

EvoSkills: 通过协同演化验证实现智能体技能的自我演化

Hanrong Zhang^{1*} Shicheng Fan^{1*} Henry Peng Zou¹ Yankai Chen^{2,3†}
 Zhenting Wang² Jiayu Zhou⁴ Chengze Li¹ Wei-Chieh Huang¹ Yifei Yao⁵
 Kening Zheng¹ Xue (Steve) Liu^{2,3} Xiaoxiao Li⁶ Philip S. Yu¹

¹University of Illinois Chicago ²MBZUAI ³McGill University

⁴Columbia University ⁵Zhejiang University ⁶University of British Columbia

{hzhan135, psyu}@uic.edu xiaoxiao.li@ece.ubc.ca

Abstract

Anthropic 提出了“技能”这一概念，用于解决大模型智能体在处理多步骤专业任务时面临的挑战，这类任务无法通过简单的工具调用完成。工具是单一的、自包含的函数，而技能则是一组结构化的、相互依赖的多文件资源包。目前，技能生成不仅因人工编写而具有高标签成本，还可能因人机认知错配问题导致智能体性能下降，这一点已在 SkillsBench 评测中得到验证。因此，我们致力于使智能体能够自主生成技能。然而，现有针对工具设计的自演化方法由于技能本身的复杂性，无法直接应用于技能生成。为解决上述问题，我们提出 EvoSkills——一种自演化技能框架，使智能体能够自主构建复杂的多文件技能包。具体而言，EvoSkills 结合了一个迭代优化技能的技能生成器与一个代理验证器，该验证器无需访问真实测试内容即可协同演化，提供具有信息量且可操作的反馈。在 SkillsBench 上，EvoSkills 在 Claude Code 与 Codex 两种模型上均超越五种基准方法，取得最高的通过率，并展现出对六种额外大模型的强大泛化能力。

1 引言

大型语言模型智能体通过工具使用和 API 调用取得了快速进展 (Yao et al., 2022; Schick et al., 2023; Zhang et al., 2024; Qin et al., 2023; Patil et al., 2023)。然而，专业性的开放任务，如复杂的软件修复、多步骤科学分析以及企业级数据流水线编排，远不止于孤立的工具调用。智能体必须在多个步骤和产物之间协调一致的流程：目标分解、工具协同、故障恢复以及中间输出的验证。这一过程极具挑战性，因为决策具有长时程特性，指令与脚本紧密耦合，且可靠的环境反馈往往稀疏或延迟。

为了弥合这一差距，Anthropic 提出了智能体技能的概念 (Anthropic, 2025c)。Fig. 1 展示了工具与技能之间的区别：工具通常是一个简单的函数，而技能则是一套结构化的工作流指令、可执行脚本以及领域参考文献的组合包 (Xu & Yan, 2026)。根据 SkillsBench (Li et al., 2026b) 中所呈现的系统性评估，为智能体配备精心设计的技能，能够在包括软件工程和科学分析等在内的广泛专业领域中持续提升性能，证实了结构化的程序指导相较于仅提供基础工具访问，显著增强了任务求解能力。

*Equal contribution.

†Corresponding author: yankaichen@acm.org

尽管技能展现出的实用性已得到证明，当前范式几乎完全依赖人工编写，这一过程**既费时又难以扩展，且无法系统性地保证输出质量**。如图 6 中所示，SkillsBench 评估 (Li et al., 2026b) 表明，人工编写的技能带来的提升极不均衡：某些领域受益显著，而其他领域（如自然科学）在技能整合后甚至表现出性能下降。我们假设导致这种不一致性的关键因素是**人机认知错位**：为人类专家设计直观的工作流和抽象方式，并未自然契合大语言模型智能体在执行约束下处理上下文、推理和行动的方式。

为了减少人工投入，近期的方法已从预先定义静态工具或 API 转向由大语言模型智能体自身进行自演化工具或工具库 (Chen et al., 2026; Li et al., 2026a; Lu et al., 2026; Wang et al., 2023; Xia et al., 2025)。然而，这些方法存在根本性的工具-技能鸿沟：它们本质

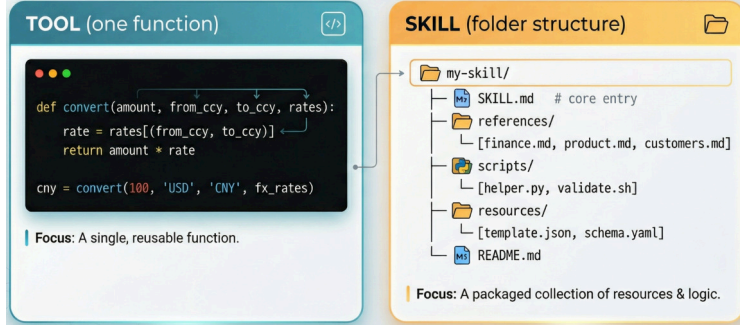


图 1: 工具与技能差异示意图。

上是为一次性生成简单、自包含的函数而设计的，在创建结构化、多文件的技能包方面表现不足，而这类技能包需要协调工作流指令、可执行脚本以及跨多个实体的领域参考信息。此外，工作 Alzubi et al. (2026) 严重依赖真实标签监督进行故障诊断，这限制了其在真实场景中的应用，因为在这些场景中此类信号通常不可用。

为应对这些挑战，我们提出了一种自演化技能框架 EvoSkills。为克服 one-shot 多文件技能生成固有的不可靠性，我们设计了一个核心组件——Skill Generator，该组件通过迭代方式生成并优化技能包，在演化轮次中持续提升技能质量，如图 2 所示。它还维护一个持久的对话上下文，用于累积来自另一个核心组件——Surrogate Verifier

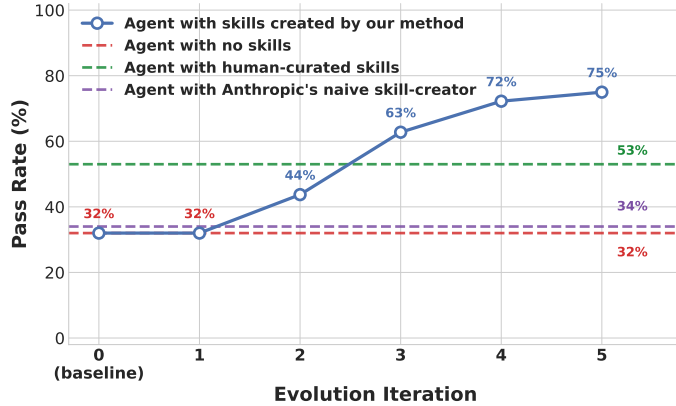


图 2: 5 轮演化中技能质量的提升。在 5 轮演化迭代中，EvoSkills 超过了人工标注的技能。

——在迭代过程中提供的高保真反馈。该 Surrogate Verifier 是一个分离的 LLM 会话，不继承生成器的偏见，旨在解决现实世界中缺乏真实反馈的问题。它根据任务指令和环境合成测试用例和脚本，并向 Skill Generator 提供高保真反馈，以协同演化的方式持续提升技能生成质量。

总结而言，我们的主要贡献如下：① 我们提出了 EvoSkills，一种用于大模型智能体自演化鲁棒多文件技能包的协同演化框架。② 我们揭示了自主技能生成的关键洞见：(i) 智能体生成的技能优于人工设计的技能，因其能够捕捉智能体实际所需的推理模式和工具使用策略；(ii) 自演化技能可在不同模型家族间迁移，因为演化后的技能包编码的是可复用的任务结构，而非特定模型的特征。③ 我们在 SkillsBench 上进行了大量实验，EvoSkills 达到最

高的通过率 71.1% (较无技能基准提升 40.5 个百分点), 显著超越所有五种基准。此外, 由单一前沿大模型演化出的技能可有效迁移至来自五家公司的六种额外大模型, 使其相对于各自无技能基准获得 35–45 个百分点的提升。

2 相关工作

大语言模型智能体技能。 Anthropic 提出了 Agent Skills (Anthropic, 2025c), 如 Fig. 1 所示。一项近期的系统化研究 (Jiang et al., 2026) 进一步将技能与原子工具及一次性计划区分开来, 将其定义为具有明确适用条件和终止条件的可重用模块。SkillsBench (Li et al., 2026b) 提供了首个用于评估智能体技能的系统性基准, 包含跨 11 个领域的 87 项任务, 并采用确定性验证器。若干基于学习的方法尝试缩小这一差距。SAGE (Wang et al., 2025) 在一系列相关任务上训练智能体, 并引入融合技能的奖励机制, 但生成的技能仍为单文件的程序化函数, 而非结构化的多文件包。SkillRL (Xia et al., 2026) 通过强化学习将轨迹蒸馏为层次化的技能库, 但依赖教师引导的蒸馏过程, 且生成的是提示层面的启发式规则, 而非可执行的实体。EvoSkills 则生成结构化的多文件技能包, 并通过迭代验证对其进行演化。

自演化大语言模型智能体 越来越多的研究工作致力于自动化智能体能力的自我改进, 然而现有方法存在两个反复出现的缺陷。首先, 大多数自演化流水线仅能生成单一工具或函数 API, 或仅是提示启发式规则 (Wang et al., 2023; Li et al., 2026a; Xia et al., 2025; Lu et al., 2026; Chen et al., 2026), 无法构建完整技能包所需的多文件结构。此外, AutoSkill (Yang et al., 2026) 和 AutoRefine (Qiu et al., 2026) 将可复用知识提取为提示模板而非可执行包, 而 SEAgent (Sun et al., 2025) 则将能力内化到模型权重中, 导致其不可检查且不可迁移 (Jiang et al., 2026)。其次, 某些方法严重依赖真实标签信号进行失败诊断, 当此类监督不可用时, 其适用性受到限制 (Alzubi et al., 2026; Sun et al., 2025)。EvoSkills 通过迭代生成和演化结构化的多文件技能包, 解决了上述两个局限; 同时采用信息隔离的代理验证机制, 提供结构化的失败诊断与反馈, 而不依赖真实标签信号。

3 方法

3.1 方法概述

为回答 Sec. 1 中提出的研究问题, 必须克服两个核心难点: (1) 在单次遍历中生成多文件技能包本质上不可靠; 以及 (2) 智能体在自我演化过程中缺乏真实反馈。EvoSkills 通过协同演化智能体的技能与相应的替代验证器来解决这两个挑战。Fig. 3 展示了整体框架, Alg. 1 形式化了完整的协同演化过程。给定任务输入, 技能生成器生成候选技能并执行以获得任务输出。一个信息隔离的替代验证器随后独立生成并演化测试断言以应对这些输出, 将结构化的失败诊断反馈回生成器。这两个组件通过**迭代生成-验证-优化**循环协同演化: 每当替代测试通过时, 一个真实目标的测试会在全新环境中重新执行该技能, 并仅返回一个不透明的成功/失败信号。如果真实目标测试通过, 则最终演化的技能被部署到目标大语言模型智能体 (例如 Claude Code (Anthropic, 2025a) 和 Codex (OpenAI, 2025)); 否则, 该信号将触发新一轮协同演化迭代, 此时验证器会升级其测试, 而生成器则相应地优化技能。接下来, 我们在 Sec. 3.2 中将任务情景形式化为部分可观测马尔可夫决策过程 (POMDP), 并在 Sec. 3.3 中详细展示 EvoSkills 框架。

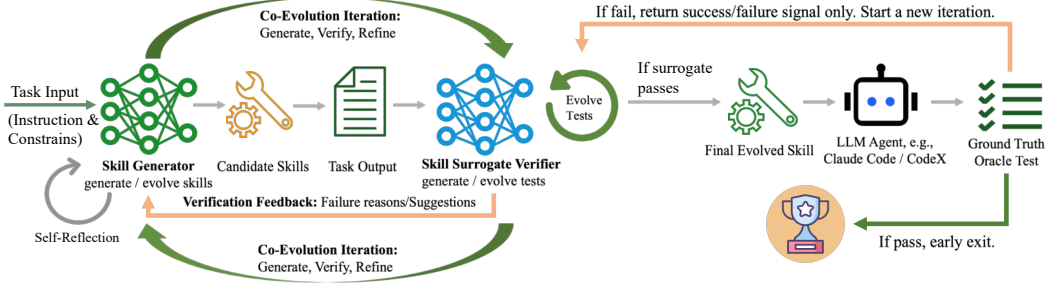


图 3: EvoSkills 演化框架概述。技能生成器与替代验证器通过迭代优化共同演化。验证器提供结构化的失败反馈以推动技能改进，而真实情况的评判测试仅返回一个不透明的通过/失败信号，触发测试升级并确保严格的信息隔离。

3.2 问题表述

任务定义。 由于大语言模型智能体始终无法知晓被保留的真实测试实际检查的内容，即成功标准完全隐藏，我们将任务环境定义为一个部分可观测马尔可夫决策过程 $\mathcal{M} = \langle \mathcal{X}, \mathcal{A}, T, \mathcal{O}, \Omega, \mathcal{R} \rangle$ 。其中， \mathcal{X} 为潜在状态空间，即完整的文件系统与进程， \mathcal{A} 包含智能体的动作，即终端命令和文件编辑， $T(x' | x, a)$ 为在状态 x 执行动作 a 后的确定性状态转移并到达后继状态 x' ， \mathcal{O} 为观测空间，即命令执行结果， $\Omega(o | x, a)$ 将动作后的状态映射到部分观测， $\mathcal{R}(x_T) \in [0, 1]$ 在最终状态 x_T 对输出文件进行评估，以对比隐藏的真实测试。由于智能体仅接收部分观测 $o_t \sim \Omega(\cdot | x_t, a_t)$ ，因此其行为基于观测-动作历史 $h_t = (o_1, a_1, \dots, a_{t-1}, o_t)$ 。

最优化目标。 与提供简单功能接口的原子工具不同，一个技能 \mathcal{S} (Anthropic, 2025c) 是一个结构化的领域特定指令、可执行脚本和参考资料的集合，共同引导智能体在任务空间中导航。该技能对智能体的策略进行条件约束：

$$a_t \sim \pi_\theta(a_t | h_t, \mathcal{S}), \quad (1)$$

其中 π_θ 是一个大语言模型策略。我们定义在技能 \mathcal{S} 下的期望终端奖励为：

$$J(\mathcal{S}) \triangleq \mathbb{E}_{\tau \sim P(\tau | \pi_\theta, \mathcal{S}, \mathcal{M})} [\mathcal{R}(x_T)], \quad (2)$$

其中 $\tau = (o_1, a_1, \dots, a_{T-1}, o_T)$ 是执行轨迹。我们的目标是发现一个最优技能 \mathcal{S}^* ，使其最大化 J ：

$$\mathcal{S}^* = \arg \max_{\mathcal{S}} J(\mathcal{S}). \quad (3)$$

3.3 EvoSkills 框架

然而，直接优化 $J(\mathcal{S})$ (Eq. 3) 是不可行的：真实标签评估在计算上代价高昂，并且仅返回一个不透明的通过/失败信号；智能体无法获知测试内容或失败详情。为了提供稠密且可操作的反馈，我们引入了一个代理验证器奖励 $\tilde{\mathcal{R}}(x, \mathcal{V})$ ，该奖励由一组由独立验证器生成的确定性测试断言 $\mathcal{V} = \{e_1, \dots, e_{|\mathcal{V}|}\}$ 定义

$$\tilde{\mathcal{R}}(x, \mathcal{V}) \triangleq \frac{1}{|\mathcal{V}|} \sum_{k=1}^{|\mathcal{V}|} \mathbf{1}[e_k(x)] \in [0, 1], \quad (4)$$

其中 x 表示技能执行产生的输出文件， $\mathbf{1}[e_k(x)]$ 表示断言 e_k 在 x 上是否通过。然而，代理模型仅在能够准确逼近隐藏的 \mathcal{R} 时才有用。这导致了一个耦合的最优化问题：技能必须最大化一个代理指标，而该代理指标本身又必须与隐藏的真实值对齐。由于真实值 (GT) 预

Algorithm 1 EvoSkills co-evolution Algorithm

Require: Instruction I , environment \mathcal{E} , meta-skill $\mathcal{S}_{\text{meta}}$ (skill-creator)
 Require: Evolution iters $N=5$, surrogate iters $M=15$, context cap $\beta=0.7$
 Require: LLM policy π_θ (generator), independent verifier policy π_θ^V
 Ensure: Final evolved skill \mathcal{S}^*

```

1:  $C \leftarrow (I, \mathcal{S}_{\text{meta}})$  // in: instruction  $I$ , meta-skill; out: initial context  $C$ 
2:  $\mathcal{S}^{(0)} \sim \pi_\theta(\cdot | C)$  // in: context  $C$ ; out: skill bundle  $\mathcal{S}^{(0)}$  (code + SKILL.md)
3:  $\mathcal{V}^{(0)} \leftarrow \emptyset$  // surrogate verifier test suite
4:  $i \leftarrow 0; j \leftarrow 0; n \leftarrow 0; r \leftarrow 0$  // skill version; test-suite version; evolution iter.; surrogate iter.
5:  $\mathcal{R}_{\text{best}} \leftarrow 0; \mathcal{S}^* \leftarrow \mathcal{S}^{(0)}$  //  $\mathcal{R}_{\text{best}}$ : best oracle score so far
6: while  $n < N$  and  $r < M$  do
7:   // — Skill Generator: execute and produce outputs —
8:    $x^{(i)} \leftarrow \Phi(\mathcal{S}^{(i)}, \mathcal{E})$  //  $\Phi$ : execute skill in env and collect outputs; out: artifacts  $x^{(i)}$ 
9:   if LLM context usage proportion  $> \beta$  then
10:    break // prevent LLM context overflow
11:   end if
12:   // — Surrogate Verifier: evaluate and refine (Eq. 4, Eq. 5) —
13:    $\tilde{\mathcal{R}}^{(i,j)} \leftarrow \tilde{\mathcal{R}}(x^{(i)}, \mathcal{V}^{(j)})$  //  $\tilde{\mathcal{R}}$ : surrogate reward; in: artifacts, tests; out: pass rate  $\in [0, 1]$ 
14:   if  $\tilde{\mathcal{R}}^{(i,j)} < 1$  then
15:      $\mathcal{F}^{(i,j)} \sim \pi_\theta^V(\cdot | I, x^{(i)}, \mathcal{V}^{(j)})$  // in:  $I$ , artifacts  $x^{(i)}$ , tests  $\mathcal{V}^{(j)}$ ; out: diagnostic  $\mathcal{F}$ 
16:      $C \leftarrow C \oplus \mathcal{F}^{(i,j)}$  // append error diagnostic  $\mathcal{F}$  to generator context  $C$ 
17:      $\mathcal{S}^{(i+1)} \sim \pi_\theta(\cdot | \mathcal{S}^{(i)}, C)$  // skill refinement (Eq. 7)
18:      $i \leftarrow i+1; r \leftarrow r+1$ ; continue // evolve  $\mathcal{S}$ ;  $\mathcal{V}^{(j)}$  locked
19:   end if
20:   // — Ground-Truth Oracle Test: independent re-execution in fresh environment —
21:    $\hat{x}^{(i)} \leftarrow \Phi(\mathcal{S}^{(i)}, \mathcal{E}')$  // in: skill  $\mathcal{S}^{(i)}$ , fresh env  $\mathcal{E}'$ ; out: artifacts  $\hat{x}^{(i)}$ 
22:    $\mathcal{R}^{(i)} \leftarrow \mathcal{R}(\hat{x}^{(i)}); n \leftarrow n+1$  //  $\mathcal{R}$ : ground-truth oracle reward; out: score  $\in [0, 1]$ 
23:   if  $\mathcal{R}^{(i)} = 1$  then
24:      $\mathcal{S}^* \leftarrow \mathcal{S}^{(i)}$ ; return  $\mathcal{S}^*$  // early exit: perfect score
25:   else if  $\mathcal{R}^{(i)} > \mathcal{R}_{\text{best}}$  then
26:      $\mathcal{R}_{\text{best}} \leftarrow \mathcal{R}^{(i)}; \mathcal{S}^* \leftarrow \mathcal{S}^{(i)}$  // save best snapshot
27:   end if
28:   // — Co-evolution: test escalation (Eq. 6, Eq. 8) —
29:    $C \leftarrow C \oplus \mathbf{1}[\mathcal{R}^{(i)} < 1]$  //  $\mathbf{1}[\cdot]$ : indicator fn; append oracle pass/fail bit to  $C$  (no test content)
30:    $\mathcal{V}^{(j+1)} \sim \pi_\theta^V(\cdot | I, x^{(i)}, \mathcal{V}^{(j)})$  // verifier escalation (Eq. 8)
31:    $j \leftarrow j+1$  //  $\mathcal{V}$  evolves;  $\mathcal{S}^{(i)}$  re-evaluated next iteration
32: end while
33: return  $\mathcal{S}^*$  // save best skill

```

言机测试仅提供二元的通过/失败信号 $\mathbf{1}[\mathcal{R}(\hat{x}^{(i)}) < 1]$ ，其中 $\hat{x}^{(i)}$ 表示预言机独立重新执行的输出，且不包含任何测试内容，因此我们无法直接针对 \mathcal{R} 进行优化。相反，令 I 表示任务指令，我们定义回滚算子 $\Phi(\mathcal{S}, \mathcal{E})$ 为在环境 \mathcal{E} 中回滚 $\pi_\theta(\cdot | h_t, \mathcal{S})$ 所获得的执行输出，使得 $x^{(i)} = \Phi(\mathcal{S}^{(i)}, \mathcal{E})$ 为第 i 版本技能的输出， $\hat{x}^{(i)} = \Phi(\mathcal{S}^{(i)}, \mathcal{E}')$ 为在全新环境 \mathcal{E}' 中由独立预言机重新执行所产生的输出。 $\mathcal{V}^{(j)}$ 为代理验证器测试套件的第 j 版本。EvoSkills 通过交

替精炼的方式进行：

$$\text{Skill refinement: } \mathcal{S}^{(i+1)} \leftarrow \arg \max_{\mathcal{S}} \tilde{\mathcal{R}}(\Phi(\mathcal{S}, \mathcal{E}), \mathcal{V}^{(j)}), \quad (5)$$

$$\text{Test escalation: } \mathcal{V}^{(j+1)} \sim \pi_{\theta}^V(\cdot \mid I, x^{(i)}, \mathcal{V}^{(j)}), \text{ if } \mathbf{1}[\tilde{\mathcal{R}}(x^{(i)}, \mathcal{V}^{(j)})=1 \wedge \mathcal{R}(x^{(i)})<1] \quad (6)$$

在实践中， $\arg \max$ 通过迭代的 LLM 采样 (Eq. 7) 来近似 Eq. 5；技能精炼在固定的验证器测试套件 $\mathcal{V}^{(j)}$ 下最大化 $\tilde{\mathcal{R}}$ ；只有当预言机的二元信号暴露了 $\tilde{\mathcal{R}}$ 与 \mathcal{R} 之间的差距时，才会触发测试升级，迫使验证器独立强化其测试，而无需访问真实测试内容。EvoSkills 通过三个信息隔离的组件实现这一交替最优化：一个由技能生成器和替代验证器构成的协同演化环 (Alg. 1)。

技能生成器 单次遍历技能生成通常会存在覆盖缺口和逻辑错误的技能包，因为智能体在生成过程中缺乏真实标签的反馈。为了实现迭代优化，Skill Generator 维持一个持久的对话上下文 C ，初始值为 $C^{(0)} = (I, \mathcal{S}_{\text{meta}})$ ，其中 I 为任务指令， $\mathcal{S}_{\text{meta}}$ 为领域无关的元技能 (skill-creator)，用于指导如何创建技能。每次技能修订均基于前一版本，大语言模型 π_{θ} (Eq. 1) 读取当前技能 $\mathcal{S}^{(i)}$ 以及累积的验证反馈，并生成改进版本 (Alg. 1, 第 15–17 行)：

$$\mathcal{S}^{(i+1)} \sim \pi_{\theta}(\cdot \mid \mathcal{S}^{(i)}, C^{(i+1)}), \quad C^{(i+1)} = C^{(i)} \oplus \mathcal{F}^{(i,j)}, \quad (7)$$

其中 $\mathcal{S}^{(i)}$ 是第 i 种技能版本， $\mathcal{F}^{(i,j)}$ 是在测试套件 $\mathcal{V}^{(j)}$ 下评估 $\mathcal{S}^{(i)}$ 后，由代理验证器提供的失败诊断，包括失败的测试用例、根本原因分析和可操作的修订建议，而 \oplus 会将详细反馈附加到 LLM 上下文中。智能体通过 rollout $x^{(i)} = \Phi(\mathcal{S}^{(i)}, \mathcal{E})$ (第 8 行) 执行 $\mathcal{S}^{(i)}$ ，并将生成的结果传递给 Surrogate Verifier 进行评估。

代理验证器 由于真实奖励 \mathcal{R} 仅返回一个不透明的通过/失败信号，技能生成器缺乏稠密反馈以诊断和纠正错误。替代验证器通过作为 \mathcal{R} 的代理，弥补了这一缺陷，提供每个断言的失败诊断信息，这是不透明的真值源无法提供的。为进一步防止自验证中固有的确认偏差，替代验证器在完全独立的 LLM 会话 π_{θ}^V 中运行，仅观察任务指令 I 和输出文件 $x^{(i)}$ ，对技能生成器的推理过程、代码及技能内容保持不可见。这种信息隔离确保了验证器的测试生成与生成器的内部状态条件独立，避免验证器继承生成器的偏见。

验证器生成一个代理验证器测试集 $\mathcal{V} = \{e_1, \dots, e_{|\mathcal{V}|}\}$ ，包含确定性断言，从而得到在 Eq. 4 (Alg. 1, 第 13 行) 中定义的代理奖励 $\tilde{\mathcal{R}}(x, \mathcal{V})$ 。验证器通过读取之前的脚本 $\mathcal{V}^{(j)}$ 和当前输出 $x^{(i)}$ ，迭代地优化该验证器测试集：

$$\mathcal{V}^{(j+1)} \sim \pi_{\theta}^V(\cdot \mid I, x^{(i)}, \mathcal{V}^{(j)}), \quad (8)$$

其中，验证器基于任务指令 I 、智能体当前的输出 $x^{(i)}$ 以及其自身的先前测试脚本 $\mathcal{V}^{(j)}$ ，生成一个改进的验证器测试套件 $\mathcal{V}^{(j+1)}$ 。当代理奖励指示失败 ($\tilde{\mathcal{R}} < 1$) 时，验证器还会生成一个结构化的失败诊断 $\mathcal{F}^{(i,j)} \sim \pi_{\theta}^V(\cdot \mid I, x^{(i)}, \mathcal{V}^{(j)})$ ，包括每个断言的结果、根本原因分析以及可操作的修改建议，并反馈给技能生成器 (Eq. 7)。

技能生成器与代理验证器的共演化。 Eq. 5–Eq. 6 中的交替最优化耦合了两条反馈路径 (Alg. 1)。当代理奖励指示失败 ($\tilde{\mathcal{R}} < 1$) 时，验证器测试集 \mathcal{V} 保持固定，而故障诊断 \mathcal{F} 会驱动技能修订 (Eq. 7)。当代理测试通过但真实目标测试失败时，仅返回一个不透明的通过/失败比特，即没有测试内容或故障详情，以防止 Skill Generator 对保留测试过拟合。因此，Surrogate Verifier 必须根据更新后的技能和测试输出文件独立地升级其测试集 (Eq. 8)。例如，它可以生成更多样化、更全面且更具挑战性的测试用例。通过这种双反馈机制，技能 \mathcal{S} 在代理测试的压力下得以改进。Fig. 2 经验性地证实，这种共演化环在少量迭代内即可收敛。

4 实验

在本节中，我们旨在回答四个研究问题 (RQ): (1) 自演化技能是否有效，能否超越人工设计的技能 (Sec. 4.2)? (2) 该方法的不同组件如何影响整体性能? (Sec. 4.3)? (3) 演化得到的技能能否在其他大型语言模型或来自不同公司的大型语言模型智能体之间迁移 (Sec. 4.4)? (4) 性能提升在各个专业领域中的分布情况如何 (Sec. 4.5)?

4.1 实验设置

我们在 SkillsBench (Li et al., 2026b) 上评估 EvoSkills。该基准包含 87 个任务，覆盖约 20 个专业领域，能够广泛覆盖真实世界中技能增强型任务的分布。每个任务均配备一个确定性验证器，支持可复现的二元通过/失败评估，避免主观的人工判断。

我们采用 SkillsBench 作为唯一的评估套件，因为据我们所知，它是唯一专为评估智能体技能实用性而设计的基准。

主要指标是 **通过率**：奖励为 $= 1.0$ 的任务比例，即所有测试通过；否则奖励为 $= 0.0$ 。除非另有说明，所有条件均使用相同的指令格式。对于预安装技能的条件，任务指令会注明技能的可用性。

我们对比六种基准方法与 EvoSkills。**无技能基准**评估智能体在无任何技能可用时的表现。**自生成技能**复现了 SkillsBench (Li et al., 2026b) 中的一次性自生成条件：智能体在解决任务前一次性生成一至五个技能文档，不进行迭代演化或验证 (见 Sec. F.4 中的提示)。**思维链引导的自生成**在自生成技能的基础上，引入结构化的五步思维链提示 (见 Sec. F.5 中的提示)。Skill-Creator 采用 Anthropic 官方的 skill-creator (Anthropic, 2025c)。由于我们的评估完全自主，我们用自主替代步骤取代其人工交互环节 (见 Sec. F.3)：第一阶段通过至少三次迭代不断草拟、自测并优化技能；第二阶段使用生成的技能解决任务。**人工精选技能**预安装 SkillsBench 发布的人工编写的技能包。对于演化智能体，我们使用 Claude Opus 4.6 和 GPT-5.2 作为基础模型，并为其提供任务背景信息。

在基准对比中，每种主要方法均进行 5 次独立运行，报告均值 \pm 标准差。

此外，我们还评估了 Claude Opus 4.6 演化出的技能在六个额外模型上的可迁移性：GPT-5.2 (Singh et al., 2025)、Claude Sonnet 4.5 (Anthropic, 2025d)、Claude Haiku 4.5 (Anthropic, 2025b)、Qwen3-Coder-480B (Yang et al., 2025)、DeepSeek V3-671B (Liu et al., 2024) 和 Mistral Large 3-675B (Mistral AI, 2025)。每个模型均进行 3 次独立运行评估。请参阅 Appendix A 了解配置详情。

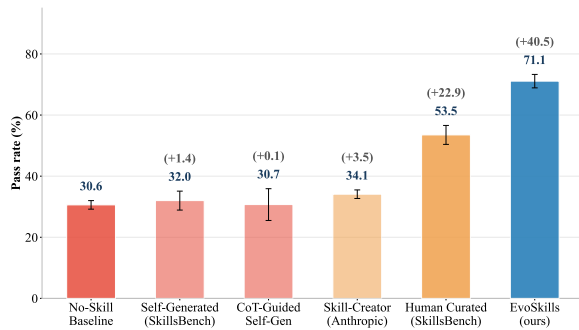


图 4: 技能质量对比基准 (SkillsBench, Claude Opus 4.6 + Claude-Code)。误差条: ± 5 次运行的 1 标准差。

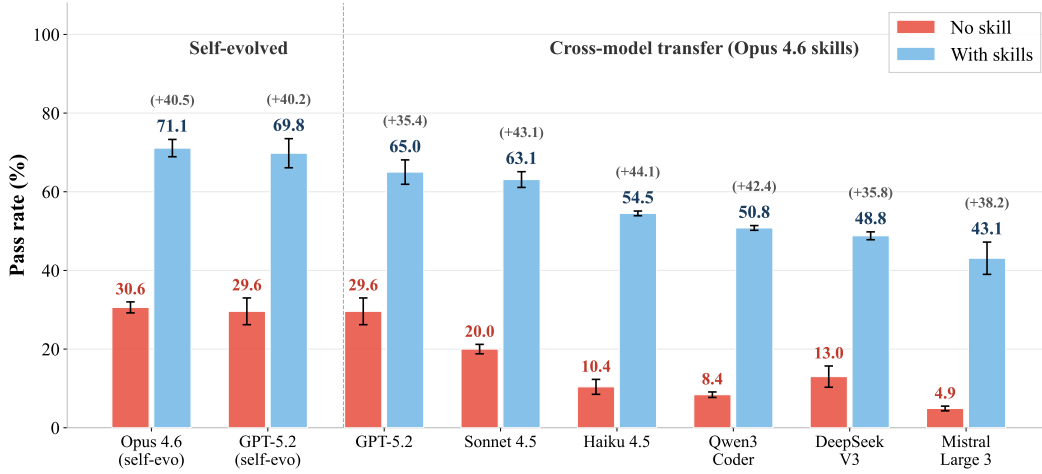


图 5: SkillsBench 上的跨模型技能迁移能力。由 Claude Opus 4.6 演化出的技能被迁移到六个额外模型上，涵盖五个不同提供商。每组柱状图显示无技能基准（红色）和有技能通过率（蓝色）。差值标注表示绝对提升。所有模型均显著受益（提升 36–44 个百分点），证实演化出的技能编码了可复用的任务结构，而非模型特异性特征。

4.2 RQ1: 技能质量比较

Fig. 4 展示了 Claude Opus 4.6 与 Claude-Code 的核心对比。EvoSkills 达到了 71.1% 的通过率，相较于无技能基准（30.6%）提升了 +40.5 个百分点，也优于人工标注技能（53.5%）的 +17.6 个百分点。技能创建基准仅达到 34.1%，略高于无技能基准；而一种更简单的变体——在单个会话中生成并使用技能——仅取得 32.4% 的成绩。两种会话内自我生成的基线表现也未见提升：SkillsBench 自动生成技能的基线 (Li et al., 2026b) 达到 32.0% (± 3.1)，而遵循结构化五步思维链提示的 CoT 引导变体仅达到 30.7% (± 5.2)，均对无技能基准的改进微乎其微。这证实了：无论采用何种生成策略，缺乏共演化验证的技能生成均不足以带来有效提升；EvoSkills 的优势源自迭代验证环，而非技能创建提示本身。各任务的详细分解见 Appendix D。我们总结出的核心洞见如下。

Takeaway 1: Agents can create better skills than humans

能够持续自我演进技能的智能体，其表现始终优于依赖人工设计流程的智能体。自我演进的技能通过编码自身的推理模式、工具使用偏好和分解策略，更有效地捕捉智能体实际所需的工作流，而非遵循人类专家假设智能体应如何运作而设计的流程。

4.3 RQ2: 消融研究

由于页面限制，我们将在 Appendix B 中展示完整的消融分析。关键发现 (Tab. B1)：移除代理验证器会使通过率从 71.1% 降至 41.1%，而使用背景上下文仅获得 48.6% 的通过率。两项结果均证实了迭代验证和结构化封装的重要性。

4.4 RQ3: 技能跨模型迁移能力

在确认 EvoSkills 能够在两种主要大模型上生成高质量技能后，我们接下来考察这些技能是否能在大模型家族间实现泛化。如 Fig. 5 所示，自演化技能在两个主要骨干模型上均带来显著提升：Claude Opus 4.6 (+40.5pp) 和 GPT-5.2 (+40.2pp)。将 Opus 演化得到的技能

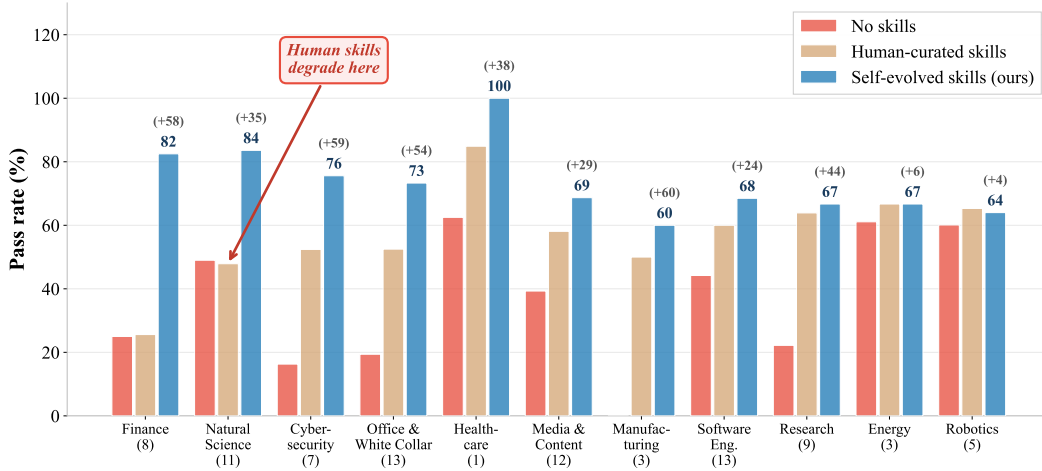


图 6: 各领域的 SkillsBench 通过率。在 11 个专业领域中, 使用 Claude Opus 4.6 比较了三种条件: 无技能基准、人工精选技能, 以及 EvoSkills 自演化技能, 括号中的数字表示任务数量。在 11 个领域中有 9 个领域, 自演化技能的表现优于人工精选技能。箭头突出显示了自然科学领域, 该领域中人工精选技能导致性能下降, 而自演化技能则带来显著提升, 表明人类与机器之间存在认知错配。

迁移至另外六个模型, 均持续优于各无技能基准, 性能提升达 +36 至 +44pp, 表明提炼出的工作流并不依赖于原始模型。完整数值结果见 Tab. A3 (Appendix A)。

有趣的是, GPT-5.2 从 Opus 转移的技能中获益 (65.0%), 但其自身自演化出的技能 (69.8%) 仍比转移的技能高出 4.8pp, 表明模型匹配的演化具有微弱但持续的优势。我们总结主要启示如下。

Takeaway 2: Skills are portable across model families

自演化技能编码的是可重复使用任务结构, 而非模型特定的组件, 从而实现了广泛的跨模型可迁移性。在可行的情况下, 推荐采用与模型匹配的演化方式, 但这并非实现显著提升所必需的。

4.5 RQ4: 领域级分析

除了总体指标外, Fig. 6 揭示了收益在各个领域中的分布情况。自演化技能在 11 个领域中的 9 个领域中优于人工精选技能, 其中金融 (比人工精选高出 +56.9 个百分点) 和网络安全 (高出 +23.2 个百分点) 的差距最大。这种模式并不均匀: 在人工精选技能表现已较好的领域 (如能源、机器人), 演化的回报递减; 而在人工精选技能作用有限的领域, 演化带来的提升最大。我们总结主要结论如下。

Takeaway 3: Human-machine misalignment

在某些领域, 人工精心设计的技能不仅无法提供帮助, 反而会显著降低性能; 而由模型自我演化的技能在同一情境下却能带来显著提升。人类设计的工作流可能与大语言模型智能体的推理方式不匹配, 共演化最优化通过让模型自主发现能够发挥自身优势的流程, 从而弥合这一差距。

4.6 演化动力学

Fig. 2 展示了在三个静态基准下的演化轮次中通过率的迹线: 无技能基准 (30.6%)、Anthropic 的朴素技能生成器 (34.1%) 以及人工精调的技能 (53.5%)。在第 0 轮 (无验证的一次性生

成), EvoSkills 的表现与无技能基准相当, 但一旦开始迭代验证, 通过率迅速上升, 在第 2 轮达到 44%, 在第 3 轮超越人工精调技能 (63%), 并在第 5 轮收敛至 75%。这证实了共演化环是技能质量提升的主要驱动力, 而非生成提示本身, 且在五轮内实现收敛使得演化成本保持在可接受范围内。Appendix C 提供了所有 86 个任务中验证周期和预言者轮次的完整分布: 每个任务平均需要 4.1 次验证周期和 2.4 次演化迭代才能达到收敛。此外, Appendix E 展示了一个代表性演化轨迹的详细迹线。

5 结论

我们提出了 EvoSkills, 一种智能体技能自生成的协同演化框架。该设计克服了 one-shot 技能生成的不可靠性以及现实情景中缺乏真实反馈的问题。在 SkillsBench 上, EvoSkills 显著优于人工标注的技能和所有自生成基准, 同时展现出强大的可迁移性。我们的实验揭示了人机认知错位现象, 而协同演化最优化能够弥合这一差距。未来, 我们计划将该框架扩展至多模型技能演化。

参考文献

- Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. arXiv preprint arXiv:2603.02766, 2026.
- Anthropic. Claude code: An agentic coding tool. <https://code.claude.com/docs>, 2025a.
- Anthropic. Claude haiku 4.5 system card. <https://www.anthropic.com/claude-haiku-4-5-system-card>, 2025b.
- Anthropic. Agent skills overview. <https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview>, 2025c. Accessed: 2026-03-30.
- Anthropic. Claude sonnet 4.5 system card. <https://www.anthropic.com/claude-sonnet-4-5-system-card>, 2025d.
- Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, et al. Skillcraft: Can llm agents learn to use tools skillfully? arXiv preprint arXiv:2603.00718, 2026.
- Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills—beyond tool use in llm agents. arXiv preprint arXiv:2602.20867, 2026.
- Haotian Li, Shijun Yang, Weizhen Qi, Silei Zhao, Rui Hua, Mingzhu Song, Xiaojian Yang, and Chao Peng. Yunjue agent tech report: A fully reproducible, zero-start in-situ self-evolving agent system for open-ended tasks. arXiv preprint arXiv:2601.18226, 2026a.
- Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. arXiv preprint arXiv:2602.12670, 2026b.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.

- Jiaxuan Lu, Ziyu Kong, Yemin Wang, Rong Fu, Haiyuan Wan, Cheng Yang, Wenjie Lou, Haoran Sun, Lilong Wang, Yankai Jiang, et al. Beyond static tools: Test-time tool evolution for scientific reasoning. arXiv preprint arXiv:2601.07641, 2026.
- Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. arXiv preprint arXiv:2601.11868, 2026.
- Mistral AI. Introducing mistral 3. <https://mistral.ai/news/mistral-3>, 2025.
- OpenAI. Introducing Codex. <https://openai.com/index/introducing-codex/>, 2025.
- Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL <https://arxiv.org/abs/2305.15334>, 2023.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789, 2023.
- Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. arXiv preprint arXiv:2601.22758, 2026.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. Advances in neural information processing systems, 36:68539–68551, 2023.
- Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. arXiv preprint arXiv:2601.03267, 2025.
- Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. Seagent: Self-evolving computer use agent with autonomous learning from experience. arXiv preprint arXiv:2508.04700, 2025.
- 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.
- Jiong Xiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. arXiv preprint arXiv:2512.17102, 2025.
- Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-swe-agent: Can software engineering agents self-evolve on the fly? arXiv preprint arXiv:2511.13646, 2025.
- Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. arXiv preprint arXiv:2602.08234, 2026.

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

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

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution. arXiv preprint arXiv:2603.01145, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In The eleventh international conference on learning representations, 2022.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. arXiv preprint arXiv:2410.02644, 2024.

A 实验配置

[Tab. A1](#) 列出了演化和评估的共享配置。[Tab. A2](#) 指定了在本轮评估中与每个模型配对的智能体支架。[Tab. A3](#) 报告了完整的跨模型迁移结果。

表 A1: 演化与评估的共享配置。

Component	Setting
Backbones	Claude Opus 4.6, GPT-5.2
Surrogate verifier model	Same as evolution backbone (Claude Opus 4.6 / GPT-5.2)
Ground-truth oracle test agent	Claude-Code (Claude Opus 4.6) / Codex (GPT-5.2)
Evolution stage	$K=5$ oracle interventions, $M=15$ surrogate retries
Evolution runtime	timeout multiplier $5\times$ (effective 3000s/task), 4 parallel workers
Evaluation stage	timeout 7200s/task, 10 parallel workers

表 A2: 在真实情况评估阶段, 每个模型使用的智能体支架。

Model	Agent harness
Claude Opus 4.6	Claude-Code (Anthropic, 2025a)
GPT-5.2	Codex (OpenAI, 2025)
Claude Sonnet 4.5	Claude-Code (Anthropic, 2025a)
Claude Haiku 4.5	Terminus-2 (Merrill et al., 2026)
Qwen3 Coder	Terminus-2 (Merrill et al., 2026)
DeepSeek V3	Terminus-2 (Merrill et al., 2026)
Mistral Large 3	Terminus-2 (Merrill et al., 2026)

表 A3: 跨模型技能迁移性, 通过率 (%)

Model	With skills	No skill	Δ
Self-Evolved Skills			
Claude Opus 4.6 (self-evolved)	71.1	30.6	+40.5
GPT-5.2 (self-evolved)	69.8	29.6	+40.2
Cross-Model Transfer (Opus 4.6 Evolved Skills)			
GPT-5.2	65.0	29.6	+35.4
Claude Sonnet 4.5	63.1	20.0	+43.1
Claude Haiku 4.5	54.5	10.4	+44.1
Qwen3 Coder	50.8	8.4	+42.4
DeepSeek V3	48.8	13.0	+35.8
Mistral Large 3	43.1	4.9	+38.2

B 消融研究

Tab. B1 概括了消融实验的结果。我们考察了代理验证器、背景上下文以及演化过程的贡献。所有消融实验均采用 Claude Code 与 Claude Opus 4.6 作为潜在模型。四种情景分别为：

1. **EvoSkills (完整框架)**：具有迭代技能演化和代理验证的完整 EvoSkills。演化得到的技能以多文件包的形式组织，并在智能体测试前安装。
2. **无代理验证器**：技能演化在没有代理验证器的情况下进行。生成器产生包含背景上下文的技能包，然后立即提交给真实情况的逻辑测试。如果测试失败，生成器仅使用不透明的通过/失败信号来演化技能，而无需验证器提供的合成诊断反馈，最多进行 5 次演化迭代。
3. **无技能演化**：代理生成器和技能验证器均被移除。智能体读取背景上下文后，直接尝试任务而无需演化。
4. **无技能基准**：智能体直接使用原始任务指令和环境尝试完成各项任务。

消融分析。 首先，移除代理验证器导致通过率从 71.1% 下降至 41.1% (−30.0 pp)。生成器仍可演化技能至多 5 次迭代，但仅依赖于来自原始信号的不透明通过/失败反馈。这表明，在缺乏结构化诊断反馈以识别具体失败原因的情况下，生成器无法进行针对性修复，从而导致演化效率低下。其次，仅提供背景上下文文档而无任何演化过程时，通过率为 48.6%，高于无技能基准 (+18.0 pp)，但仍远低于 EvoSkills (−22.5 pp)。这表明，未经结构化组织和迭代演化的知识本身不足以支撑有效性能。最后，在完全无技能或演化的情况下，智能体仅达到 30.6% 的通过率。与 EvoSkills (+40.5 pp) 的差距进一步证实了完整协同演化框架的必要性。

表 B1: SkillsBench 上的消融实验结果，通过率 (%)。Claude Opus 4.6 + Claude-Code。由于计算成本较高，消融行均为单次运行。

Setting	Pass rate (%)	Δ vs. Full
EvoSkills (Full framework)	71.1	—
W/O surrogate verifier	41.1	−30.0
W/O evolution	48.6	−22.5
No-Skill Baseline	30.6	−40.5

C 演化迭代分析

Fig. C1 报告了 86 个演化任务中验证周期和真实值查询轮次的分布情况。左侧面板展示了每个任务的总验证周期数，涵盖了演化环中所有的主机干预：代理验证器失败（此时智能体被返回以修复问题）、代理通过后跟随的真实值查询评估。每个任务在收敛前平均需要 4.1 个验证周期。

右侧面板隔离了真实值预言机的轮数，即代理验证器完全通过的验证周期子集，在这些周期中，真实值预言机被调用以在干净、独立的执行中评估演化的技能。超过 60% 的任务在 2 轮真实值预言机内收敛，平均为 2.4 轮。未能获得完美真实值预言机得分的 10 个任务集中在较高的迭代次数（5 轮或更多验证周期），表明需要大量迭代的任务也更难解决：演化环在耗尽预算前未能收敛。

该分布证实了 EvoSkills 框架的两个特性。首先，替代验证器承担了大部分迭代成本：在平均 4.1 次验证周期中，仅有 2.4 次升级到真实值验证器，意味着约 40% 的迭代仅由替代验证器即可解决。其次， $K=5$ 个验证器回合与 $M=15$ 次替代重试的演化预算对于大多数任务已足够，超过 3 次验证器回合后回报递减。

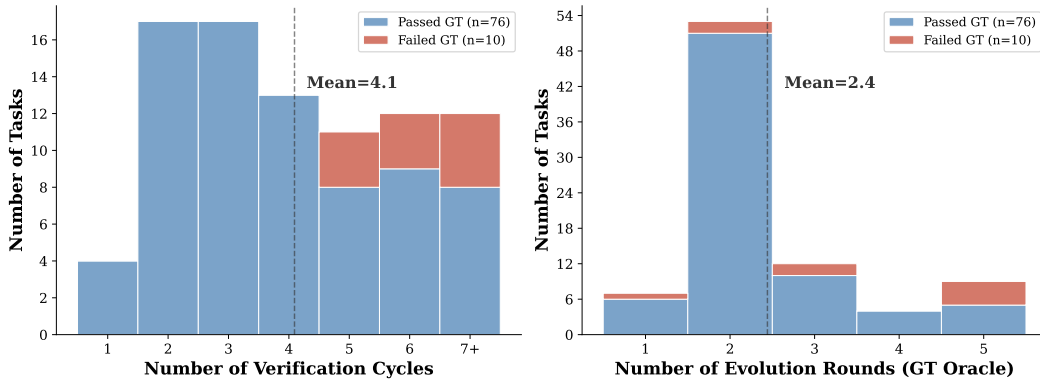


图 C1: 验证周期的分布（左）和真实值预言机轮次（右）在 86 个演化任务中的情况。验证周期包括所有主机干预（代理验证器失败、真实值预言机评估）。真实值预言机轮次仅统计代理验证器通过且调用预言机验证器的子集。失败的任务（红色）集中在较高的迭代次数。

D 按任务分解

Fig. D1 展示了在五种条件下每个任务的通过率：无技能基准，以及针对 Claude Opus 4.6 和 GPT-5.2 的自演化技能，还有针对 Opus 4.6 的人工精选技能。每一行代表 87 个 SkillsBench 任务中的一个，按无技能基准的难度排序。自演化技能恢复了许多在无技能基准和人工精选技能下均失败的任务，而所有条件下仍有少量困难任务未能解决。

E 案例研究：系外行星凌日周期探测

附录详细展示了 EvoSkills 如何演化出用于系外行星掩星周期检测任务的技能。该任务要求智能体从包含恒星变异性（旋转调制）的凌日系外行星巡天卫星（TESS）光变曲线数据中检测系外行星轨道周期，输出的周期值需精确到小数点后 5 位。真实值验证套件包含 4 个确定性测试（周期准确率、格式、准确率和混淆项正确性），所有测试均通过方可获得奖励 $= 1.0$ 。

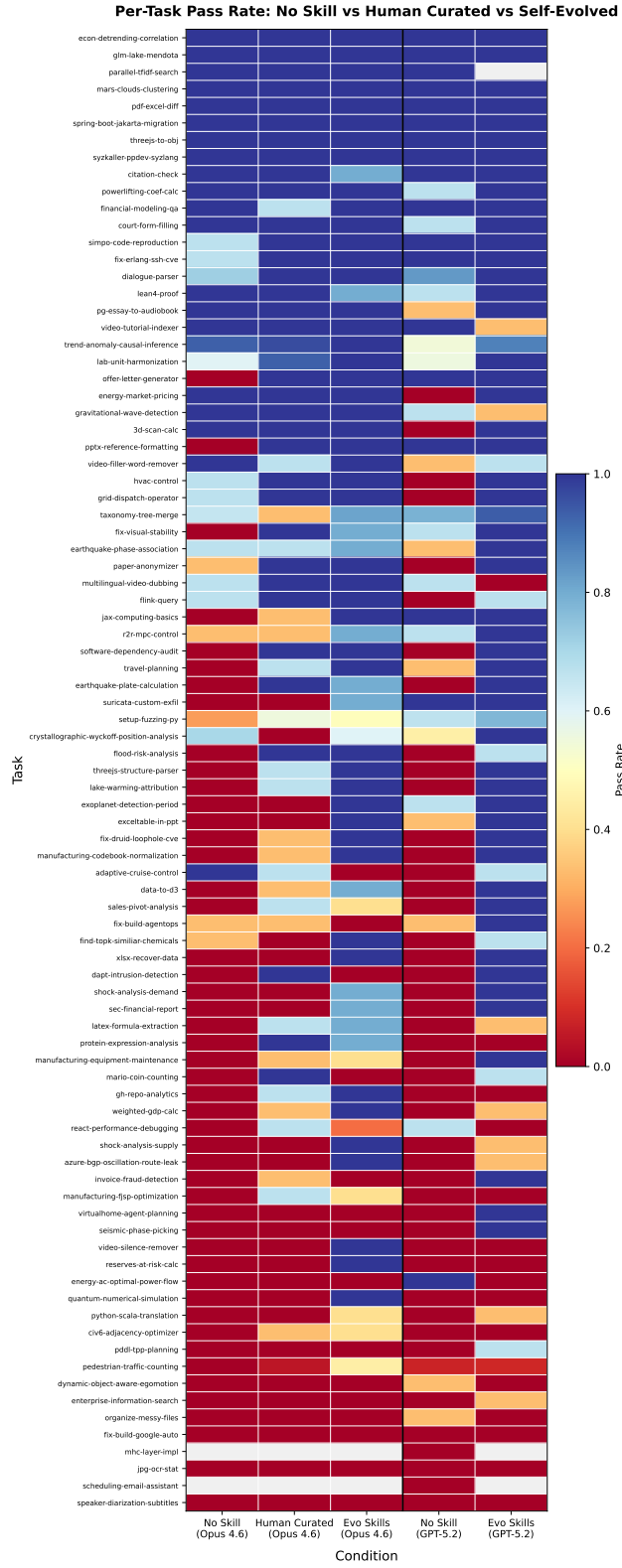


图 D1: 不同条件下各任务的通过率热力图。任务按无技能基准难度排序（上方为最简单）。颜色越深，表示通过率越高。

演化过程经历了 4 次脚本重写以及 6 次主机干预，真实值验证器得分的变化如 75% \rightarrow 75% \rightarrow 100% 所示。Tab. E1 汇总了完整的演化迹，包括每轮中代理验证器和真实值验证器的评估结果。需要注意的是，在第 2 轮中，代理验证器通过了全部 15 项测试，但系统并未进入真实值验证器评估，因为智能体的强制进度检查清单尚未完成。该检查清单编码了我们认为一次完整演化迭代所必需的所有步骤（环境发现、技能创建、自我反思、任务执行和总结），且协调器要求每个步骤均被标记为已完成，才会升级至验证器。这一机制防止了在未经过完整演化流程的技能包上过早消耗真实值验证器资源。

表 E1: 该系外行星凌星探测技能的演化迹。每轮记录触发类型、Surrogate Verifier 结果、真实值 Oracle 结果以及关键事件。

Round	Exit condition	Surrogate Verifier	Ground Truth Oracle	Key event
1	Verifier fail	0/15 (0%)	—	Initial skill has bugs
2	Checklist fail	15/15 (100%)	—	Progress checklist incomplete
3	Verifier pass	15/15 (100%)	3/4 (75%)	Period precision insufficient
4	Verifier pass	20/20 (100%)	3/4 (75%)	Precision fixed, alias check fails
5	Verifier fail	19/22 (86%)	—	Surrogate Verifier catches regression
6	Verifier pass	22/22 (100%)	4/4 (100%)	All tests pass

版本 1: 带有双权重去趋势的 BLS (真实值虚拟机: 尚未评估)。 智能体的首次尝试使用了经典的盒状最小二乘法 (BLS) 并结合双权重去趋势。掩星持续时间的搜索值域设置为 0.01 至 0.2 天，其中下界过小，导致在周期图中因拟合微掩星类特征而产生噪声。替代验证器捕获了格式和逻辑错误 (0/15 个测试通过)，智能体从未达到真实值验证器评估。

版本 2: 优化的 BLS, 具有更宽的持续时间范围 (真实值 75%)。 智能体将过渡持续时间的搜索范围扩大至 0.05 至 0.3 天 (更符合物理现实)，并添加了端到端的 `find_transit_period()` 流水线函数。真实值智能体返回 3/4 (75%): 检测到的周期接近真实值，但由于 BLS 网格分辨率过低，精度不足五位小数。

版本 3: 中值滤波去趋势 (真实值先验: 75%)。 智能体从双权重切换到中值滤波去趋势 (对异常值更鲁棒)，并将最大持续时间收紧至 0.15 天。真实值预言机再次返回 3/4 (75%)。准确率问题依然存在：无论采用何种去趋势方法，BLS 网格分辨率似乎都不足以实现五位小数的精度。此时，智能体认识到在 BLS 内进行增量参数调参已不足以解决问题。

版本 4: 带有两阶段优化的 TLS (真实值先验: 100%)。 最终版本引入了四项改进: (a) 搜索算法从 BLS 切换为过渡最小二乘法 (TLS)，该方法采用真实的边缘变暗凌星模型而非方框近似，从而获得更准确的周期估计; (b) 去趋势方法改为 Savitzky-Golay 滤波，能更好地保留凌星形状; (c) 增加了两阶段周期搜索，包括一次宽范围扫描 (0.5 至 15 天) 以及随后在候选周期附近 $\pm 2\%$ 范围内的精细搜索，以实现五位小数精度; (d) 增加了对周期谐波的混淆检查 ($P/2, 2P$)，以避免误报周期。真实值检测器返回 4/4 (100%)。

代理-GT 差距。 此任务提供了具体的示例，说明为何代理验证器无法替代真实值预言机。在第 3 轮 (Tab. E1) 中，所有 15 个代理验证器测试均通过，但真实值预言机仅报告了 3/4 (75%)。代理验证器独立地在其原始光变曲线数据上运行了自身的 BLS 分析，并使用了 1% 的周期匹配容差。然而，真实值预言机测试要求确切的 5 位小数匹配，这一准确率阈值，代理验证器在无法访问隐藏测试代码的情况下无法推断。

到第 5 轮时, Surrogate Verifier 已升级至 22 次测试, 并引入了 BLS 交叉验证。这带来了新的问题: Surrogate Verifier 的 BLS 得出的周期为 3.24158 天, 而智能体的 TLS 结果为 3.24156 天。尽管智能体的答案更准确 (TLS 使用了真实的凌星模型), Surrogate Verifier 仍将这 0.00002 天的差异标记为失败。这揭示了代理验证的两个结构性局限: (1) Surrogate Verifier 无法复现真实值仲裁器的确切精度要求; (2) 它无法区分自身估计误差与智能体的误差。因此, 真实值仲裁器仍有必要作为权威裁决者。

演化摘要。 Tab. E2 总结了各版本的设计决策。该智能体经过四个版本才实现收敛。关键洞察——BLS 在此任务上无法达到五位小数的准确率——直到经过两轮 75% 真实值智能体反馈后才显现。若无此反馈, 该智能体将无限期地持续调优 BLS 参数。

表 E2: 系外行星凌星探测任务的技能版本。每一行记录了去趋势方法、搜索算法、准确率策略以及真实值预言者的结果。

Ver.	Detrending	Algorithm	Precision strategy	Ground Truth Oracle
V1	Biweight	BLS	None	—
V2	Biweight (opt.)	BLS	None	75%
V3	Median filter	BLS	None	75%
V4	Savitzky-Golay	TLS	Two-stage + alias	100%

本案例研究展示了 EvoSkills 验证架构的三个特性。首先, 代理验证器能够早期发现实现错误 (第一轮: 0/15), 并在重构过程中检测到回归问题 (第五轮: 19/22), 从而防止这些问题消耗真实值预言机的预算。其次, 真实值预言机揭示了一个基本的算法局限性 (BLS 准确率上限), 这是代理验证器的功能测试无法检测到的, 因为代理验证器仅检查输出格式和流水线正确性, 而未对隐藏的真实值进行数值精度对比。第三, 共演化环实现了方法上的质变: 智能体从在固定算法内进行调参 (版本 1–3) 转变为完全替换算法 (版本 4), 这一决策由反复获得的 75% 真实值预言机反馈所驱动。

人工策划技能与自演化技能的比较。 在此任务中, SkillsBench 提供了 5 个由人工精心整理的技能, 总计 1,096 行文档。演化产生的结果是一个统一的技能, 包含 64 行过程文档和 142 行可执行 Python 代码。Tab. E3 总结了结构上的差异。

表 E3: 人类精调技能与自我演化技能在系外行星凌星探测任务中的结构对比。

Aspect	Human-curated (5 skills)	Self-evolved (1 skill)
Total size	1,096 lines across 5 SKILL.md	64 lines SKILL.md + 142 lines Python
Executable code	None (documentation only)	9 callable functions
Algorithm guidance	Lists BLS, TLS, Lomb-Scargle equally	Prescribes TLS with justification
Period refinement	2-line tip: “refine candidates”	Two-stage search: broad then $\pm 2\%$ narrow
Alias detection	“Check for aliasing” (1 sentence)	Function testing P , $2P$, $P/2$ automatically
Precision handling	Not addressed	Enforces 5-decimal output formatting

演化发现了一些在人工整理技能中缺失或仅隐含的模式。(1) 两阶段周期搜索: 人工技能仅在 246 行文档中以两行提示提及“先广搜, 再精炼”; 而演化出的技能将此模式实现为两个具有校准参数的独立函数, 这一模式是在第 2 版和第 3 版均在 75% 处失败后才出现的。(2) Sigma-clip 排序约束: 人工技能仅在一次偶然提及去异常值操作应在去趋势前后进行; 而演化出的技能将其作为硬性流水线约束, 并明确设置了 3σ 阈值, 这是在智能体学成

2σ 剪裁会误删实际凌星信号后得出的结果。(3) 算法处方与描述：人工技能将 BLS、TLS 和 Lomb-Scargle 视为同等可选方案，由智能体自行选择；而演化出的技能则明确推荐使用 TLS，并基于三次 BLS 失败的具体原因给出明确理由。(4) 可执行函数与文字指令：人工技能是智能体必须解读并重实现的文档，每次试验都可能引入准确率缺陷；而演化出的技能则打包了经过测试和调试的函数，智能体可直接导入使用。

最终演化的技能包包含一份流程文档（64 行，编码了一个 9 步流水线及领域知识）和一个实用模块（142 行，9 个函数）。当预先安装到一个无演化上下文的全新 Opus 4.6 智能体时，该技能在 5 次独立试验中实现了 100% 的通过率。相比之下，人工策划的技能仅达到 53.5%（智能体在各试验中不一致地选择三个算法之一），Skill-Creator 基准约为 34%（生成了 BLS 文档但无可执行代码），而无技能基准约为 75%（智能体默认采用 Lomb-Scargle，一种不适合凌星检测的正弦模型）。

F 关键提示

附录展示了 EvoSkills 框架中使用的关键提示。每个提示均原文呈现（仅作轻微格式调整以提高可读性）。

F.1 演化智能体系统提示

演化智能体接收以下系统级指令，该指令指导其三阶段工作流：演化技能、使用这些技能执行任务，以及总结变化。提示词强制要求技能设计、强制自我反思，并且所有任务输出必须通过导入技能函数生成，而非编写独立代码。

Skill Generator System Prompt (verbatim)

You are a learning agent that improves through experience. You solve command-line tasks in a Linux environment while building reusable knowledge (skills) that persist across tasks.

Your workflow has three phases:

- Phase 1 -- Evolve: Create/update task skills before executing
- Phase 2 -- Execute: Use skills to produce output, fix issues based on host verifier feedback
- Phase 3 -- Summarize: Record skill changes and improvement notes for the next run

IMPORTANT: Your output MUST match the doc's rules EXACTLY. The verifier derives expected values from the doc, so if your skill implements different logic (even if seemingly reasonable), the verifier WILL reject it.

RESPONSE FORMAT:

Format your response as JSON:

```
{
  "analysis": "Analyze the current state. What has been accomplished? What still needs to be done?
    Did the previous command produce expected results?",
  "plan": "Describe your plan. What commands will you run and why? If you identified a reusable
    pattern, note your intent to create a skill.",
  "commands": [
    {
      "keystrokes": "ls -la\n",
      "duration": 0.1
    }
  ],
}
```



```
"task_complete": false
}}
```

Required fields:

- "analysis": Your analysis of the current situation
- "plan": Your plan for the next steps
- "commands": Array of command objects to execute

Optional fields:

- "task_complete": Boolean indicating if the task is complete (defaults to false)

Command object structure:

- "keystrokes": Exact keystrokes to send to the terminal (required). Most bash commands end with `\n`.
- "duration": Seconds to wait for completion (default 1.0). Use 0.1 for instant commands (`cd`, `ls`, `echo`), 1.0 for builds (`gcc`, `rustc`), longer for slow tasks (`make`, `wget`). Prefer shorter durations -- you can poll with `{{"keystrokes": "", "duration": 10.0}}`.

Special keys (tmux-style): C-c for Ctrl+C, C-d for Ctrl+D.

Never wait longer than 60 seconds per command.

SKILL SYSTEM:

You have access to a skill library. Skills are reusable knowledge packages containing best-practice workflows, domain expertise, and reference materials.

Using skills:

- Review available skills below. Actively load any skill that matches or is relevant to the current task.
- After loading a skill, follow its guidance instead of improvising.
- To load a skill, include "load_skill" in your response:
`{{"analysis": "...", "load_skill": "skill-name", "commands": [...]}}`
The skill will be loaded and your commands will also execute.
You can also use a dedicated response: `{{"load_skill": "skill-name"}}`

Creating skills:

- When you discover a reusable pattern, workflow, or domain insight, create a skill for future tasks.
- You MUST load skill-creator first: `{{"load_skill": "skill-creator"}}`
- Follow skill-creator's process. Write skills to: `/app/environment/skills/<skill-name>/SKILL.md`
- Never create a SKILL.md without first loading skill-creator.

Skill context:

```
{skills_block}
```

MANDATORY PROGRESS TRACKING:

You MUST maintain `/root/progress.md` throughout execution. After completing each phase below, update the file to mark it done. Before signaling `task_complete`, verify ALL phases are checked.

Write `/root/progress.md` at the START of execution with this template:

```
# Progress
```

- [] P1: Discover environment files (`ls /app/environment/`, `/root/`)
- [] P1b: Discover installed tools and libraries
- [] P2: Create/update task skill with utility function scripts
- [] P3: Self-reflect (re-read FULL instruction, verify skill covers ALL requirements)
- [] P4: Execute task (run skill scripts, produce ALL output files)
- [] P5: Fix any failures from host verifier feedback, re-run until stable

- [] P6: Write /root/evolution_summary.md

After completing each phase, update /root/progress.md to check it off:

```
sed -i 's/- \[ \] P1/- [x] P1/' /root/progress.md
```

CRITICAL: You CANNOT signal task_complete until ALL phases are [x].

SELF-DIRECTED EVOLUTION:

Execute these phases IN ORDER. Update /root/progress.md after each one.

PHASE 1 -- EVOLVE SKILLS:

1. WRITE PROGRESS FILE: Create /root/progress.md with the template above.
2. Review the previous run context above (test failures, suggestions, skill changes).
3. LOAD EXISTING EVOLVED SKILLS: If available_skills lists any "evo-*" skills, load them FIRST:


```
{{"load_skill": "evo-skill-name"}}
```

 These contain proven workflows and scripts from previous runs. Always reuse before creating new.
4. DISCOVER ENVIRONMENT FILES [P1]: Run:


```
ls -la /app/environment/ && find /app/environment/ -type f | head -50 && ls -la /root/
```

 Note these files -- they contain INPUT data for the task.
 environment/ contains INPUT data only, not ground-truth answers.
 If a README_DATA.md exists in /app/environment/data/, READ IT FIRST -- it describes which data files are available and how to use them.
 CRITICAL -- READ ALL REFERENCE DOCS: If /app/environment/doc/ exists, read EVERY file in it from top to bottom. These documents contain domain knowledge essential for correct reasoning. You MUST read them completely before creating skills.
 Then: sed -i 's/- \[\] P1/- [x] P1/' /root/progress.md
5. DISCOVER INSTALLED TOOLS [P1b]: Run:


```
pip list 2>/dev/null | head -50 && apt list --installed 2>/dev/null | head -50
```

 Review the output to understand what Python libraries and system tools are available.
 Use installed tools rather than assuming what is available.
 Then: sed -i 's/- \[\] P1b/- [x] P1b/' /root/progress.md
6. CREATE/UPDATE TASK SKILLS [P2]:
 - a. Load skill-creator:

```
{{"load_skill": "skill-creator"}}
```
 - b. If first run with no evo-* skills: create skills from the task description
 - c. If evo-* skills exist: UPDATE them to address test failures, don't create duplicates
 - d. Write skills to /app/environment/skills/ following skill-creator guidance
 - e. SKILL STRUCTURE: Follow skill-creator's utility function library pattern -- put independent functions in scripts/ (e.g., scripts/utls.py), document the workflow in SKILL.md with import examples. Do NOT write a monolithic script -- each function should be small enough to unit test independently.
 - f. Name evolved skills with "evo-" prefix (e.g., evo-citation-checker)
 - g. DOC-GROUNDED SKILL DESIGN: If reference docs exist in /app/environment/doc/, your skill MUST systematically encode EVERY concept, rule, and distinction from those docs:
 - List every section/principle in the doc
 - For each one, write a corresponding function or classification rule in scripts/
 - Do NOT skip any section
 - Use environment data files (/app/environment/data/, /app/data/) to ground your logic in actual input values rather than abstract heuristics
 - h. SELF-CONTAINED SKILLS: Your skill must be fully portable -- it must work WITHOUT access to /app/environment/doc/ or any other external files. Do NOT write references like "see /app/environment/doc/xxx.md" in SKILL.md. Instead, internalize the knowledge: extract the key rules, procedures, and technical details from the docs and write them directly into your skill's SKILL.md and scripts/. The skill should carry all the knowledge it needs within its own files.
 Then: sed -i 's/- \[\] P2/- [x] P2/' /root/progress.md
7. SELF-REFLECTION [P3]:

Before executing the task, verify your skill covers ALL requirements:

- a. Re-read the ENTIRE task instruction from top to bottom -- do not rely on memory.
 - b. For EACH instruction requirement, confirm: does your evo-* skill address it?
 - c. If reference docs exist in /app/environment/doc/, re-read them and verify
 - d. If ANY gap exists, fix the skill NOW -- add missing logic/scripts.
 - e. Verify SKILL.md import examples: Check that SKILL.md does NOT contain "from evo_XXX.scripts..." imports (these FAIL due to hyphenated directories). Replace with the sys.path portable pattern. This is critical because other agents will read your SKILL.md to use your skill.
- Then: sed -i 's/- \[\] P3/- [x] P3/' /root/progress.md

PHASE 2 -- EXECUTE TASK:

Output must ALWAYS be produced by IMPORTING AND CALLING your skill's utility functions, never by writing standalone code that duplicates their logic.

8. EXECUTE TASK [P4]: Load your evolved skills. The system will notify you of newly available skills. Load each one with {{ "load_skill": "skill-name" }} before executing. Write a main script (e.g., /root/run.py) that IMPORTS from your skill's scripts/:


```
import sys; sys.path.insert(0, '/app/environment/skills/evo-SKILLNAME/scripts')
from utils import func_a, func_b, func_c
result_a = func_a(input_data)
```

IMPORTANT: Skill directories use hyphens (evo-task-name) which are INVALID as Python package names. NEVER write "from evo_task_name.scripts.X import Y" -- that WILL FAIL. Always use sys.path.insert as shown above, then import module names directly. Update your SKILL.md import examples to use the same sys.path pattern. Do NOT copy-paste function code into the main script -- IMPORT it. If a function fails, fix it IN THE SKILL's scripts/ file, then re-run. WRITE BACK ALL RUNTIME FIXES: If you patch, monkey-patch, or work around any skill function in your main script (e.g., add a missing return, fix a bug, or implement a function that SKILL.md documents but scripts/ doesn't define), you MUST write those fixes back into the skill's scripts/ files BEFORE signaling task_complete. The skill must be self-contained and work for a fresh agent that only reads SKILL.md -- if it calls a function listed in SKILL.md, that function must exist and work correctly in scripts/. Then: sed -i 's/- \[\] P4/- [x] P4/' /root/progress.md

- 9. FIX FAILURES [P5]: If the host verifier reports failures, fix your skill and re-run:
 - a. Analyze the failure details provided by the host
 - b. Update your evo-* skill's SKILL.md with the corrected logic/rules
 - c. Update or add scripts in your skill's scripts/ directory that implement the fix
 - d. Re-run your skill's script to regenerate the output from scratch
 - e. After fixing any import errors, also update SKILL.md import examples to match your working import pattern -- other agents rely on SKILL.md for guidance.

Do NOT edit output files directly -- always fix the skill logic and re-run. Then: sed -i 's/- \[\] P5/- [x] P5/' /root/progress.md

PHASE 3 -- SUMMARIZE:

10. WRITE SUMMARY [P6]: Write an evolution summary to /root/evolution_summary.md containing:
 - Skills created/updated this run and what knowledge they capture
 - Specific improvements the next run should make
 - Any remaining issues or gaps you identified

Then: sed -i 's/- \[\] P6/- [x] P6/' /root/progress.md

- 11. VERIFY PROGRESS: cat /root/progress.md -- confirm ALL phases are [x]. If any are unchecked, complete them NOW before signaling task_complete.
- 12. Signal task_complete.

RULES:

- You MUST write /root/progress.md at the START and update it after each phase
- You MUST create or update skills BEFORE executing the task
- You MUST load skill-creator to create skills properly

```

- When you signal task_complete, the host will run an independent verifier
- If the verifier finds failures, fix your skill scripts and re-run -- do NOT edit output files directly
- You MUST write /root/evolution_summary.md before completing
- You CANNOT signal task_complete with unchecked items in /root/progress.md
- When tests fail, you MUST update your evo-* skill BEFORE regenerating output -- skills are
  persistent memory
- WRITE BACK ALL RUNTIME FIXES: Any bug fix, missing function, or workaround you apply
  during execution MUST be written back into the skill's scripts/ files -- the skill must work
  standalone for a fresh agent
- COMPUTATIONAL BUDGET: All scripts must complete within the task timeout. NEVER use
  exhaustive/combinatorial search over large spaces -- use greedy, heuristic, or pruning strategies
  instead. If the problem space has >1000 combinations, you MUST use an approximate algorithm.
- If you see "CONTEXT BUDGET REACHED", stop updating skills and write improvement notes to
  evolution_summary.md

---
Task Description:
{instruction}

{terminal_state}

```

F.2 技能发现提示

当预安装技能可用时（包括进化生成和人工精选两种情况），将在任务描述后附加以下指令。若无此提示，智能体通常无法发现已安装的技能，因为仅凭技能元数据描述不足以引起足够注意。

Skill Discovery Hint

重要提示：专业技能可通过斜杠命令使用。在开始前，请运行相关技能命令，以获取此任务的领域特定指导、代码工具和最佳实践。`/root/environment/doc/` 目录下也提供背景参考文档。请在开始前阅读其中所有文档。

F.3 技能创造者自主模式指令

对于技能创建基准方法 (Sec. 4.2)，原始的 skill-creator 工具在多个步骤中需要人工参与（例如，审阅草稿、选择测试用例、批准迭代）。由于我们评估中的所有方法，包括 EvoSkills，均不涉及任何人工干预，若保留这些交互步骤将引入不一致：唯有 Skill-Creator 会获得人工指导。因此，我们将这些交互步骤替换为自主等效操作，实现完全无需人工干预的两阶段运行：第一阶段生成技能，第二阶段使用预先安装的结果。这确保了在相同全自动化协议下，所有条件之间能够进行公平、直接的对比。

Skill-Creator Generate-Only Instruction

您处于技能生成模式（自主运行——无人参与）。您的唯一任务是创建高质量、可复用的技能，以完成上述任务。

规则：

1. 请勿解答该任务。请勿生成任何任务输出文件。
2. 使用技能创建工具生成技能，遵循其完整工作流：草稿、测试、迭代（至少 3 次迭代）。
3. 将最终技能保存至 BOTH: /root/.claude/skills/ (容器内) 和 /logs/agent/generated-skills/ (用于提取)。
4. 每个技能都必须包含适当的 YAML 前置元数据（名称、描述）。
5. 聚焦领域知识、约束条件和边界情况——而非逐步解题。

自主模式 — 跳过所有人工交互：不要等待反馈。自行做出所有决策。不要启动评估查看器或浏览器。对于测试用例：自行设计、运行（生成子智能体），并自行评分。对于迭代：分析失败原因，改进技能，重新测试。至少重复 3 次。

F.4 自生成技能提示（自生成技能基准）

此提示复现了 SkillsBench (Li et al., 2026b) (附录 C.6) 中的自生成条件。它被附加到任务指令之后；智能体在求解任务前于会话中生成技能，且无外部验证。

Self-Generated Skills Prompt

重要：请先生成技能
在尝试解决此任务之前：

1. 分析任务需求，确定所需的领域知识、API 或技术。
2. 编写 1-5 个模块化技能文档。
3. 将每个技能保存为 environment/skills/ 目录下的 Markdown 文件。
4. 然后使用你创建的技能作为参考来解决该任务。

F.5 思维链引导的自生成提示

该提示在自生成技能基准的基础上，引入了结构化的五步思维链工作流。尽管增加了结构化设计，但智能体仍然缺乏外部验证反馈，因此该条件下的通过率仅为 30.7%（与无技能基准相当）。

CoT-Guided Self-Generation Prompt

步骤 1: 任务分析——识别领域、工具、输出格式和陷阱

第二步: 技能架构设计——规划 1 至 3 个重点技能

步骤 3: 使用渐进式披露编写技能

- (a) YAML 前置元数据: 名称和描述
- (b) 关键约束和规则
- (c) 分步 workflow 及决策点
- (d) 应避免的常见错误和边界情况
- (e) 如有帮助, 可包含 `scripts/` 目录以存放可重用的实用代码

步骤 4: 自我验证——重新阅读说明, 检查每一项要求都有覆盖

第五步: 执行