Shannon divergence (JSD), a symmetrized, bounded variant:

$$\mathrm{JS}(P_x, P_y) = \tfrac{1}{2}\,\mathrm{KL}(P_x \,\|\, M) + \tfrac{1}{2}\,\mathrm{KL}(P_y \,\|\, M), \qquad M = \tfrac{P_x+P_y}{2}, \tag{1}$$

which lies in $[0, 1]$ under base-2 logarithms. We convert divergence to similarity via $\psi(P_x, P_y) = 1 - \mathrm{JS}(P_x, P_y)$, so that higher values indicate stronger distributional alignment. We examine three dynamic properties of each trajectory below:

- **Plan Copying** (ϕ_{plan}). Let P_P denote the token distribution of all accumulated *Next Strategy* sections from the exploration memo, and P_s the token distribution of the final `SKILL.md`. We define plan copying as $\phi_{\text{plan}} = \psi(P_P, P_s)$. High ϕ_{plan} indicates that the skill's token emphasis closely mirrors the memo's planned actions rather than encoding independently verified knowledge.
- **Execution Grounding** (ϕ_{exec}). Let P_E denote the distribution of agent commands from the successful execution trace. We define execution grounding as $\phi_{\text{exec}} = \psi(P_E, P_s)$. High ϕ_{exec} indicates that the skill concentrates on operations by operations validated by the environment.
- **Memo Ossification** (ϕ_{oss}). We measure two cross-attempt stability signals: (i) the distributional similarity of *Verified Facts* between consecutive reflections, $\psi(P_{v_{i-1}}, P_{v_i})$, and (ii) the distributional persistence of failed test sets, $\psi(P_{f_{i-1}}, P_{f_i})$. We define memo ossification as the equal-weight average $\phi_{\text{oss}} = \frac{1}{2}\,\overline{\psi(P_{v_{i-1}}, P_{v_i})} + \frac{1}{2}\,\overline{\psi(P_{f_{i-1}}, P_{f_i})}$, where $\overline{(\cdot)}$ denotes the mean across consecutive attempt pairs. High ϕ_{oss} indicates that the trajectory is stuck, repeating the same task understanding and failure patterns without substantive update.

We aggregate these three characteristics into a single composite score. Let $z(\cdot)$ denote z-score normalization across all iterative tasks. The *Posterior Distillation Index* (PDI) is defined as:

$$\mathrm{PDI} = z(\phi_{\text{exec}}) - z(\phi_{\text{plan}}) - z(\phi_{\text{oss}})\,. \tag{2}$$

Intuitively, a high PDI indicates that the skill is distributionally aligned with environment-verified evidence (high ϕ_{exec}), diverges from memo-level plans (low ϕ_{plan}), and stems from a trajectory whose task understanding evolves rather than repeating the same failure pattern (low ϕ_{oss}). We use an equal-weight linear form because PDI is designed to be an *interpretable* and *transferable* diagnostic index rather than a benchmark-specific predictive model. Throughout the experiments, we instantiate PDI with $\mathcal{D}$=JS and a smoothing parameter α=0.002; sensitivity to α is analyzed below.

Online PDI-guided intervention. The offline index uses a completed trajectory and its final skill. During exploration, the proxy uses the current memo and observed commands and test feedback. From the second reflect step onward, each component is normalized over that task's steps observed so far; the first step uses the raw component difference. Concretely, after each reflect step k SPARK + PDI computes a weighted proxy-PDI $\hat{d}_k = w_k \cdot \mathrm{PDI}_k$, where $w_k = \min(1,\, k/W)$ is a linear warm-up ramp (W=2) that suppresses noisy early signals. When $\hat{d}_k < \tau$ (τ=−0.5), a *soft* intervention injects prompt-only guidance encouraging broader hypothesis revision while keeping the metric hidden from the agent. If two consecutive steps both trigger ($\hat{d}_{k-1} < \tau$ and $\hat{d}_k < \tau$), SPARK + PDI escalates to a *strong* intervention that additionally withholds the previous *Next Strategy* section and instructs the agent to anchor on *Verified Facts* and *Current Error Pattern* rather than continuing the stale plan. The full procedure, including the PDI-guided branch, is given in Appendix H.1; trajectory-level case studies are also provided in Appendix C.

3.3 Task Construction Pipeline

In Figure 1 right, ❶ task-construction pipeline converts a prompt-level task idea into a structured task blueprint via three sequential stages. ❷ A blueprint stage that exposes explicit intermediate structure; ❸ a repair stage that iteratively fixes violated constraints in the blueprint; ❹ a critique stage that checks semantic consistency and executable constraints; and ❺ a validation stage that accepts only tasks passing deterministic oracle verification. Overall, task generation is a build-and-verify process that ensures each generated task is self-contained rather than a single-LLM-generated step. Further, ❻ the pipeline supports cross-instance transfer evaluation by constructing unseen task variants from high-level specifications, on which a student agent is then deployed to test whether skills distilled from one trajectory generalize to related but independently constructed tasks.

4 Experiments

We adopt the task suite from the SkillsBench benchmark (Li et al., 2026), which provides 86 tasks spanning 11 domains (e.g., Software Engineering, Finance, and Cybersecurity), each paired with a natural-language instruction, a Docker execution environment, and a deterministic pytest oracle. To broaden our evaluation, SPARK (Section 3.3) can synthesize runnable task variants from natural-language prompts; we use this capability to extend the evaluation pool (n = 300) and validate the generality power (Section 4.4). For cross-domain generalization, we evaluate SPARK on ALFWorld (Shridhar et al., 2021), a text-based household environment differing from terminal-based programming tasks. A capable teacher model interacts with the environment over up to $N_{\max}$=7 attempts; the objective is to distill the procedural knowledge embedded in these interactions into a reusable skill document that improves a weaker student model (statistics are reported in Appendix G). The human-written skills used as a comparison baseline are those provided by SkillsBench (Li et al., 2026); we use them as-is without modification.

Evaluation metrics. Each task is associated with a deterministic pytest oracle that returns a reward $r \in [0, 1]$, where r=1 indicates full task completion. We report two quantities throughout the study: **task reward** $r_{m,t}$, namely the reward obtained by student model m on task t under a given condition (baseline, with SPARK skill or with human-written skill); and **skill gain** $\Delta r_{m,t} = r^{\text{+skill}}_{m,t} - r^{\text{base}}_{m,t}$, namely the change in the reward attributable to the skill, where positive values indicate improvement and negative values indicate degradation. We aggregate across tasks via the mean reward $\bar{r}_m = \frac{1}{|\mathcal{T}|}\sum_t r_{m,t}$ and the mean skill gain $\overline{\Delta r}_m = \frac{1}{|\mathcal{T}|}\sum_t \Delta r_{m,t}$.

4.1 Main Results

Figure 2 reports the mean reward $\bar{r}$ of seven student models under three conditions: no skill (baseline), SPARK-generated skills (without online PDI intervention), and human-written skills. Horizontal lines mark the unaided performance of GPT-5.4 and Claude Opus 4.6 as teacher models during exploration.

Two striking findings stand out. **First**, SPARK-generated skills consistently outperform human-written skills on the majority of student models: for instance, GPT-5.4-mini guided by SPARK skills reaches $\bar{r}$=0.52, surpassing the human-written condition (0.47). **Second**, several small student models

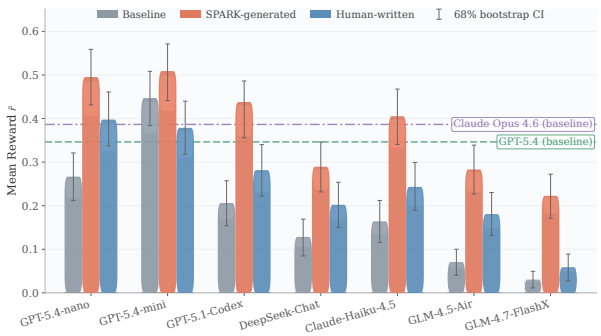


Figure 2: Mean reward $\bar{r}$ across seven student models under three conditions: no skill (baseline), SPARK-generated skills (without online PDI intervention), and human-written skills. Horizontal dotted lines mark the interaction-free performance of two strong teacher models.

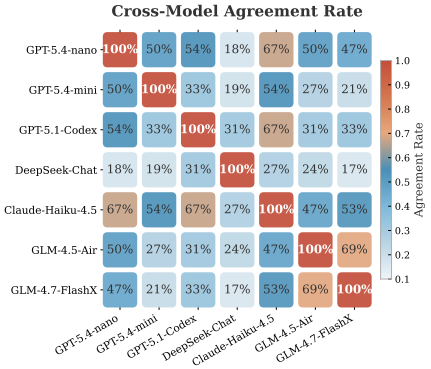


Figure 3: Cross-model agreement rate $A(m_i, m_j)$ among seven student models. Off-diagonal values are generally moderate.

equipped with SPARK skills exceed the unaided performance of the teacher models themselves. For instance, GPT-5.4-nano with SPARK skills (0.41) outperforms Claude Opus 4.6 without skills (0.37). Overall, SPARK-distilled skills capture environment-specific insights that strong models alone miss, substantially narrowing the gap between weaker and stronger models. SPARK also carries a favorable cost profile: although strong-teacher exploration is expensive (Appendix G), the upfront cost amortizes across repeated cheap student runs—student inference costs as little as $0.02 per task, over $1{,}000\times$ cheaper than teacher exploration—in some cases enabling weaker students to even surpass the teacher's own interaction-free performance.

4.2 Trajectory-Level Analysis

Although SPARK-generated skills improve student performance (Section 4.1), they do not reveal *why* some trajectories yield useful skills while others do not. Our central answer is the Posterior Distillation Index (PDI), which measures whether a distilled skill is grounded in posterior execution evidence rather than stale prior plans. Before testing PDI, we first examine simpler explanations below (Appendix B details further trajectory correlation analysis).

4.2.1 Do a Few High-Quality Skills Drive All Gains?

A handful of universally strong skills can not further improve skill utility with a moderate gain (<60%), calling for more efforts on scale, divergence, and trajectory exploration.

For each pair (m_i, m_j), we restrict to tasks where both fail the baseline ($r^{\text{base}}_{m,t}{=}0$) and at least one shows a non-zero skill-induced change, and define the agreement rate as $A(m_i, m_j) = \frac{\left|\{t|\text{sign}(\Delta r_{m_i,t})=\text{sign}(\Delta r_{m_j,t})\}\right|}{\left|\{t|\Delta r_{m_i,t}\neq 0 \vee \Delta r_{m_j,t}\neq 0\}\right|}$. Figure 3 shows that off-diagonal rates are mostly moderate (<60%). This calls for trajectory-level predictors of skill quality across models at different capacities.

4.2.2 Does More Evidence or More Attempts Improve Skills?

No, more evidence and attempts do not by themselves improve skills; excessive compression hurts and gains become unstable after the first a few rounds.

We next ask whether *quantitative* factors alone, such as the volume of exploration evidence or the number of attempts, are sufficient to determine skill quality.

Compression ratio. Let L_{traj} denote the total character length of the agent's execution outputs (stdout) across all attempts, and L_{skill} the character length of the distilled `SKILL.md`. We define the

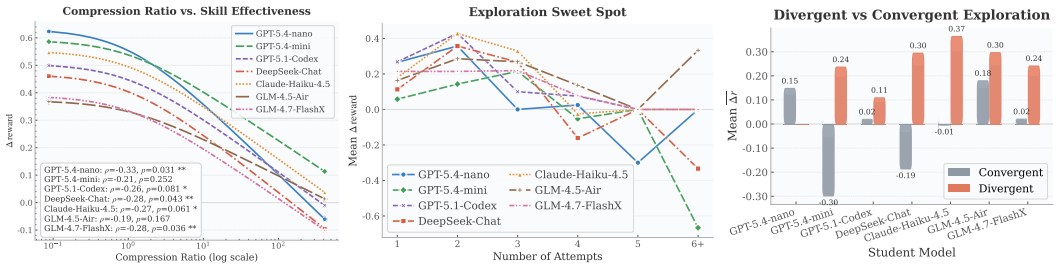


Figure 4: Three complementary views of skill quality determinants. **Left:** Compression ratio ρ_c vs. per-pair Δr; excessive compression degrades skill effectiveness. **Middle:** Mean Δr as a function of the number of exploration attempts; gains are stable for the first three attempts and become volatile thereafter. **Right:** Mean $\overline{\Delta r}$ per student model for skills distilled from convergent vs. divergent teacher trajectories.

compression ratio as $\rho_c = L_{\text{traj}}/L_{\text{skill}}$. Figure 4 (left) plots ρ_c (log scale) against per-pair Δreward for all baseline-unsolved student×task pairs. Across most student models, the Spearman correlation between ρ_c and Δreward is negative, indicating that excessive compression discards actionable details and degrades skill effectiveness.

Exploration attempt budget. For each task, let K denote the number of execution attempts the teacher performed before skill generation. For each student model m, we group all baseline-unsolved tasks by K and compute the mean skill gain per bin as $\overline{\Delta r}_m(K) = \frac{1}{|\mathcal{T}_{m,K}|}\sum_{t\in\mathcal{T}_{m,K}} \Delta r_{m,t}$, where $\mathcal{T}_{m,K}$ is the set of tasks for which student model m fails the baseline and the teacher uses exactly K attempts. Figure 4 (middle) shows that gains are positive and relatively stable for $K \leq 3$, but become volatile and model-dependent beyond that point. More exploration therefore does not by itself explain why a distilled skill becomes useful.

4.2.3 Does Convergent or Divergent Exploration Produce Better Skills?

Divergent trajectories yield more transferable skills, whereas convergent ones encode teacher-specific refinements that weaker students cannot reproduce.

We next ask whether the instability above depends on *how* the teacher explores, namely whether it stays within one line of attack or pivots to different ones.

Exploration-mode classification. Let μ_i denote the memo text after the i-th reflection. We measure the Jaccard similarity between successive memos as $J_i = \frac{|\,\text{tok}(\mu_{i-1})\cap\text{tok}(\mu_i)\,|}{|\,\text{tok}(\mu_{i-1})\cup\text{tok}(\mu_i)\,|}$, where $\text{tok}(\cdot)$ denotes word-level tokenization. We then fit a linear trend $J_i \approx \alpha + \gamma \cdot i$ and use the slope γ to classify each task: **Convergent** ($\gamma > 0$), where successive memos become increasingly similar, and **Divergent** ($\gamma \leq 0$), where memos grow less similar over time.

Figure 4 (right) reports the mean skill gain $\overline{\Delta r}$ for each student model, grouped by exploration mode. Across nearly all student models, skills distilled from divergent trajectories yield substantially larger gains than those from convergent ones; for several models, convergent-trajectory skills even produce negative Δr relative to the no-skill baseline.

This contrast reveals that exploration quality depends not only on search volume, but also on whether the trajectory accumulated genuinely new posterior evidence rather than refining the same plan.

4.2.4 What Separates Useful Iterative Skills from Useless Ones?

Environment-verified skills consistently outperform human-authored baselines, while plan-copying skills without external task-update signals fall below even interaction-free ones.

The prior analyses motivate testing whether PDI (Section 3.2) distinguishes useful iterative skills from ineffective ones across both trajectory groups and individual student-task pairs.

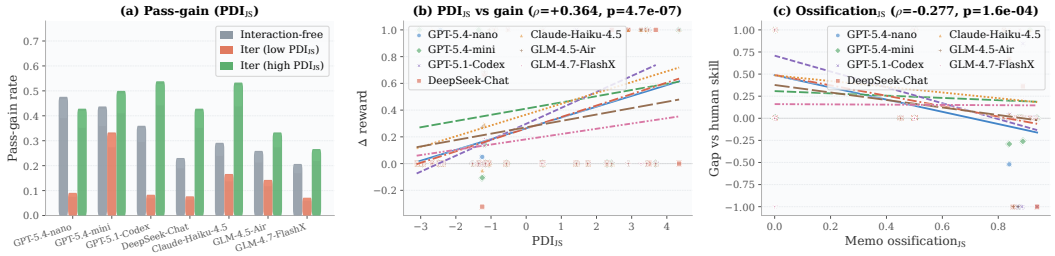


Figure 5: Trajectory-level analysis of skill quality using divergence-based PDI (α=0.002). (a) Pass-gain rate by trajectory group across seven student models: high-PDI iterative trajectories consistently outperform both interaction-free and low-PDI iterative skills. (b) PDI vs. per-pair Δr (ρ=+0.364, $p<10^{-6}$). (c) Memo ossification vs. gap relative to human-written skills (ρ=−0.277, $p<10^{-3}$): trajectories that repeat stale task understanding produce skills that fall further behind human baselines.

Table 1: Mean skill gain ($\overline{\Delta r}$) over no-skill baseline and gap to human-written skills by plan copying and execution grounding (median split).

ϕ_{plan}	ϕ_{exec}	$\overline{\Delta r}$	Gap2Human
Low	High	**+0.377**	**+0.312**
Low	Low	+0.321	+0.393
High	High	+0.143	+0.071
High	Low	+0.028	−0.040

Table 2: PDI ablations: pass rates (%) on the rerun subset (3 matched-compute seeds; left) and transfer to 300 task variants (right).

	PDI-rerun subset (3 seeds)			Task variants (n=300)			
Model	**w/o PDI**	**w/ PDI**	**Δ**	**Base**	**w/o PDI**	**w/ PDI**	**Hum.**
DeepSeek-Chat	8.9 ± 5.0	**42.1** ± 6.6	+33.2	43.0	73.0	**87.0**	80.0
GPT-5.1-Codex	1.0 ± 1.7	**22.5** ± 5.8	+21.5	17.0	63.0	**83.0**	50.0
GPT-5.4-mini	18.5 ± 4.8	**30.6** ± 5.7	+12.1	7.0	41.0	**53.0**	33.0
GPT-5.4-nano	14.9 ± 5.3	**30.1** ± 6.0	+15.2	56.7	78.0	**90.0**	83.3
GLM-4.5-Air	3.1 ± 3.1	**15.3** ± 4.7	+12.2	45.0	81.0	**90.0**	82.0
Pooled	9.3 ± 3.5	**28.1** ± 4.7	**+18.8**	33.7	67.2	**80.6**	65.7

PDI predicts student-level gains. We evaluate the predictive power of PDI through three complementary views (Figure 5), each targeting a different facet of skill usefulness.

Panel (a): Pass-gain rate. For each trajectory group $g \in$ {interaction-free, iter-low-PDI, iter-high-PDI} and student model m, we compute the pass-gain rate over baseline-unsolved pairs as $\text{PG}(g,m) = \frac{\left|\{t \in \mathcal{T}_g \mid r^{\text{base}}_{m,t}=0 \wedge r^{+\text{skill}}_{m,t}=1\}\right|}{\left|\{t \in \mathcal{T}_g \mid r^{\text{base}}_{m,t}=0\}\right|}$, where $\mathcal{T}_g$ denotes the set of tasks in trajectory group g. This quantity is the probability that a student passes with the skill given that it failed without one. Skills from high-PDI iterative trajectories consistently achieve the highest pass-gain rates across all seven student models, outperforming both interaction-free skills and low-PDI iterative skills, the latter of which often fall below 0.10.

Panel (b): PDI vs. skill gain. For each baseline-unsolved iterative pair (m,t), we plot PDI against the skill gain $\Delta r_{m,t}$. The scatter reveals a significant positive Spearman correlation (ρ=+0.364, p=4.7×10^{-7}): higher PDI is associated with larger student gains across student models.

Panel (c): Memo ossification vs. gap to human skills. We define the gap relative to human-written skills as $G_{m,t} = \Delta r^{\text{SPARK}}_{m,t} - \Delta r^{\text{human}}_{m,t}$, where $\Delta r^{\text{SPARK}}_{m,t}$ and $\Delta r^{\text{human}}_{m,t}$ are the skill gains from SPARK-generated and human-written skills, respectively. Plotting ϕ_{oss} against $G_{m,t}$ yields a negative correlation (ρ=−0.277, p=1.6×10^{-4}), confirming that cognitive stagnation during exploration directly harms downstream transfer.

A Mann–Whitney U test between high- and low-PDI task groups yields p=0.028, confirming that PDI retains discriminative power at the task level. Indeed, sign-consistent weight combinations yield significant correlations ($p<0.05$) with skill gain, and cross-validated weight fitting underperforms on held-out data (Appendix F), confirming that PDI's predictive power derives from its directional structure rather than any particular coefficient.

Effective skills ground in evidence rather than plans. To disentangle the contributions of plan copying and execution grounding, we partition iterative tasks into four quadrants by the median of ϕ_{plan} and ϕ_{exec}. Table 1 reports the mean Δr and the mean gap versus human-written skills for each quadrant. The best-performing quadrant, low plan copying combined with high execution grounding, achieves a mean Δr of +0.377 and a positive gap of +0.312 over human skills. In contrast, the worst quadrant, high plan copying with low execution grounding, yields a near-zero mean Δr of +0.028

and a gap of -0.040 below human skills. This confirms that useful skills are not summaries of what the agent *planned* to do, but distillations of what the environment *verified* to work.

Belief stagnation undermines exploration. A related failure mode is what we term the *facts–strategy gap*: the difference between the cross-attempt stability of *Verified Facts* and *Next Strategy*. A large gap indicates that the agent superficially varies its strategy across retries while its task understanding remains unchanged, a form of pseudo-exploration. Among iterative tasks, this gap is negatively correlated with mean Δreward (ρ=-0.390, p=0.033), confirming that trajectories in which "strategies change but task understanding does not" fail to produce transferable skills.

Sensitivity to the smoothing parameter. PDI is robust to the choice of α within a broad low-smoothing regime; the full sensitivity curves and significance analysis are provided in Appendix E.

4.3 PDI Makes Autonomous Skill Generation Work

The above analyses treat PDI as a descriptor applied to completed trajectories. A natural question is whether the same signal can improve autonomous skill generation *during* the process itself. To test this, we rerun skill generation, with PDI-guided online intervention explicitly enabled, on the subset of tasks where original trajectories showed persistently low proxy-PDI values. To avoid introducing confounding effects, this subset excludes tasks where the teacher agent succeeded in the first round and those where the PDI monitors remain within healthy thresholds that will not trigger interference.

Downstream evaluation. We evaluate the skills generated **with and without PDI** control on the same subset of tasks, regenerating both arms over 3 seeds under matched compute. This ablation tests whether the trajectory-level signal can improve skill generation rather than merely a post-hoc descriptor. In Table 2 (left), PDI-guided skills yield stronger pass rates than their non-PDI counterparts on all five students ($+18.8$ pooled, p<0.001). The largest improvement appears on DeepSeek-Chat (pass rate from 8.9% to 42.1%). These results indicate that the online intervention improves the downstream utility of distilled skills without introducing negative side effects (case study in Appendix D). We compare SPARK against three skill generation baselines (Trace2Skill Ni et al. (2026), AutoRefine Qiu et al. (2026), and EvoSkill Alzubi et al. (2026)) given identical trajectories; SPARK's skill gain exceeds the strongest baseline by $+0.119$ in mean reward (Appendix K).

4.4 Task-Variant Ablation

The main benchmark evaluates one skill on one task instance, leaving robustness to various within-class instances untested. To assess this, we generate 300 task variants via SPARK's pipeline, preserving the overall workflow while purposely resampling task-specific contents. This design tests whether distilled skills encode reusable procedural structure rather than overfitting to instance-specific surface form. In Table 2 (right), SPARK improves over the baseline on all student models, and the PDI intervention holds across instances it never saw ($+13.4$ pooled), confirming that distilled skills generalize across instances within the same problem class. Absolute rates on this pool are not comparable to Figure 2, since the generated variants are on average easier than the expert-authored originals (Appendix J); both arms face the identical pool, so it is their contrast that carries the claim. For cross-domain generalization, we extend to evaluate SPARK on ALFWorld (Shridhar et al., 2021), a text-based interactive household environment that differs fundamentally from programming tasks. PDI-refined skills improve overall success rate from 16.7% to 40.0%, validating both the SPARK pipeline and PDI in an out-of-distribution setting (Appendix M).

5 Conclusion

Skill generation and validation through the lens of trajectory quality are crucial to advance agentic system. In this study, our central contribution is the Posterior Distillation Index (PDI): an interpretable, process-grounded criterion that enables a trajectory-level measure of whether a generated skill is grounded in the task environment rather than relying on prior plans. We demonstrate that the SPARK framework can produce such trajectories and a PDI proxy can guide and improve skill generation online. Together, autonomous skill generation works strongly when agent learns from the environment as a prerequisite, not an afterthought, for producing vital knowledge that future agents can inherit and execute verifiably and reliably.

AI USE STATEMENT

LLMs generated the skills and runnable tasks studied here; their outputs were verified by execution in the target environments. Generative AI also assisted with manuscript readability and code edits. The authors reviewed this work, tested the code, and take responsibility for the final paper and artifacts.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. Oral.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations (ICLR)*, 2024.

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. SkillCraft: Can LLM agents learn to use tools skillfully? *arXiv preprint arXiv:2603.00718*, 2026a.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. CUA-Skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*, 2026b.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International conference on machine learning*, pp. 1607–1616. PMLR, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Can Jin, Tong Che, Hongwu Peng, Yiyuan Li, Dimitris N Metaxas, and Marco Pavone. Learning from teaching regularization: Generalizable correlations should be easy to imitate. *Advances in Neural Information Processing Systems*, 37:966–994, 2024.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How well do agentic skills work in the wild: Benchmarking LLM skill usage in realistic settings. *arXiv preprint arXiv:2604.04323*, 2026.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Liangyu Li, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. AutoRefine: Compiling trajectories into validated typed agent artifacts. *arXiv preprint arXiv:2601.22758*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, et al. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

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.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations (ICLR)*, 2025.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

Yanzhao Zheng, Zhentao Zhang, Chao Ma, Yuanqiang Yu, Jihuai Zhu, et al. SkillRouter: Skill routing for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026.

Yang Zhou, Mingyu Zhao, Zhenting Wang, Difei Gu, Bangwei Guo, Ruosong Ye, Ligong Han, Can Jin, and Dimitris N Metaxas. M^3-Bench: Multi-modal, multi-hop, multi-threaded tool-using MLLM agent benchmark. *arXiv preprint arXiv:2511.17729*, 2025a.

Yang Zhou, Shiyu Zhao, Yuxiao Chen, Zhenting Wang, Can Jin, and Dimitris N Metaxas. LED: LLM enhanced open-vocabulary object detection without human curated data generation. *arXiv preprint arXiv:2503.13794*, 2025b.

APPENDIX CONTENT

A Positioning Against Prior Skill-Centric Systems 14
B Trajectory Feature Correlation Analysis 14
C Case Studies of Online PDI-Guided Control 16
D External Transfer Case Study: `lean4-proof` 16
E Sensitivity of PDI to the Smoothing Parameter 17
F Sensitivity of PDI to Component Weights 18
G Exploration Statistics 18
H Pipeline Pseudocode 20
H.1 Skill Generation Pipeline 20
H.2 Task Construction Pipeline 20
I Generated Task Examples 20
I.1 3D Scan Mass Calculation 20
I.2 WAV Amplitude Analysis 22
I.3 Weather Station Anomaly Detection 22
J Difficulty Calibration of the Generated Variant Pool 23
K Comparison with Baseline Skill Generation Methods 23
L Comparison with Prompt- and Context-Evolution Methods 23
M External Benchmark: ALFWorld 25

A Positioning Against Prior Skill-Centric Systems

Table 3 positions SPARK relative to recent skill-centric systems along five dimensions. Prior work has typically focused on only a subset of these properties: some methods emphasize environment-grounded interaction, others study usefulness evaluation or cross-model transfer, but few go beyond demonstrating that skills can be produced or applied. In contrast, SPARK is designed not only to generate transferable skills in executable environments, but also to analyze why some trajectories yield useful downstream skills while others do not. This trajectory-level mechanistic perspective further enables analysis-guided online intervention, making SPARK distinct from prior systems that treat skill generation primarily as a black-box pipeline.

Table 3: Comparison with recent skill-centric systems. The highlighted column marks the paper's central focus beyond framework construction: explaining why some trajectories yield useful transferable skills. Binary columns: **Env.-Grounded Interaction** : the agent executes in a real environment; **Usefulness Eval.** : explicitly evaluates skill-induced Δperformance; **Cross-Model Transfer** : skills generated by one model guide a different model; **Mechanistic Analysis** : analyzes trajectory-to-skill properties that explain downstream utility; **Analysis-Guided Intervention** : uses such analysis to improve skill generation online.

System	Env.-Grounded Interaction	Usefulness Eval.	Cross-Model Transfer	Mechanistic Analysis	Analysis-Guided Intervention
AgentTrek (Xu et al., 2025)	✓	✗	✗	✗	✗
AutoSkill (Yang et al., 2026)	✗	✗	✓	✗	✗
SkillCraft (Chen et al., 2026a)	✓	✓	✗	✗	✗
SkillsBench (Li et al., 2026)	✓	✓	✓	✗	✗
SkillNet (Liang et al., 2026)	✓	✓	✓	✗	✗
SkillRL (Xia et al., 2026)	✓	✗	✗	✗	✗
Trace2Skill (Ni et al., 2026)	✓	✓	✓	✗	✗
EvoSkill (Alzubi et al., 2026)	✓	✓	✗	✗	✗
AutoRefine (Qiu et al., 2026)	✓	✓	✗	✗	✗
OEL (Ye et al., 2026)	✓	✗	✗	✗	✗
SKILL0 (Lu et al., 2026)	✓	✓	✗	✗	✗
SPARK (Ours)	✓	✓	✓	✓	✓

B Trajectory Feature Correlation Analysis

Figure 6 presents a Spearman rank-correlation heatmap between 23 trajectory-level features and the downstream skill gain $\Delta r_{m,t}$ observed for each student model. Each cell reports the Spearman ρ between a feature (column) and $\Delta r_{m,t}$ (row) computed over all tasks where (i) the baseline student failed ($r^{\text{base}}_{m,t} = 0$) and (ii) a SPARK skill was available. The bottom row (**Pooled**) aggregates all (task, student) pairs across all seven student models. Significance levels are annotated as $*$ ($p < .05$), $**$ ($p < .01$), and $***$ ($p < .001$). Red cells indicate that higher feature values are associated with larger student gains; blue cells indicate the opposite.

How to read the figure. Columns are grouped into four categories by colour: *Exploration Dynamics* (blue), *Memo Quality* (green), *Skill Structure* (purple), and *Non-predictive* (grey). A feature that shows consistent red across most student rows and a significant pooled ρ is a robust positive predictor of skill usefulness; consistent blue indicates a negative predictor. Features in the *Non-predictive* block were hypothesized to matter but showed weak or inconsistent correlations, serving as negative controls.

Feature definitions. Let $\tau = (\tau_1, \ldots, \tau_K)$ denote the sequence of K execution attempts for a task, μ_k the exploration memo after the k-th reflect step, and s the generated skill text. We use $\text{tok}(\cdot)$ for word-level tokenization and $\text{J}(a, b) = |W_a \cap W_b|/|W_a \cup W_b|$ for the Jaccard similarity between word sets W_a and W_b.

Exploration Dynamics.

- **Compression ratio**: $\rho_c = \sum_k |\tau_k^{\text{stdout}}| \;/\; |s|$. Total raw agent output divided by skill length, matching the main-text definition in Section 4.2.2.

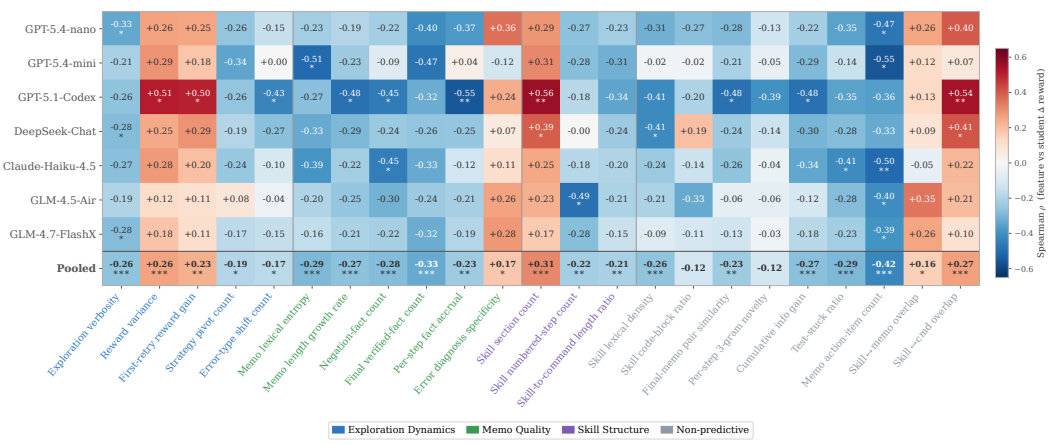


Figure 6: Spearman rank correlation between trajectory-level features (columns) and student skill gain $\Delta r_{m,t}$ (rows). Each row corresponds to a student model; the bottom row pools all (task, model) pairs. Significance: $*$ $p < .05$, $**$ $p < .01$, $***$ $p < .001$.

- **Reward variance**: $\mathrm{Var}(\{r_1, \ldots, r_K\})$.
- **First-retry reward gain**: $\Delta r^{(1)} = r_2 - r_1$.
- **Strategy pivot count**: $\sum_{k=2}^{K} \mathbf{1}[\mathrm{J}(\text{strategy}_{k-1}, \text{strategy}_k) < 0.15]$, where strategy_k is the "Next Strategy" section of μ_k.
- **Error-type shift count**: same as above, applied to the "Current Error Pattern" section.

MEMO QUALITY.

- **Memo lexical entropy**: $\bar{H} = \frac{1}{K}\sum_k H(\mu_k)$, where $H(\mu_k) = -\sum_w p_w \log_2 p_w$ is the Shannon entropy over word frequencies.
- **Memo length growth rate**: $(|\mu_K| - |\mu_1|)/|\mu_1|$. Fractional increase in memo character count.
- **Negation-fact count**: number of bullet items in the final "Verified Facts" section matching negation keywords (`not`, `never`, `failed`, `wrong`, etc.).
- **Final verified-fact count**: total bullet items in the final "Verified Facts" section.
- **Per-step fact accrual**: $\frac{1}{K-1}\sum_{k=2}^{K}(n_k^{\text{facts}} - n_{k-1}^{\text{facts}})$.
- **Error diagnosis specificity**: mean ratio of concrete references (inline code, numbers, file paths) to total words in the "Current Error Pattern" section across all memos.

SKILL STRUCTURE.

- **Skill section count**: number of Markdown heading lines (`#`, `##`, ...) in s.
- **Skill numbered-step count**: number of lines matching the pattern `/^\d+[.)]/` in s.
- **Skill-to-command length ratio**: $|s|$ / $\overline{|\mathbf{c}_{\text{succ}}|}$, where $\overline{|\mathbf{c}_{\text{succ}}|}$ is the mean character length of successful-attempt agent commands.

NON-PREDICTIVE (NEGATIVE CONTROLS).

- **Skill lexical density**: $|\mathrm{unique}(\mathrm{tok}(s))|/|\mathrm{tok}(s)|$.
- **Skill code-block ratio**: total characters inside ``` ``` ... ``` ``` blocks divided by $|s|$.
- **Final-memo pair similarity**: $\mathrm{J}(\mu_{K-1}, \mu_K)$.
- **Per-step 3-gram novelty**: mean fraction of new word 3-grams at each memo step.
- **Cumulative info gain**: sum of per-step 3-gram novelty across all steps.
- **Test-stuck ratio**: fraction of consecutive attempt pairs with identical test summaries.
- **Memo action-item count**: mean number of bullet/numbered items per memo.

- **Skill↔memo overlap**: $\mathrm{J}(s, \mu_K)$.
- **Skill↔cmd overlap**: $\mathrm{J}(s, \mathbf{c}_{\text{succ}})$.

C Case Studies of Online PDI-Guided Control

We provide two trajectory-level case studies that compare PDI-enabled runs against observe-only controls on `3d-scan-calc` and `manufacturing-codebook-normalization`. In `3d-scan-calc`, the weighted proxy-PDI triggers a soft intervention at step 4 and escalates to a strong one at step 5; the trajectory then rebounds to a clearly positive regime, whereas the observe-only control keeps drifting downward. In `manufacturing-codebook-normalization`, two earlier soft triggers (steps 3 and 5) already suffice to reverse the decline and sustain positive proxy-PDI, while the control again stays persistently negative. Together, these cases show that proxy-PDI is not merely descriptive but an actionable online signal: it detects plan-repetitive or stale-summary exploration and supports graded, prompt-only interventions that redirect search without exposing the metric to the model.

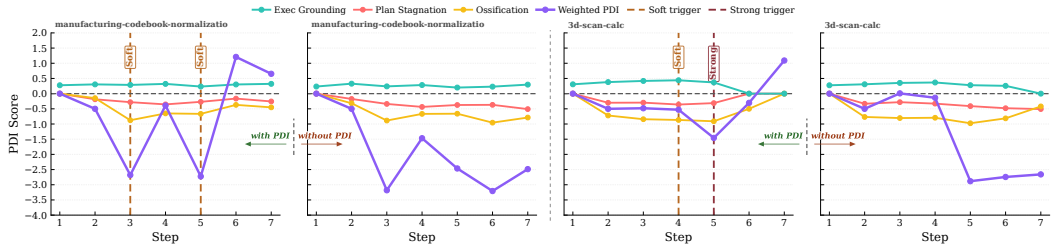


Figure 7: Two case studies of online PDI-guided control, comparing PDI-enabled runs against observe-only controls on `3d-scan-calc` and `manufacturing-codebook-normalization`. Each panel plots execution grounding (ϕ_{exec}), plan copying (ϕ_{plan}), memo ossification (ϕ_{oss}), and the warmup-weighted proxy-PDI used for intervention decisions. Vertical dashed lines mark soft and strong triggers.

D External Transfer Case Study: `lean4-proof`

Among all tasks evaluated under PDI-refined skills, `lean4-proof` exhibits the clearest behavioural shift in a held-out student model (Claude Haiku 4.5, never used during skill generation or PDI refinement). The task requires proving $S\,n \le 2$ for the geometric-series sequence $S\,0 = 1$, $S\,(n{+}1) = S\,n + 1/2^{n+1}$ in Lean 4; the proof body must start at line 15 of `solution.lean`, and the verifier runs `lake env lean -DwarningAsError=true solution.lean` with zero output indicating success. We present the analysis in three tables that progressively zoom in: skill-level differences (Table 4), step-by-step trajectory comparison (Table 5), and the resulting strategy and score changes (Table 6).

Table 4: Structural comparison between the original and PDI-refined skill documents for `lean4-proof`. The PDI-refined skill highlights the components that most directly alter the student's execution strategy. It encodes a *complete reference implementation* with the exact `norm_num` / `ring` / `linarith` tactic chain, rather than a high-level proof sketch.

Dimension	Original Skill (non-PDI)	PDI-Refined Skill
Document length	118 lines	**393 lines** (3.3×)
Structural modules	4 (overview, orientation, strategy, pitfalls)	**7** (+ step-by-step workflow, reference proof, testing)
Contains executable proof?	No	**Yes**, complete `theorem` block with exact tactic sequence
Verification command	Partial (`lake env lean solution.lean`)	**Exact**: `lake env lean -DwarningAsError=true solution.lean`
"Write to `/tmp` first" workflow	Not mentioned	**Yes**, Step 3 of workflow prevents corrupting the solution file

Table 5: Step-by-step trajectory comparison for `lean4-proof` on Claude Haiku 4.5. The PDI trajectory shows the critical behavioural differences induced by the refined skill. The non-PDI agent spends 15 commands exploring the environment and iterating on broken proofs; the PDI agent follows the skill's recipe in 5 commands with zero failures.

Step	**non-PDI Trajectory** (15 cmds, 3 failures)	**PDI Trajectory** (5 cmds, 0 failures)
0	`cat /root/spark-skills/*/SKILL.md` reads skill (high-level sketch only)	`cat /root/spark-skills/*/SKILL.md` reads PDI skill (contains full proof)
1	`cat -n solution.lean` reads template	`cat -n solution.lean` reads template
2	`cat lean-toolchain` checks toolchain version	`cat -n /app/baseline/solution.lean` verifies immutable prefix
3	`find Library -name "*.lean"` explores library structure	Writes proof to `/tmp/test.lean`, then copies to `solution.lean` follows "test in /tmp first" recipe
4	Writes proof attempt #1 → **FAIL** (type errors)	`lake env lean -DwarningAsError=true solution.lean` **SUCCESS** (exit 0)
5–14	Iterates through attempts #2–#6, each failing with tactic or type errors; 3 total failures	

Table 6: Strategy shift and quantitative changes induced by the PDI-refined skill on `lean4-proof`. The PDI Skill Cue column identifies the cues that drive each behavioural change. The bottom rows report cross-model PDI results: the largest gains appear on weaker models (GPT-5.1-Codex, GLM-4.5-Air), consistent with the hypothesis that PDI-refined skills compensate for capability gaps by converting posterior evidence into executable recipes.

Phase	**Original Skill Cue**	**PDI Skill Cue**	**Strategy Shift**	**Metric**
Task framing	High-level proof sketch: derive closed form, then prove bound	Complete tactic recipe with exact `simp [S]` + `ring` pattern	From re-deriving the plan to direct theorem construction	
Proof construction	Agent guesses tactic sequences; no reference implementation	Reference proof block with `norm_num` / `ring` / `linarith` chain	From trial-and-error to copy-and-adapt	Failed cmds: $3 \rightarrow \mathbf{0}$
Verification	`lake env lean solution.lean` (missing `-DwarningAsError`)	Exact command: `lake env lean -DwarningAsError=true`	Correct flags on first attempt	Total cmds: $15 \rightarrow \mathbf{5}$
Safety workflow	Writes directly to `solution.lean`	Writes to `/tmp` first, copies after verification	Prevents corrupting the immutable prefix	Msgs: $16 \rightarrow \mathbf{6}$
Cross-model PDI results on `lean4-proof`				
GPT-5.1-Codex: non-PDI FAIL → PDI **PASS**			$\Delta = +1.0$	
GLM-4.5-Air: non-PDI FAIL → PDI **PASS**			$\Delta = +1.0$	
Claude Haiku 4.5: stochastic (FAIL/PASS across runs)			$\Delta \in \{0, +1.0\}$	

Why PDI made the difference. The non-PDI skill tells the agent *what* to prove (closed-form identity → bound) but not *how* to write the Lean 4 tactics. The PDI-refined skill was generated after observing that weaker models fail precisely at the tactic level, so it includes: (1) the exact `simp [S]` + `ring` pattern for the inductive step; (2) the `norm_num` + `linarith` pattern for the bound derivation; and (3) a "test in `/tmp` first" workflow that prevents corrupting the solution file. This is the PDI mechanism in action: by comparing successful and failed trajectories across models, the refinement process identifies *where agents actually get stuck* and injects targeted procedural knowledge. The result is not merely a longer document, but a fundamentally different *action policy* for the student: the original skill transfers a plan; the PDI-refined skill transfers an execution recipe anchored in verified evidence.

E Sensitivity of PDI to the Smoothing Parameter

Divergence choice. KL quantifies the coding inefficiency of using Q to represent P. Its asymmetry and lack of an upper bound complicate interpretation when neither distribution is a clear reference. JSD retains sensitivity to how probability mass is allocated across tokens while remaining symmetric and well-defined even for sparse distributions.

The smoothing constant α in Section 3.2 controls the sharpness of the empirical token distributions: smaller α yields distributions closer to raw frequency counts, while larger α flattens them toward

uniform. Figure 8 reports the Spearman ρ and p-values of PDI across $\alpha \in [10^{-10}, 10]$. All three correlation metrics (task-level, pair-level, and pass-gain) exhibit a plateau for $\alpha \lesssim 10^{-3}$, where smoothing is negligible relative to observed counts, and degrade monotonically as α increases beyond 10^{-2}. The Mann–Whitney U test between high- and low-PDI groups remains significant ($p<0.05$) for $\alpha \leq 0.005$ and loses significance for $\alpha \geq 0.007$. This pattern confirms that PDI is robust to the choice of α within a broad low-smoothing regime.

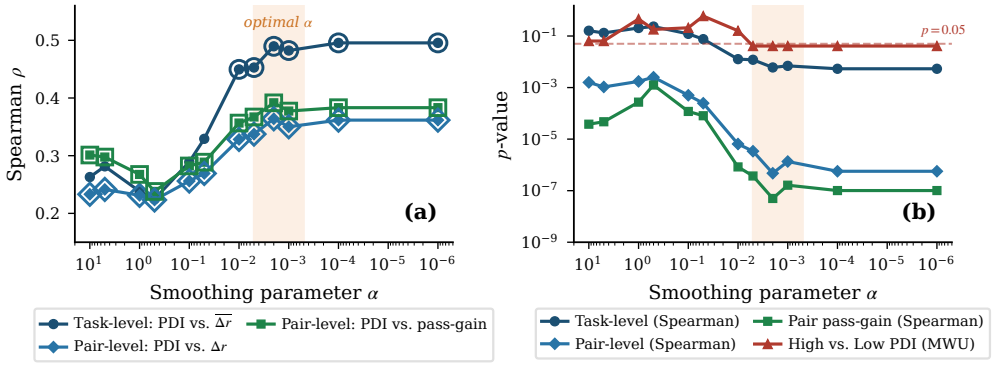


Figure 8: Sensitivity of PDI to the smoothing parameter α. (a) Spearman ρ between PDI and three outcome measures; circled points are significant at $p<0.05$. (b) Corresponding p-values on a log scale; the red dashed line marks $p=0.05$. The shaded band highlights the optimal region $\alpha \in [5\times10^{-4}, 5\times10^{-3}]$.

F Sensitivity of PDI to Component Weights

PDI combines three z-scored components with equal weights: $\text{PDI} = z(\phi_{\text{exec}}) - z(\phi_{\text{plan}}) - z(\phi_{\text{oss}})$. We examine whether this choice is justified by testing (i) whether the directional structure itself is robust, (ii) whether fitted weights generalize across held-out models and tasks, and (iii) whether each component is necessary.

Directional structure. We sweep all weight combinations (w_e, w_p, w_o) with $w_e > 0$ (preserving the sign convention that execution grounding contributes positively while plan copying and memo ossification contribute negatively). The large majority of sign-consistent combinations yield statistically significant Spearman correlation ($p<0.05$) with student skill gain, indicating that PDI's predictive power derives from its directional structure rather than from any particular weight assignment.

Cross-validated weight stability. To test whether fitted weights generalize, we perform 5-fold task cross-validation: on each fold, we grid-search the weight vector that maximizes Spearman ρ on the training tasks and evaluate it on the held-out fold. Table 7 reports the results. Grid search produces five distinct "optimal" weight vectors across five folds, yet equal weighting matches or exceeds the fitted optimum on every fold (5/5). A complementary leave-one-model-out protocol yields a similar pattern: nine folds produce five distinct optimal vectors. Together, these results confirm that weight tuning does not yield transferable gains and that the fitted optimum is an artifact of the particular train split.

Component ablation. Removing execution grounding ($w_e=0$) collapses held-out correlation to near zero; removing memo ossification ($w_o=0$) also degrades predictive power. Equal weighting thus serves as a principled, parameter-free default that preserves all three empirically validated directional signals without introducing tunable hyperparameters that risk overfitting to a particular task or model distribution.

G Exploration Statistics

Table 8 summarises the exploration cost of the SPARK skill-generation pipeline across four teacher models. All models share the same task pool and Docker-based execution environment; the maximum retry budget $N_{\max}$ is 7 for Claude Opus 4.6, GPT-5.2-Codex, GPT-5.4 and GLM-5. These figures

Table 7: 5-fold task cross-validation of PDI component weights. On each fold, the weight vector maximizing ρ on training tasks is evaluated on held-out tasks. Equal weighting $(1, 1, 1)$ matches or outperforms the fitted optimum on every fold, while the fitted vectors are unstable across folds.

Fold	Fitted (w_e, w_p, w_o)	ρ_{fitted}	ρ_{equal}	Winner
1	$(1.5,\ 0.5,\ 1.0)$	0.153	**0.241**	Equal
2	$(1.0,\ 0.5,\ 0.5)$	0.179	**0.179**	Tie
3	$(1.0,\ 0.0,\ 0.5)$	0.426	**0.492**	Equal
4	$(0.5,\ 0.0,\ 1.0)$	0.274	**0.281**	Equal
5	$(0.5,\ 0.0,\ 0.5)$	0.252	**0.252**	Tie
Mean	5 distinct vectors	0.257	**0.289**	**Equal**

should be interpreted as an upfront investment in capability transfer rather than a per-query serving cost: strong interactive teachers are expensive to run repeatedly, but once their trajectories are distilled into reusable `SKILL.md` documents, the same procedural knowledge can be deployed to much cheaper student models at inference time. In this sense, SPARK trades one-time high-cost exploration for repeated low-cost execution. The main-text results show that this trade can be favorable not only economically but also in effectiveness, since several skill-equipped students even surpass the interaction-free performance of the stronger teacher models.

Table 8: Exploration statistics of the SPARK skill-generation pipeline. Agent execution time excludes Docker build and teardown overhead. Interaction turns count agent messages plus shell commands per attempt. Cost estimates for GLM-5 and GPT-5.2-Codex are calibrated from execution time.

Metric	Claude Opus 4.6	GPT-5.4	GLM-5	GPT-5.2-Codex
Avg. attempts / task	6.00	3.93	3.25	5.37
Avg. reward (all attempts)	0.120	0.246	0.394	0.080
Avg. agent exec. time	8m50s	7m04s	4m20s	6m13s
Avg. interaction turns	30.5	32.3	16.2	23.2
Avg. cost / attempt ($)	4.11	5.70	3.03	2.70
Avg. cost / task ($)	24.72	22.35	9.90	14.49

Student inference cost and amortization. While teacher exploration is expensive, the distilled skills are consumed by much cheaper student models at inference time. Table 9 reports the per-task inference cost of representative student models, computed from the token-level usage logs recorded in each evaluation trajectory. Across all measured students, the per-task cost ranges from $0.02 (DeepSeek-Chat) to $0.18 (GLM-4.5-Air), yielding 86-task totals between $1.89 and $15.48. Compared with the cheapest teacher (GLM-5 at $9.90/task), even the most expensive student is $55\times$ cheaper per task; DeepSeek-Chat is over $1{,}100\times$ cheaper than Claude Opus 4.6. A single teacher exploration run ($852–$2,126 for 86 tasks) can therefore be amortized over hundreds of student deployments at negligible marginal cost.

Table 9: Student model inference cost per task, computed from token-level usage logs in evaluation trajectories. The teacher/student ratio uses Claude Opus 4.6 at $24.72/task as the numerator.

Student Model	Avg. cost / task ($)	86-task total ($)	Teacher / Student ratio
DeepSeek-Chat	0.022	1.89	$1{,}124\times$
GLM-4.7-FlashX	0.085	7.31	$291\times$
GLM-4.5-Air	0.180	15.48	$137\times$

H Pipeline Pseudocode

H.1 Skill Generation Pipeline

Algorithm 1 describes the iterative skill-generation loop introduced in Section 3.1. The teacher model explores each task inside a containerized sandbox for up to $N_{\max}$ attempts. Failed attempts trigger a REFLECT step that rewrites the exploration memo; a successful attempt triggers evidence assembly and skill distillation.

Memo Structure

1) Attempts Log stores a one-line LLM-distilled summary for each attempt; **2) Commands** record the key commands the agent actions in the most recent attempt; **3) Verified Facts** store the environment-confirmed outcomes across rewrites; **4) Current Error Pattern** identifies the active failure mode; and **5) Next Strategy** specifies an actionable plan that must differ from all previous approaches.

Online proxy. Let C_k, V_k, S_k, F_k denote current commands, *Verified Facts*, *Next Strategy*, and failed tests at reflection k. We compute $e_k = s(C_k, V_k)$, $p_k = s(S_{k-1}, S_k)$, and $o_k = [s(V_{k-1}, V_k) + s(F_{k-1}, F_k)]/2$, setting missing history to zero. Here s is JS similarity, or token coverage for e_k and Jaccard similarity for p_k, o_k in token-overlap mode. The running score is $\mathrm{PDI}_k = z_k(e_k) - z_k(p_k) - z_k(o_k)$, where z_k normalizes over this task's observed reflections; the first reflection uses $e_k - p_k - o_k$.

H.2 Task Construction Pipeline

Algorithm 2 details the task-construction pipeline (Section 3.3). A natural-language prompt is expanded into a structured blueprint, iteratively critiqued and repaired, rendered into a runnable task directory, and validated by executing the oracle inside the target Docker environment.

H.2.1 Integrated Techniques for Exploration Quality

SPARK integrates several complementary techniques that improve exploration efficiency and distillation fidelity. To reduce noise in the skill-generation context, it applies a lightweight command-processing pipeline: each command is (i) classified into one of five semantic categories (VERIFY, IMPLEMENT, INSPECT, PREPARE, ACTION), (ii) scored by a keyword-weighted importance function that prioritizes verification and implementation actions, and (iii) filtered to remove low-signal commands (e.g., `ls`, `pwd`, `echo`). Multi-line commands that match known domain patterns (e.g., spreadsheet manipulation via `openpyxl`, PDDL planning, bibliographic API queries) are replaced with semantic summaries. The top-k commands (default k=12) are selected by score and presented in execution order, yielding a concise yet high-signal execution chain. In addition, as the agent approaches $N_{\max}$, retry injection adds an escalating urgency signal.

I Generated Task Examples

We present three tasks produced by the SPARK task-construction pipeline (Section 3.3) to illustrate the diversity and self-contained nature of the generated benchmarks. Each task was created from a short natural-language prompt, automatically expanded into a full blueprint, critiqued, repaired, and validated by executing the oracle inside the target Docker environment. No manual editing was performed after generation.

I.1 3D Scan Mass Calculation

Prompt (input). *"Calculate the mass of a 3D printed part from a binary STL scan. The 2-byte Attribute Byte Count field stores the Material ID. Filter out scanning debris by keeping only the largest connected component, look up the density, and output the mass."*

Generated task. The pipeline produces a complete task directory containing: (i) an `instruction.md` that asks the agent to parse a binary STL file, identify the largest connected

Algorithm 1 SPARK Skill Generation Pipeline

Require: Task set $\mathcal{T}$, teacher model $\mathcal{M}_{\text{teach}}$, max attempts $N_{\max}$, parallelism P, token budgets $\mathcal{B}$
Ensure: Skill repository $\mathcal{S} = \{s_t\}_{t \in \mathcal{T}_{\text{solved}}}$

Clean stale Docker artifacts; prefetch shared base images
$\mathcal{T}_{\text{active}} \leftarrow \textsc{ResolveTasks}(\mathcal{T})$ ▷ filter blacklist, skip completed
for each task $t \in \mathcal{T}_{\text{active}}$ **in parallel** (P workers) **do**
  Initialise exploration memo $\mu_0 \leftarrow \varnothing$; attempt counter $k \leftarrow 0$
  while $k < N_{\max}$ **do**
    $k \leftarrow k + 1$
    // Stage 1: Execute
    Prepare staging directory: copy task files, inject μ_{k-1} into `instruction.md`
    if $k = N_{\max}$ **then**
      Append urgency signal to injection ▷ final-attempt warning
    end if
    $\tau_k \leftarrow \textsc{Execute}(t, \mathcal{M}_{\text{teach}})$ via Docker sandbox
    // Stage 2: Judge
    $(r_k, \mathbf{c}_k, \sigma_k) \leftarrow \textsc{Judge}(\tau_k)$ ▷ reward, commands, test summary
    if $r_k \geq 1.0$ **then** ▷ task solved
      // Stage 3: Distill
      $\mathcal{E} \leftarrow \textsc{AssembleEvidence}(t, \tau_{1:k}, \mu_{0:k-1})$ ▷ six evidence blocks
      $s_t \leftarrow \textsc{GenerateSkill}(\mathcal{E}, \mathcal{B})$ via $\mathcal{M}_{\text{teach}}$
      Save s_t as `SKILL.md`; record trajectory; **break**
    else
      // Stage 3: Reflect
      $\mu_k \leftarrow \textsc{Reflect}(\mu_{k-1}, \mathbf{c}_k, \sigma_k)$ via $\mathcal{M}_{\text{teach}}$ ▷ rewrite, not append
      // Stage 4: PDI-guided intervention
      $w_k \leftarrow \min(1,\ k/W)$ ▷ linear warm-up, $W{=}2$
      $\hat{d}_k \leftarrow w_k \cdot \text{PDI}_k(\mu_k, \mu_{k-1}, \mathbf{c}_k, \sigma_k, \sigma_{k-1})$
      if $\hat{d}_k < \tau$ **then** ▷ $\tau{=}-0.5$
        if $k > 1$ **and** $\hat{d}_{k-1} < \tau$ **then**
          Inject StrongIntervention into next retry prompt
        else
          Inject SoftIntervention into next retry prompt
        end if
      end if
    end if
  end while
  if task unsolved after $N_{\max}$ attempts **then**
    Save error report with final memo $\mu_{N_{\max}}$
  end if
end for
// Evaluation
for each student model $\mathcal{M}_s \in \{\mathcal{M}_1, \ldots, \mathcal{M}_S\}$ **do**
  Run baseline (no skill), SPARK skill, and human skill conditions
  Compute per-task $\Delta r_{s,t}$ and aggregate statistics
end for

component of the triangle mesh, extract the Material ID from the attribute field, look up the corresponding density from a provided material table, compute the volume via the signed-tetrahedron method, and write the result to a JSON file; (ii) a `build_data.py` script that deterministically synthesizes the STL asset and material table during Docker build; (iii) a pytest oracle that checks the reported mass within 0.1% relative tolerance.

This task exercises binary file parsing, computational geometry (connected components, mesh volume), and multi-source data integration, a combination unlikely to be solved by template matching alone.

Algorithm 2 SPARK Task Construction Pipeline

Require: Prompt specification $\mathcal{P}$ (text, tool hints, constraints), LLM $\mathcal{M}$, tool catalogue $\mathcal{C}$, max revisions $R_{\max}$, schema retries $S_{\max}$
Ensure: Validated task directory $\mathcal{D}_t$ or failure report

// Phase 1: Blueprint Generation
Compose generation prompt from $\mathcal{P}$, $\mathcal{C}$, and Harbor format specification
for $i = 1$ **to** $S_{\max}$ **do**
  $b \leftarrow \textsc{CallLLM}(\mathcal{M}, \text{prompt})$; parse JSON $\rightarrow$ TASKBLUEPRINT
  if schema validation passes **then**
    break
  end if
end for
// Phase 2: Iterative Critique–Repair–Validate
$critique_repairs \leftarrow 0$
for $j = 0$ **to** $R_{\max}$ **do**
  // Critique
  $I_{\text{det}} \leftarrow \textsc{DeterministicChecks}(b, \mathcal{P})$ ▷ output-path consistency, evidence validity
  $I_{\text{llm}} \leftarrow \textsc{LLMCritique}(b, \mathcal{P})$ ▷ answer leakage, oracle–verifier alignment
  $I \leftarrow I_{\text{det}} \cup I_{\text{llm}}$
  if I contains blocking issues **and** $critique_repairs < 1$ **then**
    $b \leftarrow \textsc{Repair}(b, I)$; $critique_repairs \leftarrow critique_repairs + 1$; **continue**
  end if
  // Render
  $\mathcal{D}_t \leftarrow \textsc{Render}(b)$ ▷ instruction.md, Dockerfile, oracle, verifier, support files
  // Validate
  Run `harbor tasks check` on $\mathcal{D}_t$ ▷ static structure check
  $v \leftarrow \textsc{RunOracle}(\mathcal{D}_t)$ ▷ execute oracle in Docker, run pytest verifier
  if $v.passed$ **and** no blocking issues in I **then**
    Record generation trace; **return** $\mathcal{D}_t$ ▷ success
  end if
  if $j < R_{\max}$ **then**
    $b \leftarrow \textsc{Repair}(b, I \cup v.feedback)$ ▷ repair from critique + validation
  end if
end for
Record trace with all attempts; **return** failure report

I.2 WAV Amplitude Analysis

Prompt (input). *"Parse a binary WAV file, split it into 1-second segments, compute the RMS amplitude of each segment, and identify the loudest one."*

Generated task. The agent must read a WAV file using only Python's standard library, segment the audio by sample rate, compute per-segment RMS values, and produce a structured JSON report including sample rate, channel count, duration, per-segment statistics, and the index of the loudest segment. The oracle verifies all numeric fields within 0.01% relative tolerance. The audio asset is synthesized deterministically during environment build, ensuring reproducibility without distributing binary data.

I.3 Weather Station Anomaly Detection

Prompt (input). *"Detect anomalous hourly temperature readings across multiple weather stations using a 3σ threshold."*

Generated task. The pipeline generates a CSV dataset containing hourly readings from multiple stations with injected anomalies. The agent must compute per-station population statistics, flag readings beyond three standard deviations, and output a JSON report with station count, total readings, anomaly count, and a timestamp-sorted anomaly list. The oracle validates both aggregate

counts and per-anomaly fields. This task requires careful attention to specification details (population vs. sample standard deviation, strict vs. non-strict inequality, and per-station independence) that are common sources of silent errors.

Observations. Across all three examples, the pipeline consistently produces tasks that are (i) fully self-contained (all assets are built deterministically at Docker build time), (ii) oracle-validated (the solution script passes the pytest verifier before the task is accepted), and (iii) non-trivial (each requires domain-specific reasoning beyond simple text manipulation). These properties make the generated tasks suitable for both skill-generation training and downstream evaluation.

J Difficulty Calibration of the Generated Variant Pool

The variant pool used in Section 4.4 reliably preserves the procedural workflow of each source task, but it does not preserve its difficulty. Because the generator must synthesise every asset deterministically at build time (Appendix H.2), it falls back on the simplest construction consistent with the stated workflow whenever the original asset is complex: on `3d-scan-calc`, the SkillsBench original ships a scanned mesh of 352 triangles, whereas the variant emits 36 triangles forming three axis-aligned boxes, which reduces "identify the largest connected component" from mesh traversal to separating three manifestly disjoint blocks. The generated instruction is likewise more permissive, volunteering clarifications that an expert author would withhold. Baselines on this pool are correspondingly higher than on the main benchmark, which is itself a measurement worth reporting: it indicates that expert-designed task suites such as SkillsBench remain difficult to match automatically, and that a generation pipeline is better used for what it does well, namely producing large numbers of held-out instances that share a workflow with a source task, than as a substitute for expert task design. This is precisely the role the pool plays here, and the within-pool contrast between the two arms is unaffected by the offset.

K Comparison with Baseline Skill Generation Methods

To further validate the effectiveness of PDI-guided SPARK, we compare against three recent skill generation methods, Trace2Skill (Ni et al., 2026), AutoRefine (Qiu et al., 2026), and EvoSkill (Alzubi et al., 2026), on the 86-task SkillsBench suite. All methods use the same skill generation model (`claude-sonnet-4.6`) and the same evaluation pipeline (student model: `deepseek-chat`; agent framework: `qwen-coder`; timeout multiplier: 0.5; Docker-based Harbor sandbox). The skill gain Δ is the primary comparison metric, as it controls for baseline stochasticity across independent runs.

Reproduction details. We re-implement each baseline following its published algorithm. Key configuration parameters are summarized in Table 10.

Results. Table 11 reports the comparison. SPARK achieves the highest Δ Pass Rate ($+28.8\%$) and Δ MR ($+0.291$), outperforming the strongest baseline AutoRefine by $+10.2\%$ in pass rate gain and $+0.119$ in mean reward gain. SPARK also exhibits the most favorable improvement-to-degradation ratio (21 improved vs. 3 degraded tasks), whereas AutoRefine improves 19 tasks but degrades 9. These results confirm that PDI-guided posterior distillation produces substantially stronger skills than methods that rely on lesson extraction, iterative refinement, or evolutionary search without environment-grounded trajectory analysis.

L Comparison with Prompt- and Context-Evolution Methods

A separate line of work improves an in-context artifact by iterating against downstream feedback: GEPA (Agrawal et al., 2026) evolves prompts by reflective mutation, and ACE (Zhang et al., 2026) evolves an agent context through incremental delta updates. Since these methods target the same artifact class as skills, we compare against both.

Setup. This comparison runs on a 59-task subset under one shared protocol: agent framework `qwen-coder`, generator `claude-sonnet-4`, student `deepseek-v4-flash`

Table 10: Reproduction configuration for each baseline method. All methods use `claude-sonnet-4.6` for skill generation and share the same SPARK teacher trajectories as input.

Method	Configuration
Trace2Skill	**Stage 1 (Step-level Lesson Extraction):** per-attempt lesson extraction; input truncation: commands 6 000 chars, stdout 4 000 chars; temperature 0.3, max tokens 2 048. **Stage 2 (Lesson Aggregation + Skill Synthesis):** aggregate lessons across attempts with `[SUCCESS]`/`[FAIL]` labels; aggregation truncated to 12 000 chars; temperature 0.5, max tokens 4 096.
AutoRefine	**Step 1 (Initial Skill):** generated from the first attempt; input truncation: commands 6 000 chars, stdout 4 000 chars; temperature 0.5, max tokens 4 096. **Step 2 (Iterative Refinement):** each subsequent attempt refines the current skill document (truncated to 8 000 chars); temperature 0.4, max tokens 4 096.
EvoSkill	**Population:** 4 candidates with diversity hints (workflow ordering, failure modes, environment setup, implementation details). **Generations:** 2; top-k: 2 parents retained per generation. **Fitness:** LLM-based scoring (completeness, specificity, actionability, accuracy, organization; 0–100); temperature 0.2. **Mutation:** temperature 0.6; **Crossover:** temperature 0.5.

Table 11: Comparison of skill generation methods on SkillsBench. BL: baseline (no skill); Gen: with generated skill; MR: mean reward; Impr./Degr.: number of tasks improved/degraded by the generated skill relative to baseline. Δ values measure the gain attributable to the skill.

Method	BL Pass	BL MR	Gen Pass	Gen MR	Δ Pass	Δ MR	Impr.	Degr.
Trace2Skill	16.9%	0.218	27.1%	0.348	+10.2%	+0.130	12	4
AutoRefine	22.0%	0.261	40.7%	0.434	+18.6%	+0.172	19	9
EvoSkill	20.3%	0.246	30.5%	0.353	+10.2%	+0.107	11	2
SPARK (Ours)	22.0%	0.240	**50.8%**	**0.531**	**+28.8%**	**+0.291**	**21**	**3**

(`deepseek-chat` was retired by the provider during this study). Each method runs its own exploration and its own no-skill baseline.

Implementation. We implement both methods to the letter of their published algorithms. GEPA uses Pareto-frontier candidate selection, the strict acceptance rule that a child must outscore its parent on the same minibatch, and reflective mutation targeting the lowest-scoring instances. ACE uses the Generator/Reflector/Curator separation, delta-only updates applied by deterministic merge, and grow-and-refine pruning, the mechanism that prevents context collapse.

Results. Table 12 reports the comparison. SPARK starts from the lowest baseline of the three and reaches the highest endpoint, improving 24 tasks against 17 for either method.

Table 12: End-to-end comparison with prompt- and context-evolution methods on 59 tasks. Each method runs its own exploration and its own no-skill baseline (BL). MR: mean reward; Impr./Degr.: tasks improved/degraded relative to that method's own baseline.

Method	BL MR	Gen MR	Δ MR	Δ Pass	Impr.	Degr.
GEPA (Agrawal et al., 2026)	0.352	0.538	+0.186	+18.6%	17	8
ACE (Zhang et al., 2026)	0.419	0.581	+0.163	+16.9%	17	5
SPARK (Ours)	0.351	**0.592**	**+0.241**	**+27.1%**	**24**	8

Why these methods do not substitute for PDI. GEPA and ACE both improve an artifact by observing how well a student performs with it, so a downstream reward must exist before either can act. PDI is computed from the trajectory alone and therefore fires mid-exploration, before any student has run the skill: the two are complementary, with PDI governing how evidence is gathered and reward-driven evolution refining an artifact once an evaluator is in place.

M EXTERNAL BENCHMARK: ALFWORLD

To evaluate whether SPARK generalizes beyond terminal-based programming tasks, we conduct an external evaluation on ALFWorld (Shridhar et al., 2021), a text-based interactive household environment where agents issue natural-language commands to manipulate objects. ALFWorld differs fundamentally from SkillsBench: actions are discrete text commands rather than shell operations, feedback is environment state descriptions rather than program output, and success requires multi-step physical reasoning. This evaluation simultaneously validates both the SPARK skill generation pipeline and the PDI-guided refinement mechanism in an out-of-distribution setting.

Setup. We evaluate on the ALFWorld `eval_out_of_distribution` split (134 games, 6 task types). The agent framework is ReAct (1-shot, per task type) with a maximum of 30 steps per game. Table 13 summarizes the evaluation configuration.

Table 13: ALFWorld evaluation configuration.

Parameter	Value
Benchmark	ALFWorld (eval_out_of_distribution, 134 games)
Task types	6 (Pick & Place, Clean & Place, Heat & Place, Cool & Place, Examine, Pick Two)
Teacher models	Claude Sonnet 4.5, GPT-5.5
Student model	Claude Haiku 4.5
Teacher exploration	Per teacher
Student evaluation	3 conditions (No Skill, +Skill w/o PDI, +Skill w/ PDI) $\times$ 2 teachers
Agent framework	ReAct (1-shot, per task type)
Max steps	30 steps/game

Skill generation and PDI refinement. For each teacher, we run 30 exploration games and collect trajectories. Skills are distilled per task type under the same two conditions as Section 4.3: with the PDI monitor disabled, and with PDI-guided online intervention enabled.

Results. Table 14 and Table 15 report per-task-type success rates for the two teacher models. Under both teachers, SPARK-generated skills improve overall success rate over the no-skill baseline, and PDI-guided skills consistently outperform their non-PDI counterparts. With GPT-5.5 as teacher, PDI-refined skills achieve 40.0% overall success (vs. 16.7% baseline), with gains appearing even on task types where the teacher itself never succeeded during exploration (e.g., Pick & Place, Cool & Place), indicating that PDI-guided distillation can extract transferable procedural patterns from partial successes in related task types. For brevity, we omit *Heat & Place* and *Pick Two* from the tables, since the baseline is already 0% and all three conditions remain at 0% (no change, neither gain nor drop).

Table 14: ALFWorld results with Claude Sonnet 4.5 as teacher (exploration success: 25%). Student: Claude Haiku 4.5.

Task Type	No Skill	+Skill (no PDI)	+Skill (PDI)
Pick & Place	0.0%	0.0%	0.0%
Clean & Place	0.0%	20.0%	**40.0%**
Cool & Place	0.0%	0.0%	0.0%
Examine	71.4%	85.7%	**100.0%**
Overall	16.7%	23.3%	**30.0%**

Reproduction. All experiments use the ALFWorld Python package (`pip install alfworld`) with the default `eval_out_of_distribution` split. Teacher exploration runs with 10 parallel workers. Skill distillation uses `claude-sonnet-4.5` with and without the PDI flag. Student evaluation runs across 3 conditions with 10 parallel workers per condition. Scripts are located in `extra/benchmark/scripts/`: `launch_parallel.py` (teacher explo-

Table 15: ALFWorld results with GPT-5.5 as teacher (exploration success: 23%). Student: Claude Haiku 4.5.

Task Type	No Skill	+Skill (no PDI)	+Skill (PDI)
Pick & Place	0.0%	0.0%	**25.0%**
Clean & Place	0.0%	20.0%	**40.0%**
Cool & Place	0.0%	14.3%	**28.5%**
Examine	71.4%	**100.0%**	**100.0%**
Overall	16.7%	30.0%	**40.0%**

ration), `skill_gen.py` (distillation), `launch_parallel_eval.py` (student evaluation), and `generate_tables.py` (result aggregation).