SKILLGEN: Verified Inference-Time Agent Skill Synthesis

Yuchen Ma[1]* **Yue Huang**[2]* **Han Bao**[2] **Haomin Zhuang**[2]
Swadheen Shukla[3] **Michel Galley**[3] **Xiangliang Zhang**[2] **Stefan Feuerriegel**[1]
[1]Munich Center for Machine Learning, LMU Munich
[2]University of Notre Dame [3]Microsoft Research

Abstract

Skills are a promising way to improve LLM agent capabilities without retraining, while keeping the added procedure reusable and controllable. However, high-quality skills are still largely written by hand. We introduce SKILLGEN, a multi-agent framework that synthesizes a single auditable skill from trajectories generated by a base agent. The output is a human-readable artifact that can be inspected before use. Rather than merely summarizing trajectories, SKILLGEN leverages contrastive induction over both successful and failed trajectories to identify reusable success patterns, recurring failure modes, and behaviors that appear in nearby successes but are missing from failures. SKILLGEN then generates candidate skills and iteratively refines the skill. A key novelty in SKILLGEN is that we model agent skills as interventions to empirically verify the net effect of skills on the overall performance. Specifically, we compare outcomes on the same instances with and without the skill, so that we account for both repairs (cases where the skill fixes a baseline failure) and regressions (cases where the skill breaks a baseline success). Across a broad range of agents and datasets, SKILLGEN consistently improves held-out performance, outperforms existing skill-generation baselines, and produces skills that transfer across models.

1 Introduction

Large language models (LLMs) are increasingly used to solve complex, multi-step tasks (Schick et al., 2023; Qin et al., 2024; Yao et al., 2023; Wang et al., 2023a). A common way to formalize such behavior is through ***skills***: reusable, inference-time procedures that encode task-specific guidance, such as instructions, executable code, and domain knowledge, without modifying model weights (Zhang et al., 2025; Anthropic, 2025). Skills are modular and auditable: because they are readable inference-time artifacts rather than weight updates or prompt searches, one can inspect the procedure they encode, revise it directly, and test its effect before deployment. In practice, however, high-quality skills are still largely hand-written.

Automated skill synthesis aims to learn reusable skills from agent experience (Shinn et al., 2023; Zhao et al., 2024; Ni et al., 2026; Alzubi et al., 2026; Wang et al., 2026a; Zhang et al., 2026). However, existing methods have two key shortcomings. First, existing methods primarily learn from successful trajectories, and even when failures are considered, they are typically summarized in isolation rather than contrasted against nearby successes on the same task. As a result, prior work misses a key contrastive signal between success and failure—that is, what the agent executes correctly in similar contexts and what it omits in failed roll-outs. For example, a successful trajectory may include an intermediate validation step that is absent in a failed attempt, but success-only learning does not isolate that such intermediate validation is important and would not put into a reusable pattern.

*Equal contribution. Contact emails: yuchen.ma@lmu.de and yhuang37@nd.edu.

Preprint.

Second, existing methods do not explicitly *verify* the empirical benefit of a generated skill. While a skill may repair some failures, it can also introduce new failure modes on cases that the agent previously solved correctly. As a result, skill synthesis is fundamentally an interventional problem, where one compares the net-effect on the agent's performance with and without the candidate skill. Also, such performance evaluation is necessary to eventually build iterative approaches to refine candidate skills in a principled manner.

We introduce SKILLGEN: a *multi-agent framework for automatic, inference-time skill synthesis* (see Fig. 1). SKILLGEN takes an existing dataset of LLM trajectories as input and derives a single auditable skill: a readable intervention whose task context, success procedures, and failure lessons can be inspected and whose empirical net effect is verified before deployment. The input dataset can be collected during a baseline elicitation phase to compile successful and failed trajectories. Our framework operates through three specialized agents: (1) A **contrastive induction agent** analyzes the input trajectories to extract reusable success patterns and identify recurring failure modes, with the aim to surface contrasts between successful vs. failed roll-outs. As a result, it outputs a compact and interpretable summary with task diagnostics. (2) In a generation-verification-refinement loop, the diagnostics are converted into candidate skills, and the skills are then iteratively refined based on feedback (using a **generation agent** and a **verification agent**. The final skill is selected by measuring the net-effect on the final held-out performance. This ensures that the selected skill improves the overall performance and thus accounts for *"repairs"* (i.e, when a skill fixes a failure) and *"regressions"* (i.e., when a skill breaks a correct case). To the best of our knowledge, SKILLGEN is the first agentic framework that models inference-time skill synthesis as an intervention problem to ensure a positive, empirically verified effect on performance.

We also demonstrate the effectiveness of SKILLGEN across a broad range of interactive, scientific, coding, and other tool-use benchmarks. We further evaluate SKILLGEN using several open-weight and proprietary base LLMs. As our main result, SKILLGEN improves average accuracy for all eight evaluated base LLMs, with held-out gains ranging from $+3.27$ to $+10.08$ percentage points. We also compare SKILLGEN against state-of-the-art skill-generation baselines (Ni et al., 2026; Wang et al., 2026a; Alzubi et al., 2026; Zhang et al., 2026), where SKILLGEN is consistently positive and achieves the largest average improvement by a clear margin. Our ablations show that contrastive induction, verification-guided refinement, and the verification gate each contribute to the performance gains. We also perform a cross-model transfer analysis to show that generated skills are generalizable and not tied to the LLM that produced them.

Contributions. Our main contributions are three-fold:[2](1) We formulate a general, end-to-end learning task for automatic inference-time skill synthesis: to produce a single, auditable skill that improves a base agent. (2) We introduce SKILLGEN, a multi-agent framework that learns from both failed and successful trajectories via contrastive induction, and then generates new candidate skills that are iteratively refined and verified. The final skills are selected to have a positive net-effect on the overall performance. (3) We provide an extensive empirical study to demonstrate consistent and large held-out performance gains. SKILLGEN outperforms state-of-the-art skill-generation baselines and produces skills that transfer across models without parameter updates.

2 Preliminaries

We view inference-time skills as *interventions* that modify the behavior of a base agent and thereby change its task performance. This perspective naturally induces a comparison between outcomes with and without a given skill on the same inputs.

Task setting. Let $\mathcal{X}$ be the input space, p a task distribution over $\mathcal{X}$, and $\mathcal{T}$ the space of agent trajectories. A trajectory $\tau \in \mathcal{T}$ consists of the full sequence of LLM interactions, including messages, tool calls, environment observations, and the final output. For skill synthesis, we split the training data into: (i) an induction subset $\mathcal{D}_{\text{ind}} = \{x_i\}_{i=1}^{n}$ used to analyze agent behavior, and (ii) a construction-time verification subset $\mathcal{D}_{\text{ver}} = \{\tilde{x}_j\}_{j=1}^{m}$ used for evaluating and selecting candidate skills. We consider a *base agent* $\mathcal{A}$ that maps inputs to trajectories and that we seek to improve upon. We model $\mathcal{A}$ as a stochastic trajectory kernel $P_{\mathcal{A}}(\tau \mid x; \eta)$, where x is the task instance and η is an inference-time intervention loaded into the agent's context. The empty intervention $\eta = \varnothing$ defines the "no-skill" behavior, defined by $\tau^0(x) \sim P_{\mathcal{A}}(\cdot \mid x; \varnothing)$.

[2]Code is available via https://github.com/yccm/SkillGen.

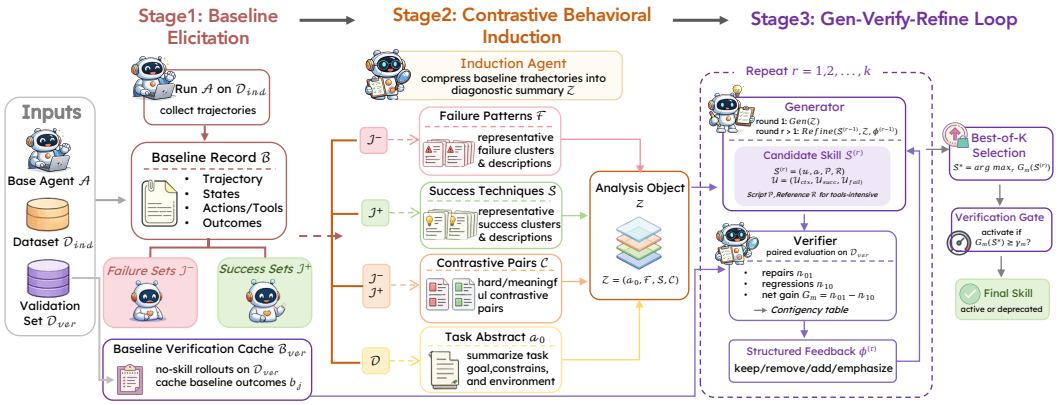

Figure 1: **SKILLGEN overview.** Our multi-agent framework synthesizes a single auditable skill from baseline trajectories. (1) It first elicits successful and failed rollouts as input. (2) It extracts reusable patterns of successful and failure modes. (3) It follows an iterative generation-verification-refinement loop to generate and refine new candidate skills.

To formalize the outcome Y, we define a task-level evaluator $\mathcal{E} : \mathcal{X} \times \mathcal{T} \to [0, 1]$. In practice, this could be an LLM-as-a-judge, a benchmark score, or a successful check against some environment outcome. As a result, the evaluator assigns a success probability to each instance–trajectory pair. The observed outcome is $Y(x, \tau) \sim \text{Bernoulli}(\mathcal{E}(x, \tau))$, with deterministic evaluators as the special case $\mathcal{E}(x, \tau) \in \{0, 1\}$. For any instance x, we define the baseline outcome $Y^0(x) = Y(x, \tau^0(x))$; for induction instances, we write $\tau_i^0 = \tau^0(x_i)$, and let y_i^0 denote the realized outcome of the base agent.

Skill interventions: We define a candidate *skill* as a inference-time intervention $s = (u, a, \mathcal{P}, \mathcal{R})$, where u is a structured prompt, a is task metadata (e.g., task description), $\mathcal{P}$ is an optional set of executable scripts, and $\mathcal{R}$ is an optional collection of auxiliary documents. Together, these components define the skill space considered by SKILLGEN.

We model skills as interventions that change the agent's behavior and thus its outcomes. To make comparative assessments of skill learning, we adopt the potential outcome framework (Rubin, 2005) as a principled manner to formalize the treatment effects. For any input x and a candidate skill s, we define two potential outcomes: the baseline outcome (i.e., $Y^0(x) = Y(x, \tau^0(x))$) and the skill-augmented outcome (i.e., $Y^s(x) = Y(x, \tau^s(x))$, $\tau^s(x) \sim P_{\mathcal{A}}(\cdot \mid x; \eta(s))$. Loading a skill s corresponds to applying an intervention $\eta(s)$ into $\mathcal{A}$.

Objective: Our goal is to measure and maximize the expected effect of a skill relative to the baseline agent:

$$\Delta(s) \;=\; \mathbb{E}_{x \sim p}\left[\mathbb{E}[Y^s(x) \mid x, s] - \mathbb{E}\big[Y^0(x) \mid x\big]\right]. \tag{1}$$

Thus, $\Delta(s)$ captures the net-effect induced by a skill intervention s: it measures how much the skill improves (or degrades) performance on the same input distribution, while accounting for both "repairs" (i.e., cases where the skill fixes a baseline failure) and "regressions" (i.e., cases where the skill breaks a baseline success). The objective for skill synthesis is therefore to select a skill with positive net-effect on held-out performance, but without relying on human-written task-specific skills. During construction, each candidate skill is evaluated on $\mathcal{D}_{\text{ver}}$ under identical inputs with and without the skill. As a result, we yield a so-called *status* $\sigma_{\text{ver}}(s; \mathcal{D}_{\text{ver}}) \in \{\text{active}, \text{deprecated}\}$. At deployment, only active skills are loaded; deprecated skills are subsumed under the empty intervention $\varnothing$.

3 SKILLGEN

Overview. SKILLGEN takes as input: a base agent, a set of observed LLM trajectories (split into an induction subset and verification subset), and a task-level evaluator. SKILLGEN then returns a single auditable skill. SKILLGEN follows an agentic, three-staged framework (see Fig. 1; pseudocode in Alg. 1 in Appendix A). *Stage* (1) baseline elicitation: this stage uses the base agent to collect successful vs. failed trajectories. *Stage* (2) contrastive induction: this stage extracts recurring failure modes to identify local patterns that distinguish successful vs. failed roll-outs; the patterns are combined into a compact, interpretable summary of task-level diagnostics (via the **induction agent**).

Stage ③ an iterative generation–verification–refinement loop: this stage turns the diagnostics into candidate skills (via the **generation agent**), tests each candidate skill on the verification subset for performance evaluation (via the **verification agent**), and finally makes refinements to the candidate skill. Finally, the candidate skill with the largest construction-time net-effect $\Delta(s)$ is returned.

3.1 *Stage* ① : Baseline Elicitation

We first run the base agent on the induction subset and store

$$\mathcal{B} = \{(x_i, \tau_i^0, y_i^0)\}_{i=1}^n, \qquad \mathcal{I}^- = \{i : y_i^0 = 0\}, \quad \mathcal{I}^+ = \{i : y_i^0 = 1\}. \tag{2}$$

Here, $\mathcal{I}^-$ indexes failed baseline rollouts and $\mathcal{I}^+$ indexes successful ones. Intuitively, failures show where the base agent would need help; and successes show procedures the base agent can already execute. Using both strata is unique to SKILLGEN and important: failures alone can produce misleading or unhelpful advice, while successes alone do not identify the capability gap.

We further cache no-skill outcomes on the construction-time verification subset, i.e.,

$$\mathcal{B}_{\text{ver}} = \{(\tilde{x}_j, \tilde{\tau}_j^0, b_j)\}_{j=1}^m, \qquad \tilde{\tau}_j^0 \sim P_{\mathcal{A}}(\cdot \mid \tilde{x}_j; \varnothing), \qquad b_j = Y(\tilde{x}_j, \tilde{\tau}_j^0). \tag{3}$$

The cached outcomes are neither used by the induction agent nor the first-round generation agent; instead, the cached outcomes are later used for construction-time verification and subsequent refinement. The main motivation is that we can later compare each candidate's skill against the behavior of the no-skill agent on the same verification subset.

3.2 *Stage* ②: Contrastive Behavioral Induction

The **induction agent** compresses baseline trajectories into an explicit summary diagnostic for skill synthesis:

$$\text{Compress}(\mathcal{B}) = \mathcal{Z} = (a_0, \mathcal{F}, \mathcal{S}, \mathcal{C}), \tag{4}$$

where a_0 is a task-level summary of the induction inputs, $\mathcal{F}$ is a set of cluster-level failure summaries, $\mathcal{S}$ is a set of cluster-level success summaries, and $\mathcal{C}$ is a set of local contrastive observations between nearby failed and successful rollouts. Any of the three set-valued components may be empty (e.g., if the corresponding success or failure stratum is empty. Stage ③ receives only $\mathcal{Z}$, so this stage converts variable-length LLM trajectories into a lower-dimensional diagnostic summary for skill generation.

• **Task summary (a_0).** The induction agent applies a fixed abstraction prompt, denoted by Abs, to the induction inputs and writes a task summary $a_0 = \text{Abs}(\{x_i\}_{i=1}^n)$. The summary is designed to describe the task family rather than any single instance, giving the generation agent a general description of what the skill is for. The component a_0 is part of the diagnostic summary $\mathcal{Z}$ and is distinct from the skill metadata a in $s = (u, a, \mathcal{P}, \mathcal{R})$.

• **Failure analysis ($\mathcal{F}$).** For each failed rollout $i \in \mathcal{I}^-$, the induction agent writes a root-cause summary ρ_i^-. Let ϕ be a text encoder applied to a serialized input–summary pair, and define $e_i^- = \phi([x_i; \rho_i^-])$. We cluster the resulting failure embeddings via

$$\Pi^- = \text{Cluster}\left(\{e_i^- : i \in \mathcal{I}^-\}\right), \qquad f_P = \text{Summ}^-\left(\{(x_i, \tau_i^0, \rho_i^-) : i \in P\}\right) \tag{5}$$

for each cluster $P \in \Pi^-$. Each f_P is a cluster-level failure summary: it describes the recurring root cause, the trajectory point at which the failure typically appears, and the corrective rule that would avoid the failure. The resulting set is $\mathcal{F} = \{f_P\}_{P \in \Pi^-}$.

• **Success analysis ($\mathcal{S}$).** For each $i \in \mathcal{I}^+$, an LLM writes a success summary ρ_i^+, and we further embed $e_i^+ = \phi([x_i; \rho_i^+])$. We apply the same embedding and clustering procedure to successful rollouts:

$$\Pi^+ = \text{Cluster}\left(\{e_i^+ : i \in \mathcal{I}^+\}\right), \qquad h_P = \text{Summ}^+\left(\{(x_i, \tau_i^0, \rho_i^+) : i \in P\}\right), \tag{6}$$

and $\mathcal{S} = \{h_P\}_{P \in \Pi^+}$. Each h_P is a cluster-level success summary: it describes the reusable procedure, the task conditions under which it appears, and checks that make the procedure robust.

• **Local contrastive analysis ($\mathcal{C}$).** The above cluster summaries can miss some of the "small" action choices that separate a success from a failure. When $\mathcal{I}^+$ is non-empty, for each failed instance $i \in \mathcal{I}^-$, we retrieve the nearest successful neighbor under embedding distance d, i.e.,

$$j(i) = \arg\min_{j \in \mathcal{I}^+} d(e_i^-, e_j^+). \tag{7}$$

The induction agent first checks whether x_i and $x_{j(i)}$ share the same task type. If they do, the induction agent compares the two full trajectories and generates a contrastive observation

$$c_i = \mathrm{Contr}(x_i, \tau_i^0, x_{j(i)}, \tau_{j(i)}^0),$$

which describes the behavior present in the successful rollout and the corresponding behavior omitted in the failed rollout. The observations whose pairs pass the same-task check form $\mathcal{C}$. Thus, $\mathcal{C}$ provides local contrastive evidence: it anchors advice in behavior that the same base agent has already demonstrated, but that was absent in a nearby failure.

3.3 *Stage* ③: Generation–Verification–Refinement Loop

Overview. Stage ③ turns the diagnostic summary $\mathcal{Z}$ into a sequence of candidate skills and uses paired verification to decide which candidate should be deployed. The loop is designed to improve baseline failures while explicitly tracking regressions on instances that the base agent already solved. It consists of four steps: **(i)** *generation*, which produces candidate skills from the diagnostic summary; **(ii)** *verification*, which evaluates each candidate on the verification subset; **(iii)** *refinement*, which updates candidates using structured feedback from repairs and regressions; and **(iv)** *selection*, which returns the candidate with the largest verified net gain for deployment. Let $K \geq 1$ denote the round budget. We index rounds by $r \in \{1, \ldots, K\}$, write $s^{(r)}$ for the candidate skill at round r, write $\Phi^{(r)}$ for the feedback produced after verifying $s^{(r)}$, and write $s^\star$ for the selected skill.

• **(i) Generation:** In each round r, the **generation agent** uses the diagnostic summary $\mathcal{Z}$ as well as feedback from the previous round (with $\Phi^{(0)} = \varnothing$) to produce a new candidate skill

$$s^{(r)} = (u^{(r)}, a^{(r)}, \mathcal{P}^{(r)}, \mathcal{R}^{(r)}). \tag{8}$$

Skill structure: To write the new candidate skill, we use a prompt template with a fixed three-part schema

$$u^{(r)} = (u_{\mathrm{ctx}}^{(r)}, u_{\mathrm{succ}}^{(r)}, u_{\mathrm{fail}}^{(r)}) \tag{9}$$

in natural language, where: (i) u_{ctx} encodes task context, i.e., a concise description of the task distribution and constraints (derived from a_0); (ii) u_{succ} encodes reusable success patterns distilled from $\mathcal{S}$ and the successful instances from the contrastive analysis $\mathcal{C}$; and (iii) u_{fail} encodes reusable failure-avoidance patterns derived from $\mathcal{F}$ and the negative instances from the contrastive analysis $\mathcal{C}$. The above schema acts as a constrained projection from the diagnostic summary to the skill space. Intuitively, the idea here is to learn patterns with reusable procedures that define successful vs failure instances, so that the refinement can help encourage the former and avoid the latter.

For tool-intensive tasks, the generation agent may additionally emit scripts $\mathcal{P}^{(r)}$ and reference documents $\mathcal{R}^{(r)}$; however, after round $r > 1$, refinement edits are restricted to the body $u^{(r)}$ in natural language, which keeps the tool interface fixed to prevent uncontrolled expansion.

• **(ii) Verification:** The **verification agent** evaluates each candidate skill on all instances in the verification subset $\mathcal{D}_{\mathrm{ver}}$. For a candidate skill s, we load the intervention $\eta(s)$ into the base agent and roll it out on each $\tilde{x}_j \in \mathcal{D}_{\mathrm{ver}}$, i.e.,

$$z_j(s) = Y(\tilde{x}_j, \tilde{\tau}_j^s), \qquad \tilde{\tau}_j^s \sim P_{\mathcal{A}}(\cdot \mid \tilde{x}_j; \eta(s)). \tag{10}$$

Causal evaluation of skill intervention : We treat a candidate skill s as an intervention on the base agent and evaluate the effect by comparing outcomes with and without the intervention on the same inputs. For each $\tilde{x}_j \in \mathcal{D}_{\mathrm{ver}}$, we observe the baseline outcome $b_j = Y^0(\tilde{x}_j)$ and the skill-augmented outcome $z_j(s) = Y^s(\tilde{x}_j)$. Applying the skill to all verification instances yields a direct comparison between Y^0 and Y^s on identical inputs. In this view, *"repairs"* correspond to $Y^0 = 0 \to Y^s = 1$, while *"regressions"* correspond to $Y^0 = 1 \to Y^s = 0$.

Comparative metrics. We aggregate outcomes via

$$n_{\alpha\beta}(s) = \sum_{j=1}^{m} \mathbf{1}\{Y^0(\tilde{x}_j) = \alpha,\, Y^s(\tilde{x}_j) = \beta\}, \tag{11}$$

with repairs $n_{01}(s)$ and regressions $n_{10}(s)$. The empirical net-effect under this comparison is $\widehat{\Delta}_m(s)$,

$$\widehat{\Delta}_m(s) = \frac{1}{m} \sum_{j=1}^{m} \big(Y^s(\tilde{x}_j) - Y^0(\tilde{x}_j)\big) = \frac{n_{01}(s) - n_{10}(s)}{m}, \qquad G_m(s) = n_{01}(s) - n_{10}(s). \tag{12}$$

For a fixed, non-adaptively chosen skill and i.i.d. verification instances, $\mathbb{E}[\widehat{\Delta}_m(s)] = \Delta(s)$.

• **(iii) Refinement:** Refinement uses structured feedback to update the skill. • *Feedback signals.* After each round, the verification agent summarizes the diagnostic evidence rather than sending raw trajectories back to the generation agent. For this, the verification agent partitions instances into

$$\mathcal{Q}^{(r)}_{\text{repair}} = \{j : b_j = 0,\, z_j(s^{(r)}) = 1\}, \quad \mathcal{Q}^{(r)}_{\text{regress}} = \{j : b_j = 1,\, z_j(s^{(r)}) = 0\}, \quad \mathcal{Q}^{(r)}_{\text{fail}} = \{j : b_j = 0,\, z_j(s^{(r)}) = 0\}, \tag{13}$$

Here, $\mathcal{Q}^{(r)}_{\text{repair}}$ contains baseline failures repaired by $s^{(r)}$, $\mathcal{Q}^{(r)}_{\text{regress}}$ contains baseline successes broken by $s^{(r)}$, and $\mathcal{Q}^{(r)}_{\text{fail}}$ contains baseline failures that remain unresolved. • *Feedback aggregation.* The verification agent creates explanations of how the skill affected selected repairs, regressions, and unresolved failures. The verification agent then aggregates these explanations into

$$\Phi^{(r)} = \big(\Phi^{(r)}_{\text{keep}}, \Phi^{(r)}_{\text{remove}}, \Phi^{(r)}_{\text{add}}, \Phi^{(r)}_{\text{emphasize}}\big), \tag{14}$$

which specifies, for the next round, which parts of the current skill to keep, remove, add, and emphasize. • *Update rule.* The refinement uses the following update (to avoid writing a new prompt from scratch):

$$s^{(r)} = \begin{cases} \text{Gen}(\mathcal{Z}), & r = 1, \\ \text{Refine}(s^{(r-1)}, \mathcal{Z}, \Phi^{(r-1)}), & r > 1. \end{cases} \tag{15}$$

• **(iv) Final skill selection:** Since later refinement rounds need not improve empirical performance, SKILLGEN performs a best-of-K selection over the candidate sequence $\{s^{(r)}\}_{r=1}^{K}$ and returns the candidate skill with the largest construction-time net gain G_m, i.e.,

$$r^\star = \arg\max_{1 \le r \le K} G_m(s^{(r)}), \qquad s^\star = s^{(r^\star)}. \tag{16}$$

Verification gate. A candidate skill is marked 'active' only if it satisfies

$$G_m(s^\star) \ge \gamma_m, \qquad \gamma_m = \max\{g_{\text{abs}}, \lceil g_{\text{rel}} m \rceil, 1\}. \tag{17}$$

Otherwise, it is marked 'deprecated' and replaced by the empty intervention.[3] The threshold due to γ_m defines the construction-time deployment rule used by SKILLGEN: across refinement rounds, we first select the candidate with the largest G_m, and then mark the selected skill as active if it satisfies Eq. (17); otherwise, the skill is deprecated.

Deployment: At runtime, an active skill is injected into a dedicated slot of the system prompt. Reference documents are retrieved via on-demand loading through `skill_load_reference`; executable scripts encode only declared top-level functions with prefixed `skill_`. This ensures that the deployed capability matches the verified skill.

4 Experiments

We evaluate SKILLGEN on held-out test instances across interactive, scientific, coding, web, and tool-use benchmarks; full implementation details are in Appendix C. All claims use paired held-out evaluations: after construction is complete, the same task instances are rolled out with and without the generated skill.

RQ1 Does SKILLGEN improve base agents across model families and benchmark domains?

Before any held-out rollout, each skill and its active/deprecated status is fixed using only the skill-training dataset: the induction subset for trajectory analysis and the construction-time verification subset for refinement and selection. Table 1 reports the no-skill baseline accuracy, the skill-augmented accuracy, and the absolute accuracy change over 80 held-out benchmark–split–model combinations.

Table 1 shows three main patterns: (i) SKILLGEN improves average accuracy for all eight base agents, with gains from $+3.27$ to $+10.08$ percentage points; (ii) the effect holds across both open-weight

[3] Here, $g_{\text{abs}} \in \mathbb{Z}_{\ge 0}$ is an absolute minimum number of net repairs, and $g_{\text{rel}} \in [0, 1]$ is a relative minimum as a fraction of the construction-time verification subset. The gate is a simple construction-time safeguard: the absolute term prevents deploying candidates whose gain is negligible in count, the relative term requires the gain to scale with the size of the verification subset, and the final lower bound of 1 requires a strictly positive construction-time net gain.

Table 1: **Main results across open-weight and proprietary models.** For each model, we report the no-skill baseline accuracy (BASE), the skill-augmented accuracy (SKILL), and the absolute accuracy change (Δ) on held-out test instances. Values are from the paired rollout per instance under the split protocol in Appendix C.2.

Open-weight models

Dataset	Split	Gemma-4-26B			Llama-3.1-8B			Mistral-Nemo			Qwen-2.5-7B		
		BASE	SKILL	Δ	BASE	SKILL	Δ	BASE	SKILL	Δ	BASE	SKILL	Δ
ALFWorld	IOD	78.67%	86.00%	7.33%	68.00%	70.67%	2.67%	63.33%	62.67%	-0.67%	64.00%	77.33%	13.33%
	OOD	82.35%	93.73%	11.37%	67.45%	65.10%	-2.35%	58.82%	67.06%	8.24%	69.41%	82.35%	12.94%
LiveCodeBench	–	83.33%	83.33%	0.00%	16.00%	20.00%	4.00%	11.33%	14.00%	2.67%	25.33%	25.33%	0.00%
MCPBench	All	70.00%	65.00%	-5.00%	17.50%	40.00%	22.50%	47.50%	50.00%	2.50%	40.00%	40.00%	0.00%
	Single	12.50%	12.50%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	6.25%	6.25%	0.00%
Mind2Web	–	52.00%	53.00%	1.00%	24.00%	24.00%	0.00%	20.00%	35.00%	15.00%	27.00%	28.00%	1.00%
PubMedQA	–	76.00%	78.00%	2.00%	62.00%	69.00%	7.00%	61.00%	67.00%	6.00%	66.00%	66.00%	0.00%
ScienceWorld	–	18.00%	34.00%	16.00%	33.00%	36.00%	3.00%	31.00%	41.00%	10.00%	23.00%	26.00%	3.00%
SocialMaze	FTS	98.00%	98.00%	0.00%	48.00%	48.00%	0.00%	62.00%	66.00%	4.00%	54.00%	54.00%	0.00%
	UPI	14.00%	14.00%	0.00%	16.00%	16.00%	0.00%	16.00%	16.00%	0.00%	12.00%	18.00%	6.00%
Avg.	–	**58.49%**	**61.76%**	**3.27%**	**35.20%**	**38.88%**	**3.68%**	**37.10%**	**41.87%**	**4.77%**	**38.70%**	**42.33%**	**3.63%**

Proprietary models

Dataset	Split	Claude-Haiku-4.5			GPT-5.4-Nano			GPT-5.4-Mini			Grok-4-Fast		
		BASE	SKILL	Δ	BASE	SKILL	Δ	BASE	SKILL	Δ	BASE	SKILL	Δ
ALFWorld	IOD	60.00%	66.67%	6.67%	82.67%	94.00%	11.33%	89.33%	96.67%	7.33%	66.67%	77.33%	10.67%
	OOD	61.18%	64.71%	3.53%	72.94%	94.90%	21.96%	93.33%	97.25%	3.92%	65.10%	80.39%	15.29%
LiveCodeBench	–	56.00%	56.00%	0.00%	59.33%	65.33%	6.00%	59.33%	65.33%	6.00%	84.67%	84.67%	0.00%
MCPBench	All	67.50%	82.50%	15.00%	80.00%	80.00%	0.00%	75.00%	75.00%	0.00%	47.50%	65.00%	17.50%
	Single	68.75%	81.25%	12.50%	18.75%	31.25%	12.50%	43.75%	43.75%	0.00%	12.50%	12.50%	0.00%
Mind2Web	–	60.00%	59.00%	-1.00%	51.00%	57.00%	6.00%	57.00%	61.00%	4.00%	67.00%	72.00%	5.00%
PubMedQA	–	74.50%	74.50%	0.00%	67.00%	69.00%	2.00%	68.00%	72.00%	4.00%	80.00%	80.00%	0.00%
ScienceWorld	–	41.00%	49.00%	8.00%	29.00%	54.00%	25.00%	40.00%	43.00%	3.00%	43.00%	56.00%	13.00%
SocialMaze	FTS	58.00%	80.00%	22.00%	64.00%	80.00%	16.00%	56.00%	80.00%	24.00%	98.00%	100.00%	2.00%
	UPI	22.00%	20.00%	-2.00%	18.00%	18.00%	0.00%	10.00%	14.00%	4.00%	24.00%	24.00%	0.00%
Avg.	–	**57.73%**	**62.52%**	**4.79%**	**54.27%**	**64.35%**	**10.08%**	**59.17%**	**64.80%**	**5.63%**	**58.84%**	**65.19%**	**6.35%**

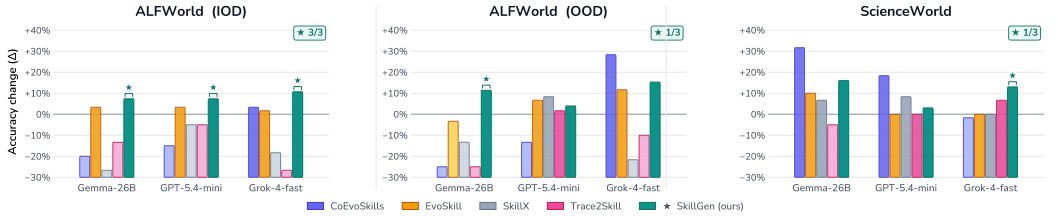

Figure 2: **Comparison with skill-generation baselines.** Accuracy improvement (Δ) from adding a generated skill across representative benchmark–model entries. Mini, Grok, and Gemma denote `GPT-5.4-Mini`, `Grok-4-Fast`, and `Gemma-4-26B`, respectively. All methods use the same evaluation harness.

models (+3.27 to +4.77 pp) and proprietary models (+4.79 to +10.08 pp); and (iii) out of 80 held-out benchmark–split–model entries, 50 improve, 25 remain unchanged, and only 5 show regressions. The largest gains appear on procedural, multi-step benchmarks: `ALFWorld` improves in 14 of 16 entries, and `ScienceWorld` improves for all eight agents. Further, SKILLGEN is especially useful when the base model has enough task capability to execute a learned procedure but still has room to improve.

RQ2 How does SKILLGEN compare with state-of-the-art automatic skill-generation baselines?

We compare SKILLGEN against four recent skill-generation baselines: **Trace2Skill** (Ni et al., 2026), **SkillX** (Wang et al., 2026a), **EvoSkill** (Alzubi et al., 2026), and **CoEvoSkills**, a co-evolutionary baseline instantiated from EvoSkills (Zhang et al., 2026). The implementation details of baselines are in Appendix C.6. We evaluate on `ALFWorld` IOD, `ALFWorld` OOD, and `ScienceWorld`, using three agent models spanning different capability tiers and providers. Figure 2 summarizes Δ accuracy

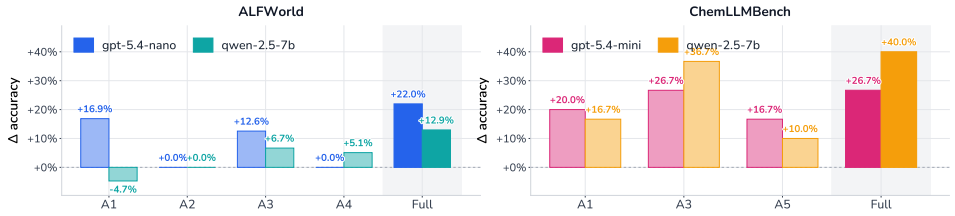

Figure 3: **SKILLGEN ablations.** Δ accuracy over a shared no-skill baseline on `ALFWorld` (OOD) and `ChemLLMBench` yield prediction. **A1**: ICL ($k = 3$) instead of the induced skill; **A2**: no refinement; **A3**: no verification gate; **A4**: no Failure Lessons; **A5**: plain-text skill (no script+reference bundle); **Full**: complete SKILLGEN. **Full** wins on every dataset–model pair, showing that each component contributes.

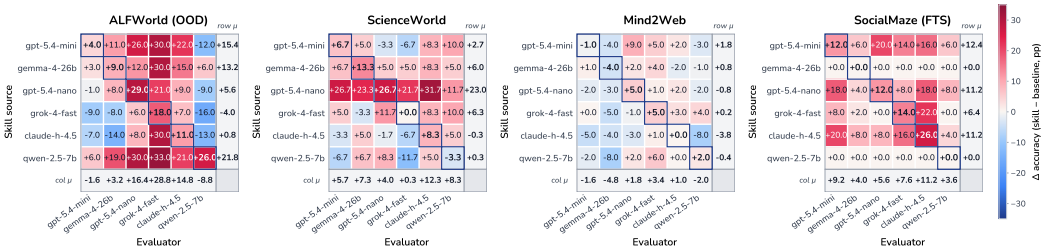

Figure 4: **Cross-model skill transferability.** Each heatmap reports Δ accuracy when a skill generated by a source model (row) is executed by an evaluator model (column). Diagonal cells are self-transfer, while off-diagonal cells are cross-model transfer. Right and bottom margins show transfer-out and transfer-in means, respectively; color saturates at ±30 pp. The transfer matrix is evaluated on a shared pool of 100 held-out instances per benchmark, distinct from the main evaluation split, to ensure that baseline trajectories are consistent across all 36 (source, evaluator) pairs.

across the benchmark–model entries. We observe that SKILLGEN leads to consistent gain across settings and achieves the largest overall improvement.

RQ3 Which components are necessary for reliable skill construction?

Figure 3 compares SKILLGEN against several ablations on representative prediction tasks and shows that each component is relevant for overall performance. We find: (i) the induced skill outperforms simple $k = 3$ demonstration reuse, so the improvement is not just retrieval; (ii) refinement and the verification gate are both needed for reliable interactive-task gains, because early candidates can repair some failures while introducing regressions; and (iii) task-specific skill structure is also relevant (e.g., the failure patterns help on `ALFWorld` OOD and the script+reference bundle helps on `ChemLLMBench`). The complete SKILLGEN system achieves the best result on every dataset–model pair in the ablation study.

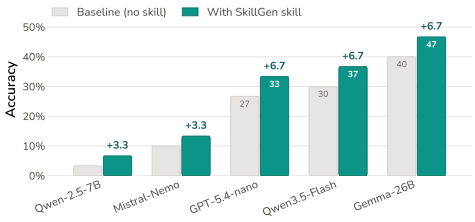

Figure 5: **Insights for τ-`Bench`.** Held-out accuracy on τ-`Bench` retail for the five models where the SKILLGEN verification gate activated. Gray bars are no-skill baselines and teal bars apply the induced skill; deltas are absolute percentage-point changes.

RQ4 Are generated skills transferable across agents?

We evaluate transfer by reusing the final SKILLGEN skill from one source model without retraining it, then executing it with a different evaluator model on `ALFWorld` OOD, `ScienceWorld`, `Mind2Web`, and `SocialMaze` FTS. Each transferred skill is compared against the evaluator's own no-skill baseline; skills marked 'deprecated' by the source pipeline are retained as no-op skills. Figure 4 shows that SKILLGEN produces skills that often transfer across models, but relevant is the choice of skill-generating model. Across 120 off-diagonal comparisons, 70% are non-negative, and 42% exceed +5 pp. We see a clear pattern: transferable skills are not simply written by the strongest baseline agents; on `ALFWorld`, `Qwen-2.5-7B` is the best skill-generating model on average, while, on `ScienceWorld`, `GPT-5.4-Nano` is best.

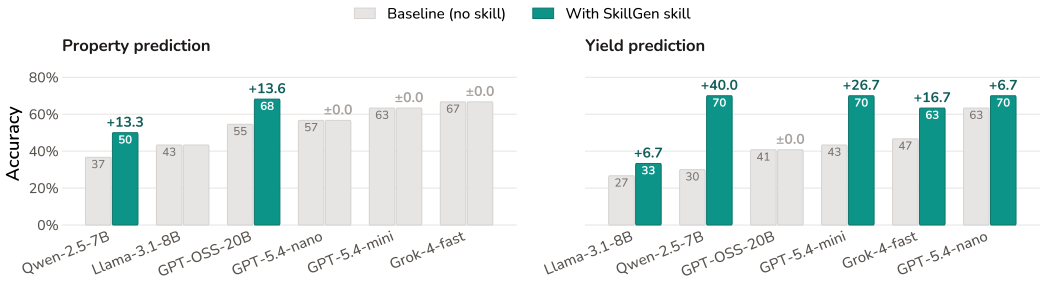

Figure 6: **Insights for `ChemLLMBench`.** Held-out accuracy on `ChemLLMBench` property prediction (left) and yield prediction (right). Gray bars are no-skill baselines and teal bars apply the SKILLGEN skill; bars labeled "±0.0" or "gate off" indicate no measurable change or rejection by the verification gate.

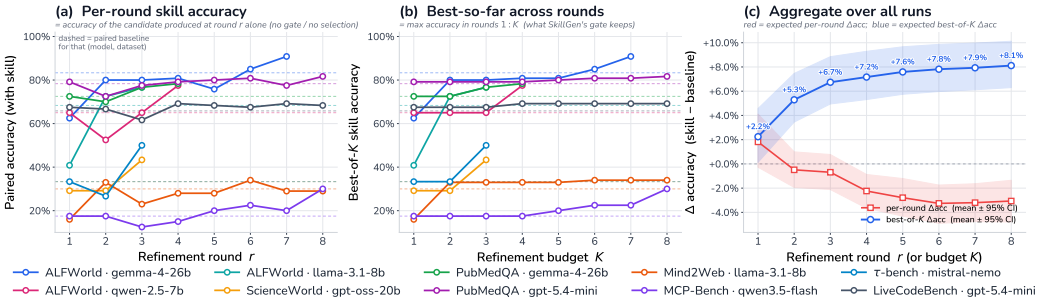

Figure 7: **Refinement rounds vs. skill accuracy.** Each refinement round produces one candidate skill evaluated on the construction-time verification subset. **(a)** Per-round candidate accuracy for representative runs, with dashed no-skill baselines. **(b)** Best-so-far accuracy under a budget of K rounds. **(c)** Aggregate mean Δ accuracy over all runs with 95% bootstrap confidence intervals.

RQ5 In which additional task regimes does SKILLGEN provide useful gains?

Figure 5 evaluates long-horizon retail tool use on τ-`Bench`. SKILLGEN improves every model whose skill passes the verification gate, with an average gain of $+5.3$ pp; models whose candidates fail the verification gate are left unchanged rather than exposed to an unverified intervention.

Figure 6 evaluates script- and reference-augmented skills on `ChemLLMBench`. Here, yield prediction benefits, with an average improvement of $+16.1$ pp across six models, while property prediction is more knowledge-bound and improves only for a small subset of agents. Together, these results suggest that resource-bundle skills are especially useful when the task is procedurally learnable, rather than being primarily a matter of recalling domain facts.

RQ6 Why select the best verified refinement round instead of using the latest candidate?

Figure 7 provides an empirical justification for treating refinement as best-of-K search rather than using the latest candidate. Per-round candidates are noisy: by round 8, the latest candidate has expected $\Delta = -3.1$ pp, while the best verified candidate reaches $+8.1$ pp (i.e. a gap of ~ 11 pp). Thus, the verification gate is not just a safety check; it is also the selection mechanism that turns unstable refinement trajectories into a reliable final skill.

RQ7 What qualitative failure modes and insights emerge from the generated skills?

We also inspect manually logged benchmark summaries and per-model skill-analysis reports; a detailed qualitative analysis is in Appendix C.9. Our analysis supports three takeaways: (i) the verification gate removes many harmful candidate skills, although accepted skills can still overgeneralize on held-out instances; (ii) residual failures in interactive environments are usually incomplete procedure execution rather than single local action mistakes; and (iii) chemistry and coding failures often reflect grounding or global-structure limits that a reusable inference-time skill cannot always repair. These observations align with the design of SKILLGEN: summary diagnostics for contrastive learning identify recurring procedural gaps, while construction-time verification limits the deployment of harmful skill interventions.

Acknowledgments and Disclosure of Funding

This paper is supported by the DAAD program "Konrad Zuse Schools of Excellence in Artificial Intelligence", sponsored by the Federal Ministry of Education and Research.

References

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

Anthropic. Agent skills overview, 2025. URL https://agentskills.io/home. Accessed: 2026.

Han Bao, Yue Huang, Yanbo Wang, Jiayi Ye, Xiangqi Wang, Xiuying Chen, Yue Zhao, Tianyi Zhou, Mohamed Elhoseiny, and Xiangliang Zhang. Autobench-v: Can large vision-language models benchmark themselves? *arXiv preprint arXiv:2410.21259*, 2024.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Yue Huang, Hang Hua, Yujun Zhou, Pengcheng Jing, Manish Nagireddy, Inkit Padhi, Greta Dolcetti, Zhangchen Xu, Subhajit Chaudhury, Ambrish Rawat, Liubov Nedoshivina, Pin-Yu Chen, Prasanna Sattigeri, and Xiangliang Zhang. Building a foundational guardrail for general agentic systems via synthetic data. *arXiv preprint arXiv:2510.09781*, 2025a.

Yue Huang, Zhengzhe Jiang, Xiaonan Luo, Kehan Guo, Haomin Zhuang, Yujun Zhou, Zhengqing Yuan, Xiaoqi Sun, Jules Schleinitz, Yanbo Wang, Shuhao Zhang, Mihir Surve, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. ChemOrch: Empowering LLMs with chemical intelligence via synthetic instructions. *arXiv preprint arXiv:2509.16543*, 2025b.

Yue Huang, Siyuan Wu, Chujie Gao, Dongping Chen, Qihui Zhang, Yao Wan, Tianyi Zhou, Jianfeng Gao, Chaowei Xiao, Lichao Sun, et al. DataGen: Unified synthetic dataset generation via large language models. In *International Conference on Learning Representations*, 2025c.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic skills – beyond tool use in LLM agents. *arXiv preprint arXiv:2602.20867*, 2026.

Simran Kaur, Simon Park, Anirudh Goyal, and Sanjeev Arora. Instruct-SkillMix: A powerful pipeline for LLM instruction tuning. In *International Conference on Learning Representations*, 2025.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, Yuxian Gu, Xin Cheng, Xun Wang, Si-Qing Chen, Li Dong, Wei Lu, Zhifang Sui, Benyou Wang, Wai Lam, and Furu Wei. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*, 2024.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. AgentInstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4. *arXiv preprint arXiv:2306.02707*, 2023.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, 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, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.

Donald B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.

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.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford CRFM Blog, 2023. URL https://crfm.stanford.edu/2023/03/13/alpaca.html.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a.

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*, 2023a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023b.

Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, Chetan Bansal, Dongmei Zhang, Saravan Rajmohan, Jianfeng Gao, and Huaxiu Yao. WebXSkill: Skill learning for autonomous web agents. *arXiv preprint arXiv:2604.13318*, 2026b.

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.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, 2024.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned LLMs with nothing. In *International Conference on Learning Representations*, 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*, 2023.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

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

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*, pages 19632–19642, 2024.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: Less is more for alignment. In *Advances in Neural Information Processing Systems*, 2023.

A Algorithm

Algorithm 1 summarizes the full SKILLGEN construction procedure.

Algorithm 1 SKILLGEN: contrastive induction with generation-verification-refinement loop

Require: induction subset $\mathcal{D}_{\text{ind}}$, construction-time verification subset $\mathcal{D}_{\text{ver}}$, base agent $\mathcal{A}$, evaluator $\mathcal{E}$, round budget $K \geq 1$, gate parameters $g_{\text{abs}} \in \mathbb{Z}_{\geq 0}$ and $g_{\text{rel}} \in [0, 1]$

Ensure: skill $s^\star$ marked active or deprecated

Stage ① Baseline elicitation

Collect induction trajectories $\mathcal{B} = \{(x_i, \tau_i^0, y_i^0)\}_{i=1}^n$ by rolling out $\mathcal{A}$ on $\mathcal{D}_{\text{ind}}$

Cache verification baselines $\mathcal{B}_{\text{ver}} = \{(\tilde{x}_j, \tilde{\tau}_j^0, b_j)\}_{j=1}^m$ by rolling out $\mathcal{A}$ on $\mathcal{D}_{\text{ver}}$

Stage ② Contrastive behavioral induction

Induction agent analyzes trajectories into $\mathcal{Z} = (a_0, \mathcal{F}, \mathcal{S}, \mathcal{C})$

Stage ③ Generation–verification–refinement

Initialize $\Phi^{(0)} \leftarrow \varnothing,\ g^\star \leftarrow -\infty,\ s^\star \leftarrow \varnothing$

Set gate threshold $\gamma_m \leftarrow \max\{g_{\text{abs}}, \lceil g_{\text{rel}} m \rceil, 1\}$

for $r = 1, \ldots, K$ **do**

  Generation agent computes $s^{(r)} \leftarrow \begin{cases} \text{Gen}(\mathcal{Z}), & \text{for } r = 1, \\ \text{Refine}(s^{(r-1)}, \mathcal{Z}, \Phi^{(r-1)}), & \text{for } r > 1 \end{cases}$

  Verification agent evaluates $s^{(r)}$ on all $\tilde{x}_j \in \mathcal{D}_{\text{ver}}$ and computes $G_m(s^{(r)})$

  Verification agent builds feedback $\Phi^{(r)}$ from repairs, regressions, and unresolved failures

  if $G_m(s^{(r)}) > g^\star$ **then**

    $g^\star \leftarrow G_m(s^{(r)}),\ s^\star \leftarrow s^{(r)}$

  end if

end for

Mark $s^\star$ as active if $g^\star \geq \gamma_m$, else mark it as deprecated

return $s^\star$

B Related Work

Agent skills. Early LLM agents augmented models with external tool use (Schick et al., 2023; Qin et al., 2024) or tool creation (Qian et al., 2023), while interaction-based agents such as ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and Voyager (Wang et al., 2023a) showed that trajectories can support reusable reasoning and action routines.

Agent skills provide a first-class abstraction that often follows the Anthropic Agent Skills standard (Zhang et al., 2025; Anthropic, 2025), which defines skills as composable bundles of instructions, scripts, and resources loaded dynamically at inference time. Recent surveys systematize skill architecture, acquisition, security, and deployment (Xu and Yan, 2026; Jiang et al., 2026). On the construction side, Trace2Skill (Ni et al., 2026), EvoSkill (Alzubi et al., 2026), EvoSkills (Zhang et al., 2026), and SkillX (Wang et al., 2026a) synthesize skills from agent experience, while SkillWeaver (Zheng et al., 2025), WebXSkill (Wang et al., 2026b), AutoSkill (Yang et al., 2026), and SkillRL (Xia et al., 2026) study web, dialogue, and RL deployment regimes. SKILLGEN differs by making failure analysis central: it clusters error patterns, identifies capability boundaries, and verifies induced skills through multi-agent collaboration with a construction-time deployment rule over paired repairs and regressions.

Synthetic data for LLM agents. Self-Instruct (Wang et al., 2023b) developed LLM-bootstrapped instruction generation, which inspiring later extension such as Alpaca (Taori et al., 2023), WizardLM (Xu et al., 2024), and further quality improvements through reasoning traces, curation, synthetic textbooks, and taxonomy-driven generation (Mukherjee et al., 2023; Zhou et al., 2023; Gunasekar et al., 2023; Li et al., 2024). Recent pipelines scale synthetic data with agentic flows or self-synthesis (Mitra et al., 2024; Xu et al., 2025), controllable generation and verification (Huang et al., 2025c), domain-specific tool-aware construction (Huang et al., 2025b), verified visual question answering (Bao et al., 2024) and risk-injected safety trajectories (Huang et al., 2025a).

For agent-specific capabilities, AgentTuning (Zeng et al., 2024), Agent-FLAN (Chen et al., 2024), FireAct (Chen et al., 2023), and Instruct-SkillMix (Kaur et al., 2025) construct trajectory or skill-composition data for instruction tuning. These methods produce data for *model fine-tuning*; in contrast, SKILLGEN synthesizes *inference-time skills*—structured bundles loaded without parameter updates—that target capability gaps exposed by systematic failure analysis.

C Experimental Details

C.1 Model Details

Table 2: Base agent models used across the reported experiments. Open-weight indicates that model weights are publicly available; proprietary models are accessed through hosted APIs.

Model	Provider	Open-weight
`Gemma-4-26B`	Google	✓
`Llama-3.1-8B`	Meta	✓
`Mistral-Nemo`	Mistral AI	✓
`Qwen-2.5-7B`	Alibaba Cloud	✓
`GPT-OSS-20B`	OpenAI	✓
`Qwen3.5-Flash`	Alibaba Cloud	×
`Claude-Haiku-4.5`	Anthropic	×
`GPT-5.4-Nano`	OpenAI	×
`GPT-5.4-Mini`	OpenAI	×
`Grok-4-Fast`	xAI	×
`Nemotron-120B`	NVIDIA	×

C.2 Datasets and Splits

All reported accuracy changes are based on the following comparative assessment: for each held-out test instance, we roll out the same base agent once without a skill and once with the generated skill, using the same instance identifier and random seed. Unless otherwise specified, we use seed 42 and keep the skill-training dataset and held-out test pool disjoint. Within the skill-training dataset, SKILLGEN uses an induction subset for trajectory analysis and a construction-time verification subset for refinement and selection, matching the usual train/validation/test separation. Table 3 summarizes the controlled split protocol for the benchmark-specific studies that require explicit sampling.

Table 3: Controlled split protocol for benchmark-specific studies.

Benchmark	Held-out test split	Constr.	Test	Notes
τ-Bench	Retail, task-disjoint	30	30	User simulator: `GPT-5.4-Nano`
`ALFWorld`	IOD (`valid_seen`)	500	150	In-distribution; task types overlap with train
`ALFWorld`	OOD (`valid_unseen`)	500	255	Out-of-distribution; novel task types
`ChemLLMBench`	Property / yield prediction	30	30	Subtasks scored independently
`LiveCodeBench`	`test_release_v6`	50	150	Sampled from release v6; seed 42
`MCPBench`	All tools	64	40	Multi-tool coverage
`MCPBench`	Single tool	40	16	Single-tool coverage
`Mind2Web`	Task-level disjoint	100	100	Web navigation; task-level split
`PubMedQA`	Sampled	150	100	Biomedical QA; seed 42
`ScienceWorld`	`dev`	150	100	Long-horizon multi-step tasks
`SocialMaze`	FTS	60	50	Fixed-target social navigation
`SocialMaze`	UPI	60	50	Unpredictable persona interaction

C.3 Model Routing and Auxiliary Roles

Each baseline or skill-augmented rollout is executed by the base agent model listed in Table 2; the auxiliary models used inside SKILLGEN never replace the base agent during rollout. To isolate the effect of the generated skill from the capability of the skill-writing model, we use a fixed auxiliary model, `GPT-5.4-Mini`, for the induction agent, generation agent, and verification agent across all base agents. Non-OpenAI model calls are routed through OpenRouter. Embeddings for clustering and skill-card merging use `text-embedding-3-small`. Decoding is deterministic with temperature 0; the default output budget is 4,096 tokens, increased to 16,384 tokens for skill generation.

C.4 SKILLGEN Hyperparameters

Unless otherwise noted, all runs use the same benchmark-specific configuration template. The induction stage uses at most eight failure clusters and eight success clusters, with adaptive k-means clustering over $k \in [2, 8]$ and a target cluster size of 15. The contrastive module keeps up to 20 nearest failure–success pairs. The generation prompt receives up to six failure clusters, six success clusters, and eight contrastive observations; web search is disabled.

The main experiments use a maximum refinement budget of eight rounds. For candidate verification, the verification gate evaluates uniformly sampled construction-time verification instances from the skill training dataset, using a 70/30 induction/verification split with at least four verification instances when the pool is small. The deployment decision follows the rule in §3.3: a skill is accepted only if its construction-time verification net gain $G_m(s)$ satisfies $G_m(s) \geq \max\{2, \lceil 0.05m \rceil, 1\}$. We additionally run up to 30 baseline-success guard checks to expose regressions on already-solved instances. Skills that fail the gate are persisted with status `deprecated`; downstream evaluation treats them as empty interventions, so cells labeled "gate off" report zero change rather than an unverified skill. The pipeline uses four workers for independent runs, and the verification agent's feedback stage uses eight workers.

C.5 Token Cost Analysis

Table 4: Token cost of SKILLGEN. *Train* is the one-time construction budget; BASE and SKILL are average tokens per call.

Benchmark	Train (M tok)	BASE (tok/call)	SKILL (tok/call)
`ScienceWorld`	2.2	1,630	1,977
`PubMedQA`	2.7	1,173	2,429
`Mind2Web`	5.2	4,482	5,919
`MCPBench`	7.5	4,847	6,000
τ-`Bench`	10.2	5,813	6,358
Mean	**5.6**	**3,589**	**4,537**
Median	**5.2**	**4,482**	**5,919**

Table 4 separates one-time construction cost from per-call inference overhead. All values are computed per model and then averaged within each benchmark. The skill-construction pipeline, including baseline trajectory collection, induction, generation, refinement, and verification, is a one-time cost per model–benchmark pair, ranging from 2.2M tokens on `ScienceWorld` to 10.2M on τ-`Bench` (mean 5.6M). Using `GPT-5.4-Mini` standard API prices ($0.75/M input tokens and $4.50/M output tokens),[4] and the prompt/output mix observed in our training logs, the mean construction budget corresponds to approximately $8.2 per generated skill. This cost is paid once for a model–benchmark pair, after which the same skill can be reused across subsequent rollouts and repeated evaluations. The per-call columns show that retrieval keeps inference prompts in the same few-thousand-token regime: the median skill-augmented call is 5,919 tokens, and the largest absolute per-call average in the table is 6,358 tokens on τ-`Bench`.

Compute resources. All experiments are orchestrated locally but executed through hosted LLM APIs routed through OpenRouter. The base-agent and auxiliary-model routing is reported in Appendix C.3; token budgets are reported in Table 4; and concurrency settings are reported in Appendix C.4. Because model inference is served by third-party API providers, the underlying accelerator type, memory configuration, and provider-side scheduling are not exposed to us. We therefore report reproducible API-level compute usage in terms of model calls, token budgets, and worker concurrency. Local compute is used only for orchestration, logging, clustering, and evaluation bookkeeping.

[4] `https://openai.com/api/pricing`, accessed April 2026.

C.6 Skill-Generation Baselines

Trace2Skill. For Trace2Skill (Ni et al., 2026), we run one no-skill rollout over the training pool. Success-branch and error-branch analysts process trajectories in parallel, and their proposed patches are consolidated through hierarchical LLM merging. We preserve the original prompt structure and do not impose an additional output schema beyond the shared Markdown skill wrapper.

SkillX. For SkillX (Wang et al., 2026a), we run two refinement rounds. Each round rolls out the current library, extracts skill cards from successful trajectories, clusters cards by cosine similarity at threshold 0.80 using `text-embedding-3-small`, merges clusters with an LLM, and filters cards with an LLM quality score threshold of 3/5. The retained library is capped at 12 cards before being canonicalized into the final skill.

EvoSkill. For EvoSkill (Alzubi et al., 2026), we maintain a frontier of $k = 3$ candidate programs for four iterations. A proposer chooses among `add_new`, `edit`, and `keep` operations based on failures from a fixed validation subset, and a builder emits the next candidate library. Admission requires the new candidate to outperform the weakest frontier member on the same validation subset.

CoEvoSkills. We instantiate the co-evolutionary baseline from EvoSkills (Zhang et al., 2026) using an information-isolated surrogate verifier. The surrogate writes binary natural-language assertions, judges rollouts against those assertions, and returns structured diagnostics to the skill generator. To match the rollout budget of the other baselines, we use three outer iterations and two inner verifier iterations. Because the shared evaluation interface consumes Markdown-formatted skills rather than executable multi-file bundles, the surrogate assertions are LLM-judged rather than executed as code.

Scope of the baseline comparison. Our baseline comparison evaluates all methods under the same deployment problem studied in this paper: synthesize one fixed, auditable inference-time skill for each benchmark–model pair, and evaluate that fixed intervention on held-out instances. This requires adapting methods that natively construct or retrieve from multi-skill libraries, such as SkillX and EvoSkill, into the same single-skill interface used by SKILLGEN. We emphasize that this controlled adaptation is not intended to measure the full native deployment potential of those systems under all possible library sizes, retrieval policies, or routing strategies. Instead, it supports a like-for-like comparison of skill synthesis quality when the deployed artifact must be one fixed skill.

This design also reduces evaluation instability and selection bias. If library-based methods were allowed to select different skills at test time, held-out performance would conflate skill construction with additional design choices such as library size, retrieval scoring, context-budget allocation, routing policy, and stochastic skill selection. Those choices are important in their own right, but they introduce extra degrees of freedom that are not shared by all methods and can make a comparative evaluation less stable (and potentially unfair). The resulting comparison should therefore be interpreted as a controlled single-skill adaptation of each baseline, rather than as a claim that the adapted baselines exhaust their native multi-skill capabilities. All reproduced baselines are adapted to this shared single-skill evaluation interface: each method emits one Markdown-formatted skill per benchmark–model pair, which is then injected and evaluated by the shared paired rollout harness. The base agent model is used for task rollouts, while `GPT-5.4-Mini` is used for auxiliary extraction, merging, proposal, and judging steps.

No-tool comparison protocol. For the skill-generation baseline comparison, we use only benchmark settings in which the evaluated skill does not rely on external tools, generated helper scripts, or reference-resource loading. To ensure a like-for-like comparison, SKILLGEN's optional script and reference components are disabled in these runs: every method, including SKILLGEN, emits a single Markdown-formatted natural-language skill, i.e., $s = (u, a, \varnothing, \varnothing)$. No method is allowed to provide executable helper functions, generated tools, retrieval documents, or calls to `skill_load_reference`. All skills are injected through the same prompt slot and evaluated with the same paired rollout harness. Thus, the comparison isolates the quality of the synthesized natural-language skill rather than differences in tool availability.

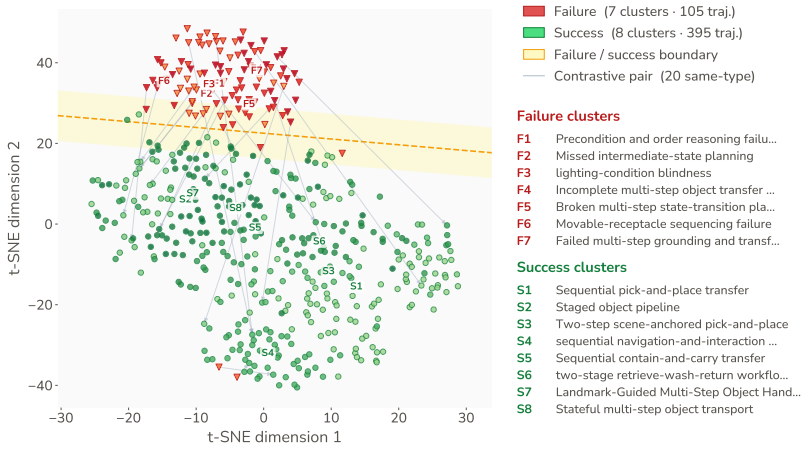

Figure 8: t-SNE visualization of SkillGen's induction on ALFWorld (gpt-5.4-nano). Red triangles (F1–F7) are failure trajectories; green circles (S1–S8) are successes. Gray arrows link each failure to its nearest same-type success (20 contrastive pairs). The yellow band marks the decision boundary between the two populations. Failures cluster compactly in the upper region (recurring planning errors), while successes spread broadly (diverse solving strategies), motivating the contrastive analysis that drives skill generation.

C.7 Evaluation Metrics and Gate-Off Handling

For every evaluated benchmark–split–model cell, we report baseline accuracy, skill-augmented accuracy, and the paired difference $\Delta = \mathrm{acc}_{\mathrm{skill}} - \mathrm{acc}_{\mathrm{base}}$ in percentage points. We also record repair counts (baseline wrong, skill correct), regression counts (baseline correct, skill wrong), and net gain as repair minus regression. When SkillGen marks a skill as `deprecated` during construction-time verification, evaluation reuses the no-skill baseline for the skill side. This convention makes rejected skills explicit and prevents an unverified prompt from introducing hidden regressions.

C.8 t-SNE Visualizations

Fig. 8 shows the t-SNE visualization of the contrastive induction of SkillGen on ALFWorld (gpt-5.4-nano).

C.9 Failure Analysis

We complement the aggregate gains in Section 4 with a qualitative inspection of the archived benchmark summaries and per-model skill-analysis reports. Rather than listing isolated examples, we distill four recurring findings about when skill augmentation still fails.

The verification gate substantially reduces harm, but accepted skills can still regress on held-out instances. The clearest evidence is the contrast between rejected and accepted negative skills. In LiveCodeBench, rejected skills would have reduced `Qwen-2.5-7B` from 25.33% to 16.00% and `Gemma-4-26B` from 83.33% to 80.67%; these cells are therefore reported as zero-delta after gating. Similar rejected regressions appear in Mind2Web and PubMedQA. However, filtering is not perfect. On ALFWorld OOD, an accepted skill for `Llama-3.1-8B` still lowers accuracy from 67.45% to 65.10%, with 51 repairs but 57 regressions, and on ChemLLMBench yield prediction `Mistral-Nemo` drops from 43.33% to 20.00%, with only three repairs against ten regressions. The failure mode here is overgeneralization: a skill that looks beneficial on the verification subset can still perturb correct baseline behavior on the final split.

In interactive environments, the dominant residual error is incomplete procedure execution rather than local action selection. Both ALFWorld and ScienceWorld exhibit this pattern. For `Llama-3.1-8B` on ALFWorld, the training-time analysis records all failures, and the largest cluster, with 65 cases, is labeled *incomplete dependency planning*. These trajectories often begin the first

plausible subgoal but omit a later prerequisite, such as turning on a lamp before inspection or using an intermediate receptacle before transport. ScienceWorld shows the same phenomenon at the level of experimental procedure. For `Gemma-4-26B`, the analysis contains 110 failures out of 150 construction instances, with large clusters labeled *ungrounded action planning* (25), *incomplete task-sequence planning* (20), and *incomplete goal-to-action planning* (19). In both benchmarks, the agent usually recognizes the task theme, but fails to maintain the full ordered procedure needed to finish it.

On chemistry tasks, failures are driven less by missing facts than by incorrect grounding of reaction roles and decision criteria. For `Qwen-2.5-7B` on ChemLLMBench yield prediction, the training analysis reports 27 failures out of 30 examples, with two dominant clusters: *superficial reaction-feasibility assessment* (14) and *reaction-role misparsing in cross-coupling* (13). A typical failure is that the model recognizes a familiar reaction family, but confuses substrates with catalysts, bases, solvents, or additives, and then predicts yield from a generic template rather than checking the actual electrophile, nucleophile, ligand/base combination, and substrate scope. This helps explain why resource-bundle skills are especially effective on yield prediction: the gain comes from enforcing a grounded checking procedure, not from merely restating chemical knowledge.

On code-generation tasks, skill augmentation cannot fully compensate for missing global problem structure. LiveCodeBench failures for `Qwen-2.5-7B` are dominated by upstream reasoning errors rather than surface-level implementation mistakes. The training analysis reports 113 failures out of 150 problems, with major clusters labeled *incomplete algorithmic modeling* (19), *incomplete algorithm realization* (14), and *structure-mapping failure* (6). Typical trajectories either pursue brute-force search where an invariant or transformation is required, apply a local greedy heuristic where dynamic programming is needed, or emit truncated code after only partially deriving the solution. This suggests that a general skill can regularize recurring reasoning habits, but it cannot reliably rescue cases where the model never identifies the right combinatorial structure in the first place.

D Broader Impacts

SKILLGEN aims to make LLM agents more reliable and easier to adapt without retraining, which could reduce the manual effort required to build task-specific agent skills and make procedural knowledge more inspectable through human-readable skill artifacts. This may benefit scientific, coding, web, and tool-use workflows where reusable guidance and verification can improve consistency. At the same time, any method that improves agent performance can also improve agents used for harmful or unintended purposes, and generated skills may overgeneralize, introduce regressions on unseen cases, or amplify mistakes in domains where incorrect tool use has real consequences. The risks are especially relevant when skills are deployed in open-ended environments, safety-sensitive applications, or workflows involving external tools and resources. SKILLGEN's paired verification provides a partial check against these harms by making regressions visible during construction. However, these checks are limited to the evaluated task distribution, so they should be paired with application-specific safety evaluation, human review of generated skills, access controls, and ongoing monitoring before deployment.