SKILLCERTAINTY: UNCERTAINTY QUANTIFICATION FOR SELF-EVOLVING AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents increasingly rely on reusable skills to solve complex tasks. However, existing skill-evolution methods typically use task-level rewards or uncalibrated self-reflection, providing limited guidance on whether a skill should be revised and which execution steps caused failure. We introduce SKILLCERTAINTY, an uncertainty-aware framework that uses step-level conformal prediction to identify behavior deviations during skill evolution. The framework constructs behavioral references from successful trajectories and applies position-stratified conformal calibration to accommodate heterogeneous, variable-length executions. Action-importance weights are learned to emphasize behaviorally informative actions and are fixed during online skill evolution. A trajectory-level score combines task performance with behavioral adherence: conforming executions support direct skill distillation, whereas nonconforming executions trigger targeted revision. Step-level diagnostics then localize deviations from successful behavior and condition the skill-generation prompt, grounding each revision in explicit behavioral evidence. When the conforming branch is reached, the distilled skill is evaluated in a separate fresh validation run. Experiments across diverse tasks and LLM backbones show competitive task performance, stronger behavioral adherence, and improved evolution efficiency over existing skill-evolution methods. These results establish conformal uncertainty as an actionable signal for self-evolving agent skills rather than merely a confidence estimate.

1 INTRODUCTION

Large Language Model (LLM) agents have demonstrated remarkable capabilities across a wide range of complex, multi-step tasks (Xu et al., 2026; Liu et al., 2024; Mialon et al., 2024). To improve efficiency and generalization performance, agent skills have been widely explored recently. These structured knowledge documents encode task-specific strategies, API patterns, and domain rules in a format that agents can load and apply at runtime (Zhao et al., 2024; Tan et al., 2024; Wang et al., 2024; Tan et al., 2026). When skills are well-designed, they significantly improve agent performance by providing detailed guidance for the whole workflow. However, a fundamental question remains unanswered: how does an agent know when its behavior is sufficiently aligned with a standard skill, and when should a skill be revised?

Existing skill generation and evolution methods (Yang et al., 2026; Ma et al., 2026; Zhou et al., 2026; Alzubi et al., 2026) improve skills through trajectory feedback, failure analysis, iterative refinement, or validation. While effective, there are two critical limitations. First, task-level outcomes and qualitative feedback may indicate that something went wrong, but not where in the trajectory the agent deviated from expected behavior, nor which specific behaviors were anomalous (Wang et al., 2023; Zhao et al., 2024; Wang et al., 2024; Lightman et al., 2024). Second, it is difficult to determine whether an observed deviation is significant or simply reflects normal variation among successful executions without calibrated behavioral references (Shorinwa et al., 2025). Together, these limitations suggest that effective skill evolution requires more than observing task-level improvement. It requires both identifying where behavior deviations occur and assessing whether they exceed the range of variation normally observed in successful executions.

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

Meeting these requirements calls for trajectory-aware uncertainty quantification that turns behavioral deviations into empirically calibrated behavioral evidence for skill revision. Conformal Prediction (CP) (Angelopoulos et al., 2024) provides a principled foundation for such calibration by offering distribution-free coverage guarantees under exchangeability. It has been applied to uncertainty calibration in classification, regression, and sequential decision-making (Angelopoulos & Bates, 2023; Tibshirani et al., 2019; Lekeufack et al., 2024). However, there are two challenges to applying CP to agent trajectories: (1) trajectories are variable-length sequences of heterogeneous actions; (2) the notion of "nonconformity" must consider both task outcome and behavior adherence with a skill reference.

In this paper, we introduce SKILLCERTAINTY, a new uncertainty-aware formulation of self-evolving agent skills. We formulate it as a calibrated behavior inference problem instead of document rewriting driven by terminal rewards: the system determines whether a new execution conforms to certified successful behaviors; when it does not, it identifies the specific decisions responsible for the deviation. This formulation turns uncertainty estimation from a passive confidence measure into an operational mechanism that decides where and how a skill evolves.

To realize this formulation, we develop a step-level conformal framework for evaluating agent behavior. SkillCertainty builds a position-aware behavioral reference from certified successful trajectories of varying lengths. A new trajectory is evaluated by these two aspects: trajectory-level conformity determines whether the agent enters the conforming or nonconforming branch, while step-level diagnostics identify where the agent deviates from successful behavior and use this information to guide skill generation. During Agent Trajectory Uncertainty Quantification (ATUQ), it constructs a global Skill Reference Vector, a position-stratified StepLibrary of conformal thresholds, and a learned action geometry that emphasizes failure-predictive behavior through contrastive training. During Conformal-Guided Self-Evolving Agent Skills Optimization (CSASO), conforming trajectories are distilled into skills and validated, whereas nonconforming trajectories are routed through Step-CP diagnostics that identify the most anomalous decisions and inject them into skill generation as empirically calibrated behavioral evidence. Our main contributions are:

- *The first uncertainty-aware formulation of self-evolving agent skills.* To the best of our knowledge, SKILLCERTAINTY is the first framework to cast skill evolution as calibrated behavioral inference, using calibrated deviations from successful reference behaviors to determine when a skill should be revised and to guide its evolution.
- *A unified step-level Conformal Prediction framework for targeted skill evolution.* We combine position-aware calibration and learned action geometry to capture local action conformity and global skill adherence in heterogeneous, variable-length trajectories. Calibrated step-level diagnostics localize anomalous decisions and directly guide skill generation, establishing a closed uncertainty-to-evolution loop.
- *A comprehensive evaluation of uncertainty-aware skill evolution.* Experiments across diverse tasks and LLM agents demonstrate gains in task performance, behavioral conformity, and evolution efficiency over skill-generation baselines, complemented by empirical calibration analysis, ablation studies, and cross-model evaluation.

2 RELATED WORK

Skill construction and skill evolution. Equipping LLM agents with reusable skill libraries has emerged as a promising approach for improving performance and generalization. Voyager (Wang et al., 2023) maintains an expanding library of executable skills for open-ended embodied exploration, while GITM (Zhu et al., 2023) uses structured knowledge and reusable procedures for long-horizon interaction. More recent work has focused on automatically constructing and refining skills from agent experience. SkillGen (Ma et al., 2026) performs contrastive skill induction from both successful and failed trajectories and empirically verifies generated skills through subsequent executions. SkillOpt (Yang et al., 2026) treats skill refinement as text-space optimization, deriving bounded edits from scored rollouts and accepting updates based on held-out validation. Memento-Skills (Zhou et al., 2026) continually constructs and updates externalized skills through memory-based reflective learning. EvoSkill (Alzubi et al., 2026) iteratively analyzes execution failures, proposes skill revisions, and retains skills according to held-out performance. These methods demonstrate that trajectory feedback, failure analysis, and validation can substantially improve skill

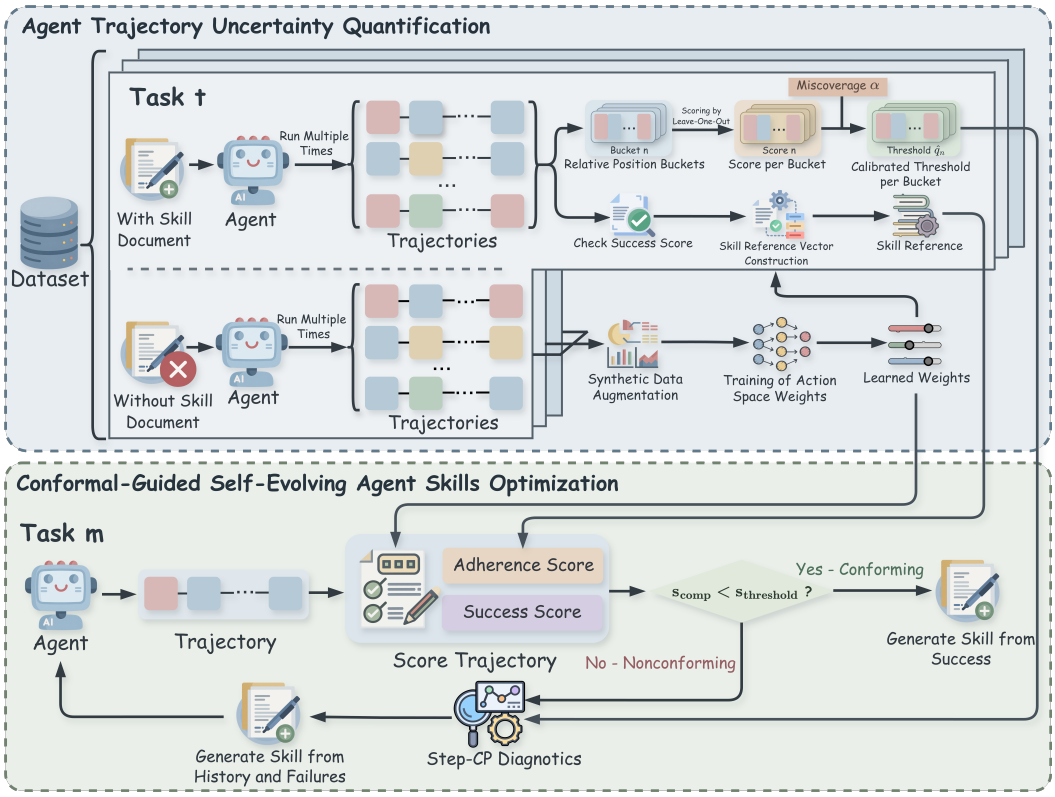


Figure 1: Overview of SkillCertainty. The framework comprises two phases: (top) Agent Trajectory Uncertainty Quantification, which builds a Skill Reference Vector, per-bucket conformal thresholds, and learned action importance weights; and (bottom) Conformal-Guided Self-Evolving Agent Skills Optimization, which iteratively scores new trajectories, routes conforming runs to direct skill distillation, and routes nonconforming runs through Step-CP Diagnostics before skill generation.

quality. SkillCertainty addresses a complementary question: how to quantify whether a new agent trajectory behaviorally conforms to certified successful executions and how to localize statistically unusual decisions. We therefore calibrate position-aware step-level behavioral deviations against successful reference trajectories and use these diagnostics directly to guide skill evolution.

Conformal prediction for LLM responses. CP (Angelopoulos et al., 2024) has attracted growing attention as a principled framework for uncertainty quantification in LLM outputs, offering distribution-free coverage guarantees without relying on internal model probabilities. Existing work applies CP at the token level (Kotla & Kotla, 2025), claim level (Cherian et al., 2024), and full sequence level (Karimi et al., 2026; Su et al., 2024; Noorani et al., 2026), with nonconformity scores derived from logit-based confidence (Kumar et al., 2023; Vishwakarma et al., 2024), sampling frequency (Su et al., 2024; Kaur et al., 2024), semantic embedding distance (Pang et al., 2025; Mohri & Hashimoto, 2024). Coverage guarantees range from static marginal (Su et al., 2024; Karimi et al., 2026) to dynamic conditional (Cherian et al., 2024) and domain-shift-aware formulations (Lin et al., 2025). However, no existing work addresses how CP applies to agent skills evolution. SkillCertainty fills this gap by introducing step-level Mondrian CP for agent trajectory evaluation, applying position-stratified calibration over action sequences and using the resulting conformal diagnostics to drive targeted skill generation.

3 Preliminaries

Agent Trajectories. Let $\mathcal{T}$ denote a set of tasks. For each task $t \in \mathcal{T}$, an LLM agent executes a trajectory $\tau = (s_1, s_2, \ldots, s_n)$, where each step s_i is a tool call characterized by an action kind

$k_i \in \mathcal{K}$ and an execution status $\sigma_i \in \{\texttt{completed}, \texttt{failed}\}$. We define the action vocabulary as $\mathcal{V} = \mathcal{K} \times \{\texttt{completed}, \texttt{failed}\}$. Each task t is associated with a grading score $\text{GS}(\tau) \in [0, 1]$, where $\text{GS} = 1.0$ denotes full success and $\text{GS} = 0.0$ denotes failure.

Conformal Prediction. Conformal Prediction (CP) (Angelopoulos et al., 2024) is a distribution-free framework for constructing prediction sets with finite-sample coverage guarantees. Given a calibration set $\mathcal{D}_{\text{cal}} = \{z_1, \ldots, z_m\}$ of exchangeable samples and a nonconformity score function $A : \mathcal{Z} \to \mathbb{R}$, the conformal quantile at miscoverage level α is:

$$\hat{q} = \text{Quantile}\left(\{A(z_i)\}_{i=1}^{m};\ \frac{\lceil (m+1)(1-\alpha) \rceil}{m}\right) \tag{1}$$

For any test point z_{test} drawn exchangeably from the same distribution:

$$\Pr[A(z_{\text{test}}) \leq \hat{q}] \geq 1 - \alpha \tag{2}$$

This marginal coverage guarantee requires only exchangeability. Mondrian CP (Vovk et al., 2005) partitions the sample space into strata $\mathcal{B} = \{B_1, \ldots, B_K\}$ and calibrates a separate threshold $\hat{q}_k$ for each, preserving conditional coverage within each stratum.

SkillCertainty adopts this quantile-calibration principle in a trajectory-structured setting. However, because multiple steps from the same trajectory can be dependent, standard pointwise conformal coverage does not directly follow without additional assumptions. We therefore distinguish the trajectory-level exchangeability assumption used in our calibration procedure from exchangeability among individual steps.

4 SkillCertainty

SkillCertainty is a two-phase framework for uncertainty-aware, self-evolving skill generation in LLM agents, illustrated in Figure 1. ATUQ builds a calibrated reference model of certified successful behavior through offline calibration; CSASO uses that reference for skill evolution at test time. The complete algorithm for the end-to-end workflow is provided in Algorithm 2 in Appendix A, and theoretical analyses are provided in Appendix B.

4.1 Agent Trajectory Uncertainty Quantification

The target of ATUQ is to construct three artifacts from a collection of successful reference trajectories together with contrastive failed executions: a StepLibrary of per-bucket conformal thresholds, a Skill Reference Vector encoding the behavioral signature of certified executions, and learned action importance weights that amplify failure-predictive signals in the embedding space. Then, CSASO uses these artifacts to evaluate any new trajectory against the certified reference.

Data collection and reference set construction. For each task $t \in \mathcal{T}$, ATUQ collects two sets of N_{cal} executions using the SkillsBench evaluation harness. With-skill executions $\mathcal{D}_t^+ = \{\tau_1, \ldots, \tau_{N_{\text{cal}}}\}$ are generated using an existing skill document, whereas without-skill executions $\mathcal{D}_t^- = \{\tau'_1, \ldots, \tau'_{N_{\text{cal}}}\}$ are generated without skill guidance. The successful reference set is $\mathcal{D}_t^* = \{\tau \in \mathcal{D}_t^+ : \text{GS}(\tau) = 1.0\}$, which is used to construct the StepLibrary and Skill Reference Vector and serves as the positive trajectory set for action-weight learning. For contrastive weight learning, we define the failed without-skill set as $\mathcal{D}_t^{\text{fail}} = \{\tau \in \mathcal{D}_t^- : \text{GS}(\tau) = 0\}$.

StepLibrary: position-stratified conformal calibration. For each relative position in a trajectory, the StepLibrary $\mathcal{L}_t$ encodes the distribution of action tokens observed in certified successful runs and a calibrated threshold for flagging anomalous steps. Each trajectory is partitioned into B relative-position buckets: step i in a trajectory of length n maps to bucket $b(i, n) = \min(\lfloor i \cdot B/n \rfloor, B - 1)$, making calibration position-invariant to trajectory length variation. For each bucket b, action-frequency counts $\text{freq}(v, b)$ are accumulated from $\mathcal{D}_t^*$.

Calibration nonconformity scores are computed via leave-one-out (LOO) for tasks with $|\mathcal{D}_t^*| > 1$. For sparse-reference cases, a single successful trajectory is scored in-sample with a minimum threshold of 0.25, while tasks with no successful reference trajectories use neutral Step-CP diagnostics.

Algorithm 1 Step-CP Diagnostics (`identify_failing_steps`)

Require: trajectory $\tau^{(\ell)}$, task t, StepLibrary $\mathcal{L}_t$, margin threshold γ, top-K

1: Assign each step s_i to its position bucket $b(i, n)$.
2: Compute step nonconformity $A(s_i)$ against the bucket frequency distribution.
3: Retrieve the calibrated threshold $\hat{q}_{b(i,n)}$ from $\mathcal{L}_t$.
4: Retain steps with significant exceedance: $\mathcal{F} = \left\{ s_i : A(s_i) > \hat{q}_{b(i,n)},\ A(s_i) - \hat{q}_{b(i,n)} > \gamma \right\}$
5: Sort $\mathcal{F}$: failed-status steps first, then by $A(s_i)$ descending.
6: Deduplicate by action key (k_i, σ_i), retaining the highest nonconformity per key.
7: **return** top-K steps with their indices, action keys, $A(s_i)$, $\hat{q}_b$, margin, step titles, and truncated output summaries.

For each step s_i in reference run τ_j^*, the contribution of τ_j^* is temporarily removed from the frequency table before evaluating:

$$A(s_i) = \underbrace{1 - \frac{\text{freq}_{-ij}(a_i, b)}{\text{total}_{-ij}(b)}}_{\text{rarity at position}} + \underbrace{\beta \cdot \mathbf{1}[\sigma_i = \texttt{failed}]}_{\text{failure penalty}} \tag{3}$$

capped at 1.0, where $\beta > 0$ controls the penalty for failed-status steps. LOO prevents a reference trajectory from directly contributing to the frequency table used to score its own steps. Each reference trajectory is therefore evaluated against behavioral frequencies constructed from the remaining successful trajectories. The resulting pooled LOO scores within each relative-position bucket are then used to set an empirical calibration threshold at level α:

$$\hat{q}_b = \max\left(\text{Quantile}\left(\{A(s_i) : b(i, n_j) = b,\ \tau_j^* \in \mathcal{D}_t^*\};\ \frac{\lceil (|\mathcal{S}_b| + 1)(1 - \alpha) \rceil}{|\mathcal{S}_b|} \right),\ \eta \right) \tag{4}$$

where the floor $\eta > 0$ prevents degenerate near-zero thresholds under sparse calibration. The full StepLibrary $\mathcal{L}_t = \{(b, \hat{q}_b)\}_{b=0}^{B-1}$ is held fixed throughout CSASO.

Action importance weight training: discriminative embeddings via contrastive learning. The CSASO adherence score uses the weighted action embedding $\text{embed}_{\mathbf{w}}(\tau)$. Since different action tokens carry different behavioral information, we first learn $\mathbf{w} \in \mathbb{R}_{>0}^{|\mathcal{V}|}$ directly from successful and failed trajectory action sequences. The Skill Reference Vector $\boldsymbol{\tau}_{\text{skill}}$ is not used during weight training; it is constructed only after $\mathbf{w}$ has been learned and frozen. The embedding is

$$\text{embed}_{\mathbf{w}}(\tau) = \frac{\mathbf{v}_{\mathbf{w}}}{\|\mathbf{v}_{\mathbf{w}}\|_2}, \qquad [\mathbf{v}_{\mathbf{w}}]_j = w_j \cdot \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[a_i = v_j]. \tag{5}$$

Let $\mathcal{P}$ denote positive pairs of successful abstract action sequences and $\mathcal{N}$ denote contrastive pairs containing successful and failed action sequences. The contrastive objective $\mathcal{L} = \mathcal{L}_{\text{pos}} + \mathcal{L}_{\text{neg}} + \mathcal{L}_{\text{reg}}$ is defined as

$$\mathcal{L}_{\text{pos}} = \frac{1}{|\mathcal{P}|} \sum_{(\tau_a, \tau_b) \in \mathcal{P}} \left[1 - \cos(\text{embed}_{\mathbf{w}}(\tau_a), \text{embed}_{\mathbf{w}}(\tau_b)) \right], \tag{6}$$

$$\mathcal{L}_{\text{neg}} = \frac{1}{|\mathcal{N}|} \sum_{(\tau_a, \tau_b) \in \mathcal{N}} \max(0,\ \cos(\text{embed}_{\mathbf{w}}(\tau_a), \text{embed}_{\mathbf{w}}(\tau_b)) - m), \tag{7}$$

$$\mathcal{L}_{\text{reg}} = \mu \sum_{v_j \in \mathcal{V}} (w_j - w_{0,j})^2. \tag{8}$$

Here, m is the similarity margin, μ is the regularization strength, and $\mathbf{w}_0 = \mathbf{1}$ is the uniform initialization. Positivity is enforced by $w_j = \log(1 + \exp(\tilde{w}_j))$. After optimization, $\mathbf{w}$ is frozen and $\boldsymbol{\tau}_{\text{skill}}$ is subsequently constructed from the weighted embeddings of certified successful trajectories.

Skill Reference Vector: behavioral centroid of certified executions. After the action weights $\mathbf{w}$ are learned and frozen, the Skill Reference Vector $\boldsymbol{\tau}_{\text{skill}}$ is constructed from the unit-normalized weighted embeddings of certified successful trajectories. When $|\mathcal{D}_t^*| \geq 2$, we compute

$$\bar{\mathbf{e}} = \frac{1}{|\mathcal{D}_t^*|} \sum_{\tau \in \mathcal{D}_t^*} \text{embed}_{\mathbf{w}}(\tau), \qquad \boldsymbol{\tau}_{\text{skill}} = \frac{\bar{\mathbf{e}}}{\|\bar{\mathbf{e}}\|_2}. \tag{9}$$

When $|\mathcal{D}_t^*| < 2$, a fallback reconstructs an approximate action sequence by parsing the task's `SKILL.md` front-matter and matching the Core Workflow section against a keyword-to-action mapping. $\boldsymbol{\tau}_{\text{skill}}$ serves as the adherence target in CSASO's composite nonconformity score.

4.2 Conformal-Guided Self-Evolving Agent Skills Optimization

With the StepLibrary, Skill Reference Vector, and learned weights fixed from ATUQ, CSASO runs an online self-evolution loop. At each iteration ℓ, the agent executes task t to produce trajectory $\tau^{(\ell)}$, which is scored against the calibrated reference and routed to one of two branches: a conforming trajectory is distilled directly into a skill, while a nonconforming trajectory triggers Step-CP diagnostics that condition skill generation on specific failure steps.

Composite nonconformity scoring and routing. The trajectory $\tau^{(\ell)}$ is embedded using the learned weights from ATUQ, and three nonconformity scores are computed (Figure 6, Appendix C):

$$\text{embed}(\tau^{(\ell)}) = \frac{\mathbf{v}^{(\ell)}}{\|\mathbf{v}^{(\ell)}\|_2}, \qquad v_j^{(\ell)} = w(v_j) \cdot \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[a_i = v_j] \tag{10}$$

$$s_{\text{out}}^{(\ell)} = 1 - \text{GS}(\tau^{(\ell)}), \quad s_{\text{adh}}^{(\ell)} = 1 - \cos\Big(\text{embed}(\tau^{(\ell)}),\ \boldsymbol{\tau}_{\text{skill}}\Big), \quad s_{\text{comp}}^{(\ell)} = \lambda \cdot s_{\text{out}}^{(\ell)} + (1-\lambda) \cdot s_{\text{adh}}^{(\ell)} \tag{11}$$

where λ balances task outcome quality and behavioral adherence. The composite score $s_{\text{comp}}^{(\ell)}$ therefore considers both whether the agent succeeds and whether its behavior is consistent with certified successful trajectories. A trajectory may achieve partial success while still following an unusual action sequence, so it should still be diagnosed. If $s_{\text{comp}}^{(\ell)} \leq \delta$, the trajectory is treated as *conforming* and sent to the success distillation branch; otherwise, it is treated as *nonconforming* and sent to the Step-CP diagnostic branch.

Nonconforming branch: Step-CP diagnostics and targeted skill generation. When $s_{\text{comp}}^{(\ell)} > \delta$, SkillCertainty applies Step-CP Diagnostics (Algorithm 1) to identify the steps that deviate most from the calibrated reference. Each step s_i is compared with the frequency distribution of its corresponding bucket. Steps whose scores exceed the threshold $\hat{q}_{b(i,n)}$ by more than γ are ranked by severity, deduplicated by action key, and the top-K are returned with their indices, scores, margins, and truncated outputs. These diagnostics are then inserted directly into the skill generation prompt (Appendix D), so the new skill is guided by localized, position-aware evidence rather than only the raw trajectory. The resulting `SKILL.md` is used by the agent at iteration $\ell + 1$.

Conforming branch: skill distillation and validation. If $s_{\text{comp}}^{(\ell)} \leq \delta$, the trajectory satisfies the composite conformity criterion and is distilled, together with the task instruction, into a reusable candidate skill $S^\star$. Because $S^\star$ is generated only after $\tau^{(\ell)}$ has been observed, the score of $\tau^{(\ell)}$ does not directly evaluate the newly distilled skill. SkillCertainty therefore executes one fresh validation run with $S^\star$ enabled and reports the resulting trajectory and verifier score. The loop terminates after this validation run regardless of its outcome.

5 Experiments

5.1 Settings

Dataset. We evaluate SkillCertainty on SkillsBench (Li et al., 2026), a benchmark of diverse real-world agent tasks covering scientific computing, data analysis, document processing, financial modeling, and engineering control. Each task is graded by an automated verifier producing a granular score GS $\in [0, 1]$. We report results across 10 tasks for the multi-model comparison and 13 tasks for the per-model per-task analysis.

Agent and Models. We evaluate across six LLM backends: Claude Haiku-4.5, Gemini 3.6 Flash, GPT-5.5, GPT-5.1, GPT-5.1 Codex Mini, and GPT-4.1. ATUQ collects $N_{\text{cal}} = 10$ runs per condition (with-skill and without-skill). CSASO terminates when the trajectory enters the conforming branch or when the specified update budget T is reached. We use a default routing threshold $\delta = 0.25$ and CP miscoverage level $\alpha = 0.10$. Additional implementation details and hyperparameter settings are provided in Appendix E.

Table 1: Multi-model comparison on SkillsBench. ΔGS: improvement in granular score; GS_f: final granular score; s_{adh}: adherence nonconformity (↓); s_{comp}: composite nonconformity (↓); n_{iters}: mean evolution iterations; Conv: fraction of tasks reaching the conforming branch; Mono: fraction of tasks with non-decreasing GS; ΔGS/iter: gain per evolution iteration.

Agent	Method	ΔGS	GS_f	s_{adh}	s_{comp}	n_{iters}	Conv	Mono	ΔGS/iter
Claude Haiku-4.5	**Ours**	**0.488**	**0.800**	**0.224**	**0.213**	**1.2**	**0.9**	**0.9**	**0.388**
	SkillOpt	0.329	0.641	0.375	0.367	1.0	—	0.7	0.329
	SkillGen	0.476	0.787	0.485	0.349	1.0	—	0.9	0.476
	Memento	0.452	0.764	0.493	0.365	1.0	—	0.8	0.452
	EvoSkill	0.289	0.600	0.306	0.353	2.6	—	0.2	0.080
Gemini 3.6 Flash	**Ours**	**0.098**	**0.931**	**0.074**	**0.072**	**0.8**	**1.0**	**1.0**	**0.123**
	SkillOpt	0.065	0.898	0.759	0.431	1.0	—	1.0	0.065
	SkillGen	0.083	0.916	0.739	0.412	1.0	—	1.0	0.083
	Memento	0.075	0.908	0.728	0.410	1.0	—	1.0	0.075
	EvoSkill	0.110	0.943	0.736	0.397	3.0	—	1.0	0.037
GPT-5.5	**Ours**	**0.195**	**0.923**	**0.176**	**0.127**	**0.8**	**0.8**	**0.9**	**0.198**
	SkillOpt	0.164	0.891	0.786	0.447	1.0	—	0.8	0.164
	SkillGen	0.141	0.868	0.728	0.430	1.0	—	0.8	0.141
	Memento	0.182	0.910	0.713	0.402	1.0	—	0.9	0.182
	EvoSkill	0.220	0.948	0.749	0.441	3.0	—	0.8	0.073
GPT-5.1	**Ours**	**0.200**	**0.927**	**0.188**	**0.130**	**0.8**	**0.8**	**0.9**	**0.200**
	SkillOpt	0.126	0.854	0.746	0.446	1.0	—	0.8	0.126
	SkillGen	0.182	0.910	0.713	0.402	1.0	—	0.9	0.182
	Memento	0.144	0.871	0.704	0.416	1.0	—	0.8	0.144
	EvoSkill	0.200	0.927	0.706	0.353	3.0	—	0.7	0.067
GPT-5.1 Codex Mini	**Ours**	**0.200**	**0.927**	**0.170**	**0.121**	**0.8**	**0.8**	**0.9**	**0.200**
	SkillOpt	0.065	0.792	0.823	0.515	1.0	—	0.8	0.065
	SkillGen	0.122	0.850	0.849	0.500	1.0	—	0.8	0.122
	Memento	0.139	0.867	0.729	0.431	1.0	—	0.8	0.139
	EvoSkill	0.200	0.927	0.710	0.355	3.0	—	0.7	0.067
GPT-4.1	**Ours**	**0.200**	**0.927**	**0.180**	**0.126**	**0.8**	**0.8**	**1.0**	**0.200**
	SkillOpt	0.155	0.882	0.802	0.460	1.0	—	0.9	0.155
	SkillGen	0.182	0.910	0.765	0.428	1.0	—	0.9	0.182
	Memento	0.055	0.782	0.730	0.474	1.0	—	0.8	0.055
	EvoSkill	0.200	0.927	0.724	0.362	3.0	—	0.6	0.067

Baselines. We compare against four skill generation baselines: SkillOpt (Yang et al., 2026), SkillGen (Ma et al., 2026), Memento (Zhou et al., 2026), and EvoSkill (Alzubi et al., 2026). Detailed descriptions are in Appendix E.2.

Metrics. We report: ΔGS (improvement in granular score from baseline), GS_f (final granular score), s_{adh} (skill adherence nonconformity), s_{comp} (composite nonconformity), n_{iters} (mean iterations to termination), Conv (fraction of tasks converging to the conforming branch), Mono (fraction of tasks with non-decreasing GS across iterations), and ΔGS/iter (per-iteration efficiency).

5.2 Main Results

Table 1 reports the multi-model comparison across all methods. SkillCertainty achieves competitive final task performance across all six LLM backends, obtaining the highest or tied-highest GS_f on four of the six backends.

Performance gains. On Claude Haiku-4.5, SkillCertainty achieves $GS_f = 0.800$ with ΔGS$= 0.488$, outperforming the strongest baseline in final score. On GPT-5.1, GPT-5.1 Codex Mini, and GPT-4.1, SkillCertainty matches the highest GS_f of 0.927. EvoSkill achieves slightly higher final scores on Gemini 3.6 Flash (0.943 vs. 0.931) and GPT-5.5 (0.948 vs. 0.923). Overall, SkillCertainty maintains competitive task performance while requiring fewer evolution iterations.

Conformity improvement. SkillCertainty achieves the lowest s_{adh} and s_{comp} across all six tested backends, indicating that its resulting trajectories are more closely aligned with the certified behavioral reference. For example, on Claude Haiku-4.5, SkillCertainty reaches $s_{adh} = 0.224$, compared with 0.485 for SkillGen, corresponding to a 54% reduction in adherence nonconformity.

Table 2: Ablation study on action weight learning. ΔGS = GS change after skill evolution (negative = degradation). **Bold** indicates best per task.

Task	Learned	Uniform
Shock Analysis (Demand)	**+0.200**	+0.000
Shock Analysis (Supply)	**+0.000**	-0.333
Lab Unit Harmonization	**+0.042**	+0.021
Mean	**+0.081**	-0.104

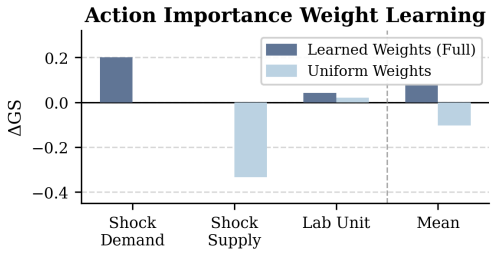


Figure 2: Action Importance Weight Learning ablation.

Evolution-update efficiency. SkillCertainty terminates in 0.8–1.2 iterations on average, compared with 2.6–3.0 for EvoSkill. It achieves the highest ΔGS/iter on five of the six backends. On Claude Haiku-4.5, SkillCertainty achieves 0.388 improvement per iteration, which is 4.9× that of EvoSkill (0.080), although SkillGen achieves a higher per-iteration gain on this backend (0.476). These results show that SkillCertainty generally achieves strong gains with fewer skill-evolution updates.

Cross-model consistency. Across all six backends, SkillCertainty consistently maintains low adherence nonconformity and requires relatively few evolution iterations, suggesting that its behavioral calibration and routing mechanisms are effective across the tested model families.

Table 3: Ablation study on Step-CP diagnostic injection. ΔGS = granular score improvement after one self-evolving skill loop. **Bold** indicates best per task.

Task	Full	No Diagnostics
Crystallographic Wyckoff	**+1.000**	+0.640
Gravitational Wave Det.	**+0.111**	+0.000
Lab Unit Harmonization	**+0.042**	−0.021
Mean	**+0.384**	+0.206

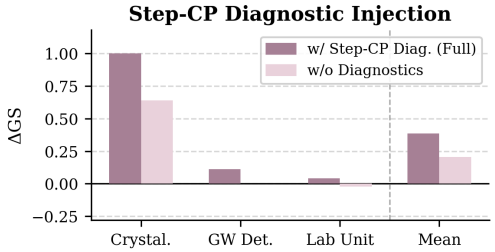


Figure 3: Step-CP Diagnostic Injection ablation.

5.3 Per-Task Analysis

Table 5 (Appendix E.4) reports per-task results for Claude Haiku-4.5. SkillCertainty improves over the repository skills baseline on all 13 tasks, with gains ranging from +0.050 (Mars Clouds Clustering) to +0.900 (Energy Market Pricing). The largest improvements occur on tasks where repository skills provide incomplete or misspecified guidance, with HVAC Control (0.200 → 1.000) analyzed in detail in Figure 9. Notably, SkillCertainty also improves on tasks where the baseline is already strong (e.g., SEC Financial Report, 0.900 → 1.000), confirming that Step-CP diagnostics are beneficial even when the agent is partially successful. Per-task discussion is in Appendix E.4.

5.4 Qualitative Analysis

Figure 9 (Appendix F.1) illustrates the core mechanism of SkillCertainty on the HVAC control task. The five repository skills collectively omit the simulator's fixed time step (`dt = 0.5 s`), the tighter IMC parameter required by the overshoot constraint ($\lambda = \tau/2$), a complete anti-windup implementation, and the exact output filenames required by the evaluator. After one nonconforming iteration, Step-CP Diagnostics pinpoints the failing steps and injects targeted diagnostics into the skill generation prompt; the resulting unified `SKILL.md` encodes all four gaps and the agent achieves GS = 1.0. This illustrates how Step-CP localizes *why* steps fail, enabling evidence-grounded skill generation that generic summarization cannot provide. Appendix F provides analogous analyses for four additional tasks (Figures 10–13).

Table 4: Ablation 3: Effect of fresh candidate-skill validation. "Detection" reports the ΔGS at the conforming detection point, before the distilled skill is executed. "Validation" reports ΔGS from a fresh execution using the newly distilled candidate skill. The comparison measures the discrepancy between the triggering trajectory and the behavior of the generated skill.

Task	Detection ΔGS	Fresh Validation ΔGS
Crystallographic Wyckoff Position Analysis	**+1.000**	+0.550
Edit PDF	**+0.000**	**+0.000**
Energy Market Pricing	**+1.000**	+0.000
Exoplanet Detection Period	**+0.000**	**+0.000**
Financial Modeling QA	**+0.000**	**+0.000**
Gravitational Wave Detection	**+0.111**	**+0.111**
Lab Unit Harmonization	-0.062	**+0.000**
Mean	$+0.293$	$+0.094$

5.5 Sensitivity Analysis

Sensitivity analysis shows that the routing threshold δ has a substantial effect on evolution behavior, whereas the calibration level α is relatively stable across the tested settings. On the crystallographic-wyckoff-position-analysis task, the default $\delta = 0.25$ avoids unnecessary evolution for trajectories that already satisfy the conformity criterion, while smaller values trigger additional skill updates. Across the tested α values, the empirical step acceptance rate remains above 0.78. Full results are reported in Appendix E.5.

5.6 Ablation Studies

Step-CP Diagnostic Injection. Table 3 and Figure 3 compare Full (step-level CP diagnostics injected) against No Diagnostics (raw trajectory only). Full outperforms No Diagnostics by 86% in mean ΔGS ($+0.384$ vs. $+0.206$), with the largest gaps on tasks where a single misaligned step prevents completion — No Diagnostics even causes active degradation on Lab Unit Harmonization (-0.021) while Full achieves $+0.042$. Detailed per-task analysis is in Appendix E.6.

Effect of Learned Action Importance Weights. On Shock Analysis (Demand), learned weights provide a more discriminative trajectory-level adherence representation, leading to $\Delta\mathrm{GS} = +0.200$, whereas uniform weights yield no improvement. On Shock Analysis (Supply), uniform weights degrade performance from 0.333 to 0.000, while learned weights preserve the baseline. On Lab Unit Harmonization, learned weights also yield a larger improvement ($+0.042$ vs. $+0.021$). These results indicate that learned action weights improve trajectory-level adherence modeling and composite routing rather than directly modifying Step-CP nonconformity scores.

Conforming Branch Validation. Table 4 compares the score observed at conforming detection with a fresh execution using the newly distilled candidate skill. The mean ΔGS decreases from $+0.293$ at detection to $+0.094$ on the fresh validation execution, showing that the triggering trajectory can be an optimistic proxy for the behavior of the generated skill. The largest discrepancy occurs on Energy Market Pricing, where the detection trajectory reaches $\mathrm{GS} = 1.000$ but the newly distilled skill obtains $\mathrm{GS} = 0.000$ on the fresh validation run. This result motivates evaluating the generated skill directly rather than inferring its quality from the trajectory used for distillation. Because each candidate is evaluated only once, we do not infer its underlying success probability from the validation outcome.

6 Conclusion

We presented SkillCertainty, a step-level Conformal Prediction framework for self-evolving agent skills. By calibrating per-bucket thresholds from certified successful trajectories and injecting localized diagnostics into skill generation, SkillCertainty moves beyond coarse task-level feedback. Across six LLM backends, it achieves competitive final task performance, consistently low adherence nonconformity, and strong evolution-update efficiency. The results highlight conformal prediction as a principled and practical foundation for uncertainty-aware skill evolution.

AI USE STATEMENT

Generative AI tools were used solely to improve writing clarity, grammar, and presentation. All research ideas, methods, experiments, analyses, and conclusions were developed and verified by the authors. All AI-assisted outputs were reviewed and verified by the authors, who take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the complete SkillCertainty workflow, including the ATUQ and CSASO procedures, in Algorithm 2. The paper specifies the nonconformity scores, calibration procedure, routing criterion, action-weight learning objective, and major hyperparameters used in our experiments. We additionally provide the skill-generation prompt template, baseline descriptions, sensitivity analyses, and per-task qualitative examples in the appendix. Experiments are conducted using the SkillsBench evaluation harness with automated granular-score verifiers, and the model backends, calibration settings, and evaluation protocols are reported in Section 5 and Appendix E. We plan to publicly release the implementation code and experimental configurations upon publication.

REFERENCES

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

Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023.

Anastasios N Angelopoulos, Rina Foygel Barber, and Stephen Bates. Theoretical foundations of conformal prediction. *arXiv preprint arXiv:2411.11824*, 2024.

John J Cherian, Isaac Gibbs, and Emmanuel J Candès. Large language model validity via enhanced conformal prediction methods. *Advances in Neural Information Processing Systems*, 37:114812–114842, 2024.

Hamed Karimi, Vaishali Meyappan, and Reza Samavi. Llms uncertainty quantification via adaptive conformal semantic entropy. *arXiv preprint arXiv:2605.04295*, 2026.

Ramneet Kaur, Colin Samplawski, Adam D Cobb, Anirban Roy, Brian Matejek, Manoj Acharya, Daniel Elenius, Alexander M Berenbeim, John A Pavlik, Nathaniel D Bastian, et al. Addressing uncertainty in llms to enhance reliability in generative ai. *arXiv preprint arXiv:2411.02381*, 2024.

Yoshith Roy Kotla and Varshith Roy Kotla. Conformal prediction sets for next-token prediction in large language models: Balancing coverage guarantees with set efficiency. *arXiv preprint arXiv:2512.22682*, 2025.

Bhawesh Kumar, Charlie Lu, Gauri Gupta, Anil Palepu, David Bellamy, Ramesh Raskar, and Andrew Beam. Conformal prediction with large language models for multi-choice question answering. *arXiv preprint arXiv:2305.18404*, 2023.

Jordan Lekeufack, Anastasios N Angelopoulos, Andrea Bajcsy, Michael I Jordan, and Jitendra Malik. Conformal decision theory: Safe autonomous decisions from imperfect predictions. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11668–11675. IEEE, 2024.

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

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Zhexiao Lin, Yuanyuan Li, Neeraj Sarna, Yuanyuan Gao, and Michael von Gablenz. Domain-shift-aware conformal prediction for large language models. *arXiv preprint arXiv:2510.05566*, 2025.

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.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Christopher Mohri and Tatsunori Hashimoto. Language models with conformal factuality guarantees. *arXiv preprint arXiv:2402.10978*, 2024.

Sima Noorani, Shayan Kiyani, George J Pappas, and Hamed Hassani. Conformal prediction beyond the seen: A missing mass perspective for uncertainty quantification in generative models. *Advances in Neural Information Processing Systems*, 38:32318–32351, 2026.

Lingyou Pang, Lei Huang, Jianyu Lin, Tianyu Wang, Akira Horiguchi, Alexander Aue, and Carey E Priebe. Unsupervised conformal inference: Bootstrapping and alignment to control llm uncertainty. *arXiv preprint arXiv:2509.23002*, 2025.

Ola Shorinwa, Zhiting Mei, Justin Lidard, Allen Z Ren, and Anirudha Majumdar. A survey on uncertainty quantification of large language models: Taxonomy, open research challenges, and future directions. *ACM Computing Surveys*, 58(3):1–38, 2025.

Jiayuan Su, Jing Luo, Hongwei Wang, and Lu Cheng. API is enough: Conformal prediction for large language models without logit-access. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 979–995, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.54.

Shangyin Tan, Lakshya A Agrawal, Rohit Sandadi, Dan Klein, Koushik Sen, Alexandros G Dimakis, and Matei Zaharia. Automatically learning skills for coding agents. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 1310–1315, 2026.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. Conformal prediction under covariate shift. *Advances in neural information processing systems*, 32, 2019.

Harit Vishwakarma, Alan Mishler, Thomas Cook, Niccolo Dalmasso, Natraj Raman, and Sumitra Ganesh. Prune'n predict: Optimizing llm decision-making with conformal prediction. *arXiv preprint arXiv:2501.00555*, 2024.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer, 2005.

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

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38, 2026.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-Skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the Minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

A COMPLETE SKILLCERTAINTY ALGORITHM

Algorithm 2 presents the complete SkillCertainty procedure, integrating the Agent Trajectory Uncertainty Quantification (ATUQ) with the Conformal-Guided Self-Evolving Agent Skills Optimization (CSASO). ATUQ constructs the calibrated behavioral reference from certified successful trajectories, while CSASO uses this reference to evaluate new executions, localize anomalous steps, and guide iterative skill generation.

Algorithm 2 End-to-end Workflow of SkillCertainty

Require: Task t with instruction x_t, agent $\mathcal{A}$, verifier GS;
reference skill S_{ref}, initial online skill $S^{(0)}$;
$N_{\mathrm{cal}}, B, \alpha, \beta, \eta, \lambda, \delta, \gamma, K, m, \mu$, update budget T.
Ensure: Returned skill, its last execution trajectory, and its verifier score.

Phase I: Agent Trajectory Uncertainty Quantification

Collect and grade N_{cal} executions with S_{ref} to form $\mathcal{D}_t^+$.
Collect and grade N_{cal} executions without skills to form $\mathcal{D}_t^-$.
$\mathcal{D}_t^* \leftarrow \{\tau \in \mathcal{D}_t^+ : \mathrm{GS}(\tau) = 1\}$.
$\mathcal{D}_t^{\mathrm{fail}} \leftarrow \{\tau \in \mathcal{D}_t^- : \mathrm{GS}(\tau) = 0\}$.
Assign step i in a length-n trajectory to $b(i,n) = \min(\lfloor iB/n \rfloor, B-1)$.
Build bucket-wise action-token frequencies F_t from $\mathcal{D}_t^*$.
for $b = 0, \ldots, B-1$ **do**
  $\mathcal{C}_b \leftarrow \textsc{ReferenceStepScores}(\mathcal{D}_t^*, b, B, \beta)$.
  $\widehat{q}_b \leftarrow \max\left\{ \mathrm{Quantile}\left(\mathcal{C}_b; \frac{\lceil (|\mathcal{C}_b|+1)(1-\alpha) \rceil}{|\mathcal{C}_b|} \right), \eta \right\}$.
end for
$\mathcal{L}_t \leftarrow \{(b, \widehat{q}_b)\}_{b=0}^{B-1}$.
$\mathbf{w} \leftarrow \textsc{FitActionWeights}(\mathcal{D}_t^*, \mathcal{D}_t^{\mathrm{fail}}, m, \mu)$.
Freeze $\mathbf{w}$.
$\boldsymbol{\tau}_{\mathrm{skill}} \leftarrow \textsc{BuildSkillReference}(\mathcal{D}_t^*, \mathbf{w}, S_{\mathrm{ref}})$.
Freeze F_t, $\mathcal{L}_t$, and $\boldsymbol{\tau}_{\mathrm{skill}}$ for online evolution.

Phase II: Conformal-Guided Self-Evolving Agent Skills Optimization

Initialize skill-and-failure history $\mathcal{H} \leftarrow [\,]$.
$\tau^{(0)} \leftarrow \textsc{Execute}(\mathcal{A}, t, S^{(0)})$.
for $\ell = 0, \ldots, T-1$ **do**
  $s_{\mathrm{out}}^{(\ell)} \leftarrow 1 - \mathrm{GS}(\tau^{(\ell)})$.
  $s_{\mathrm{adh}}^{(\ell)} \leftarrow 1 - \cos\Big(\mathrm{embed}_{\mathbf{w}}(\tau^{(\ell)}), \tau_{\mathrm{skill}} \Big)$.
  $s_{\mathrm{comp}}^{(\ell)} \leftarrow \lambda s_{\mathrm{out}}^{(\ell)} + (1-\lambda) s_{\mathrm{adh}}^{(\ell)}$.
  if $s_{\mathrm{comp}}^{(\ell)} \le \delta$ **then**
    $S_\star \leftarrow \textsc{DistillSkill}(x_t, \tau^{(\ell)})$.
    $\tau_{\mathrm{val}} \leftarrow \textsc{Execute}(\mathcal{A}, t, S_\star)$.
    return $(S_\star, \tau_{\mathrm{val}}, \mathrm{GS}(\tau_{\mathrm{val}}))$.
  end if
  $n \leftarrow |\tau^{(\ell)}|$.
  for $i = 1, \ldots, n$ **do**
    $a_i \leftarrow (k_i, \sigma_i)$; $b_i \leftarrow b(i,n)$.
    $A_i \leftarrow \min\{1,\ 1 - p_{F_t}(a_i \mid b_i) + \beta \mathbf{1}[\sigma_i = \text{failed}]\}$.
  end for
  $\mathcal{F} \leftarrow \{i : A_i > \widehat{q}_{b_i},\ A_i - \widehat{q}_{b_i} > \gamma\}$.
  Sort $\mathcal{F}$: failed-status steps first, then descending A_i.
  Deduplicate by (k_i, σ_i), retaining the largest A_i for each key.
  $\mathcal{E} \leftarrow \textsc{FormatDiagnostics}(\tau^{(\ell)}, \mathrm{TopK}(\mathcal{F}, K), \{A_i\}, \{\widehat{q}_{b_i}\})$.
  $S^{(\ell+1)} \leftarrow \textsc{GenerateSkill}(x_t, S^{(\ell)}, \tau^{(\ell)}, \mathrm{GS}(\tau^{(\ell)}), \mathcal{E}, \mathcal{H})$.
  Append $(S^{(\ell)}, \tau^{(\ell)}, \mathcal{E})$ to $\mathcal{H}$.
  $\tau^{(\ell+1)} \leftarrow \textsc{Execute}(\mathcal{A}, t, S^{(\ell+1)})$.
end for
return $(S^{(T)}, \tau^{(T)}, \mathrm{GS}(\tau^{(T)}))$.

B Theoretical Analysis and Proofs

We formalize three properties of SkillCertainty: the symmetry and empirical calibration behavior of LOO scoring, deterministic bounds induced by the composite routing criterion, and the role of fresh candidate-skill validation.

Lemma 1 (Trajectory-Level Leave-One-Out Calibration Symmetry)**.** *Let $\mathcal{D}_t^* = \{\tau_1^*, \ldots, \tau_M^*\}$ be exchangeable successful reference trajectories collected for the same task under the same execution protocol. For each trajectory τ_j^*, let F^{-j} denote the bucket-wise action-frequency tables constructed from $\mathcal{D}_t^* \setminus \{\tau_j^*\}$, and let $A^{-j}(s_i)$ be the nonconformity score of a step $s_i \in \tau_j^*$ computed using F^{-j}.*

Then the leave-one-out construction is symmetric with respect to permutations of the reference trajectories: every trajectory is evaluated by the same scoring rule against a reference constructed from all other trajectories, and no step from τ_j^* contributes to the frequency table used to score τ_j^* itself.

For bucket b, let

$$\mathcal{S}_b = \left\{ A^{-j}(s_i) : s_i \in \tau_j^*,\ b(i, n_j) = b,\ j = 1, \ldots, M \right\}.$$

If $\hat{q}_b$ is chosen as the empirical $(1-\alpha)$ quantile of $\mathcal{S}_b$ (with the threshold floor η applied as in Eq. (4)), then the resulting threshold satisfies the corresponding empirical calibration property

$$\frac{1}{|\mathcal{S}_b|} \sum_{a \in \mathcal{S}_b} \mathbf{1}[a \le \hat{q}_b] \ge 1 - \alpha,$$

up to the finite-sample discretization of the empirical quantile.

Proof. Because the reference trajectories are exchangeable and the leave-one-out construction applies the same procedure to every trajectory, permuting the trajectory indices simply permutes the corresponding leave-one-out score collections. Moreover, when scoring τ_j^*, the frequency table F^{-j} is constructed without any step from τ_j^*, so the trajectory does not directly contribute to its own behavioral reference. Finally, $\hat{q}_b$ is defined as the empirical $(1-\alpha)$ quantile of the pooled leave-one-out scores in bucket b. By the definition of the empirical quantile, at least a $(1-\alpha)$ fraction of these calibration scores do not exceed $\hat{q}_b$; applying the floor η can only increase this fraction. □

Remark 1. *The role of leave-one-out scoring is to prevent a reference trajectory from directly contributing to the frequency table used to evaluate its own steps. This reduces self-inclusion effects and makes the calibration scores reflect cross-trajectory behavioral consistency rather than within-trajectory reuse. Importantly, the exchangeability assumption is imposed on successful trajectories collected for the same task, not on individual steps within a trajectory. Lemma 1 applies to the multi-trajectory LOO setting; sparse-reference cases use the fallback described in Section 4.1. Since multiple steps within a trajectory may be dependent, this empirical calibration property should not be interpreted as a standard finite-sample $1-\alpha$ predictive-coverage guarantee for an arbitrary future step.*

Proposition 2 (Bounds Induced by the Composite Routing Criterion)**.** *Let $s_{\text{comp}}^{(\ell)} = \lambda s_{\text{out}}^{(\ell)} + (1-\lambda) s_{\text{adh}}^{(\ell)}, \lambda \in (0,1)$. If a trajectory satisfies $s_{\text{comp}}^{(\ell)} \le \delta$, then $\text{GS}(\tau^{(\ell)}) \ge 1 - \frac{\delta}{\lambda}$, and $\cos\big(\text{embed}(\tau^{(\ell)}), \boldsymbol{\tau}_{\text{skill}}\big) \ge 1 - \frac{\delta}{1-\lambda}$.*

Proof. Since both $s_{\text{out}}^{(\ell)}$ and $s_{\text{adh}}^{(\ell)}$ are non-negative, $s_{\text{comp}}^{(\ell)} \le \delta$ implies $\lambda s_{\text{out}}^{(\ell)} \le \delta$ and $(1-\lambda) s_{\text{adh}}^{(\ell)} \le \delta$. Using $s_{\text{out}}^{(\ell)} = 1 - \text{GS}(\tau^{(\ell)})$ gives $\text{GS}(\tau^{(\ell)}) \ge 1 - \frac{\delta}{\lambda}$. Similarly, since $s_{\text{adh}}^{(\ell)} = 1 - \cos\big(\text{embed}(\tau^{(\ell)}), \boldsymbol{\tau}_{\text{skill}}\big)$, we obtain $\cos\big(\text{embed}(\tau^{(\ell)}), \boldsymbol{\tau}_{\text{skill}}\big) \ge 1 - \frac{\delta}{1-\lambda}$. □

Remark 2. *Proposition 2 gives the routing threshold δ an interpretable relationship with task outcome quality and behavioral adherence. At the default setting $\lambda = 0.5$ and $\delta = 0.25$, any trajectory entering the conforming branch satisfies $\text{GS}(\tau^{(\ell)}) \ge 0.5$ and $\cos\big(\text{embed}(\tau^{(\ell)}), \boldsymbol{\tau}_{\text{skill}}\big) \ge 0.5$. These bounds follow deterministically from the routing criterion and should not be interpreted as statistical guarantees of future task success or skill generalization.*

Proposition 3 (Fresh Validation Evaluates the Distilled Candidate)**.** *Let* $S^\star = \textsc{DistillSkill}(x_t, \tau^{(\ell)})$ *be the candidate skill distilled after observing trajectory* $\tau^{(\ell)}$*, where* $\tau^{(\ell)} = \textsc{Execute}(A, t, S^{(\ell)})$*. Because* $\tau^{(\ell)}$ *is generated before* $S^\star$ *exists,* $\mathrm{GS}(\tau^{(\ell)})$ *evaluates the behavior induced by* $S^{(\ell)}$*, rather than the newly generated candidate* $S^\star$*.*

Let $\tau_{\mathrm{val}} = \textsc{Execute}(A, t, S^\star)$ *be a fresh validation execution sampled under the same evaluation protocol and execution distribution conditional on* $S^\star$*.*

If executions are stochastic and $\mu(S^\star) = \mathbb{E}\left[\mathrm{GS}\left(\textsc{Execute}(A, t, S^\star)\right) \mid S^\star\right]$*, then* $\mathbb{E}\left[\mathrm{GS}(\tau_{\mathrm{val}}) \mid S^\star\right] = \mu(S^\star)$*.*

Remark 3. *A single validation execution provides only one observation of the candidate skill's behavior and therefore does not certify its true success probability or generalization performance. Its role is to prevent the triggering trajectory's score from being used as a direct proxy for the performance of the newly distilled skill.*

C Framework Diagrams

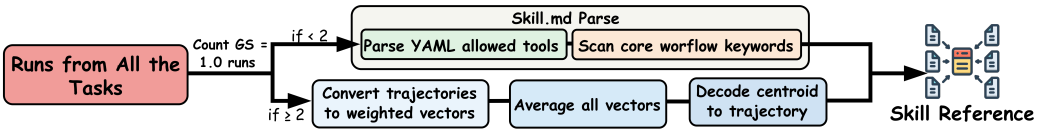


Figure 4: Framework of skill reference construction.

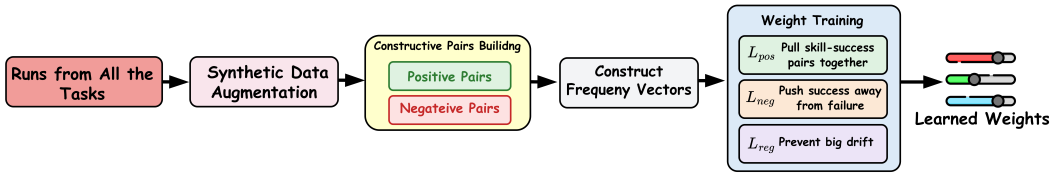


Figure 5: Framework of action space weight training.

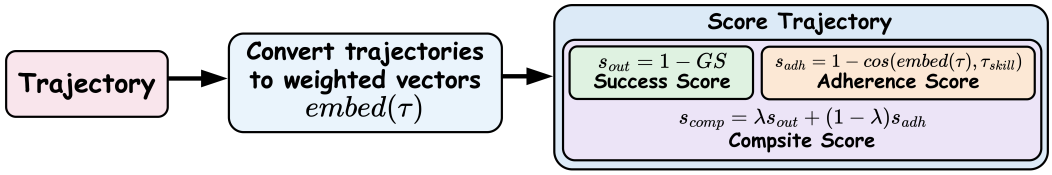


Figure 6: Framework of trajectory scoring.

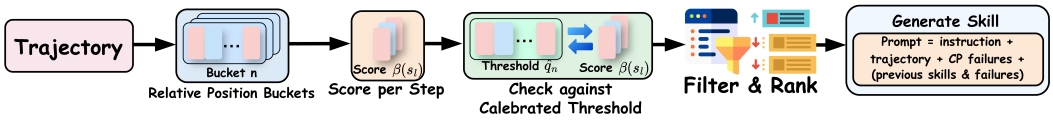


Figure 7: Framework of Step-CP diagnostics.

D Skill Generation Prompt Template

The following shows the full prompt template used in the nonconforming branch of CSASO. The Step-CP Diagnostics block (highlighted) is the key addition that distinguishes SkillCertainty from

prior skill generation methods: it injects statistically grounded, step-level failure evidence directly into the generation prompt rather than relying on raw trajectory summaries.

```
## Task: {task name}  ## Outcome: PARTIAL (GS={x.xx})
## Task Instruction {instruction text}
## Agent Trajectory {formatted step sequence with titles and outputs}
## Step-Level CP Diagnostics (F/N steps fail CP threshold)
- Step i [k:failed] A=0.83 q̂=0.42 (margin=+0.41): {step title}
Output: {truncated step output} ...
Note: High margin means this action type was RARELY seen at this
position in GS=1.0 runs.
## Previous Skills and Failures {accumulated skill history}
Distill the key lessons into SKILL.md. Specifically address the
failure points identified above.
```

E EXPERIMENTS

E.1 IMPLEMENTATION DETAILS AND HYPERPARAMETERS

Unless otherwise stated, we use $N_{\text{cal}} = 10$ executions per condition, $B = 10$ relative-position buckets, CP miscoverage level $\alpha = 0.10$, failed-step penalty $\beta = 0.3$, minimum calibration threshold $\eta = 0.25$, composite-score weight $\lambda = 0.5$, routing threshold $\delta = 0.25$, diagnostic margin $\gamma = 0.15$, and $K = 5$ diagnostic steps. For action-weight learning, we use similarity margin $m = 0.3$ and regularization strength $\mu = 2.0$.

E.2 BASELINE DESCRIPTIONS

We compare against four skill-evolution baselines spanning complementary design strategies.

SkillOpt (Yang et al., 2026) performs iterative text-space skill optimization using scored rollouts and held-out validation to accept beneficial edits.

SkillGen (Ma et al., 2026) synthesizes skills through contrastive induction over successful and failed trajectories and verifies their effect through subsequent executions.

Memento-Skills (Zhou et al., 2026) continually constructs and updates externalized skills through memory-based reflective learning.

EvoSkill (Alzubi et al., 2026) iteratively analyzes execution failures, proposes skill additions or revisions, and selects skills using held-out validation performance.

Unlike these baselines, SkillCertainty explicitly calibrates position-aware behavioral deviations against certified successful trajectories and uses the resulting diagnostics for conformity-based routing and targeted skill revision.

E.3 CROSS-MODEL COMPARISON

Figure 8 reports skill adherence nonconformity s_{adh} and per-iteration efficiency $\Delta\text{GS}/\text{iter}$ across all six model backends. SkillCertainty achieves the lowest s_{adh} on every backend, confirming that generated skills conform more tightly to the certified behavioral reference regardless of the underlying LLM. Per-iteration efficiency is highest on five of the six backends; Claude Haiku-4.5 is the only backend on which another method (SkillGen) achieves a higher ΔGS/iter.

E.4 PER-TASK ANALYSIS: DETAILS

SkillCertainty achieves the largest gains on tasks where repository skills provide incomplete or misspecified guidance:

- **Energy Market Pricing** ($+0.900$): repository skills provide no task-specific pricing formulas; Step-CP diagnostics identify the missing computation steps and the generated skill encodes the full pricing pipeline.

- **HVAC Control** ($+0.800$, $0.200 \rightarrow 1.000$): the agent fails with generic repository guidance but achieves perfect score after one Step-CP iteration, as analyzed in Figure 9.
- **Edit PDF** ($+0.778$): repository skills miss the specific API calls required by the task's PDF library; Step-CP pinpoints the failing `fetch:failed` steps and generates a skill that directly addresses the correct API usage.

Even on tasks where the baseline performs reasonably well, SkillCertainty further improves performance: SEC Financial Report ($0.900 \rightarrow 1.000$) and Mars Clouds Clustering ($0.700 \rightarrow 0.750$) both benefit from targeted diagnostic injection, demonstrating that Step-CP adds value across the full performance range.

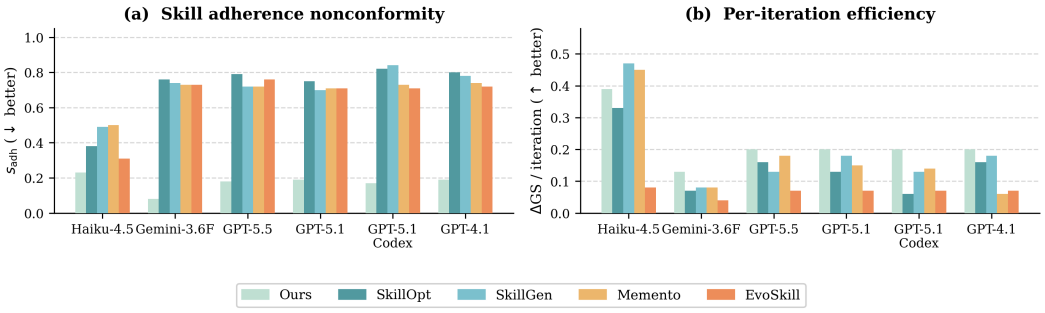


Figure 8: **Cross-model comparison of skill adherence and per-iteration efficiency.** (a) Skill adherence nonconformity s_{adh} (↓ better): SkillCertainty achieves the lowest score on every backend. (b) Per-iteration efficiency $\Delta\text{GS}/\text{iter}$ (↑ better): SkillCertainty yields the highest gain per iteration on five of six models.

Table 5: Step-CP vs. repository skills (with-skills baseline) for Claude Haiku 4.5 on all tasks where Step-CP improves performance.

Task	Repo Skills	Step-CP (Ours)	ΔGS
Energy Market Pricing	$0.000_{\pm 0.000}$	$0.900_{\pm 0.137}$	**+0.900**
XLSX Recover Data	$0.000_{\pm 0.000}$	$0.875_{\pm 0.000}$	**+0.875**
HVAC Control	$0.200_{\pm 0.422}$	$1.000_{\pm 0.000}$	**+0.800**
Edit PDF	$0.000_{\pm 0.000}$	$0.778_{\pm 0.091}$	**+0.778**
Invoice Fraud Detection	$0.100_{\pm 0.316}$	$0.700_{\pm 0.274}$	**+0.600**
Lake Warming Attribution	$0.000_{\pm 0.000}$	$0.400_{\pm 0.224}$	**+0.400**
Offer Letter Generator	$0.400_{\pm 0.516}$	$0.750_{\pm 0.000}$	**+0.350**
Travel Planning	$0.600_{\pm 0.516}$	$0.909_{\pm 0.074}$	**+0.309**
Powerlifting Coef. Calc.	$0.700_{\pm 0.483}$	$1.000_{\pm 0.000}$	**+0.300**
Reserves at Risk Calc	$0.000_{\pm 0.000}$	$0.200_{\pm 0.000}$	**+0.200**
Weighted GDP Calc	$0.636_{\pm 0.505}$	$0.741_{\pm 0.174}$	**+0.105**
SEC Financial Report	$0.900_{\pm 0.316}$	$1.000_{\pm 0.000}$	**+0.100**
Mars Clouds Clustering	$0.700_{\pm 0.483}$	$0.750_{\pm 0.500}$	**+0.050**
Mean	0.326	0.769	**+0.443**

E.5 Sensitivity Analysis

Composite routing threshold δ. Table 6 reports sensitivity to δ on the crystallographic-wyckoff-position-analysis task. The routing threshold substantially affects whether the agent enters the conforming branch or continues skill evolution. For GPT-family models whose initial trajectories already achieve high task scores, smaller thresholds can trigger additional updates and reduce final performance, whereas $\delta = 0.25$ allows early termination. For Claude Haiku-4.5 and Gemini 3.6 Flash, intermediate thresholds provide a different trade-off between continued evolution and early termination. On this task, $\delta = 0.25$ provides a practical default for balancing these behaviors.

Calibration level α. Table 7 evaluates $\alpha \in \{0.05, 0.10, 0.20, 0.30\}$ on the same task. Performance varies across models, but the empirical fraction of steps below their calibrated thresholds remains above 0.78 in all tested settings. This analysis characterizes the sensitivity of the empirical Step-CP thresholds to α rather than validating a standard pointwise conformal coverage guarantee.

E.6 Ablation Studies: Per-Task Details

Step-CP Diagnostic Injection. On *Crystallographic Wyckoff Analysis*, Full resolves the task completely in one iteration (ΔGS $= +1.000$) whereas No Diagnostics achieves only partial improvement ($+0.640$). On *Gravitational Wave Detection*, step-level diagnostics pinpoint the matched-filtering operation as the critical failure point, enabling Full to close the gap ($+0.111$) that No Diagnostics cannot ($+0.000$) within the evaluated update budget. On *Lab Unit Harmonization*, No Diagnostics produces a skill that actively degrades performance (-0.021), whereas Full achieves $+0.042$ by targeting the specific unit-conversion steps flagged by CP.

Effect of Learned Action Importance Weights. On Shock Analysis (Demand), learned weights improve the trajectory-level adherence representation and composite routing, leading to a GS improvement from 0.200 to 0.400 (ΔGS $= +0.200$), whereas uniform weights yield no improvement. On Shock Analysis (Supply), uniform weights degrade GS from 0.333 to 0.000 (ΔGS $= -0.333$), while learned weights preserve the baseline. On Lab Unit Harmonization, learned weights also achieve a larger gain ($+0.042$ vs. $+0.021$). Step-CP localization remains determined by position-dependent action frequencies and execution status.

Conforming Branch Validation. On Energy Market Pricing, the score changes from 1.000 at conforming detection to 0.000 when the newly distilled skill is executed in the fresh validation run, illustrating that the triggering trajectory score is not necessarily representative of the generated skill's subsequent behavior. Crystallographic Wyckoff shows a similar, though smaller, discrepancy. In contrast, Gravitational Wave Detection obtains the same improvement in both measurements. These cases illustrate why a fresh execution is useful for directly evaluating the candidate skill after distillation.

F Per-Task Qualitative Analysis

For each task we describe the failure modes of the repository skills, the Step-CP diagnostics that identified them, and how the generated skill addresses each gap.

F.1 HVAC Control

Repository skill gaps. Five repository skills cover IMC tuning theory, scipy curve fitting, excitation signal design, first-order model fitting, and safety interlocks. None specifies that the simulator advances in fixed steps of `dt = 0.5 s`; using a wrong step rate corrupts all parameter estimates obtained from the calibration phase. The IMC tuning skill recommends $\lambda = \tau$ as the default, but the task imposes a $<10\%$ overshoot constraint that requires the more aggressive $\lambda = \tau/2$; no skill derives this from the constraint. The safety-interlocks skill mentions anti-windup only as a tip with no implementation, so integral saturation during the ramp phase produces large overshoot in practice. Finally, no skill specifies the five output filenames (`calibration_log.json`, `estimated_params.json`, `tuned_gains.json`, `control_log.json`, `metrics.json`) required by the evaluator; missing any one causes evaluation failure.

Step-CP diagnosis. At iteration 0 the agent uses $\lambda = \tau$ and omits anti-windup, producing overshoot that violates the $<10\%$ bound. Step-CP flags the control-loop step as high-nonconformity, identifies the overshoot-check step as `execute:failed`, and injects diagnostics pointing to both the IMC gain selection and the missing back-calculation logic. A secondary diagnosis flags missing output files as the cause of evaluator failure even when the controller itself converges.

Generated skill. After one iteration the skill encodes the fixed simulator time-step (`dt = 0.5 s`) as a hard constant, derives $\lambda = \tau/2$ from the overshoot constraint with an explanatory comment, provides a complete anti-windup back-calculation loop, and lists all five required output filenames with their exact paths. GS improves from 0.200 to 1.000 (ΔGS $= +0.800$).

Table 6: Sensitivity of SkillCertainty to the composite routing threshold δ on crystallographic-wyckoff-position-analysis. † marks the default setting ($\delta = 0.25$).

Model	δ	GS_f	ΔGS	n_{iters}	Conv.(%)
GPT-5.1	0.05	$0.638_{\pm 0.194}$	$-0.362_{\pm 0.194}$	$5.0_{\pm 0.0}$	0%
	0.10	$0.510_{\pm 0.107}$	$-0.490_{\pm 0.107}$	$5.0_{\pm 0.0}$	0%
	0.15	$0.542_{\pm 0.259}$	$-0.458_{\pm 0.259}$	$5.0_{\pm 0.0}$	0%
	0.20	$0.400_{\pm 0.416}$	$-0.600_{\pm 0.416}$	$5.0_{\pm 0.0}$	0%
	0.25†	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{0.000}_{\pm 0.000}$	$\mathbf{0.0}_{\pm 0.0}$	**100%**
GPT-5.4	0.05	$0.454_{\pm 0.133}$	$-0.546_{\pm 0.133}$	$5.0_{\pm 0.0}$	0%
	0.10	$0.530_{\pm 0.103}$	$-0.470_{\pm 0.103}$	$5.0_{\pm 0.0}$	0%
	0.15	$0.548_{\pm 0.225}$	$-0.452_{\pm 0.225}$	$5.0_{\pm 0.0}$	0%
	0.20	$0.566_{\pm 0.137}$	$-0.434_{\pm 0.137}$	$5.0_{\pm 0.0}$	0%
	0.25†	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{0.000}_{\pm 0.000}$	$\mathbf{0.0}_{\pm 0.0}$	**100%**
GPT-5.5	0.05	$0.634_{\pm 0.260}$	$-0.366_{\pm 0.260}$	$5.0_{\pm 0.0}$	0%
	0.10	$0.200_{\pm 0.276}$	$-0.800_{\pm 0.276}$	$5.0_{\pm 0.0}$	0%
	0.15	$0.000_{\pm 0.000}$	$-1.000_{\pm 0.000}$	$5.0_{\pm 0.0}$	0%
	0.20	$0.274_{\pm 0.387}$	$-0.726_{\pm 0.387}$	$5.0_{\pm 0.0}$	0%
	0.25†	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{0.000}_{\pm 0.000}$	$\mathbf{0.0}_{\pm 0.0}$	**100%**
GPT-4.1	0.05	$0.800_{\pm 0.254}$	$-0.200_{\pm 0.254}$	$5.0_{\pm 0.0}$	0%
	0.10	$0.690_{\pm 0.222}$	$-0.310_{\pm 0.222}$	$5.0_{\pm 0.0}$	0%
	0.15	$0.746_{\pm 0.236}$	$-0.254_{\pm 0.236}$	$5.0_{\pm 0.0}$	0%
	0.20	$0.600_{\pm 0.191}$	$-0.400_{\pm 0.191}$	$5.0_{\pm 0.0}$	0%
	0.25†	$\mathbf{1.000}_{\pm 0.000}$	$\mathbf{0.000}_{\pm 0.000}$	$\mathbf{0.0}_{\pm 0.0}$	**100%**
Gemini 3.6 Flash	0.05	$0.712_{\pm 0.195}$	$0.712_{\pm 0.195}$	$4.2_{\pm 1.8}$	20%
	0.10	$0.636_{\pm 0.412}$	$0.636_{\pm 0.412}$	$3.2_{\pm 2.0}$	60%
	0.15	$0.874_{\pm 0.080}$	$0.874_{\pm 0.080}$	$2.2_{\pm 1.1}$	100%
	0.20	$\mathbf{0.892}_{\pm 0.099}$	$\mathbf{0.892}_{\pm 0.099}$	$\mathbf{2.2}_{\pm 1.3}$	**100%**
	0.25†	$0.712_{\pm 0.148}$	$0.712_{\pm 0.148}$	$2.0_{\pm 1.2}$	100%
Claude Haiku 4.5	0.05	$0.510_{\pm 0.345}$	$0.510_{\pm 0.345}$	$5.6_{\pm 0.9}$	20%
	0.10	$0.510_{\pm 0.356}$	$0.510_{\pm 0.356}$	$6.0_{\pm 0.0}$	0%
	0.15	$0.638_{\pm 0.358}$	$0.638_{\pm 0.358}$	$4.6_{\pm 1.9}$	40%
	0.20	$\mathbf{0.820}_{\pm 0.180}$	$\mathbf{0.820}_{\pm 0.180}$	$\mathbf{4.2}_{\pm 1.6}$	**60%**
	0.25†	$0.784_{\pm 0.207}$	$0.784_{\pm 0.207}$	$2.8_{\pm 0.4}$	100%

F.2 Crystallographic Wyckoff Position Analysis

Repository skill gaps. Two repository skills describe pymatgen's API and SymPy rational arithmetic respectively. The pymatgen skill recommends verifying the installation with `print(pymatgen.__version__)`, which raises `AttributeError` in this environment and crashes the agent before any crystal structure is loaded. The SymPy skill suggests `nsimplify` for fraction conversion, but SymPy is not installed in the task environment; furthermore, neither skill specifies the `max_denominator=12` bound required by the evaluator, the parsing logic for `wyckoff_symbols` (last character = letter, leading integer = multiplicity), or the expected output dictionary schema.

Step-CP diagnosis. At iteration 0 the agent crashes at `pymatgen.__version__` before reaching the analysis step. Step-CP identifies this as an `execute:failed` step with an `AttributeError` traceback, and separately flags the `import sympy` line as a `ModuleNotFoundError`.

Generated skill. After one iteration the skill encodes the import quirk as a hard rule (`# do NOT check __version__`), replaces SymPy with `fractions.Fraction(max_denominator=12)`, provides the complete Wyckoff parsing loop, and includes a concrete expected-output example for self-verification. GS improves from 0.000 to 1.000 (ΔGS $= +1.000$).

Table 7: Sensitivity of Step-CP to CP miscoverage level α on `crystallographic-wyckoff-position-analysis` (5 trials per α). GS_f and ΔGS are $\mathrm{mean}_{\pm\mathrm{std}}$. Conv. denotes the fraction of trials where $s_{\mathrm{comp}} \leq 0.25$; Mono. denotes the fraction of trials with non-decreasing GS across iterations; Step Cov. denotes the empirical fraction of final-iteration steps below their calibrated thresholds; it is not a finite-sample predictive-coverage estimate. **Bold** marks the best α per model; † marks the default (α=0.10).

Model	α	GS_f	ΔGS	n_{iters}	Conv.	Mono.	Step Cov.
Claude Haiku 4.5	0.05	$0.800_{\pm 0.276}$	$0.800_{\pm 0.276}$	$2.8_{\pm 2.0}$	0.60	0.80	0.878
	0.10†	$0.620_{\pm 0.217}$	$0.620_{\pm 0.217}$	$3.6_{\pm 1.5}$	0.60	0.80	0.792
	0.20	$0.566_{\pm 0.324}$	$0.566_{\pm 0.324}$	$3.2_{\pm 2.0}$	0.60	0.60	0.838
	0.30	$0.820_{\pm 0.246}$	$0.820_{\pm 0.246}$	$3.0_{\pm 1.9}$	0.60	1.00	0.780
GPT-5.1	0.05	$0.416_{\pm 0.106}$	$0.416_{\pm 0.106}$	$5.0_{\pm 0.0}$	0.00	0.60	1.000
	0.10†	$0.710_{\pm 0.268}$	$0.710_{\pm 0.268}$	$3.6_{\pm 1.9}$	0.40	0.40	1.000
	0.20	$0.710_{\pm 0.268}$	$0.710_{\pm 0.268}$	$4.2_{\pm 1.8}$	0.40	0.60	1.000
	0.30	$0.470_{\pm 0.045}$	$0.470_{\pm 0.045}$	$5.0_{\pm 0.0}$	0.00	0.60	1.000
Gemini 3.6 Flash	0.05	$0.694_{\pm 0.163}$	$0.694_{\pm 0.163}$	$1.2_{\pm 0.4}$	1.00	1.00	0.884
	0.10†	$0.584_{\pm 0.139}$	$0.584_{\pm 0.139}$	$1.8_{\pm 1.8}$	0.80	0.80	0.951
	0.20	$0.712_{\pm 0.148}$	$0.712_{\pm 0.148}$	$1.6_{\pm 1.3}$	1.00	1.00	0.893
	0.30	$0.476_{\pm 0.277}$	$0.476_{\pm 0.277}$	$3.0_{\pm 1.6}$	0.80	0.60	0.779

F.3 Gravitational Wave Detection

Repository skill gaps. Two repository skills provide correct but generic PyCBC conditioning and matched-filtering recipes. Neither documents the astropy attribute clash that causes `from pycbc.frame import read_frame` to raise an ImportError at import time. The matched-filtering skill uses example masses $(25, 20\,M_\odot)$ without specifying the $m_1 \geq m_2$ convention, and provides no concrete mass grid, set of approximants, or output-file specification.

Step-CP diagnosis. At iteration 0 the agent crashes immediately at `read_frame` with an ImportError. Step-CP flags this failed import step and exposes the associated traceback in the diagnostic context. Subsequent diagnostics localize the remaining matched-filtering failures, providing targeted evidence for skill revision.

Generated skill. After one iteration, the generated skill adds the required astropy pre-import workaround, enforces $m_1 \geq m_2$, specifies the three required approximants (`SEOBNRv4_opt`, `IMRPhenomD`, and `TaylorT4`), the 10–40 $M_\odot$ mass grid, and the required output CSV path. It retains the explicit `.numpy()` conversion already present in the repository matched-filtering recipe. GS improves from 0.889 to 1.000 ($\Delta GS = +0.111$).

F.4 GLM Lake Mendota Calibration

Repository skill gaps. Three repository skills cover GLM's namelist format, calibration parameter ranges, and output coordinate conventions. The basics skill states `cd /root && glm` but does not specify that the binary must be invoked as `/usr/local/bin/glm` and that the working directory must be `/root/` — running bare `glm` from any other directory fails silently. The calibration skill provides Nelder-Mead optimization over parameter ranges, but each GLM call takes ∼60 s, making iterative optimization infeasible within a single episode; and the default `wind_factor=1.0` yields RMSE ≈ 2.41 °C, above the 2.0 °C threshold. The output skill converts GLM depth using a fixed `LAKE_DEPTH` constant, but GLM's simulated surface elevation varies per timestep as water fills or drains; a constant produces wrong depth matching across the 7-year simulation period.

Step-CP diagnosis. At iteration 0 the agent runs GLM with default parameters and achieves RMSE = 2.09 °C (above threshold). Step-CP localizes the failing RMSE-check and calibration steps; conditioned on these diagnostics, the skill generator selects the wind factor as a high-leverage adjustment.

Generated skill. After one iteration the skill directly encodes the verified parameter set (`Kw=0.4`, `coef_mix_hyp=0.25`, `wind_factor=0.9`, `lw_factor=0.85`) that achieves RMSE $\approx$ 1.86–1.93 °C, specifies the exact binary path and working directory, and reads surface elevation per-timestep from `elev[idx, nlayers-1]` rather than using a fixed constant. GS improves from 0.000 to 1.000 (ΔGS $= +1.000$).

F.5 Energy Market Pricing (DC-OPF)

Repository skill gaps. Four repository skills address DC power flow, economic dispatch, LMP extraction via CVXPY, and network data parsing. None mentions that system Python lacks `numpy` and `scipy`, requiring a virtual environment before any numerical work can proceed. The DC power flow skill implies that Pmax/Pmin are in per-unit (requiring division by `baseMVA`), whereas the task's MATPOWER JSON stores them already in MW; applying the scaling corrupts all generator bounds. No skill formulates joint dispatch-and-reserve co-optimization or the 5% reserve-adequacy constraint. The LMP skill uses CVXPY (`cp.CLARABEL`), which is unavailable in the restricted environment; the correct replacement is `scipy.optimize.linprog` with `eqlin.marginals`, whose sign convention requires negation (`lmp = -eqlin.marginals`). Finally, the counterfactual branch lookup treats the branch list as directed, missing the case where the two endpoint bus IDs appear in reverse order.

Step-CP diagnosis. At iteration 0 the agent crashes immediately with `ModuleNotFoundError: numpy`, which Step-CP attributes to the missing venv. At iteration 1 an infeasibility error arises from double-scaled Pmax bounds; Step-CP identifies the unit-conversion step as the source and injects the MW note. Residual output mismatches at iteration 1 flag the sign error on dual values and the unidirectional branch search.

Generated skill. After two iterations the skill encodes venv creation and activation as Step 0; adds an explicit MW-unit note with the 2-vs-3-coefficient handling for `gencost`; formulates the full LP with reserve variables and the 5% adequacy constraint; negates `eqlin.marginals` for LMPs; and searches both directions for the counterfactual branch. GS improves from 0.000 to 1.000 (ΔGS $= +0.900$).

Repository Skills (5 generic files)

imc-tuning-rules/SKILL.md

```
# IMC Tuning for First-Order Systems
# Process: G(s) = K / (tau*s + 1)
Kp = tau / (K * lambda)
Ki = Kp / tau        # = 1 / (K * lambda)
Kd = 0.0

# Choosing lambda (tuning parameter):
#  0.1*tau  ->  very aggressive
#  0.5*tau  ->  aggressive
#  1.0*tau  ->  moderate  <-- DEFAULT
#  2.0*tau  ->  conservative

class PIController:
    def compute(self, measurement, dt):
        error = self.setpoint - measurement
        self.integral += error * dt
        output = self.Kp*error +
              self.Ki*self.integral
        # Anti-windup: mentioned as tip, not
              implemented
        return max(out_min, min(out_max,
              output))
```

scipy-curve-fit/SKILL.md

```
# scipy.optimize.curve_fit for Parameter
     Estimation
y_initial = ...   # initial output value
u         = ...   # step-test input magnitude

def step_response(t, K, tau):
    return y_initial + K*u*(1 -
          np.exp(-t/tau))

popt, _ = curve_fit(step_response, t_data,
      y_data, p0=[K_guess, tau_guess])
K_est, tau_est = popt
```

excitation-signal-design/SKILL.md

```
# Step Test for System Identification
# - Hold >= 3-5 time constants
# - 10-20 samples per time constant
for step in range(num_steps):
    result = system.step(input_value)
    data.append({"time": result["time"],
          "output": result["output"]})
# No simulator-specific dt constraint given
```

first-order-model-fitting/SKILL.md

```
# Step Response: y(t) = y_amb +
      K*u*(1-exp(-t/tau))
# K   = (y_steady - y_ambient) / u
# tau = time to reach 63.2% of final change
def model(t, K, tau):
    return y_ambient + K*u*(1 -
          np.exp(-t/tau))
# Target: R^2 > 0.9 (no task-specific bound
     given)
```

safety-interlocks/SKILL.md

```
# Safety: check BEFORE applying control
     output
def apply_safety_limits(measurement,
      command, max_limit, max_output,
                        min_output):
    if measurement >= max_limit:
        command = min_output   # emergency
              cutoff
    return max(min_output, min(max_output,
          command))
# No concrete limits specified for this task
```

Step-CP Skill (1 unified file, generated after iter 1)

hvac-pid-temperature-control/SKILL.md

```
# HVAC PID Temperature Control
# Full pipeline: calibration -> fitting -> tuning ->
     anti-windup PI -> metrics

## 1. Calibration
sim = HVACSimulator()
T0 = sim.get_temperature()
sim.set_heater_power(50.0)
data = [{"time": 0.0, "temperature": T0,
         "heater_power": 0.0}]
for i in range(1, 121):   # 60 s at dt=0.5 s
    sim.step()
    data.append({"time": i*0.5,
                 "temperature": sim.get_temperature(),
                 "heater_power": 50.0})
```

Simulator dt = 0.5 s is fixed (no repo skill mentions this). Wrong step rate corrupts all parameter estimates.

```
## 2. First-Order Parameter Estimation
# T(t) = T_inf - (T_inf - T0)*exp(-t/tau)
# T_inf = T0 + K * u
def first_order(t, K, tau):
    T_inf = T0 + K * heater_power
    return T_inf - (T_inf - T0)*np.exp(-t/tau)

popt, _ = curve_fit(first_order, times, temps,
     p0=[0.1, 30.0], maxfev=10000)
K, tau = popt   # e.g. K~0.116, tau~38 s
# Require R^2 > 0.95; extend calibration if not met

## 3. IMC Tuning
# lambda = tau/2  ->  balanced (meets overshoot bound)
# lambda = tau/3  ->  aggressive, risks overshoot
lambda_c = tau / 2.0
Kp = tau / (K * (lambda_c + dt)) # ~17
Ki = Kp / tau                    # ~0.45
Kd = 0.0                         # avoids noise amp.
```

Repo defaults $\lambda = \tau$; this task requires $\lambda = \tau/2$ to satisfy the $<10\%$ overshoot constraint. Step-CP localizes the overshoot-related failure; the generated skill incorporates the tighter tuning rule.

```
## 4. Anti-Windup PI Control Loop
setpoint = <TARGET_TEMP># read from task input

integral = 0.0
out_min, out_max = 0.0, 100.0

for step in range(300):    # 150 s at dt=0.5
    T     = sim.get_temperature()
    error = setpoint - T
    integral += error * dt
    u = Kp * error + Ki * integral
    u = np.clip(u, out_min, out_max)
    # Back-calculate anti-windup
    if u == out_max and error > 0:
        integral -= error * dt
    if u == out_min and error < 0:
        integral -= error * dt
    sim.set_heater_power(u)
    sim.step()
```

Repo only mentions anti-windup as a tip; no code given. Without back-calculation the integral saturates during the ramp, producing large overshoot.

```
## 5. Metrics  (max-temp constraint: 30 C)
band = 0.02 * (setpoint - T0)
settled = [t for t,T in zip(times,temps)
           if abs(T - setpoint) <= band]
settling_time = settled[0] if settled else None
overshoot = max(0, max(temps)-setpoint)/(setpoint-T0)
sse = abs(np.mean(temps[-20:]) - setpoint)
# smooth with last 20 pts; noise std = 0.1 C

## Required output files (all 5 must be written):
# calibration_log.json  estimated_params.json
# tuned_gains.json  control_log.json  metrics.json
```

No repo skill specifies required output filenames. Missing any one file causes evaluation failure.

Figure 9: **HVAC Control.** Five repository skills cover IMC theory, curve fitting, and safety interlocks in isolation, omitting simulator-specific constants, anti-windup implementation, and output-file specification. After one Step-CP iteration, a single unified skill encodes the fixed time-step (`dt = 0.5 s`), the tighter IMC rule ($\lambda = \tau/2$), a complete anti-windup loop, and the exact five required filenames. These task-specific corrections are incorporated after Step-CP localizes the relevant failure points.

Repository Skills (2 generic files)

pymatgen/SKILL.md

```
# Pymatgen - Python Materials Genomics
# Materials science toolkit: crystal
      structures (CIF,
# POSCAR), phase diagrams, band structure,
      DOS,
# Materials Project integration.

## Installation
uv pip install pymatgen
# Verify:
import pymatgen
print(pymatgen.__version__)    # check
      installed version

## Basic Structure Operations
from pymatgen.core import Structure
structure =
      Structure.from_file("material.cif")
analyzer = SpacegroupAnalyzer(structure)
symm = analyzer.get_symmetrized_structure()
# symm.wyckoff_symbols  ->  list of Wyckoff
      labels
```

No mention that `pymatgen.__version__` raises `AttributeError` in this environment (only `from pymatgen.core import Structure` works). Verification via `print(pymatgen.__version__)` will crash the agent.

sympy/SKILL.md

```
# SymPy Rational Numbers
from sympy import Rational, nsimplify

# Convert float to exact fraction
r = Rational(0.333).limit_denominator(100)
# -> 1/3

# Or use nsimplify
val = nsimplify(0.375)    # -> 3/8
```

No mention of `fractions.Fraction` from the standard library, which requires only `max_denominator=12` per task spec. SymPy is not installed; importing it will fail.

Step-CP Skill (1 unified file, generated after iter 1)

crystallographic-wyckoff-position-analysis/SKILL.md

```
# Crystallographic Wyckoff Position Analysis
# When: given CIF files, extract Wyckoff labels,
# multiplicities, and fractional coords as rationals.

## CRITICAL: pymatgen import quirk
# pymatgen.__version__  ->  AttributeError (do NOT
      check)
# from pymatgen.core import Structure  ->  works fine
from pymatgen.core import Structure
from pymatgen.symmetry.analyzer import
      SpacegroupAnalyzer
from fractions import Fraction   # stdlib; always
      available
```

Step-CP flags the import failure observed in iteration 0; the generated skill encodes the corresponding workaround as a hard rule.

```
## Rationalization (max_denominator=12, task-required)
def float_to_rational(v: float, denom: int = 12) ->
      str:
    frac = Fraction(v).limit_denominator(denom)
    return str(frac) if frac.denominator > 1 \
           else str(frac.numerator)
# 0.9999 -> "1",  0.375 -> "3/8",  0.333 -> "1/3"
```

Repo skill suggests SymPy (not installed) and no denominator bound. The generated skill uses the task-required max-denominator value of 12 and replaces unavailable SymPy with `fractions.Fraction`.

```
## Wyckoff extraction pipeline
structure  = Structure.from_file(filepath)
analyzer   = SpacegroupAnalyzer(structure)
symm       = analyzer.get_symmetrized_structure()

wyckoff_multiplicity_dict = {}
wyckoff_coordinates_dict  = {}

for sites, label in zip(symm.equivalent_sites,
                        symm.wyckoff_symbols):
    letter = label[-1]          # e.g. "a", "c"
    mult   = int(label[:-1])    # e.g. 4, 8
    if letter not in wyckoff_multiplicity_dict:
        wyckoff_multiplicity_dict[letter] = mult
        wyckoff_coordinates_dict[letter]  = [
            float_to_rational(float(c))
                for c in sites[0].frac_coords
        ]
return {"wyckoff_multiplicity_dict":
      wyckoff_multiplicity_dict,
        "wyckoff_coordinates_dict":
             wyckoff_coordinates_dict}
```

Repo skill shows `wyckoff_symbols` exists but gives no parsing logic. The generated skill encodes the required Wyckoff parsing logic and output-dictionary schema.

```
## Correct workflow - write then test immediately
# Step 1: write solution.py (all logic above)
# Step 2: test inline (skip version checks entirely)
python3 -c "
from solution import analyze_wyckoff_...
print(analyze_wyckoff_...('FeS2_mp-226.cif'))
"
# Expected: {'wyckoff_multiplicity_dict':
      {'a':4,'c':8},
#   'wyckoff_coordinates_dict':
      {'a':['0','1/2','1/2'],
#
      'c':['3/8','1/9','8/9']}}
```

No repo skill specifies the output dict schema or a concrete test case. The generated skill includes a concrete task-provided output example for self-verification.

Figure 10: **Crystallographic Wyckoff Position Analysis.** The two repository skills cover pymatgen broadly and SymPy rational arithmetic, but neither identifies the `pymatgen.__version__` crash, the `fractions.Fraction(max_denominator=12)` requirement, the exact `wyckoff_symbols` parsing logic, or the task-required output schema. After one diagnostic-guided iteration, the generated skill incorporates these task-specific details, lifting GS from 0.00 to 1.00.

Repository Skills (2 generic files)

conditioning/SKILL.md

```
# GW Data Conditioning (PyCBC)
from pycbc.filter import highpass,
     resample_to_delta_t

strain = highpass(strain, 15.0)     # remove
     <15 Hz noise
strain = resample_to_delta_t(strain,
     1.0/2048)
strain = strain.crop(4, 4)          # remove
     edge artifacts

# PSD estimation
from pycbc.psd import welch
seg_len = int(4 * strain.sample_rate)
psd = welch(strain, seg_len=seg_len,
            seg_stride=seg_len // 2)
```

matched-filtering/SKILL.md

```
# Matched Filtering (PyCBC)
from pycbc.waveform import get_td_waveform
from pycbc.filter import matched_filter

hp, hc = get_td_waveform(
    approximant='IMRPhenomD',
    mass1=25, mass2=20,
    delta_t=conditioned.delta_t,
    f_lower=20
)
hp.resize(len(conditioned))
template =
     hp.cyclic_time_shift(hp.start_time)

snr = matched_filter(template, conditioned,
                     psd=psd,
                          low_frequency_cutoff=20)
snr = snr.crop(4 + 4, 4)

peak_idx = np.argmax(abs(snr).numpy())
peak_snr = abs(snr[peak_idx])
```

The repository recipe already includes .numpy() for SNR extraction, but does not specify the $m_1 \geq m_2$ convention or the required grid-search configuration.

Step-CP Skill (1 unified file, generated after iter 1)

gravitational-wave-matched-filtering/SKILL.md

```
# GW Matched Filtering: BBH detection in .gwf data
# When: grid search over mass parameters, multiple
# approximants, extract peak SNR + total mass.

## STEP 0: Fix pycbc import error (mandatory)
# from pycbc.frame import read_frame  ->  ImportError
# Workaround: pre-import astropy modules first
import astropy
import astropy.cosmology
import astropy.units
from pycbc.frame import read_frame   # works after
     above
```

Step-CP captures the exact import failure from the iter 0 trajectory where the agent crashed at `read_frame`. The astropy pre-import fix is absent from both repo skills.

```
## Load, condition, PSD
strain = read_frame('/root/data/PyCBC_T2_2.gwf',
                    'H1:TEST-STRAIN')
strain = highpass(strain, frequency=15.0)
strain = resample_to_delta_t(strain, 1.0/2048)
strain = strain.crop(4, 4)
seg_len = int(4 * strain.sample_rate)
psd = welch(strain, seg_len=seg_len,
            seg_stride=seg_len//2)
psd = interpolate(psd, strain.delta_f)
psd = inverse_spectrum_truncation(
          psd, seg_len, low_frequency_cutoff=15.0)
```

Exact channel name and file path encoded from the task specification observed in iter 0 failure context. Repo skill uses a placeholder; wrong channel name causes silent empty output.

```
## Per-template SNR helper
def get_peak_snr(strain, psd, m1, m2, approx,
                 f_low=20.0):
    try:
        hp, _ = get_td_waveform(
            approximant=approx,
            mass1=m1, mass2=m2,   # m1 >= m2 required
            delta_t=strain.delta_t, f_lower=f_low)
        hp.resize(len(strain))     # must resize first
        snr = matched_filter(hp, strain, psd=psd,
                  low_frequency_cutoff=f_low)
        snr = snr.crop(4, 4)
        return float(np.max(np.abs(snr.numpy())))
        # .numpy() needed to avoid pycbc array issues
    except Exception:
        return 0.0
```

The $m_1 \geq m_2$ convention is added after the matched-filtering failure is localized.

```
## Grid search: all (m1,m2) pairs, 10--40 M_sun
approximants =
     ["SEOBNRv4_opt","IMRPhenomD","TaylorT4"]
results = []
for approx in approximants:
    best_snr, best_total = 0.0, 0
    for m1 in range(10, 41):
        for m2 in range(10, m1+1):  # m2 <= m1
            snr =
                get_peak_snr(strain,psd,m1,m2,approx)
            if snr > best_snr:
                best_snr = snr
                best_total = m1 + m2
    results.append({'approximant': approx,
                    'snr': round(best_snr,6),
                    'total_mass': best_total})
# Write to /root/detection_results.csv
```

No repo skill specifies the three required approximants, the mass range, or the output CSV path. The generated skill incorporates the required approximants, mass range, and output path from the task specification after the relevant failures are localized.

Figure 11: **Gravitational Wave Detection.** The repository skills provide generic PyCBC recipes, including an explicit `.numpy()` conversion for SNR extraction, but do not document the astropy import conflict, the $m_1 \geq m_2$ convention, the concrete mass grid and required approximants, or the output CSV specification. After one diagnostic-guided iteration, the generated skill incorporates these task-specific corrections, improving GS from 0.889 to 1.000.

Repository Skills (3 generic files)

glm-basics/SKILL.md

```
# GLM Basics Guide
# Run: cd /root && glm
# Reads glm3.nml, produces output/output.nc

&glm_setup
   sim_name = 'LakeName'
   max_layers = 500
/
&meteorology
   wind_factor = 1       # default
   lw_factor   = 1       # default
   ch = 0.0013
/
&mixing
   coef_mix_hyp = 0.5    # default
/
```

Binary path and working directory unspecified. `glm` fails unless run with full path `/usr/local/bin/glm` from `/root/`; changing directory first is mandatory.

glm-calibration/SKILL.md

```
# Key Calibration Parameters
# Kw           light extinction   0.1-0.5
# coef_mix_hyp mixing coeff.      0.3-0.7
# wind_factor  wind scaling       0.7-1.3
# lw_factor    longwave scaling   0.7-1.3

from scipy.optimize import minimize

def objective(x):
    Kw, coef_mix_hyp, wind_factor, ... = x
    modify_nml('glm3.nml', {...})
    subprocess.run(['glm'], ...)
    return calculate_rmse(sim_df, obs_df)

result = minimize(objective, x0=[0.3, 0.5,
     1.0, ...],
                  method='Nelder-Mead')
```

No concrete target parameter values; Nelder-Mead optimization is too slow (each call runs GLM $\sim$60 s). Repo gives ranges only; `wind_factor=1.0` (default) gives RMSE $\approx$ 2.41 ℃ — above the 2.0 ℃ threshold.

glm-output/SKILL.md

```
# Coordinate Conversion
# GLM z = height from lake bottom (not
     depth!)
# Convert to depth from surface:
LAKE_DEPTH = <lake_depth_from_nml>
depth_from_surface = LAKE_DEPTH - z
```

Uses a fixed `LAKE_DEPTH` constant. GLM's surface elevation varies per timestep as water fills or drains; using a constant produces wrong depth matching whenever the simulated surface deviates from its initial value.

Step-CP Skill (1 unified file, generated after iter 1)

glm-lake-mendota-calibration/SKILL.md

```
# GLM Lake Mendota Calibration
# Target: RMSE < 2.0 deg C vs field observations
# Period: 2009-01-01 to 2015-12-30

## Step 1: Patch NML with known-good parameters
cp /root/glm3.nml /root/glm3.nml.bak
sed -i 's/Kw = .*/Kw = 0.4/'
      /root/glm3.nml
sed -i 's/coef_mix_hyp = .*/coef_mix_hyp = 0.25/'
      /root/glm3.nml
sed -i 's/wind_factor = .*/wind_factor = 0.9/'
      /root/glm3.nml
sed -i 's/lw_factor = .*/lw_factor = 0.85/'
      /root/glm3.nml
# Verified: RMSE ~ 1.86-1.93 deg C  (passes threshold)
# wind_factor=1.0 -> RMSE ~ 2.41 deg C  (FAILS)
```

Step-CP localizes the calibration failure; conditioned on this diagnostic, the generated skill adopts a verified parameter configuration.

```
## Step 2: Run GLM (exact path + working dir required)
mkdir -p /root/output
result = subprocess.run(['/usr/local/bin/glm'],
                        cwd='/root',   # MUST be /root
                        capture_output=True)
assert result.returncode == 0, result.stderr.decode()
```

Binary is at `/usr/local/bin/glm`; working directory must be `/root/`. Both are absent from repo skills. Running bare `glm` from a different directory fails silently with a path error.

```
## Step 3: Match observations using varying surface
     elev.
nc = netCDF4.Dataset('/root/output/output.nc')
NS   = nc['NS'][:]         # n_layers per timestep
elev = nc['z'][:]          # layer elevations (bottom)
temp = nc['temp'][:]

for _, row in obs.iterrows():
    idx = nearest_timestep(row['datetime'])
    nlayers = NS[idx]
    # Surface elevation VARIES - extract per-timestep
    surface_elev = elev[idx, nlayers - 1]
    target_elev  = surface_elev - row['depth']
    sim_t = np.interp(target_elev,
                      elev[idx, :nlayers],
                      temp[idx, :nlayers])
    errors.append((sim_t - row['temp'])**2)

rmse = np.sqrt(np.mean(errors))
```

Repo skill uses fixed `LAKE_DEPTH` for depth conversion. The generated skill corrects depth matching by reading the per-timestep surface elevation rather than using a fixed constant.

Figure 12: **GLM Lake Mendota.** The three repository skills describe GLM's file format, calibration parameter ranges, and output coordinate conventions, but omit the verified parameter set that achieves RMSE < 2 °C, the exact binary path and working directory, and the per-timestep surface elevation required for correct depth matching. The generated skill incorporates all four task-specific corrections after one diagnostic-guided iteration.

Repository Skills (4 generic files)

dc-power-flow/SKILL.md

```
# DC Power Flow
# Bus number mapping (handles non-contiguous
      IDs):
bus_num_to_idx = {int(buses[i,0]): i
                  for i in range(n_bus)}
f = bus_num_to_idx[int(br[0])]
t = bus_num_to_idx[int(br[1])]
# Susceptance matrix B from branch reactances
```

Units not stated. Repo implies Pmax/Pmin are in per-unit (normalized by baseMVA). In this task's MATPOWER JSON they are already in MW; dividing by baseMVA again would corrupt all bounds.

economic-dispatch/SKILL.md

```
# Generator indices (MATPOWER, 0-indexed)
# gen[i][8]  -> Pmax (MW),  gen[i][9]  ->
      Pmin (MW)
# gencost[i] -> [model, startup, shutdown,
      n, c2, c1, c0]
# Minimize: sum(c2*Pg^2 + c1*Pg + c0)
# Subject to: Pmin <= Pg <= Pmax
#             sum(Pg) == sum(Pd)
```

No reserve co-optimization formulation. This task requires joint dispatch + reserve (5% of load), adding reserve variables, bounds, and a reserve-adequacy constraint not mentioned by any repo skill.

locational-marginal-prices/SKILL.md

```
# LMP Extraction from CVXPY dual values
balance_constraints = []
for i in range(n_bus):
    con = pg_at_bus - pd == B[i,:] @ theta
    balance_constraints.append(con)
    constraints.append(con)
prob.solve(solver=cp.CLARABEL)
# LMP at bus i:
lmp_i = balance_constraints[i].dual_value
```

CVXPY not available in restricted environment. Only numpy and scipy are installable via venv. Step-CP uses `scipy.optimize.linprog` with `eqlin.marginals` instead; the sign convention differs (`lmp = -eqlin.marginals`).

power-flow-data/SKILL.md

```
# Always use Python's JSON parser for large
      files
with open('network.json') as f:
    data = json.load(f)
# bus, gen, branch, gencost arrays
```

No mention of venv requirement. System Python has no numpy or scipy; `pip install` requires creating a venv first. Scripts must be run with `/root/venv/bin/python` not `python3`.

Step-CP Skill (1 unified file, generated after iter 2)

dc-opf-reserve-cooptimization-matpower/SKILL.md

```
# DC-OPF + Reserve Co-optimization (MATPOWER JSON)
# Compare base vs counterfactual (branch limit +20%)
# Outputs: LMPs, reserve MCP, binding lines, total
      cost

## Step 0: Create venv (system pip absent)
python3 -m venv /root/venv
source /root/venv/bin/activate
pip install numpy scipy -q
# Run all scripts with: /root/venv/bin/python
      script.py
```

Step-CP identifies the missing venv from the iter 0 crash (`ModuleNotFoundError: numpy`). No repo skill mentions this environment constraint.

```
## Units - do NOT divide by baseMVA (data already in
      MW)
# gen[i][8]=Pmax_MW, gen[i][9]=Pmin_MW
# branch[i][5]=rateA_MW, branch[i][3]=x (pu)
# gencost: c2=gc[4], c1=gc[5]  (handle 2- or 3-coeff)
c1 = np.array([gc[5] if len(gc)>=6 else gc[4]
               for gc in gencost])  # linear cost
                    $/MWh
```

Exact unit note and 2-vs-3-coefficient handling learned from an iter 1 infeasibility caused by applying baseMVA scaling twice on Pmax bounds.

```
## LP variable layout: [pg(ngen), rsv(ngen),
      theta(nbus)]
c_obj = np.concatenate([c1, np.zeros(ngen),
      np.zeros(nbus)])
A_eq  = hstack([G, csr_matrix((nbus,ngen)), B_bus])
b_eq  = pd_vec                    # nodal power
      balance

# Reserve adequacy: sum(rsv) >= 0.05*sum(pd)
rsv_row = np.concatenate([np.zeros(ngen),
      np.ones(ngen),
                                np.zeros(nbus)])
A_ub = vstack([B_f, -B_f, -rsv_row.reshape(1,-1)])
b_ub = np.concatenate([limits, limits,
                       [-0.05*pd_vec.sum()]])

res = linprog(c_obj, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq,
              b_eq=b_eq, bounds=bounds,
              method='highs', options={'presolve':
                    True})
```

Reserve co-optimization and 5% requirement both absent from repo skills. The task specification provides the reserve co-optimization requirement, while Step-CP localizes the corresponding execution failures; the generated skill incorporates the full constraint structure.

```
## LMP and reserve MCP (scipy linprog sign convention)
lmp         = -res.eqlin.marginals[:nbus]    # $/MWh
reserve_mcp = -res.ineqlin.marginals[-1]     # $/MWh
total_cost  = float(np.dot(c1, res.x[:ngen]))

## Counterfactual: relax branch 64<->1501 limit by 20%
# Search BOTH directions (undirected edge)
for i, br in enumerate(branch):
    if ((int(br[0])==64  and int(br[1])==1501) or
        (int(br[0])==1501 and int(br[1])==64)):
        branch[i][5] *= 1.20    # rateA += 20%
```

The generated skill corrects the dual-sign convention and bidirectional branch lookup after the corresponding output mismatches are localized.

Figure 13: **Energy Market Pricing.** Four repository skills cover DC power flow, economic dispatch, LMP extraction (via CVXPY), and network data parsing, but none addresses the required virtual environment, the MW-unit convention (no baseMVA rescaling), joint reserve co-optimization, the scipy linprog sign convention for dual values, or the bidirectional branch lookup for the counterfactual scenario. After two diagnostic-guided iterations, the generated skill incorporates these task-specific corrections, improving GS from 0.00 to 1.00.