QWEN
BUSINESS UNIT

Skill Self-Play: Pushing the Frontier of LLM Capability with Co-Evolving Skills

Siyuan Huang[†1,2], **Pengyu Cheng**[§1], **Haotian Liu**[†1,3], **Tao Chen**[†1,4], **Yihao Liu**[†1,5], **Jingwei Ni**[†1,6,7], **Shijie Zhou**[†1,8], **Ziyi Yang**[1], **Gangwei Jiang**[1], **Mengyu Zhou**[1], **Yu Cheng**[§2], **Xiaoxi Jiang**[1] **and Guanjun Jiang**[1]

[1]Qwen Large Model Application Team, Alibaba, [2]The Chinese University of Hong Kong, [3]Renmin University of China, [4]Sun Yat-sen University, [5]Peking University, [6]ETH Zürich, [7]University of Zurich, [8]University at Buffalo

[†]Work done during an internship at Alibaba. [§]Corresponding author.

LLM training is shifting from manual design and annotation to interaction-driven self-evolution. However, existing self-evolutionary methods face a fundamental dilemma between task diversity and verification reliability: environment-bound methods obtain precise feedback but confine learning to narrow domains, while open-ended self-generation broadens the task space but lacks reliable verification, allowing misleading rewards to pollute the training loop. We identify agent skills as a powerful middle ground to reconcile this tension: each skill ensures deep, verifiable execution in a specific scenario, while dynamic routing across skills maintains open-ended task variety. Leveraging this insight, we introduce Skill Self-Play (Skill-SP), a co-evolutionary framework comprising a proposer, a solver, and a dynamic skill controller. Orchestrated via a reinforcement learning loop, these components co-evolve in a continuous self-play loop: the proposer generates challenging tasks conditioned on dynamically sampled skills; the solver explores candidate solutions to push its capability boundaries; and the skill controller collects execution feedback to update and expand the skill library. This interactive co-evolution effectively bridges the gap between structured verification and open-ended exploration. Empirical evaluations on tool-use and reasoning benchmarks demonstrate that Skill-SP, serving as a robust evolution engine, consistently pushes the performance ceiling of competent backbones while catalyzing striking turnarounds for initially misaligned models. Our code is available at https://github.com/Qwen-Applications/skill-self-play.

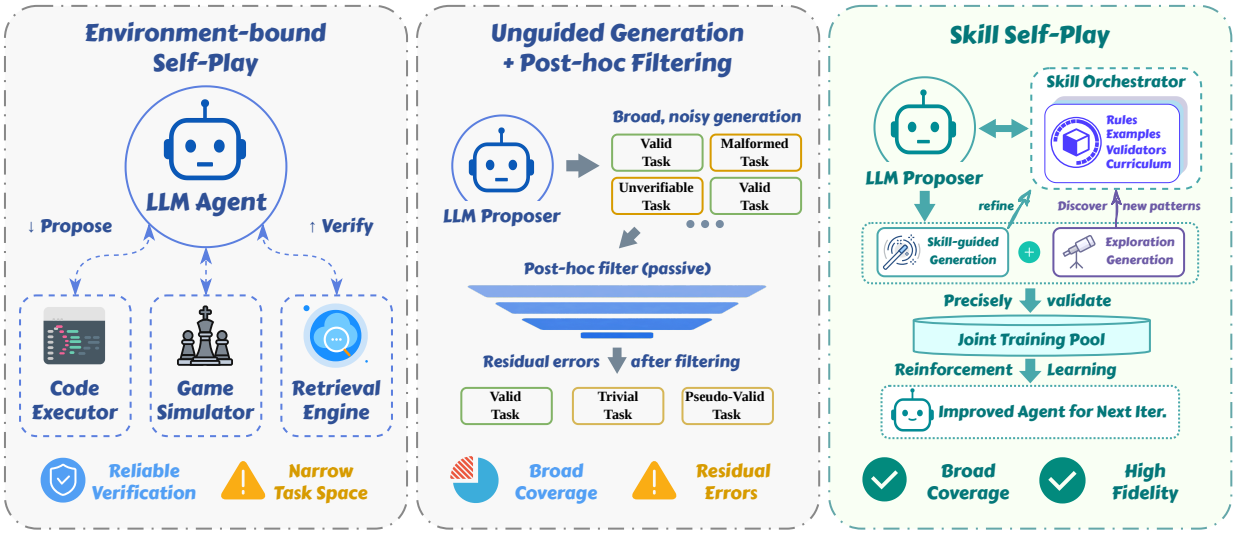


Figure 1 | Motivation of **Skill Self-play**. **Left:** *Environment-bound methods* provide reliable verification but severely restrict the task space. **Middle:** *Unguided generation* broadens coverage but depends on passive post-hoc filtering, allowing residual errors to degrade data fidelity. **Right:** *Skill Self-Play* uses a proactive Skill Orchestrator with dual streams for skill-guided generation and open-ended exploration, enabling iterative skill evolution with both broad coverage and high fidelity.

1. Introduction

Self-evolution methods, which empower Large Language Models (LLMs) to autonomously generate tasks, find solutions, and evaluate outcomes, have shifted the post-training paradigm from heavy data-driven annotation to self-improvement without any supervision (Tao et al., 2024; Patel et al., 2024; Jiang et al., 2024a; Chen et al., 2025; Xu et al., 2025a). Along this new scaling dimension, self-play has emerged as a foundational technique, in which the learning LLM engages in gameplay as different role-playing agents and updates its policies via multi-agent reinforcement learning (MARL) (Yang et al., 2025b; Yuan et al., 2025; Liu et al., 2025a; Jiang et al., 2026a). Recent works have demonstrated the surprising power of self-play across various scenarios, significantly boosting LLM capabilities in domains such as logical reasoning (Cheng et al., 2024b; Huang et al., 2025a), deep search (Lu et al., 2025; Liu et al., 2025b), visual question answering (Wang et al., 2025), and software engineering (Wei et al., 2025; Zhao et al., 2026).

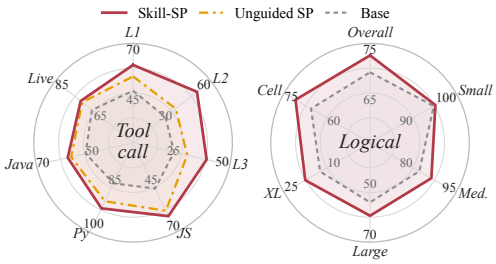


Figure 2 | **Performance footprint.** Skill-SP broadly expands Qwen3-4B-Ins capabilities across tool-calling and logical reasoning. Notably, Unguided SP is absent from the latter as it fails to synthesize valid puzzles.

However, the success of self-play in each specific domain depends heavily on well-crafted environments and rewards to provide precise validation (Yang et al., 2026a; Song et al., 2026; Yan et al., 2026; Huang et al., 2026b; Wang et al., 2026a). While external verification ensures accuracy, it confines agent training to narrow operational boundaries, leaving the model unable to generalize to broader, open-ended tasks (Tang et al., 2025; Zhao et al., 2025). To broaden the task space, alternative approaches employ unguided task generation paired with post-hoc filtering (e.g., format checks or majority voting) (Wang et al., 2023; Xu et al., 2024; Nguyen et al., 2024; Xu et al., 2025b; An et al., 2025; Yu et al., 2025a). Unfortunately, open-ended task generation severely compromises quality control (Long et al., 2024; Jiang et al., 2025a). Without structural guidance, task generators frequently synthesize ill-posed tasks or overfit to simple templates (Yu et al., 2025b; Li et al., 2026a). Over successive training iterations, accumulated errors and biases cause synthetic data collapse (Shumailov et al., 2024; Kazdan et al., 2025). Ultimately, post-hoc filtering acts merely as a *passive sieve*: it rejects glaring formatting errors but cannot actively guide the generator to produce reusable, logically valid, and diverse task patterns. Consequently, a fundamental challenge remains: *How can self-play autonomously generate and verify diverse agent tasks without collapsing into either narrow domains or chaotic synthetic noise?*

To resolve this dilemma and bridge the gap between rigid environments and passive filtering (Khattab et al., 2023; Dong et al., 2025; Prabhakar et al., 2026; Gao et al., 2026a), we argue that verifiable self-play can be significantly enhanced by the concept of *skills*. As introduced by Anthropic (2025), a skill is a modular unit of procedural knowledge that bundles task-specific instructions and supporting resources into a reusable block. By organizing expertise into these self-contained units, skills allow agents to dynamically load specialized knowledge exactly when needed. When formalized as a *task-pattern interface*, this proactive abstraction serves as the ideal medium of evolution. While raw execution rollouts in self-play contain valuable insights, this evolutionary knowledge is typically scattered across isolated trajectories. Simply aggregating these experiences by stuffing historical trajectories into a single task-generator prompt leads to severe context bloat and dilutes structural control. In contrast, a skill interface distills this scattered experience into a compact, reusable, and co-evolvable unit. Rather than acting as a passive post-hoc filter, this interface actively orchestrates the entire self-play loop: injecting structural priors to guide the task generator *before* synthesis, providing explicit executable mechanisms for rigorous verification *after* response generation, and tracking historical difficulty to govern curriculum progression *across* successive iterations.

To realize this paradigm, we introduce ***Skill Self-Play*** (**Skill-SP**), a reinforcement learning framework driven by an evolving library of modular skill packages (Figure 1). At its core, Skill-SP orchestrates self-play through three core components: a *Proposer*, a *Solver*, and a *Controller*. More specifically, the Proposer synthesizes targeted challenges guided by routed skill packages; the Solver learns to execute and solve these tasks; and the Controller analyzes execution trajectories to continually manage the skill library. Within this loop, the skill library undergoes a continuous evolutionary cycle: the Controller automatically refines existing skill packages based on execution feedback, archives obsolete ones, and induces entirely new skills from open-ended exploration. This dynamic routing of modular skills ensures strict structural quality control without context

bloat, while the discovery of new skill packages continuously expands the task frontier. By isolating governance in this evolving library, Skill-SP successfully resolves the fundamental tension between task diversity and verification reliability. We empirically validate Skill-SP on two families of verifiable agent tasks: tool calling (API-Bank (Li et al., 2023) and BFCL (Patil et al., 2025)) and logical reasoning (ZebraLogic (Lin et al., 2025)). Across various open-source LLM backbones, Skill-SP delivers substantial performance breakthroughs, yielding absolute gains of up to **+42.9** points on tool use and **+12.0** points on logical reasoning, consistently outperforming unguided self-play (as in Figure 2). Finally, diagnostic analyses confirm that Skill-SP successfully anchors task generation at the model's evolving learning frontier by continuously distilling novel patterns into reusable skills. The proposed skill-driven self-play brings a fundamental paradigm shift to model self-improvement, unlocking a sustainable and truly open-ended frontier for LLM self-evolution.

2. Related Work

Self-play with external verifiers. Self-play has long improved agents in closed games (Silver et al., 2017, 2018), inspiring recent language-model work on alignment, instruction following, and reasoning through self-generated data or verifiable rewards (Cheng et al., 2024b; Chen et al., 2024; Cheng et al., 2024a; Huang et al., 2025b; Kuba et al., 2025; Ren et al., 2026). Recent self-evolving systems often obtain reliable feedback by grounding task generation in external environments. For example, Absolute Zero (Zhao et al., 2026) and verified-code pipelines (Wilf et al., 2025) rely on code executors; SPIRAL and MARSHAL (Liu et al., 2025a; Yuan et al., 2025) use game simulators; and Search Self-play (Lu et al., 2025) leverages retrieval augmentation. While these works demonstrate the value of reliable verification, their specialized environments inherently bottleneck the task distribution, confining the agent's learning to narrow, predefined operational domains.

Synthetic task generation and filtering. To broaden the task space, general self-improvement pipelines commonly combine synthetic task generation with automated verification (Yang et al., 2026a). Representative mechanisms include unit tests for coding tasks (Jimenez et al., 2024; Gehring et al., 2024; Jiang et al., 2025b), success signals in interactive web environments (Liu et al., 2024; Zhou et al., 2024; Yao et al., 2024; Song et al., 2026), schema checks for tool use (Li et al., 2023; Patil et al., 2025), and deterministic constraint checkers for reasoning (Zhou et al., 2023; Jiang et al., 2024b; Qin et al., 2024; Wen et al., 2024; Pyatkin et al., 2025). Although these mechanisms make synthetic data auditable, they primarily operate as passive, *post-hoc* filters. They dictate which tasks survive but provide no proactive structural guidance to the proposer, causing unguided generation to frequently collapse into ill-posed tasks or redundant templates.

Skill interfaces for agents. Recent work studies agent skills as reusable procedural interfaces, with an emphasis on retrieval, compression, and progressive disclosure (Ling et al., 2026; Jiang et al., 2026b; Cho et al., 2026; Li et al., 2026c; Huang et al., 2026a; Gao et al., 2026b; Chen et al., 2026b,a; He et al., 2026). Another line of research automatically constructs or evolves skills from interaction traces and execution feedback (Mi et al., 2026; Liu et al., 2026b; Wang et al., 2026b; Shen et al., 2026; Yang et al., 2026b; Liu et al., 2026d; Gautam et al., 2026; Wang et al., 2026c; Liu et al., 2026c; Ni et al., 2026), although preserving the fidelity of automatically generated skills remains challenging (Li et al., 2026b; Zhong et al., 2026; Zhou et al., 2026). Most prior work uses skills to support inference-time execution, while a few recent studies explore their role during training (Lu et al., 2026). In contrast, Skill-SP uses an evolving skill library as a training-time interface for task synthesis, enabling structural guidance and verification to co-evolve with the self-play curriculum. The following section formalizes this framework.

3. Methodology

We propose ***Skill Self-Play*** (**Skill-SP**), a training-time framework that transforms generic self-play into a proactive curriculum-construction process (Figure 3). At each iteration, Skill-SP orchestrates the joint optimization of a proposer policy π_{propose} and a solver policy π_{solve}, alongside a skill controller managing an evolving library $\mathcal{S}$. Guided by routed skills, the proposer is updated to synthesize valid tasks at the solver's learning frontier, while the solver is optimized on the resulting verified curriculum. Concurrently, the controller distills execution feedback to update the library, driving a continuous co-evolutionary loop that yields progressively more informative challenges.

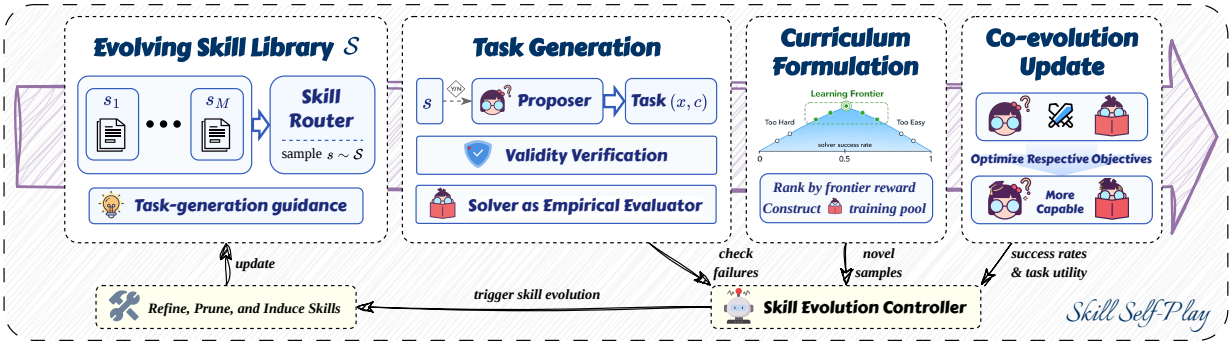


Figure 3 | **Overview of Skill Self-Play.** An evolving skill library routes task-generation guidance to the proposer, which generates candidate tasks that undergo validity verification. Valid candidates are ranked by frontier reward to construct the solver curriculum. The resulting signals drive co-evolutionary proposer and solver updates, while validation failures, novel samples, and task-level statistics trigger skill refinement, pruning, and induction, distilling training feedback into reusable skills for subsequent task generation.

3.1. Skill Self-Play Objective

We formalize a verifiable agent task as a tuple $(\boldsymbol{x}, \boldsymbol{c})$, where $\boldsymbol{x}$ is the standard prompt visible to the solver, and $\boldsymbol{c}$ is a hidden, machine-readable verification contract (e.g., unit tests, reference answers) evaluated exclusively by the environment. Given a solver's response $\boldsymbol{y}$, the environment returns a verification reward $\mathcal{R}_{\text{solve}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \in [0, 1]$.

A standard self-play loop trains the proposer to generate challenging tasks and the solver to resolve them. For a generated task instance $(\boldsymbol{x}, \boldsymbol{c})$, the solver's expected success rate is defined as:

$$v_{\text{solve}}(\boldsymbol{x}, \boldsymbol{c}; \pi_{\text{solve}}) = \mathbb{E}_{\boldsymbol{y} \sim \pi_{\text{solve}}(\cdot|\boldsymbol{x})} \left[\mathcal{R}_{\text{solve}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \right]. \tag{1}$$

To drive continuous improvement, a natural objective for the proposer is to target the solver's learning frontier via a medium-difficulty score, formulated as $1 - 2|v_{\text{solve}} - 0.5|$. However, optimizing the proposer solely toward this metric frequently invites *reward hacking*, where the proposer synthesizes ill-posed or unsolvable contracts to fabricate artificial difficulty. To prevent this, Skill-SP formalizes task generation via a *gated curriculum reward*. We introduce a binary task quality filter that retains only candidates with structurally sound contracts, thereby explicitly gating the proposer's reward:

$$\mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi_{\text{solve}}) = \mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \text{ is valid}\}} \cdot \left(1 - 2\left|v_{\text{solve}}(\boldsymbol{x}, \boldsymbol{c}; \pi_{\text{solve}}) - \frac{1}{2}\right|\right), \tag{2}$$

where the indicator $\mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \text{ is valid}\}}$ represents this binary filter, which is further detailed in Eq. 4 of Section 3.2.

To proactively guide task generation, Skill-SP introduces an evolving skill library $\mathcal{S}$. Task synthesis is conditioned on a sampled skill $\boldsymbol{s} \sim \mathcal{S}$, which equips the proposer with reusable structural priors and programmatic validators. Consequently, Skill-SP can be formulated as a bi-level optimization problem. The outer objective jointly optimizes the skill library $\mathcal{S}$ and the proposer π_{propose} to synthesize valid, frontier-targeted tasks, subject to an inner objective where the solver maximizes its execution success on these tasks:

$$\begin{aligned} \max_{\pi_{\text{propose}}, \mathcal{S}} \quad & \mathbb{E}_{\boldsymbol{s} \sim \mathcal{S},\ (\boldsymbol{x},\boldsymbol{c}) \sim \pi_{\text{propose}}(\cdot|\boldsymbol{s})} \left[\mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi^*_{\text{solve}}) \right] \\ \text{s.t.} \quad & \pi^*_{\text{solve}} = \arg\max_{\pi} \mathbb{E}_{\boldsymbol{s} \sim \mathcal{S},\ (\boldsymbol{x},\boldsymbol{c}) \sim \pi_{\text{propose}}(\cdot|\boldsymbol{s}),\ \boldsymbol{y} \sim \pi(\cdot|\boldsymbol{x})} \left[\mathcal{R}_{\text{solve}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \right]. \end{aligned} \tag{3}$$

To orchestrate this high-fidelity curriculum loop, the subsequent sections detail the policy optimization of the proposer and the solver, alongside the dynamic evolution of the skill library $\mathcal{S}$.

3.2. Proposer Optimization via Skill-Conditioned Generation

To operationalize the outer objective (Eq. 3), the proposer must continuously synthesize valid, frontier-targeted tasks. We structure this process into three stages: skill-conditioned generation to inject structural priors, rigorous verification to ensure task validity, and policy optimization to drive curriculum evolution.

Skill-Conditioned Generation. A skill $s \in \mathcal{S}$ serves as a structural interface defined by the tuple $s = \langle m, r, h, e, \nu, \sigma \rangle$. Here, m denotes routing metadata; r, h, and e provide procedural rules, generation hints, and few-shot examples to contextually condition the proposer; ν specifies executable validators; and σ tracks historical usage statistics. During generation, we dynamically sample a skill $s \sim \mathcal{S}$ based on σ, balancing the exploitation of high-yield skills with an exploration bonus for under-tested ones (detailed in Appendix B). The proposer then synthesizes candidates via a *skill stream* $(x, c) \sim \pi_{\text{propose}}(\cdot \mid s)$, which injects strong structural priors. To continuously chart novel task spaces and prevent mode collapse, this is complemented by an open-ended *exploration stream* $(x, c) \sim \pi_{\text{propose}}(\cdot \mid \varnothing)$ without skill constraints.

Validity Verification. The binary task quality filter $\mathbb{1}_{\{(x,c) \text{ is valid}\}}$ gating the proposer reward (Eq. 2) enforces rigorous structural and logical checks. For tasks generated via the skill stream, this valid event is defined as the intersection of three conditions:

$$\{(x, c) \text{ is valid}\} = \{\text{schema compliant}\} \cap \{\text{contract valid}\} \cap \{\text{probe consistent}\}. \tag{4}$$

Here, the first condition ensures global schema compliance; the second applies validators defined in s to filter ill-posed contracts; and the third verifies *probe consistency*. Specifically, this consistency requires that K rollouts drawn from the current solver $\pi_{\text{solve}}(\cdot \mid x)$ yield a unique majority answer matching the proposer-defined reference in c. For the exploration stream, the valid event omits this skill-specific validation.

Policy Update. At each training iteration t, we fix the current solver $\pi^{(t)}_{\text{solve}}$ to serve as an empirical evaluator. For each generated candidate, we compute the gated reward $\mathcal{R}_{\text{propose}}$ by combining its validity mask $\mathbb{1}_{\{(x,c) \text{ is valid}\}}$ with the empirical success rate $v_{\text{solve}}(x, c; \pi^{(t)}_{\text{solve}})$, efficiently estimated by reusing the K verification rollouts. The proposer policy $\pi^{(t)}_{\text{propose}}$ is then updated via Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to maximize the expected reward:

$$\mathcal{J}_{\text{propose}}(\pi) = \mathbb{E}_{s \sim \mathcal{S},\, (x,c) \sim \pi(\cdot \mid s)} \left[\mathcal{R}_{\text{propose}}(x, c; \pi^{(t)}_{\text{solve}}) \right]. \tag{5}$$

Optimizing this objective yields the next-iteration proposer $\pi^{(t+1)}_{\text{propose}}$, ensuring that as the solver evolves, the proposer dynamically tracks its shifting learning frontier to continuously synthesize challenging, reliable tasks.

3.3. Solver Optimization via Dynamic Curriculum Construction

To operationalize the inner objective (Eq. 3), the solver is optimized on a dynamic curriculum that systematically advances its capabilities. We construct this high-quality training pool by selecting the most challenging, frontier-targeted tasks from both generation streams.

Curriculum Construction. Let $\mathcal{T}^{(t)}_{\text{skill}}$ and $\mathcal{T}^{(t)}_{\text{explore}}$ denote the sets of strictly valid candidates (x, c) synthesized by the skill and exploration streams, respectively. To form the final training pool $\mathcal{D}^{(t)}$ of size M, we rank these candidates by their proposer reward $\mathcal{R}_{\text{propose}}(x, c; \pi^{(t)}_{\text{solve}})$, effectively pinpointing the solver's learning frontier, and select the top-scoring tasks according to a blending ratio $\alpha \in (0, 1)$:

$$\mathcal{D}^{(t)} = \text{Top}_{\alpha M}\left(\mathcal{T}^{(t)}_{\text{skill}}; \mathcal{R}_{\text{propose}}\right) \cup \text{Top}_{(1-\alpha)M}\left(\mathcal{T}^{(t)}_{\text{explore}}; \mathcal{R}_{\text{propose}}\right). \tag{6}$$

Policy Update. Using the constructed curriculum $\mathcal{D}^{(t)}$, the solver policy $\pi^{(t)}_{\text{solve}}$ is updated via GRPO to maximize the environment verification reward:

$$\mathcal{J}_{\text{solve}}(\pi) = \mathbb{E}_{(x,c) \sim \mathcal{D}^{(t)},\, y \sim \pi(\cdot \mid x)} \left[\mathcal{R}_{\text{solve}}(x, y, c) \right]. \tag{7}$$

Optimizing this objective yields the next-iteration solver $\pi^{(t+1)}_{\text{solve}}$.

3.4. Continuous Evolution of the Skill Library

To prevent curriculum stagnation, the skill library $\mathcal{S}^{(t)}$ dynamically co-evolves with the proposer and solver policies at each iteration t. This evolution comprises three operations: refining existing skills, pruning obsolete ones, and inducing novel structural priors from open-ended exploration.

Algorithm 1: Skill Self-Play (Skill-SP)

Input: Initial proposer $\pi^{(0)}_{\text{propose}}$, solver $\pi^{(0)}_{\text{solve}}$, skill library $\mathcal{S}^{(0)}$, and number of self-play iterations T.
Output: Trained solver $\pi^{(T)}_{\text{solve}}$.

foreach *self-play iteration* t **do**
  Sample $\boldsymbol{s} \sim \mathcal{S}^{(t)}$ using skill statistics; generate candidates with $\pi^{(t)}_{\text{propose}}(\cdot \mid \boldsymbol{s})$ and $\pi^{(t)}_{\text{propose}}(\cdot \mid \varnothing)$;
  Verify candidates and compute $\mathcal{R}_{\text{propose}}$; retain valid candidates for the solver curriculum;
  Rank valid candidates by $\mathcal{R}_{\text{propose}}$ to build the mixed frontier curriculum $\mathcal{D}^{(t)}$;
  Update $\pi^{(t)}_{\text{propose}} \to \pi^{(t+1)}_{\text{propose}}$ using $\mathcal{J}_{\text{propose}}$, and $\pi^{(t)}_{\text{solve}} \to \pi^{(t+1)}_{\text{solve}}$ on $\mathcal{D}^{(t)}$ using $\mathcal{J}_{\text{solve}}$;
  Evolve the skill library by updating skill statistics, refining existing skills from failure traces, inducing new skills $\mathcal{S}^{(t)}_{\text{new}}$ from induction candidates $\mathcal{T}^{(t)}_{\text{induce}}$, and pruning obsolete skills $\mathcal{S}^{(t)}_{\text{prune}}$: $\mathcal{S}^{(t+1)} \leftarrow (\mathcal{S}^{(t)} \setminus \mathcal{S}^{(t)}_{\text{prune}}) \cup \mathcal{S}^{(t)}_{\text{new}}$;
end
return $\pi^{(T)}_{solve}$;

Skill Refinement. Using generation feedback, Skill-SP updates the tracking statistics σ of each sampled skill to modulate future routing (Appendix B). Concurrently, it analyzes execution trajectories from invalid generation attempts to diagnose systematic failures and automatically refine the core content of the skills, maintaining the library as a robust repository of structural priors.

Skill Pruning. To maximize sample efficiency, the active library undergoes periodic pruning. By monitoring σ, Skill-SP identifies a pruning set $\mathcal{S}^{(t)}_{\text{prune}}$ of saturated skills that consistently yield trivial tasks, indicated by an expected frontier reward falling below a threshold γ_{prune}:

$$\mathcal{S}^{(t)}_{\text{prune}} = \left\{ \boldsymbol{s} \in \mathcal{S}^{(t)} \;\middle|\; \mathbb{E}_{(\boldsymbol{x},\boldsymbol{c})\sim\pi^{(t)}_{\text{propose}}(\cdot|\boldsymbol{s})} \left[\mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}}) \right] < \gamma_{\text{prune}} \right\}. \tag{8}$$

Archiving $\mathcal{S}^{(t)}_{\text{prune}}$ conserves the generation budget and keeps the proposer focused on the learning frontier.

Skill Induction and Update. To systematically expand the curriculum, Skill-SP extracts an induction set $\mathcal{T}^{(t)}_{\text{induce}}$ of novel candidates from the exploration stream, conditioned on a minimum frontier reward threshold γ_{induce}:

$$\mathcal{T}^{(t)}_{\text{induce}} = \left\{ (\boldsymbol{x}, \boldsymbol{c}) \in \mathcal{T}^{(t)}_{\text{explore}} \;\middle|\; \mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}}) \geq \gamma_{\text{induce}} \right\}. \tag{9}$$

Skill-SP then leverages the skill controller π_{control}, instantiated by the initial base policy, to abstract the underlying task patterns within $\mathcal{T}^{(t)}_{\text{induce}}$ into candidate packages. These are systematically filtered for structural integrity and semantic novelty against the current library to yield the newly induced skills $\mathcal{S}^{(t)}_{\text{new}}$:

$$\mathcal{S}^{(t)}_{\text{new}} = \left\{ \boldsymbol{s} \sim \pi_{\text{control}}(\cdot|\mathcal{T}^{(t)}_{\text{induce}}) \;\middle|\; \text{Integrity}(\boldsymbol{s}) \wedge \text{Novelty}(\boldsymbol{s}, \mathcal{S}^{(t)}) \right\}, \tag{10}$$

where $\text{Integrity}(\boldsymbol{s})$ checks package completeness, and $\text{Novelty}(\boldsymbol{s}, \mathcal{S}^{(t)})$ rejects candidates duplicating existing skills by thresholding lexical similarity. The active library is subsequently updated via $\mathcal{S}^{(t+1)} = (\mathcal{S}^{(t)} \setminus \mathcal{S}^{(t)}_{\text{prune}}) \cup \mathcal{S}^{(t)}_{\text{new}}$. This mechanism distills successful open-ended exploration into reusable structural priors, driving an ever-expanding curriculum. Algorithm 1 summarizes the resulting training loop.

4. Experiments

4.1. Experimental Setup

Task Families. We instantiate the framework on two domains. The first is *tool-call prediction*, evaluating the solver's schema adherence and tool selection. We benchmark on API-Bank Levels 1–3 (Li et al., 2023) and four BFCL categories (Patil et al., 2025): *bfcl_simple_javascript*, *bfcl_simple_python*, *bfcl_simple_java*, and *bfcl_live_simple*. The second is *logical reasoning*, evaluating the solver's ability to satisfy complex constraints and deduce unique solutions. We benchmark on ZebraLogic (Lin et al., 2025), which formalizes tasks as Zebra-style grid puzzles spanning four complexity scales determined by their search space size.

Method	API-Bank				BFCL					Avg.
	L1	L2	L3	Avg.	JS	Py	Java	Live	Avg.	
Qwen3-4B-Ins	58.1	42.5	35.6	51.4	57.8	91.8	59.8	75.6	71.2	60.2
+ Unguided SP	60.2$_{+2.1}$	46.3$_{+3.8}$	40.8$_{+5.2}$	54.4$_{+3.0}$	65.8$_{+8.0}$	95.0$_{+3.2}$	63.0$_{+3.2}$	77.9$_{+2.3}$	75.4$_{+4.2}$	64.1$_{+3.9}$
+ Skill-SP	**64.6**$_{+6.5}$	**54.7**$_{+12.2}$	**44.0**$_{+8.4}$	**58.9**$_{+7.5}$	**65.5**$_{+7.7}$	**96.0**$_{+4.2}$	**63.5**$_{+3.7}$	**78.5**$_{+2.9}$	**75.9**$_{+4.7}$	**66.7**$_{+6.5}$
Qwen3-8B	73.3	61.4	43.9	65.5	69.0	95.1	64.9	77.9	76.7	69.4
+ Unguided SP	74.3$_{+1.0}$	64.4$_{+3.0}$	48.3$_{+4.4}$	67.5$_{+2.0}$	71.8$_{+2.8}$	95.8$_{+0.7}$	64.9$_{+0.0}$	77.8$_{-0.1}$	77.5$_{+0.8}$	71.0$_{+1.6}$
+ Skill-SP	**75.9**$_{+2.6}$	**67.2**$_{+5.8}$	**50.0**$_{+6.1}$	**69.2**$_{+3.7}$	**72.8**$_{+3.8}$	**96.0**$_{+0.9}$	**65.1**$_{+0.2}$	**78.7**$_{+0.8}$	**78.2**$_{+1.5}$	**72.2**$_{+2.8}$
Ministral-3-8B	37.6	35.6	20.7	33.7	8.2	16.4	9.2	17.3	12.8	20.7
+ Unguided SP	38.5$_{+0.9}$	36.4$_{+0.8}$	20.8$_{+0.1}$	34.4$_{+0.7}$	7.5$_{-0.7}$	15.4$_{-1.0}$	8.5$_{-0.7}$	18.8$_{+1.5}$	12.5$_{-0.3}$	20.8$_{+0.1}$
+ Skill-SP	**68.5**$_{+30.9}$	**56.2**$_{+20.6}$	**42.9**$_{+22.2}$	**61.5**$_{+27.8}$	**48.5**$_{+40.3}$	**90.8**$_{+74.4}$	**58.1**$_{+48.9}$	**80.0**$_{+62.7}$	**69.3**$_{+56.5}$	**63.6**$_{+42.9}$
Ministral-3-14B	31.1	15.5	15.7	26.0	14.5	32.9	18.6	27.2	23.3	22.2
+ Unguided SP	67.2$_{+36.1}$	50.6$_{+35.1}$	46.3$_{+30.6}$	60.8$_{+34.8}$	36.8$_{+22.3}$	88.7$_{+55.8}$	46.1$_{+27.5}$	77.2$_{+50.0}$	62.2$_{+38.9}$	59.0$_{+36.8}$
+ Skill-SP	**70.9**$_{+39.8}$	**57.5**$_{+42.0}$	**48.9**$_{+33.2}$	**64.5**$_{+38.5}$	**42.0**$_{+27.5}$	**89.9**$_{+57.0}$	**59.0**$_{+40.4}$	**83.1**$_{+55.9}$	**68.5**$_{+45.2}$	**64.5**$_{+42.3}$
Granite-4.1-3B	57.5	53.7	43.2	53.9	66.5	66.5	57.4	55.9	61.6	57.2
+ Unguided SP	59.0$_{+1.5}$	54.3$_{+0.6}$	42.9$_{-0.3}$	54.9$_{+1.0}$	63.5$_{-3.0}$	70.6$_{+4.1}$	57.9$_{+0.5}$	59.5$_{+3.6}$	62.9$_{+1.3}$	58.2$_{+1.0}$
+ Skill-SP	**63.2**$_{+5.7}$	**57.1**$_{+3.4}$	**47.0**$_{+3.8}$	**59.0**$_{+5.1}$	**67.5**$_{+1.0}$	**78.7**$_{+12.2}$	**57.9**$_{+0.5}$	**65.9**$_{+10.0}$	**67.5**$_{+5.9}$	**62.5**$_{+5.3}$

Table 1 | **Main tool-call prediction results (avg@8).** Models are evaluated on API-Bank across three difficulty levels (L1–L3) and on BFCL across diverse programming languages and live scenarios. Absolute changes over base models are shown as subscripts: gains in green (**+Gain**), drops in red (**-Drop**).

Backbones. We evaluate five backbones spanning 3B to 14B parameters: Qwen3-4B-Instruct, Qwen3-8B (Yang et al., 2025a), Ministral-3-8B-Instruct, Ministral-3-14B-Instruct (Liu et al., 2026a), and Granite-4.1-3B (IBM Research, 2026). For each backbone, the identical checkpoint initializes both the proposer and the solver. Unless otherwise stated, skill refinement and induction rely solely on the backbone itself, without any external stronger teacher.

Baselines & Evaluation Strategy. To isolate the end-to-end impact of our framework, the main results (Section 4.2) evaluate the final solvers trained via Skill-SP against both the initial *Base* checkpoints and an *Unguided SP* baseline, which performs standard self-play relying solely on passive post-hoc filtering without proactive skill guidance. Crucially, because unguided generation structurally collapses on complex tasks like synthesizing valid logical puzzles, Unguided SP fails to bootstrap a viable training loop in the reasoning domain and is therefore evaluated exclusively on tool calling. Further mechanistic diagnostics and comparisons against other controlled variants are detailed in the ablation study (Section 4.3).

Training Protocol & Implementation Details. All experiments run for five iterations, optimizing policies via GRPO (4 proposer and 5 solver rollouts). The per-iteration training pool contains $M = 8{,}000$ tasks ($\alpha = 0.5$) via $K = 10$ probes for tool calling, and $M = 1{,}920$ tasks via a deterministic checker for reasoning. The proposer and solver undergo 5 and 15 update steps respectively for tool calling (3 and 10 for reasoning). Solver rewards target exact correctness and format compliance; the proposer receives a -1 penalty for structural parsing failures. Final benchmark results are reported as avg@8. The full hyperparameters, initial library $\mathcal{S}^{(0)}$, and formal algorithm are detailed in Appendices C, D, and E, respectively.

4.2. Main Results

Tool-call prediction. Table 1 reports the end-to-end performance on tool calling. Skill-SP universally improves all five backbones across diverse tool-calling scenarios. For competent base models like Qwen and Granite, the framework yields steady average enhancements of 2.8 to 6.5 absolute points. Crucially, Unguided SP not only yields consistently smaller overall gains, but also behaves inconsistently, occasionally causing capability degradation on specific subtasks. In contrast, Skill-SP maintains strictly positive and superior gains across the board. More strikingly, Skill-SP rescues initially misaligned models from severe zero-shot schema adherence failures. On Ministral-3-8B, for example, it drives a massive 42.9-point absolute gain where Unguided SP

Method	Overall Grid-level acc.	• Small $< 10^3$	■ Medium $10^3 \sim 10^6$	♦ Large $10^6 \sim 10^9$	♦♦ X-Large $> 10^9$	Cell-level Acc.
Qwen3-4B-Ins	72.1	97.2	88.6	62.0	18.7	70.3
+ **Skill-SP**	**73.5**$_{+1.4}$	**97.3**$_{+0.1}$	**89.7**$_{+1.1}$	**64.6**$_{+2.6}$	**21.9**$_{+3.2}$	**74.2**$_{+3.9}$
Qwen3-8B	23.6	67.2	7.3	0.3	0.0	43.8
+ **Skill-SP**	**32.4**$_{+8.8}$	**82.0**$_{+14.8}$	**20.7**$_{+13.4}$	**1.8**$_{+1.6}$	**0.1**$_{+0.1}$	**49.1**$_{+5.4}$
Ministral-3-8B	5.0	14.1	1.6	0.0	0.0	3.7
+ **Skill-SP**	**11.2**$_{+6.2}$	**32.8**$_{+18.7}$	**2.5**$_{+0.9}$	**0.1**$_{+0.1}$	**0.0**$_{+0.0}$	**23.7**$_{+20.0}$
Ministral-3-14B	5.4	14.9	2.2	0.2	0.0	7.3
+ **Skill-SP**	**17.4**$_{+12.0}$	**50.2**$_{+35.3}$	**4.6**$_{+2.4}$	**0.2**$_{+0.0}$	**0.0**$_{+0.0}$	**26.4**$_{+19.1}$
Granite-4.1-3B	11.6	35.4	0.8	0.1	0.0	34.3
+ **Skill-SP**	**12.6**$_{+1.0}$	**38.4**$_{+3.0}$	**1.2**$_{+0.4}$	**0.1**$_{+0.0}$	**0.0**$_{+0.0}$	**35.1**$_{+0.8}$

Table 2 | **Main logical reasoning results (avg@8).** Models are evaluated on the ZebraLogic benchmark across Small, Medium, Large, and X-Large task scales, where scale is determined by search space size. Overall Grid-level accuracy and Cell-level accuracy are reported. Absolute gains of Skill-SP over the corresponding base models are shown as subscripts (**+Gain**) next to the scores.

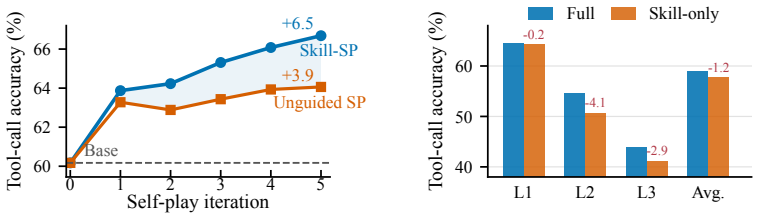
remains completely stagnant. This stagnation occurs because the base model completely fails to synthesize valid tasks independently, starving the unguided loop of meaningful learning signals. These results confirm that proactive skill orchestration translates reliably into generalized tool-use capabilities.

Logical reasoning. Table 2 confirms that Skill-SP successfully transfers to rigorous constraint-satisfaction reasoning, universally improving overall accuracy across all five backbones. While competent models like Qwen3-4B-Instruct show steady enhancements across all task scales, the framework drives striking turnarounds for logically misaligned models such as Ministral-3-14B, delivering remarkable grid-level gains of up to 12.0 points overall and exceeding 35 points on smaller puzzles. Naturally, progress on the extreme Large and X-Large scales remains limited for initially weak models, as pure self-play requires a minimal capability threshold to bootstrap valid learning signals. This confirms that guaranteeing the structural accuracy and broad diversity of synthesized puzzles robustly translates into advanced logical deduction.

4.3. Ablations

To isolate the contribution of each design choice, we evaluate controlled variants on Qwen3-4B-Instruct shown in Figure 4 and Tables 3 and 4. The results systematically validate four core design dimensions:

Naive self-play. The *Unguided SP* variant removes skill orchestration entirely. As depicted in Figure 4a, this approach plateaus early during training and yields a severe overall performance drop of 2.6 absolute points. This confirms that standard self-play without structural guidance is insufficient for complex tasks.



(a) Learning trajectory. (b) Skill-only ablation.

Figure 4 | **Tool-call ablations.** Skill-SP improves more steadily than Unguided SP, while skill-only data underperforms the mixed pool.

Over-specialization. Generating tasks exclusively from the skill library via the *Skill-only pool* degrades generalized evaluation performance on API-Bank (Figure 4b). This validates our dual-stream architecture: mixing open-ended exploration is crucial to preserve task diversity and prevent mode collapse.

Static skill orchestration. Both *Uniform routing* and *Frozen skills* strictly underperform the full system, causing overall accuracy drops of 1.9 and 2.3 points respectively (Table 3). This proves that the performance

gains stem from dynamic curriculum allocation and continuous library evolution rather than merely from injecting static structural constraints.

Ablation Variant	API-Bank				BFCL					Overall
	L1	L2	L3	Avg.	JS	Py	Java	Live	Avg.	
Full System	**64.6**	**54.7**	**44.0**	**58.9**	**65.5**	**96.0**	**63.5**	**78.5**	**75.9**	**66.7**
Unguided SP	60.2$_{-4.4}$	46.3$_{-8.4}$	40.8$_{-3.2}$	54.4$_{-4.5}$	65.8$_{+0.3}$	95.0$_{-1.0}$	63.0$_{-0.5}$	77.9$_{-0.6}$	75.4$_{-0.5}$	64.1$_{-2.6}$
Uniform routing	60.8$_{-3.8}$	50.4$_{-4.3}$	39.7$_{-4.3}$	55.0$_{-3.9}$	64.0$_{-1.5}$	95.9$_{-0.1}$	64.1$_{+0.6}$	78.5$_{+0.0}$	75.6$_{-0.3}$	64.8$_{-1.9}$
Frozen skills	61.3$_{-3.3}$	50.2$_{-4.5}$	40.2$_{-3.8}$	55.4$_{-3.5}$	61.5$_{-4.0}$	95.5$_{-0.5}$	63.1$_{-0.4}$	78.9$_{+0.4}$	74.8$_{-1.1}$	64.4$_{-2.3}$

Table 3 | **Quantitative ablations on the self-play data loop.** All variants are initialized with the Qwen3-4B-Instruct backbone, with each disabling a specific component of the proposal-side orchestration. Subscripts (-Drop) indicate the absolute performance degradation compared to the full system.

Co-evolutionary updates. Table 4 evaluates the necessity of continuous policy optimization. The *Frozen proposer* variant fixes the generator at initialization, causing a 2.1-point overall drop. This shows that the proposer must learn to leverage the evolving skill library via RL. More critically, the *Frozen feedback solver* variant fixes the solver used for probing and proposer rewards. This severs the dynamic frontier-tracking mechanism, supplying the proposer with outdated difficulty signals and leading to a severe 3.0-point degradation. Finally, *Frozen both* disables the co-evolutionary loop, yielding the largest performance drop (-3.2 points). This confirms that proactive curriculum generation requires both policies to co-evolve: the proposer must adapt to synthesize harder tasks, while the feedback solver must advance to provide accurate boundary evaluations.

Ablation Variant	API-Bank				BFCL					Overall
	L1	L2	L3	Avg.	JS	Py	Java	Live	Avg.	
Full System	**64.6**	**54.7**	**44.0**	**58.9**	**65.5**	**96.0**	**63.5**	**78.5**	**75.9**	**66.7**
Frozen proposer	62.0$_{-2.6}$	53.0$_{-1.7}$	43.8$_{-0.2}$	57.0$_{-1.9}$	55.5$_{-10.0}$	94.0$_{-2.0}$	62.6$_{-0.9}$	78.1$_{-0.4}$	72.6$_{-3.3}$	64.6$_{-2.1}$
Frozen feedback solver	59.9$_{-4.7}$	45.2$_{-9.5}$	39.3$_{-4.7}$	53.7$_{-5.2}$	65.3$_{-0.3}$	95.5$_{-0.5}$	62.8$_{-0.8}$	78.2$_{-0.3}$	75.4$_{-0.5}$	63.7$_{-3.0}$
Frozen both	59.2$_{-5.4}$	47.6$_{-7.1}$	39.2$_{-4.8}$	53.5$_{-5.4}$	63.5$_{-2.0}$	94.3$_{-1.8}$	62.0$_{-1.5}$	78.5$_{+0.0}$	74.6$_{-1.3}$	63.5$_{-3.2}$

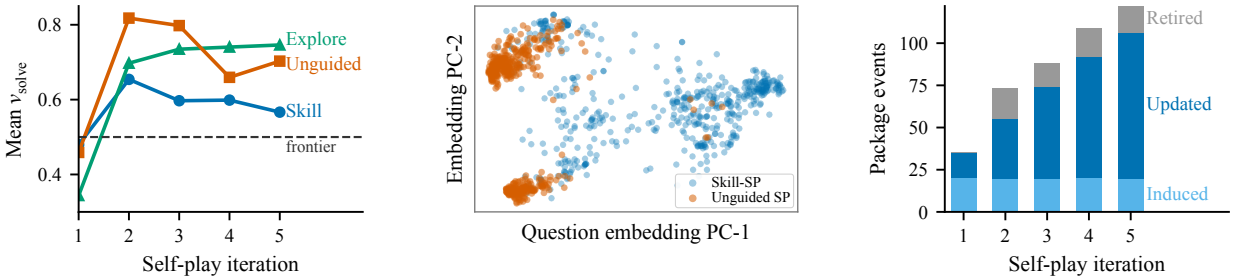
Table 4 | **Quantitative ablations on the co-evolutionary update loop.** All variants are initialized with the Qwen3-4B-Instruct backbone, with each disabling a specific component of the co-evolutionary update loop. Subscripts (-Drop) indicate the absolute performance degradation compared to the full system.

4.4. Data-Loop Diagnostics

Final accuracy alone obscures the underlying mechanism of self-improvement. To verify that Skill-SP constructs a controlled curriculum rather than merely acting as a passive data filter, we audit the accepted records and the skill-library lifecycle illustrated in Figure 5.

Curriculum Quality. Figure 5a tracks the empirical solver success rate v_{solve}. The skill-routed stream consistently generates tasks closer to the solver's learning frontier with a mean v_{solve} of approximately 0.57. This distinctly outperforms the Unguided SP and exploration streams, which drift to 0.70 and 0.75 respectively. In Figure 5b, we encode the generated questions using all-MiniLM-L6-v2 (Reimers & Gurevych, 2019) and project their embeddings via PCA. The visualization reveals that Unguided SP tasks cluster into narrow semantic regions, whereas the Skill-SP mixed pool demonstrates broader coverage in the embedding space. This demonstrates that skill orchestration simultaneously pushes task difficulty and fosters structural diversity.

Library Evolution. This frontier-targeted and diverse curriculum is driven by a highly dynamic interface. Across five iterations, Figure 5c shows that Skill-SP continuously induces roughly 20 new packages per round, updates existing ones based on execution traces, and retires obsolete skills. These diagnostics show that the skill library functions as an adaptive curriculum engine that expands agent capabilities.



(a) Frontier proximity. (b) Task diversity. (c) Skill lifecycle.

Figure 5 | **Data-loop diagnostics for Qwen3-4B-Instruct tool-call self-play.** Skill-stream records stay closer to the empirical frontier than both the exploration stream and Unguided SP, while the overall Skill-SP pool yields broader task coverage in the question-embedding space, and the skill library is continually induced, updated, and retired across self-play iterations.

Skill Utilization. A growing library must actively diversify training to be useful. As shown in Figure 6, the number of *active* skills (yielding ≥ 1 accepted record) expands to 86 across five iterations. To rigorously discount rare long-tail usage, we track *effective* skills via exponentiated Shannon entropy, $N_{\text{eff}} = \exp(-\sum_j p_j \log p_j)$ (where p_j is the accepted frequency fraction), which steadily grows to 46. This continuous growth confirms that Skill-SP successfully converts an expanding library into a broad and balanced curriculum, ensuring the generator does not idle on a few dominant patterns.

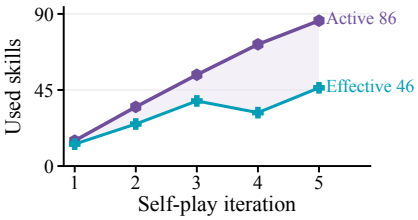


Figure 6 | **Skill utilization.** Active and effective skills grow steadily, driving broad curriculum expansion.

5. Conclusion

We presented ***Skill Self-Play*** (**Skill-SP**), a co-evolutionary framework designed to resolve the fundamental tension between task diversity and verification reliability in LLM self-evolution. By leveraging a dynamically evolving library of modular skills as proactive task-pattern interfaces, Skill-SP successfully steers the proposer to synthesize high-fidelity curricula tailored precisely to the solver's current learning frontier. This structured abstraction effectively mitigates the context bottlenecks of raw trajectory histories, while equipping the self-play loop with explicit, executable boundaries for rigorous verification. Our extensive evaluations across complex tool-calling and logical reasoning domains confirm that this dynamic orchestration systematically expands the performance horizons of diverse language models. Ultimately, our findings demonstrate that co-evolving skills serve as powerful training-time scaffolds, unlocking a sustainable, structured, and truly open-ended pathway for the autonomous self-evolution of large language models.

References

Kaikai An, Li Sheng, Ganqu Cui, Shuzheng Si, Ning Ding, Yu Cheng, and Baobao Chang. Ultraif: Advancing instruction following from the wild. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18722–18737, 2025.

Anthropic. Introducing Agent Skills | Claude — claude.com. https://claude.com/blog/skills, 2025.

Tao Chen, Gangwei Jiang, Pengyu Cheng, Siyuan Huang, Yihao Liu, Jingwei Ni, Jiaqi Guo, Mengyu Zhou, Kai Tang, Junling Liu, et al. Skill-rm: Unifying heterogeneous evaluation criteria via agent skill. *arXiv preprint arXiv:2606.03980*, 2026a.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution. *arXiv preprint arXiv:2510.23595*, 2025.

Zhiyu Chen, Zihan Guo, Bo Huang, Bingwei Lu, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. Skilljuror: Measuring how agent skill organization changes runtime behavior. *arXiv preprint arXiv:2606.11543*, 2026b.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

Jiale Cheng, Xiao Liu, Cunxiang Wang, Xiaotao Gu, Yida Lu, Dan Zhang, Yuxiao Dong, Jie Tang, Hongning Wang, and Minlie Huang. Spar: Self-play with tree-search refinement to improve instruction-following in large language models. *arXiv preprint arXiv:2412.11605*, 2024a.

Pengyu Cheng, Yong Dai, Tianhao Hu, Han Xu, Zhisong Zhang, Lei Han, Nan Du, and Xiaolong Li. Self-playing adversarial language game enhances llm reasoning. *Advances in Neural Information Processing Systems*, 37: 126515–126543, 2024b.

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. Skillret: A large-scale benchmark for skill retrieval in llm agents. *arXiv preprint arXiv:2605.05726*, 2026.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39286–39313, 2025.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Wei-Chen Wang, Shusheng Xu, Hanrui Wang, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents. *arXiv preprint arXiv:2601.22607*, 2026a.

Yudong Gao, Zongjie Li, Zimo Ji, Pingchuan Ma, Shuai Wang, et al. Skillreducer: Optimizing llm agent skills for token efficiency. *arXiv preprint arXiv:2603.29919*, 2026b.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

Zefeng He, Siyuan Huang, Xiaoye Qu, Yafu Li, Tong Zhu, Yu Cheng, and Yang Yang. Gems: Agent-native multimodal generation with memory and skills. *arXiv preprint arXiv:2603.28088*, 2026.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025a.

Siyuan Huang, Xiaoye Qu, Yafu Li, Yun Luo, Zefeng He, Daizong Liu, and Yu Cheng. Spotlight on token perception for multimodal reinforcement learning. *arXiv preprint arXiv:2510.09285*, 2025b.

Siyuan Huang, Xiaoye Qu, Yafu Li, Tong Zhu, Zefeng He, Muxin Fu, Daizong Liu, Wei-Long Zheng, and Yu Cheng. Persistent visual memory: Sustaining perception for deep generation in lvlms. *arXiv preprint arXiv:2605.00814*, 2026a.

Yixu Huang, Xinglei Yu, and Zhongyu Wei. Ace: Self-evolving llm coding framework via adversarial unit test generation and preference optimization. *arXiv preprint arXiv:2605.16299*, 2026b.

IBM Research. Granite 4.1 language models. https://huggingface.co/blog/ibm-granite/granit-4-1, 2026. Accessed: 2026-04-28.

Bowen Jiang, Taiwei Shi, Ryo Kamoi, Yuan Yuan, Camillo J Taylor, Longqi Yang, Pei Zhou, and Sihao Chen. One model, all roles: Multi-turn, multi-agent self-play reinforcement learning for conversational social intelligence. *arXiv preprint arXiv:2602.03109*, 2026a.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. Self-[in] correct: Llms struggle with discriminating self-generated responses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24266–24275, 2025a.

Xue Jiang, Yihong Dong, Mengyang Liu, Hongyi Deng, Tian Wang, Yongding Tao, Rongyu Cao, Binhua Li, Zhi Jin, Wenpin Jiao, et al. Coderl+: Improving code generation via reinforcement with execution semantics alignment. *arXiv preprint arXiv:2510.18471*, 2025b.

Xun Jiang, Feng Li, Han Zhao, Jiahao Qiu, Jiaying Wang, Jun Shao, Shihao Xu, Shu Zhang, Weiling Chen, Xavier Tang, et al. Long term memory: The foundation of ai self-evolution. *arXiv preprint arXiv:2410.15665*, 2024a.

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*, 2026b.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024b.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Joshua Kazdan, Rylan Schaeffer, Apratim Dey, Matthias Gerstgrasser, Rafael Rafailov, David L. Donoho, and Sanmi Koyejo. Collapse or thrive: Perils and promises of synthetic data in a self-generating world. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=buwLCdOHxO.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training. *arXiv preprint arXiv:2509.07414*, 2025.

Gengsheng Li, Jinghan He, Shijie Wang, Dan Zhang, Ruiqi Liu, Renrui Zhang, Zijun Yao, Junfeng Fang, Haiyun Guo, and Jinqiao Wang. R-diverse: Mitigating diversity illusion in self-play llm training. *arXiv preprint arXiv:2602.13103*, 2026a.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Yanchao Li, Wanhao Liu, Ben Gao, Jiaqing Xie, Zhehong Ai, Na Zou, Yuqiang Li, and Tianfan Fu. Skillsinjector: Dynamic skill context construction for llm agents. *arXiv preprint arXiv:2605.29794*, 2026c.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *arXiv preprint arXiv:2602.08004*, 2026.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026a.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025a.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025b.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing llm agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026b.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution. *arXiv preprint arXiv:2605.18401*, 2026c.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, et al. Skillrevise: Improving llm-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026d.

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11065–11082, 2024.

Hongliang Lu, Yuhang Wen, Pengyu Cheng, Ruijin Ding, Jiaqi Guo, Haotian Xu, Chutian Wang, Haonan Chen, Xiaoxi Jiang, and Guanjun Jiang. Search self-play: Pushing the frontier of agent capability without supervision. *arXiv preprint arXiv:2510.18821*, 2025.

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.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Procmem: Learning reusable procedural memory from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026.

Thao Nguyen, Jeffrey Li, Sewoong Oh, Ludwig Schmidt, Jason E Weston, Luke Zettlemoyer, and Xian Li. Better alignment with instruction back-and-forth translation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13289–13308, 2024.

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.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*, 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38, 2026.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2025.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13025–13048, 2024.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Qingyu Ren, Qianyu He, Jiajie Zhu, Xingzhou Chen, Jingwen Chang, Zeye Sun, Han Xia, Fei Yu, Jiaqing Liang, and Yanghua Xiao. Seif: Self-evolving reinforcement learning for instruction following. *arXiv preprint arXiv:2605.07465*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. *arXiv preprint arXiv:2601.05808*, 2026.

Yunhao Tang, Sid Wang, Lovish Madaan, and Rémi Munos. Beyond verifiable rewards: Scaling reinforcement learning in language models to unverifiable data. *Advances in Neural Information Processing Systems*, 38: 74421–74448, 2025.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

Aozhe Wang, Yuchen Yan, Nan Zhou, Zhengxi Lu, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Code-a1: Adversarial evolving of code llm and test llm via reinforcement learning. *arXiv preprint arXiv:2603.15611*, 2026a.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026b.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. Skillgrad: Optimizing agent skills like gradient descent. *arXiv preprint arXiv:2605.27760*, 2026c.

Qinsi Wang, Bo Liu, Tianyi Zhou, Jing Shi, Yueqian Lin, Yiran Chen, Hai Helen Li, Kun Wan, and Wentian Zhao. Vision-zero: Scalable vlm self-improvement via strategic gamified self-play. *arXiv preprint arXiv:2509.25541*, 2025.

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)*, pp. 13484–13508, 2023.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward training superintelligent software agents through self-play swe-rl. *arXiv preprint arXiv:2512.18552*, 2025.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Alex Wilf, Pranjal Aggarwal, Bryan Parno, Daniel Fried, Louis-Philippe Morency, Paul Pu Liang, and Sean Welleck. Propose, solve, verify: Self-play through formal verification. *arXiv preprint arXiv:2512.18160*, 2025.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pp. 30745–30766, 2024.

Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Qiushi Sun, Kanzhi Cheng, Junxian He, Jun Liu, and Zhiyong Wu. Genius: A generalizable and purely unsupervised self-training framework for advanced reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13153–13167, 2025a.

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*, volume 2025, pp. 76346–76382, 2025b.

Zhiling Yan, Dingjie Song, Hanrong Zhang, Wei Liang, Yuxuan Zhang, Yutong Dai, Lifang He, Philip S Yu, Ran Xu, Xiang Li, et al. Openskill: Open-world self-evolution for llm agents. *arXiv preprint arXiv:2606.06741*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Haoyan Yang, Mario Xerri, Solha Park, Huajian Zhang, Yiyang Feng, Sai Akhil Kogilathota, and Jiawei Zhou. Self-improvement of large language models: A technical overview and future outlook. *arXiv preprint arXiv:2603.25681*, 2026a.

Min Yang, Jinghua Piao, Xu Xia, Xiaochong Lan, Jiaju Chen, Yongshun Gong, and Yong Li. Skillmaster: Toward autonomous skill mastery in llm agents. *arXiv preprint arXiv:2605.08693*, 2026b.

Ziyi Yang, Weizhou Shen, Chenliang Li, Ruijun Chen, Fanqi Wan, Ming Yan, Xiaojun Quan, and Fei Huang. Spell: Self-play reinforcement learning for evolving long-context language models. *arXiv preprint arXiv:2509.23863*, 2025b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Ping Yu, Jack Lanchantin, Tianlu Wang, Weizhe Yuan, Olga Golovneva, Ilia Kulikov, Sainbayar Sukhbaatar, Jason Weston, and Jing Xu. Cot-self-instruct: Building high-quality synthetic prompts for reasoning and non-reasoning tasks. *arXiv preprint arXiv:2507.23751*, 2025a.

Wenhao Yu, Zhenwen Liang, Chengsong Huang, Kishan Panaganti, Tianqing Fang, Haitao Mi, and Dong Yu. Guided self-evolving llms with minimal human supervision. *arXiv preprint arXiv:2512.02472*, 2025b.

Huining Yuan, Zelai Xu, Zheyue Tan, Xiangmin Yi, Mo Guang, Kaiwen Long, Haojia Hui, Boxun Li, Xinlei Chen, Bo Zhao, et al. Marshal: Incentivizing multi-agent reasoning via self-play with strategic llms. *arXiv preprint arXiv:2510.15414*, 2025.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. *arXiv preprint arXiv:2505.19590*, 2025.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

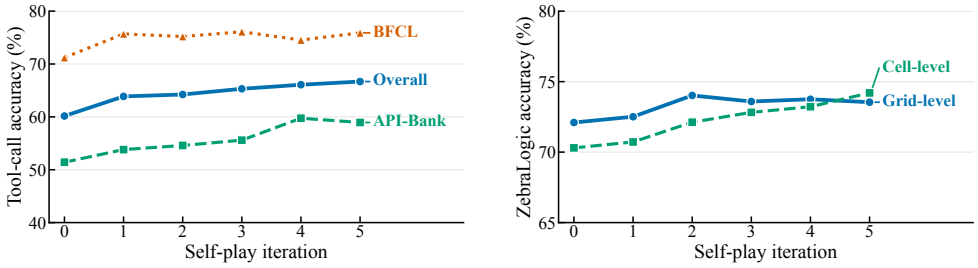
Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Yifan Zhou, Zhentao Zhang, Ziming Cheng, Shuo Zhang, Qizhen Lan, Zhangquan Chen, Zhi Yang, Ronghao Chen, Huacan Wang, Sen Hu, et al. Skillgenbench: Benchmarking skill generation pipelines for llm agents. *arXiv preprint arXiv:2605.18693*, 2026.

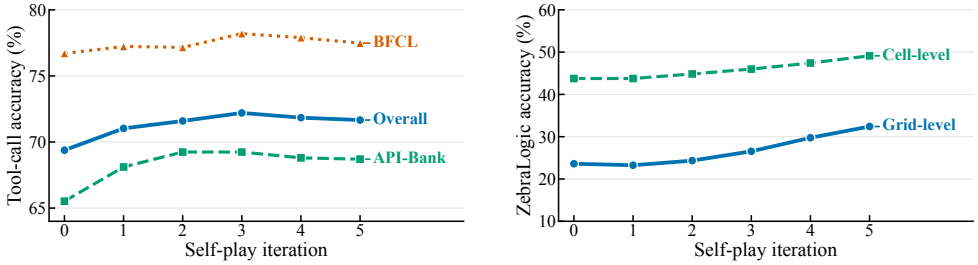
A. Iteration-wise Evaluation Trajectories

Figures 7–11 report raw held-out performance after each self-play iteration, with iteration 0 denoting the initial solver. Tool-call panels track the seven-subset overall score, API-Bank, and BFCL; logical-reasoning panels track ZebraLogic grid-level and cell-level accuracy.



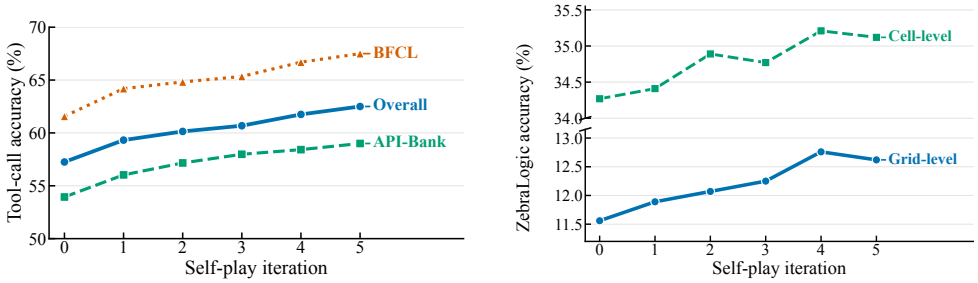
(a) Tool-call **(b) Logical reasoning**

Figure 7 | **Qwen3-4B-Instruct.** Held-out tool-call (left) and logical-reasoning (right) trajectories.



(a) Tool-call **(b) Logical reasoning**

Figure 8 | **Qwen3-8B.** Held-out tool-call (left) and logical-reasoning (right) trajectories.



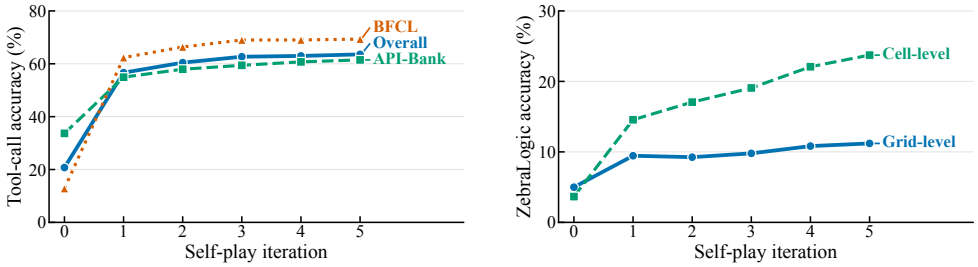
(a) Tool-call **(b) Logical reasoning**

Figure 9 | **Granite-4.1-3B.** Held-out tool-call (left) and logical-reasoning (right) trajectories.

B. Details of Dynamic Skill Routing

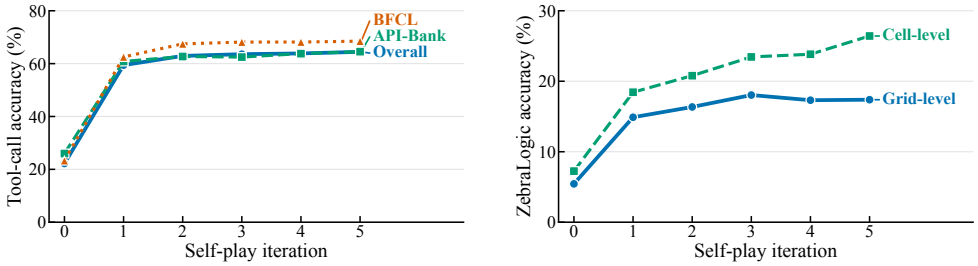
To ensure that the proposer efficiently focuses on structural priors that yield high-quality tasks, Skill-SP routes each generation attempt by dynamically sampling a skill $s \sim \mathcal{S}$. This routing mechanism must carefully balance exploiting historically reliable skills with exploring newly induced or under-tested ones.

For each skill $s_j \in \mathcal{S}$, its tracking statistics σ_j record the total number of historical selection attempts a_j. It also maintains the specific success counts across three crucial stages of validity verification (detailed in Section 3.2): the number of structurally verified candidates n_j^{ver}, solver-consistent candidates n_j^{con}, and frontier-difficulty tasks n_j^{fd}.



(a) Tool-call **(b) Logical reasoning**

Figure 10 | **Ministral-3-8B.** Held-out tool-call (left) and logical-reasoning (right) trajectories.



(a) Tool-call **(b) Logical reasoning**

Figure 11 | **Ministral-3-14B.** Held-out tool-call (left) and logical-reasoning (right) trajectories.

To mitigate the high variance in early sampling estimates, we apply symmetric Beta smoothing with a prior strength κ. The smoothed success rate for each outcome $o \in \{\text{ver}, \text{con}, \text{fd}\}$ is calculated as:

$$\widehat{r}_j^{\,o} = \frac{n_j^o + \kappa/2}{a_j + \kappa}. \tag{11}$$

We then compute a composite quality score $v_{\text{skill}}(\boldsymbol{s}_j)$ by taking a weighted average of these smoothed rates, prioritizing the fundamental structural validity while equally rewarding consistency and frontier difficulty:

$$v_{\text{skill}}(\boldsymbol{s}_j) = \frac{1}{2}\widehat{r}_j^{\,\text{ver}} + \frac{1}{4}\widehat{r}_j^{\,\text{con}} + \frac{1}{4}\widehat{r}_j^{\,\text{fd}}. \tag{12}$$

Finally, to prevent the routing distribution from prematurely collapsing onto a few dominant skills, we augment the sampling weight with a decaying exploration bonus. The sampling weight $w(\boldsymbol{s}_j)$ and the resulting selection probability $P(\boldsymbol{s}_j)$ are formulated as:

$$w(\boldsymbol{s}_j) = \text{clip}\big(v_{\text{skill}}(\boldsymbol{s}_j), w_{\min}, w_{\max}\big)\,\big(1 + \beta \exp(-a_j/\tau)\big), \qquad P(\boldsymbol{s}_j) = \frac{w(\boldsymbol{s}_j)}{\sum_{\boldsymbol{s}_\ell \in \mathcal{S}} w(\boldsymbol{s}_\ell)}, \tag{13}$$

where the clipping bounds $[w_{\min}, w_{\max}]$ ensure numerical stability, β controls the initial magnitude of the exploration bonus, and τ dictates its decay rate. This mechanism guarantees that under-tested skills are sampled sufficiently until their true empirical yield converges.

C. Training and Implementation Details

This section reports the settings that materially affect the generated curriculum, solver optimization, and final benchmark scores. Each training run uses eight NVIDIA A800 GPUs.

Task rewards. For tool-call prediction, the solver reward combines exact tool-call correctness and output format compliance:

$$\mathcal{R}_{\text{tool}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) = 0.9 \times \mathbf{1}\{\text{ExactCall}(\boldsymbol{y}, \boldsymbol{c})\} + 0.1 \times \mathbf{1}\{\text{ValidFormat}(\boldsymbol{y})\}. \tag{14}$$

Here, ValidFormat($\boldsymbol{y}$) indicates that the output contains at least one parseable tool call, and ExactCall($\boldsymbol{y}, \boldsymbol{c}$) indicates that at least one parsed call exactly matches the reference function name and JSON arguments in the hidden contract $\boldsymbol{c}$. For logical reasoning, the solver reward is a sparse puzzle-level signal:

$$\mathcal{R}_{\text{logic}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) = 0.9 \times \mathbf{1}\{\text{FullPuzzle}(\boldsymbol{y}, \boldsymbol{c})\} + 0.1 \times \mathbf{1}\{\text{ValidStructure}(\boldsymbol{y}, \boldsymbol{c})\}. \tag{15}$$

ValidStructure($\boldsymbol{y}, \boldsymbol{c}$) requires the output to parse into a legal Zebra grid with the correct house and attribute schema, while FullPuzzle($\boldsymbol{y}, \boldsymbol{c}$) requires all cells to match the hidden solution. Cell accuracy is logged as a diagnostic metric but is not used in the training reward.

Proposer repetition penalty. To encourage generation diversity and prevent batch-level mode collapse, we subtract a repetition penalty $\rho(\boldsymbol{x})$ from the proposer's RL reward during optimization. Within a proposer rollout batch of B candidates, the penalty for a generated prompt $\boldsymbol{x}_i$ is computed as $\rho(\boldsymbol{x}_i) = |C_i|/B$, where C_i is the greedy cluster of task texts formed using a Jaccard similarity threshold of 0.5.

For tool-call prediction, the frontier filter is applied after both structural parsing and solver probing. Skill-stream records additionally use package-level validators, while exploration-stream records rely on global schema validity and solver consistency before entering $\mathcal{D}^{(t)}$. For logical reasoning, the deterministic puzzle checker provides a stronger structural gate: generated puzzles must be parseable, valid, and uniquely solvable before they are used for solver training. Across both domains, the final solver never receives skill-package context at training or evaluation time; it only observes the standard prompt $\boldsymbol{x}$.

Setting	Tool-call prediction	Logical reasoning
Self-play loop	$T = 5$ iterations	$T = 5$ iterations
Proposer update	GRPO with four rollouts per prompt and five update steps per iteration	GRPO with four rollouts per prompt and three update steps per iteration
Solver update	Five rollouts per task group and 15 GRPO steps per iteration	Five rollouts per task group and 10 GRPO steps per iteration
Training pool	M = 8,000 accepted tasks per iteration with $\alpha = 0.5$ skill-stream records and 0.5 exploration-stream records	M = 1,920 checker-verified puzzles per iteration
Candidate filtering	Probe each candidate with $K = 10$ frozen-solver responses; retain records with $v_{\text{solve}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}}) \in [0.25, 0.75]$, solver consistency, and a unique majority answer	Enforce parser validity, puzzle validity, and uniqueness through the deterministic puzzle checker before solver training
Skill routing	$\kappa = 4$; $[w_{\min}, w_{\max}] = [0.25, 1.75]$; $\beta = 1$, $\tau = 8$; quality and exploration weighting enabled, age decay disabled	Same as tool-call prediction
Optimization	AdamW with learning rate 10^{-6}, weight decay 10^{-2}, gradient clipping at 1.0, and KL coefficient 10^{-2}	Same as tool-call prediction
Context and generation	8192-token prompt and response limits by default; the 14B Ministral tool-call runs use a 16384-token prompt limit	8192-token prompt and response limits
Skill evolution	Induce at least 20 new packages from accepted exploration records and refine packages after each iteration	Induce at least 10 new packages from verified puzzle records and refine packages after each iteration
Benchmark evaluation	avg@8 sampling with temperature 1.0, top-p = 1.0, and top-k = 40	avg@8 sampling with temperature 1.0, top-p = 1.0, and top-k = 40

Table 5 | **Training and evaluation details.** We report the main settings used for both task families.

D. Initial Skill Construction

Construction. We construct each task family's initial skill library $\mathcal{S}^{(0)}$ once and share it across all model backbones. The reported libraries contain 15 generic tool-call packages and 8 generic ZebraLogic packages. An LLM proposes these packages from domain-level task formats and verifier interfaces using the same representation as Skill-SP, $\boldsymbol{s} = \langle m, r, h, e, \nu, \sigma \rangle$. We retain only packages whose required fields can be parsed and whose domain-specific components pass structural and verification checks.

Effect of online evolution. The *Frozen skills* ablation in Table 3 isolates the contribution of this shared initialization by holding $\mathcal{S}^{(0)}$ fixed while retaining skill-guided generation. Frozen skills improves only marginally over Unguided SP (64.4 vs. 64.1 overall) and remains below the full system (66.7). Thus, most of the gain arises from online skill evolution rather than the initial library alone.

E. Full Pseudocode for Skill-SP

Algorithm 2 expands the high-level workflow in Section 3 into a single training loop.

F. Computational Overhead

A complete five-iteration Skill-SP run takes slightly over one day on average in our setup. Table 6 reports the average wall-clock breakdown across complete runs. Proposer policy optimization includes its on-policy rollouts, reward computation, and GRPO update, whereas solver-curriculum construction is the subsequent collection pass that generates, probes, verifies, and mixes the training pool $\mathcal{D}^{(t)}$. Explicit skill induction and refinement together account for only 6.5% of end-to-end runtime, indicating that evolving the skill library introduces limited computational overhead.

G. Analysis of Evaluation Benchmark

Our benchmark suite contains two complementary tool-call benchmarks, API-Bank and BFCL, and one logical-reasoning benchmark, ZebraLogic. Together, they cover two forms of verifiable agent behavior: selecting and grounding executable actions from dialogue context, and solving deterministic constraint-satisfaction problems with a unique answer.

API-Bank. API-Bank (Li et al., 2023) is the primary benchmark for evaluating tool-call prediction in our experiments. Each instance is normalized into a standard record containing a system prompt with tool descriptions, a user-side dialogue history, and a hidden reference answer specifying the correct tool name and arguments. The solver receives only the system and user messages and must output a single tool invocation. Evaluation parses the predicted `<tool_call>` block and exact-matches the normalized tool name and parameters against the reference answer. We report accuracy on Levels 1–3, which lets us separate easier tool-routing cases from harder dialogue states that require more careful argument propagation and context grounding. This benchmark therefore measures whether self-play improves the core operational skill that Skill-SP trains for: producing executable, schema-valid actions from standard prompts.

BFCL. BFCL (Patil et al., 2025) complements API-Bank by evaluating tool-call generalization on a distinct benchmark source. In the main paper, we report the displayed BFCL categories used in the seven-subset tool-call average: *bfcl_simple_javascript*, *bfcl_simple_python*, *bfcl_simple_java*, and *bfcl_live_simple*. These categories stress different function-schema conventions, type systems, and argument normalization requirements. Unlike API-Bank, BFCL is not merely a level-wise slice of the same benchmark family; it tests whether the solver's learned tool-use behavior transfers to new function descriptions and answer formats. We use the benchmark-specific scorer to compare predicted calls against the provided possible answers after category-aware normalization. BFCL therefore serves as a transfer check: improvements should persist when the solver faces different function descriptions, category conventions, and answer-normalization rules.

Algorithm 2: SKILL SELF-PLAY (Skill-SP)

Input: Initial proposer $\pi^{(0)}_{\text{propose}}$, solver $\pi^{(0)}_{\text{solve}}$, and skill library $\mathcal{S}^{(0)}$; self-play iterations T; target pool size M; curriculum ratio α; solver probe count K.

Output: Final solver $\pi^{(T)}_{\text{solve}}$.

for $t = 0, 1, \ldots, T-1$ **do**
  Initialize $\mathcal{T}^{(t)}_{\text{skill}} \leftarrow \emptyset$ and $\mathcal{T}^{(t)}_{\text{explore}} \leftarrow \emptyset$;
  foreach *skill-conditioned proposal attempt* **do**
    Sample $\boldsymbol{s} \sim \mathcal{S}^{(t)}$ using its routing statistics σ;
    Generate $(\boldsymbol{x}, \boldsymbol{c}) \sim \pi^{(t)}_{\text{propose}}(\cdot \mid \boldsymbol{s})$;
    Draw K responses from $\pi^{(t)}_{\text{solve}}(\cdot \mid \boldsymbol{x})$ to evaluate $v_{\text{solve}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}})$, $\mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \text{ is valid}\}}$, and $\mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}})$;
    if $\mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \textit{ is valid}\}} = 1$ **then**
      Add $(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{s})$ to $\mathcal{T}^{(t)}_{\text{skill}}$;
    end
  end
  foreach *open-ended proposal attempt* **do**
    Generate $(\boldsymbol{x}, \boldsymbol{c}) \sim \pi^{(t)}_{\text{propose}}(\cdot \mid \varnothing)$;
    Draw K responses from $\pi^{(t)}_{\text{solve}}(\cdot \mid \boldsymbol{x})$ to evaluate $v_{\text{solve}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}})$, $\mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \text{ is valid}\}}$, and $\mathcal{R}_{\text{propose}}(\boldsymbol{x}, \boldsymbol{c}; \pi^{(t)}_{\text{solve}})$;
    if $\mathbb{1}_{\{(\boldsymbol{x},\boldsymbol{c}) \textit{ is valid}\}} = 1$ **then**
      Add $(\boldsymbol{x}, \boldsymbol{c})$ to $\mathcal{T}^{(t)}_{\text{explore}}$;
    end
  end
  Construct $\mathcal{D}^{(t)}$ from $\mathcal{T}^{(t)}_{\text{skill}}$ and $\mathcal{T}^{(t)}_{\text{explore}}$ using Eq. 6;
  Update $\pi^{(t)}_{\text{propose}} \rightarrow \pi^{(t+1)}_{\text{propose}}$ with GRPO according to Eq. 5, keeping $\pi^{(t)}_{\text{solve}}$ fixed;
  Update $\pi^{(t)}_{\text{solve}} \rightarrow \pi^{(t+1)}_{\text{solve}}$ with GRPO on $\mathcal{D}^{(t)}$ according to Eq. 7;
  Update σ from skill-conditioned attempts; refine skills from invalid attempts and accepted execution traces;
  Build $\mathcal{T}^{(t)}_{\text{induce}}$ by retaining frontier-targeted samples from $\mathcal{T}^{(t)}_{\text{explore}}$;
  Induce $\mathcal{S}^{(t)}_{\text{new}}$ from $\mathcal{T}^{(t)}_{\text{induce}}$ and identify $\mathcal{S}^{(t)}_{\text{prune}}$;
  Update the active library as $\mathcal{S}^{(t+1)} \leftarrow (\mathcal{S}^{(t)} \setminus \mathcal{S}^{(t)}_{\text{prune}}) \cup \mathcal{S}^{(t)}_{\text{new}}$;
end
return $\pi^{(T)}_{solve}$;

Pipeline component	**Runtime share**
Proposer policy optimization	24.3%
Solver-curriculum construction	57.0%
Skill-stream generation and verification	20.0%
Exploration-stream generation and verification	37.0%
Solver policy optimization	10.2%
Skill-library evolution	6.5%
Skill induction	1.3%
Skill refinement	5.2%
Held-out benchmark evaluation	1.9%
Other orchestration	0.1%

Table 6 | **Average wall-clock breakdown.** Indented rows decompose their parent stage. Skill-library evolution occupies only a small fraction of the complete pipeline.

Benchmark	Task Form	Main Capability	Reported Metrics
API-Bank (Li et al., 2023)	Dialogue-conditioned next tool call	Tool selection, argument grounding, schema adherence	Level 1–3 accuracy and average
BFCL (Patil et al., 2025)	Function-calling prompts across programming and live-tool categories	Cross-benchmark tool-call generalization	Category accuracy and BFCL average
ZebraLogic (Lin et al., 2025)	Zebra-style grid puzzles with deterministic constraints	Constraint tracking, uniqueness reasoning, complete solution synthesis	Grid-level and cell-level accuracy

Table 7 | **Evaluation benchmark roles.** API-Bank and BFCL evaluate tool-call prediction from different benchmark sources, while ZebraLogic evaluates a separate logical-reasoning task family with deterministic verification.

ZebraLogic. ZebraLogic (Lin et al., 2025) evaluates a different verifiable agent task family: logical reasoning over grid-structured constraint puzzles. Each instance provides a natural-language puzzle whose solution is a complete assignment of attributes to houses. The evaluator checks whether the model's output can be parsed into a valid grid, whether the predicted schema matches the reference schema, and whether each cell agrees with the hidden solution. We report both grid-level accuracy, which requires the entire puzzle to be solved correctly, and cell-level accuracy, which measures partial progress. The benchmark is further stratified by search-space scale into Small, Medium, Large, and X-Large subsets. This stratification is useful for Skill-SP because it separates superficial format compliance from genuine constraint tracking: gains on larger or full-puzzle metrics indicate that the generated curriculum helps the solver learn harder reasoning patterns rather than only improving local answer formatting. ZebraLogic also exposes the limitation of unguided self-play most clearly, since generating valid, uniquely solvable puzzles requires proactive structural guidance rather than passive post-hoc filtering.

H. Limitations and Future Work

While Skill-SP is highly effective for autonomous agent self-evolution, discovering entirely novel task patterns inherently requires the base model to possess a minimum foundational capability to bootstrap valid learning signals. For extremely complex domains, the framework might initially benefit from a small set of human demonstrations to jumpstart the evolving library. Furthermore, the current instantiation relies on fixed heuristics, such as the static skill-stream mixing ratio α and pre-defined difficulty bounds, which may require empirical tuning for new task families.

Building on these observations, future work will explore several promising directions. Replacing fixed routing heuristics with learnable, dynamic curriculum schedulers could autonomously optimize the data orchestration process. Alongside this, we aim to investigate the fully automated co-induction of generative rules and executable validators directly from raw environment interactions. Broadening the scope of the framework, transferring evolved skill libraries across different model architectures could enable strong frontier models to bootstrap smaller, efficient agents, presenting an exciting avenue for scalable and democratized alignment.

I. Case Studies of Induced Skills

All skill packages shown in this section were induced during Skill-SP runs initialized from Qwen3-4B-Instruct.

I.1. Tool-Call Prediction

Each package records a reusable transition from an observed dialogue state to one valid next tool call, together with generation constraints and a local validator. The following examples illustrate library-level pattern induction.

Each complete listing is organized into routing metadata and curriculum state (m, σ), proposer-visible construction fields (r, h, e), and the package-local validator ν. In the colored listings, blue denotes metadata and curriculum state, orange denotes procedural construction resources, green denotes hints and validation, and gray denotes a package example. Any listed solver procedure is an offline diagnostic oracle used to inspect generated samples.

Table 8 presents two packages whose metadata records `source=skill_induction`. Both were induced in iteration 2 and specialize different forms of observation grounding. `skill_038` turns a previously identified top-rated restaurant into the `restaurant_id` of a reservation, while `skill_042` uses a confirmed availability observation, rather than the user's broader time window, to construct a meeting interval.

Induced package	Observed evidence	Induced grounding rule	Local validation and action
`skill_038` *Discovery → reservation* Iteration 2 74/82 filtered/routed	The user asks to reserve for four at 7 PM. A prior observation identifies the top-rated Italian restaurant and its identifier, `TRT-8842`.	Select `book_table` after a rated discovery; copy `restaurant_id` from the observation and normalize the requested date and time.	Require the observed identifier, complete arguments, ISO date/time, and a positive guest count. `book_table`: `TRT-8842`, `2023-09-16`, `19:00`, 4 guests
`skill_042` *Availability → scheduling* Iteration 2 1,102/1,993 filtered/routed	The user requests a meeting for Sarah and Alex between 10 AM and 2 PM, on "project updates." A prior observation reports that only 10:30–11:30 is available.	Select `ScheduleMeeting` after the availability check; use the confirmed slot rather than the requested window, and copy the participants and topic.	Require ISO timestamps inside the requested window and evidence-grounded participants and topic. `ScheduleMeeting`: Sarah, Alex; `10:30-11:30`; project updates

Table 8 | **Tool-call case studies from the induced skill library.** All packages were created from self-play trajectories and saved with `source=skill_induction`. Each package binds prior evidence to a next-action rule and checks that the resulting call remains grounded and schema-valid. The final line in each row compactly renders the verified call from its package example; filtered/routed reports the package statistics accumulated during routing.

I.1.1. Complete induced package: `skill_038`

The complete listing of `skill_038` is organized into four color-coded blocks so that each package component remains readable at paper scale. Together, they reproduce all package-specific fields and one complete proposer example.

Routing Metadata and Curriculum State (m, σ) — **`skill_038`**

Package	`skill_038` — `book_table_from_discovered_restaurant`; added in iteration 2 with `source=skill_induction`; `induction_novelty=specialization`.
Description	When a user requests a dinner reservation with a prior restaurant discovery and specifies guests, date, and time, the system should book a table at the top-rated restaurant from the discovery list.
Induction reason	The dialogue state shows a clear pattern of task progression from restaurant discovery to table booking, grounded in user intent, prior observations, and explicit constraints. The tool selection is logically derived from the sequence of user requests and available observations, with parameters filled from prior findings and normalized into valid formats.
Statistics	82 attempts; 82 consistent; 74 boundary and verified records; 8 too-easy records; no too-hard or inconsistent records; mean $v_{\text{solve}} = 0.572$; last updated in iteration 4.

Proposer Construction Rules (r)	skill_038
Trigger	The user requests a table booking after a restaurant discovery and specifies a date, time, and guest count.
Required evidence	A prior observation contains a top-rated restaurant with an identifier, cuisine, and price constraint; the user specifies the date, time, and number of guests.
Distractors	`find_restaurant` and `check_schedule` are distractors because the restaurant has already been identified.
Tool rule	Select `book_table` only when the user explicitly requests a table after restaurant discovery and rating.
Parameter rule	Copy the identifier from the top-rated observation; normalize the date to `YYYY-MM-DD`, the time to `HH:MM`, and the guest count to a number.
Answer checks	Require `restaurant_id`, `date`, `time`, and `guests`; require valid ISO date/time formats and an identifier from the top-rated observation.

Proposer Hint, Validator, and Offline Oracle (h, ν)	skill_038
Generator hint	Vary the city, cuisine, maximum price, date, and time while ensuring that a preceding restaurant discovery with a top rating exists.
Validator	Tool name must be `book_table`; `restaurant_id` must be a string from a prior top-rated observation; `date` must use `YYYY-MM-DD`; `time` must use `HH:MM`; `guests` must be a positive integer; and all parameters must be present.
Offline recognition	Look for a user request to book a table after a prior restaurant discovery.
Offline evidence extraction	Extract the identifier from a top-rated observation, together with the number of guests, date, and time.
Offline decision rule	Choose `book_table` if a top-rated restaurant with an identifier is observed and the user requests a table booking.
Offline parameter recovery	Fill `restaurant_id` from the top-rated observation; normalize date to `YYYY-MM-DD`, time to `HH:MM`, and guests to an integer.
Offline final check	Ensure that every parameter is present, valid, and derived from prior evidence.

Proposer Example (e)	skill_038
Available tools	`get_weather(location, units)`, `find_restaurant(city, cuisine, max_price)`, `book_table(restaurant_id, date, time, guests)`, `check_schedule(user_email, date)`, and `send_message(recipient, message)`.
Dialogue state	`<user>` "I want to plan a dinner for my friends this weekend in Portland. We're looking for a nice Italian place that doesn't cost more than $30 per person." `<response>` "I can help you find some Italian restaurants in Portland with a max price of $30." `<obs>` "Found 5 Italian restaurants in Portland with a max price of $30." `<user>` "Can you book a table for 4 of us on Saturday at 7 PM?" `<response>` "Which restaurant would you like me to book a table for?" `<user>` "Can you pick one of the top-rated ones from the list?" `<obs>` "Top-rated Italian restaurant in Portland with max price $30: Trattoria al Sole, ID: `TRT-8842` (rating: 4.7)." `<user>` "Based on our conversation above, please only make one tool call to solve my need."
Verified call	`name=book_table; restaurant_id=TRT-8842; date=2023-09-16; time=19:00; guests=4`.

Table 9 | **Complete induced package `skill_038`.** The four blocks instantiate $(m, r, h, e, \nu, \sigma)$: routing metadata, construction rules, hint and validation, and a proposer example.

I.1.2. Complete induced package: `skill_042`

`skill_042` was induced in the same iteration as `skill_038`, but captures a different dependency: a proposed tool call must follow the confirmed availability observation rather than simply mirror the user's requested time range. Its four-block listing uses the same organization as above.

Routing Metadata and Curriculum State (m, σ)	**skill_042**
Package	`skill_042` — `ScheduleMeetingWithTimeSlot`; added in iteration 2 with `source=skill_induction`; `induction_novelty=refinement`.
Description	This pattern captures scheduling a meeting between two or more participants within a specified time window, using confirmed start and end times derived from user-provided slots and availability observations. The meeting topic is extracted directly from the user's request.
Induction reason	The dialogue state involves a clear, structured request to schedule a meeting with specific participants, time constraints, and a topic, with the tool selection being unambiguous and grounded in explicit user input and observation. The pattern generalizes beyond this instance to any meeting scheduling request with defined participants, time window, and topic, making it reusable and nontrivial.
Statistics	1,993 attempts; 1,993 consistent; 1,102 boundary and verified records; 891 too-easy records; no too-hard or inconsistent records; mean $v_{\text{solve}} = 0.695$; last updated in iteration 4.

Proposer Construction Rules (r)	**skill_042**
Trigger	The user requests a meeting between participants with a time range and topic, and an observation confirms an available time slot.
Required evidence	The user specifies participants, a time range (for example, 10 AM to 2 PM), and a topic; an observation confirms a particular available slot within that range.
Distractors	`SearchProduct` and `GetWeather` are irrelevant because the dialogue contains no evidence of product search, weather, or account balance.
Tool rule	Choose `ScheduleMeeting` when the user explicitly requests a meeting with participants and a time slot, and an observation confirms availability.
Parameter rule	Copy participants from user input; normalize `start_time` and `end_time` to ISO 8601 using the confirmed slot (for example, 10:30–11:30 becomes `2023-10-12T10:30:00` and `2023-10-12T11:30:00`); copy the topic from user input.
Answer checks	Require `ScheduleMeeting`, an array of participants, ISO 8601 `start_time` and `end_time`, and a topic; all values must be grounded in dialogue or observation and match the schema.

These examples are more structured than a direct keyword-to-tool association. They bind the current action to evidence accumulated in earlier turns: a selected candidate becomes an identifier, and an observed slot becomes the executable interval. The corresponding package statistics show that the patterns are not merely retained as descriptions: both contributed filtered frontier records during routing. More broadly, the induced library contains analogous search-to-action and context-to-parameter patterns across restaurant, calendar, flight, and reminder workflows. This qualitative trace supports the mechanism underlying Skill-SP's evolving curriculum coverage: exploration can be consolidated into reusable, verifiable task patterns.

I.2. Logical Reasoning

Logical-reasoning packages encode reusable constraint topologies rather than tool-selection rules. Each package combines a procedural construction rule with a declarative compiler specification that guides the fixed logical runtime in assembling constraint blueprints. The runtime then admits an instance only when its schema, constraints, and unique solution pass deterministic checks. The exported logical library does not populate per-package routing statistics; accordingly, we report saved accepted structural-evidence records below rather than treating these counts as solver performance.

Proposer Hint, Validator, and Offline Oracle (h, ν)	skill_042
Generator hint	Vary participants (for example, John and Lisa), time ranges (for example, 9 AM to 5 PM), and topics (for example, Q3 review or budget planning), while ensuring that a confirmed time slot is provided in an observation.
Validator	Tool name must be `ScheduleMeeting`; participants must be a non-empty string array; `start_time` and `end_time` must be ISO 8601 timestamps; start must precede end; both timestamps must lie within the requested window; the topic must be non-empty; and no parameter may be invented.
Offline recognition	Look for requests involving "schedule a meeting" or "plan a meeting" with participants and time constraints.
Offline evidence extraction	Extract the participants, the requested start and end range, and the topic.
Offline decision rule	If an observation confirms a specific available slot within the requested window, select `ScheduleMeeting`.
Offline parameter recovery	Normalize times to ISO 8601; copy participants and topic directly; use the confirmed slot as the start and end times.
Offline final check	Ensure that all parameters are present, valid, derived from dialogue or observation, and free of schema violations.

Proposer Example (e)	skill_042
Available tools	`GetWeather(location, units)` with `units` in {`celsius`, `fahrenheit`}; `SearchProduct(query, category)` with `category` in {`electronics`, `clothing`, `books`, `home`}; `ScheduleMeeting(participants, start_time, end_time, topic)`; `GetAccountBalance(account_id)`; `SendEmail(to, subject, body)`; and `FindRoute(origin, destination, mode)` with `mode` in {`driving`, `walking`, `cycling`}.
Dialogue state	`<user>` "I need to plan a meeting between Sarah and Alex next week. The meeting should be between 10 AM and 2 PM on Thursday, and the topic is project updates." `<response>` "I can help you schedule that meeting. Could you confirm the participants and preferred time slot?" `<user>` "Yes, participants are Sarah and Alex. Confirm the meeting on Thursday, 10 AM to 2 PM." `<response>` "I'll check if that time works. I need to schedule the meeting now." `<obs>` "Available time slots for Sarah and Alex on Thursday between 10:00 and 14:00 are fully booked except for 10:30 to 11:30." `<user>` "That works. Let's go with 10:30 to 11:30." `<response>` "Great! I'll proceed with scheduling the meeting at 10:30 AM to 11:30 AM on Thursday for Sarah and Alex on the topic of project updates." Final request: "Based on our conversation above, please only make one tool call to solve my need."
Verified call	`name=ScheduleMeeting; participants=[Sarah, Alex]; start_time=2023-10-12T10:30:00; end_time=2023-10-12T11:30:00; topic=project updates.`

Table 10 | **Complete induced package `skill_042`.** The four blocks instantiate $(m, r, h, e, \nu, \sigma)$: routing metadata, construction rules, hint and validation, and a proposer example.

Induced package	Constraint topology	Compiler-enforced structure	Saved structural evidence
`zebra_skill_022` *Fixed-anchor adjacency bridge* Iteration 1	A fixed entity, a cross-attribute `next_to` relation, and a `same_house` bridge resolve a positional ambiguity.	`anchored_component` → `component_bridge` → `same_house_chain`; requires at least one `fixed`, `same_house`, and `next_to` clue.	333 accepted and 4 rejected evidence records.
`zebra_skill_031` *Chain-anchor-bridge* Iteration 2	A directed three-value chain is anchored at one endpoint, then connected across attributes through `same_house` and `next_to` bridges.	`directed_chain` → `same_house_chain` → `component_bridge`; requires at least two `directly_left_of`, one `same_house`, and one `next_to` clue.	178 accepted and 2 rejected evidence records.

Table 11 | **Induced logical-reasoning skill patterns.** Both packages are saved with `source=zebra_skill_induction`. Counts are accepted or rejected structural-evidence records, not solver scores. Every accepted record reported here has valid schema and constraints, a unique puzzle, and a capped solution count of one.

I.2.1. Complete induced package: `zebra_skill_022`

We first expand `zebra_skill_022`, an early-stage fixed-anchor adjacency bridge induced in iteration 1. A fixed entity, a cross-attribute `next_to` relation, and a `same_house` component jointly resolve a non-local positional dependency. The following five blocks give its complete package-specific content using the $(m, r, h, e, \nu, \sigma)$ organization.

Routing Metadata and Induction Record (m, σ)	`zebra_skill_022`
Package	`zebra_skill_022` — `Fixed-Anchor Adjacency Bridge`; added in iteration 1 with `source=zebra_skill_induction`.
Description	A `next_to` relation links a fixed-position entity to a value in another attribute, while a `same_house` component anchors a value pair. Together, the fixed anchor and adjacency bridge resolve a non-local positional dependency.
Induction reason	The inducing trace contained directed chains and cross-attribute order relations but no reusable pattern combining a fixed entity, cross-attribute adjacency, and a `same_house` anchor. This package adds that missing adjacency-based bridge.
Induction novelty	The bridge resolves relative positions through both adjacency and shared-house structure, rather than isolated `left_of` or `directly_left_of` constraints.
Evidence state	The exported `stats.json` has no populated routing counters (the σ fields are zero). Independently saved refinement evidence contains 333 accepted and 4 rejected records; each accepted record satisfies schema validity, constraint validity, uniqueness, and capped solution count.

I.2.2. Complete induced package: `zebra_skill_031`

We next expand `zebra_skill_031`, a later chain–anchor–bridge pattern induced in iteration 2. It extends the fixed-anchor construction above with a directed chain and two cross-attribute bridge types. The following five blocks expose its package fields using the structure defined in Section 3.2, together with one accepted unique puzzle instance.

The logical examples illustrate a different form of library evolution from the tool-call setting. Induction adds reusable constraint topologies and their compiler constraints, enabling subsequent generations to jointly structure directed chains, cross-attribute bridges, and uniqueness conditions.

Proposer Construction Resource (r): Declarative Compiler **zebra_skill_022**

Compiler spec	`version=1`; `fixed_limit=1`; preferred relation types are `fixed`, `same_house`, and `next_to`.
Seed program	(1) `anchored_component(anchors=1, component_width=2)`; (2) `component_bridge(component_width=2, relation=next_to)`; (3) `same_house_chain(width=2)`.
Compiler guards	`prune=true` and `ensure_relation_diversity=true`; `derived_from=zebra_skill_010`.

Proposer Construction Rules (r) **zebra_skill_022**

Generation trigger	Use this pattern when a fixed-position entity is linked by `next_to` to a value in another attribute and a `same_house` constraint anchors a value pair.
Construction rule	Start from a fixed value in a specific house. Add a cross-attribute `next_to` relation, then a `same_house` pair in a different house so that the adjacency and shared-house components jointly resolve positions.
Difficulty controls	Increase difficulty by using a non-extreme fixed house, adding a `not_house` exclusion that blocks one adjacent position, or choosing a non-adjacent `same_house` component.

Proposer Hint and Package Validator (h, ν) **zebra_skill_022**

Generator hint	Anchor a fixed value in a specific house; connect a name to a hobby or sport through `next_to`; and add a `same_house` pair for two attributes. Keep the `next_to` relation adjacent to the fixed entity, and avoid placing the `same_house` pair in the fixed entity's house unless required.
Type guard	Require at least one `fixed`, one `same_house`, and one `next_to` relation.
Local validator	Verify the fixed house, adjacency, and shared-house relation; require the resulting constraint system to have a unique solution.
Offline oracle	Assign the fixed value to its house; enumerate adjacent houses for the `next_to` value; apply the `same_house` pair; eliminate fixed-anchor and adjacency conflicts; then solve the remaining CSP under all-different and adjacency constraints.

Proposer Example (e) and Accepted Offline Instance	zebra_skill_022
Stored package example	Two houses; `Name`={Elena, Felix}, `Nationality`={German, Japanese}, `Drink`={Coffee, Tea}, `Hobby`={Gardening, Reading}, and `FavoriteSport`={Badminton, Soccer}. Clues: (1) Elena directly left of Felix; (2) Felix in House 2; (3) Elena left of Felix; (4) German same house as Soccer; (5) Tea left of Coffee; (6) Elena next to Badminton; (7) Gardening not in House 1.
Example rationale	Felix fixed in House 2 and Elena next to Badminton force Elena to House 1 and Badminton to House 2; the `same_house` component then resolves the remaining positional bridge.
Accepted blueprint	Three houses; `Name`={Leo, Mia, Ryan}, `Nationality`={French, Indian, Swedish}, `Drink`={Latte, GreenTea, Cappuccino}, `Hobby`={Photography, Cooking, Reading}, and `FavoriteSport`={Tennis, Badminton, Volleyball}; difficulty `medium`; theme *City Apartment Residents*.
Compiled clues	(1) Leo in House 2; (2) Leo same house as Indian; (3) Mia same house as Swedish; (4) Indian same house as Cappuccino; (5) Swedish next to Cappuccino; (6) Ryan same house as French; (7) GreenTea same house as Volleyball; (8) Swedish next to Tennis; (9) Photography left of Reading; (10) Volleyball directly left of Indian; (11) French same house as Cooking; (12) Photography left of Cooking.
Unique solution	House 1: GreenTea, Volleyball, Photography, Mia, Swedish; House 2: Cappuccino, Tennis, Reading, Leo, Indian; House 3: Latte, Badminton, Cooking, Ryan, French.
Structural checks	`logical_generation_mode=blueprint`; `prompt_valid=1`; `schema_valid=1`; `constraints_valid=1`; `puzzle_unique=1`; `solution_count_capped=1`.

Table 12 | **Complete induced package `zebra_skill_022`.** The five blocks instantiate $(m, r, h, e, \nu, \sigma)$: metadata, compiler resource, construction rules, hint and validation, and a stored example with an accepted offline instance.

Routing Metadata and Induction Record (m, σ)	zebra_skill_031
Package	`zebra_skill_031` — `Chain-Anchor-Bridge Pattern`; added in iteration 2 with `source=zebra_skill_induction`; `induction_novelty` is a hybrid chain-solving structure.
Description	A directed adjacency chain is anchored by a fixed value and reinforced with cross-attribute `same_house` and `next_to` bridges, so deduction propagates through value-sharing and adjacency rather than position alone.
Induction reason	The inducing trace contains a linear `directly_left_of` chain (Elara–Gita–Leo), anchored by Leo in House 3 and extended through cross-attribute links. The pattern adds `same_house` and `next_to` bridges that the source directed-chain skill does not model.
Evidence summary	The exported `stats.json` has no populated routing counters (the σ fields are zero). The saved refinement evidence contains 178 accepted and 2 rejected records. Every accepted record has `constraints_valid=1`, `schema_valid=1`, `puzzle_unique=1`, and `solution_count_capped=1`.

Proposer Construction Resource (r): Declarative Compiler	zebra_skill_031
Runtime spec	`version=1`; `fixed_limit=1`; preferred relation types are `directly_left_of`, `same_house`, and `next_to`.
Seed program	(1) `directed_chain(length=3, anchor_end=true)`; (2) `same_house_chain(width=2)`; (3) `component_bridge(component_width=2, relation=next_to)`.
Compiler guards	`prune=true` and `ensure_relation_diversity=true`.
Lineage	`derived_from=zebra_skill_002`.

Proposer Construction Rules (r)	zebra_skill_031
Generation trigger	Use this pattern when a directed adjacency chain is anchored by a fixed entity and must be resolved through cross-attribute `same_house` or `next_to` constraints that link values across chain members.
Construction rule	Begin with a directed chain of at least three `directly_left_of` relations anchored at one end by `fixed`. Add non-redundant, non-symmetric `same_house` or `next_to` bridges that link a chain member to another value or a fixed house.
Offline oracle	Resolve positional offsets from the anchor; then apply `same_house` or `next_to` to determine value assignments across the chain. Propagate shared or adjacent values, and resolve conflicts using all-different and cross-attribute consistency.
Difficulty controls	Extend the chain to four values, add multiple bridges, or place bridges between non-consecutive members. Avoid symmetric or isolated constraints.

Proposer Hint and Package Validator (h, ν)	zebra_skill_031
Generator hint	Anchor a directed chain of length three or four with a fixed entity. Add a `same_house` or `next_to` bridge to a non-trivial attribute such as nationality or hobby, rather than only a name or drink. Avoid symmetric or isolated bridges.
Local validator	Require `directly_left_of`, `same_house`, and `next_to`, with minimum counts 2, 1, and 1, respectively. Every constraint must hold, the bridge logic must yield a unique solution, and at least one non-redundant `same_house` or `next_to` connection must link to a chain member.
Offline filtering oracle	(1) Resolve the directed chain from the anchor. (2) Apply `same_house` or `next_to` to determine value positions. (3) Propagate values through shared attributes. (4) Use adjacency around known values. (5) Apply all-different elimination. (6) Validate uniqueness by full constraint satisfaction.

Proposer Example (e) and Accepted Offline Instance	zebra_skill_031
Package example	Chain: Elara → Gita → Leo, with Leo fixed in House 3. Cross-attribute bridges connect Cooking to Mia, Mia to Reading through `same_house`, and Brazilian to Coffee through `next_to`. The resulting propagation resolves Mia, Reading, Brazilian, Coffee, and Tea without relying only on positional logic.
Accepted blueprint	Six houses; attributes `Name` = {Clara, Dale, Eva, Finn, Gia, Leo} and `Hobby` = {Reading, Cooking, Gardening, Photography, Painting, Writing}; difficulty `medium`; theme *Summer Reading Club*.
Compiled clues	(1) Dale directly left of Gia; (2) Gia directly left of Leo; (3) Leo in House 6; (4) Eva same house as Cooking; (5) Eva next to Gardening; (6) Finn directly left of Dale; (7) Reading next to Gia; (8) Gardening left of Painting; (9) Gardening directly left of Dale; (10) Leo same house as Photography.
Unique solution	House 1: Clara, Writing; House 2: Eva, Cooking; House 3: Finn, Gardening; House 4: Dale, Reading; House 5: Gia, Painting; House 6: Leo, Photography.
Structural checks	`logical_generation_mode=blueprint`; `prompt_valid=1`; `schema_valid=1`; `constraints_valid=1`; `puzzle_unique=1`; `solution_count_capped=1`.

Table 13 | **Complete induced package `zebra_skill_031`.** The five blocks instantiate $(m, r, h, e, \nu, \sigma)$: metadata, compiler resource, construction rules, hint and validation, and a package example with an accepted offline instance.