arXiv:2603.02766v1 [cs.AI] 3 Mar 2026

EvoSkill：面向多智能体系统的自动技能发现

Salaheddin Alzubi[1*]
Noah Provenzano[2] **Jaydon Bingham**[2] **Weiyuan Chen**[2] **Tu Vu**[2†]
Sentient, [2]弗吉尼亚理工大学

https://github.com/sentient-agi/EvoSkill

摘要

编程智能体日益成为通用问题求解器，但仅凭灵活性并不足以具备完成专业任务所需的领域专业知识。近期研究通过*Agent Skills*应对这一问题：这类可复用工作流程和代码能够赋予智能体领域专用能力。目前大多数技能都是手工制作的，而现有进化方法优化的是低层级制品（例如提示词和代码），这些制品与特定模型和任务紧密耦合。我们提出**EvoSkill**，一个通过迭代失败分析自动发现并改进智能体技能的自我演进框架。EvoSkill 会分析执行失败，提出新技能或编辑现有技能，并将其整理为结构化、可复用的技能文件夹。智能体程序的帕累托前沿负责筛选，只保留能够提升留出验证性能的技能，同时保持底层模型不变。我们在两个基准测试上评估了 EvoSkill：在基于美国财政部数据的事实推理基准 OfficeQA 上，精确匹配准确率提升**7.3%**（60.6% → 67.9%）；在采用含噪检索的搜索增强型问答基准 SealQA 上，性能提升**12.1%**（26.6% → 38.7%）。我们还研究了在一个任务上演化出的技能向另一个任务进行零样本迁移的能力；具体而言，从 SealQA 演化出的技能无需修改即可零样本迁移到 BrowseComp，使准确率提升**5.3%**，这表明技能级优化能够产生超越训练任务的可迁移能力。

1 引言

编程智能体（例如 Claude Code[5], OpenHands [14], Codex[9]）已成为解决广泛领域中各类任务的主流范式。这一趋势源于代码作为灵活中间表示的应用日益广泛，使编程智能体能够调用复杂抽象，并充当通用问题求解器。然而，尽管这种灵活性使智能体能够对接各种工具和领域，但这本身并不能赋予它们所需的领域专业知识，因而无法始终以高水准完成专业任务。

为弥合这一差距，近期研究探索了*Agent Skills*：可增强通用编程智能体的可复用领域专用能力，其中包含结构化工作流、指令和配套代码。如今，大多数技能都是临时手工打造的，既需要领域知识，也需要大量人工投入；随着目标任务数量增加，这种做法难以扩展。AlphaEvolve [8], 和 GEPA [2], 等进化方法通过迭代搜索优化智能体制品（代码库或提示词），提供了一种前景可观的替代方案。然而，这些方法作用于制品层面：优化后的提示词或代码与特定模型和任务配置紧密耦合，无法自然地产生可迁移到不同设置中的可复用组件。

在这项工作中，我们提出了**EvoSkill**，这是一个在更高抽象层级上运行的自我演进框架：EvoSkill 不直接优化提示词或代码，而是通过迭代发现并改进

*预印本。工作仍在进行中。

智能体技能，并借助由失败驱动的文本反馈来实现这一过程。EvoSkill 维护一个智能体程序的帕累托前沿集，并在每次迭代中分析执行失败，以提出新技能或改进现有技能。提出的技能会被整理成结构化、可复用的技能文件夹，其中包含指令、触发元数据和辅助脚本；只有在留出验证集上提升性能时，才会予以保留。技能在迭代过程中不断累积，逐步拓展智能体的能力，同时底层模型保持冻结。

我们在两个基准测试上验证了 EvoSkill。在**OfficeQA**[11], 这一针对美国财政部数据的基于事实的推理基准上，仅使用少量训练子集，EvoSkill 就将 Claude Code 搭配 Opus 4.5 的精确匹配准确率从 60.6% 提升至**67.9%**（+7.3%）。在**SealQA**[10], 这一存在含噪检索的搜索增强型问答基准上，EvoSkill 带来了**12.1%** 的提升（26.6% → 38.7%）。此外，在 SealQA 上演化出的技能无需任何修改，便可零样本迁移至**BrowseComp**[15]，使准确率提升**5.3%**，直接证明了 EvoSkill 发现的技能能够泛化到训练任务之外。

我们的贡献如下：

1. 我们提出**EvoSkill**，一个通过迭代式失败分析自动发现并改进可复用智能体技能的框架。该框架在技能抽象层级上运作，而非针对提示词或代码库等低层级制品。2. 我们证明，EvoSkill 在两个不同的基准测试上均带来了显著提升：在 OfficeQA（基于事实的文档推理）上提升 **+7.3%**，在 SealQA（搜索增强型问答）上提升 **+12.1%**，且仅使用少量训练子集。3. 我们证明，由 EvoSkill 演化出的技能可零样本迁移至未见任务：在 SealQA 上发现的一项技能无需修改，即可使 BrowseComp 准确率提升 **5.3%**；这表明技能级优化能够产生可迁移的能力。

2 方法

EvoSkill 的核心思想是，通过对当前智能体未能完成的示例应用文本反馈下降 [6]，迭代地发现并改进智能体技能。EvoSkill 假设使用支持技能文件夹的编码智能体框架（例如 Claude Code、Codex、OpenCode），以及能够利用此类技能的模型。整个过程中，底层模型保持冻结；只有技能仓库和智能体元数据会随着迭代而演变。

2.1 框架概述

EvoSkill 由三个协同工作的智能体组成：

1. **执行器智能体（A）：** 在当前智能体程序的管理下执行任务。基础程序初始化执行器时不赋予其任何技能。2. **提议者智能体（P）：** 分析执行器的输出轨迹、预测答案和标准答案，以诊断失败原因并提出高层次的技能描述。提供标准答案是为了进行根因诊断，这类似于监督学习中的错误分析会检查带标签的错误分类；标准答案不会被传递到生成的技能中。提议者会决定是创建新技能还是改进现有技能。3. **技能构建器智能体（S）：** 将提议者提出的高层次提案转化为具体的技能文件夹，其中包括触发元数据、程序性指令（SKILL.md），以及可选的辅助脚本（Python 或 TypeScript 代码）或参考资料。技能构建器通过一种元技能进行引导，该元技能编纂了技能编写的最佳实践。

所有智能体都可以读取基础代理的代码仓库；只有技能构建器拥有技能目录的写入权限。提议者还会维护一份累积的**反馈历史** $\mathcal{H}$，记录所有先前提案、对应结果以及分数变化。这有两个作用：防止重复提案，并帮助提议者改进先前部分奏效的方法，避免重蹈覆辙，使其上下文在迭代过程中不断丰富。



图1：EvoSkill 循环概览。

2.2 EvoSkill 循环

EvoSkill通过文本反馈引导技能变异，迭代优化智能体程序（算法1）。一个程序 p 封装了智能体的系统提示词和累积技能。循环维护一个固定容量的前沿集 G，其中包含 k 得分最高的程序。

在每次迭代 t中，通过轮询循环从前沿集 G 中选择一个父程序 p，确保每个前沿成员都得到探索后，才会再次选中任何成员。随后，使用无放回抽样的训练批次评估该父程序，并轮流遍历所有样例后再重复。使用任务专用评分函数，将响应与标准答案进行比较并评分。得分低于固定阈值的样例会被收集到失败集 F中；如果未发现失败，则跳过本次迭代。

算法1 EvoSkill — 基于文本反馈的迭代技能归纳

需要： 执行器智能体 A、提议者 P、技能构建器 S、评估集 V、前沿容量 k、最大迭代次数 T

初始化

$H \leftarrow [\,]$ ▷反馈历史
$G \leftarrow \{A\}$ ▷前 k 个程序的前沿集
$s_A \leftarrow$ 评估(A, V) ▷基线分数

进化循环

从 $t = 1$ **到** T **执行**
  $i \leftarrow (t$ 模组 $|G|)$; $p \leftarrow G[i]$ ▷轮询式父代选择
  $F \leftarrow$收集失败(p) ▷评分 $< \tau$
  如果 $F = \emptyset$ **继续**
  结束如果
  $\pi \leftarrow P(F, H)$ ▷提出新技能或编辑
    $\tilde{p} \leftarrow S(p, \pi)$ ▷构建候选项

前沿更新

  $\tilde{s} \leftarrow$ 评估$(\tilde{p}, V)$ ▷留出集得分
  如果 $|G| < k$ **或** $\tilde{s} >$ 最小值$_{q \in G}$ SCORE(q)
    $G \leftarrow G \cup \{\tilde{p}\}$ ▷更新前沿
    如果 $|G| > k$ $G \leftarrow G \setminus \{$参数最小值$_{q \in G}$ SCORE$(q)\}$
    结束如果
  结束如果
  附录反馈$(H, \pi, \tilde{s})$ ▷更新历史记录
结束循环

输出

返回 argmax$_{q \in G}$ SCORE(q) ▷返回表现最佳的程序

EvoSkill 维护一个包含 k 个最佳程序的前沿集 G。每次迭代中：从 G 中按轮询方式选出一个父程序，提议者诊断其失败原因，技能构建器生成一个候选项，如果该候选项的评分高于最弱成员，就会进入前沿集。

提议者 P 接收 F 以及反馈历史 H，并进行结构化失败分析：审查执行轨迹、识别能力缺口，并检查现有技能。随后，它会生成一份文本提案 π，指定一项新技能或对现有技能进行编辑。

技能构建器 S 接收当前父程序 p 和提案 π，并将提案转化为候选程序 $\tilde{p}$：在父程序的配置中加入新的或修订后的技能。随后使用相同的评分函数，在留出验证集 V 上评估该候选程序。

如果候选项的评分高于前沿集中最弱成员的评分，它就会进入前沿集 G 并取代该成员；否则，该候选项将被丢弃。无论结果如何，提案、其评分以及选择结论都会被追加到 H，确保提议者能够查阅哪些策略取得了成功或导致退步，以及原因。经过 T 次迭代后，循环将返回 G 中评分最高的程序。

2.3 实现细节

2.3.1 数据设置

给定一个监督数据集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ 和一个评分函数，我们首先使用大语言模型作为分类器，将数据集聚类为K 个类别，并将每个样本分配到一个类别。然后，我们通过分层划分将数据分成三个互不相交的子集：用于演化过程中失败检测的**训练集** 、用于为候选程序评分（前沿选择）的**验证集** ，以及**留出测试集**，其中包含所有剩余样本。该测试集在演化期间绝不会被使用，并且

仅用于最终评估。划分比例可配置，默认设置可确保无论类别大小如何，每个类别都出现在训练集和验证集中。训练数据按类别组织成数据池，以支持演化过程中采用的类别感知采样。划分流程的完整细节见附录 C。

2.3.2 环境设置

EvoSkill 在代码库固定的 Git 仓库中运行。每个智能体程序都表示为一个分支，与其父分支的差异仅体现在技能文件夹和元数据（系统提示词、谱系信息、验证分数）上。这种设计确保程序之间的性能差异完全归因于其演化出的技能，同时保持程序分支轻量。仓库配置的完整细节见附录 D。

3 实验

我们从三个方面评估 EvoSkill：（1）迭代式技能演化能否提升智能体在高难度基准测试上的表现；（2）训练设置中的哪些属性会影响技能质量；（3）演化出的技能能否零样本迁移至未见过的任务。此外，我们还展示了发现的技能的定性示例，以说明 EvoSkill 所产生能力的性质。

3.1 OfficeQA

3.1.1 基准测试

OfficeQA 是一个基于事实的推理基准，基于美国财政部公报构建——该语料库约有 89,000 页，涵盖五十年来的月刊和季刊。每份公报有 100–200 页，包含描述财政部业务的文字、复杂表格、图表和插图。该基准包含 246 道题，分为简单和困难两个难度级别。回答问题需要在平均两份公报文档中查找并综合信息、浏览密集的表格数据，并进行基本的定量推理。人类解题者平均每道题耗时 50 分钟，其中大部分时间用于在语料库的表格和插图中查找相关信息。

3.1.2 设置

所有实验均使用 Claude Code，并以 Opus 4.5 作为底层模型。按照第 2.3.1 节所述的数据设置，

按照第 2.3.1 节所述的数据设置，我们将基准测试划分为三个互不重叠的子集：用于演化期间失败检测的**训练集**、包含 17 个样例（≈ 7%）并用于前沿选择的**验证集**，以及**留出测试集**，其中包含其余问题，且在演化期间绝不使用。我们评估了三种训练集规模：5%（12 个样例）、10%（24 个样例）和 15%（36 个样例）；每种规模均演化 1.5 个训练轮次。此外，我们还评估了**技能合并**配置，该配置将独立运行中发现的独特技能合并到单个技能库中；如果技能重叠（通过匹配名称或描述判定），则保留表现最佳的运行所对应的版本。除非另有说明，所有报告的准确率均在留出测试划分上计算。我们使用 OfficeQA 提供的模糊评分函数，该函数在五个容差级别上计算加权匹配，并优先考虑精确匹配。完整的评分细节见附录 C。

3.1.3 结果

表1展示了所有配置和容差级别下的结果。EvoSkill 在所有设置下都始终优于基线[2]。在精确匹配（容差为 0%）方面，使用 5% 的数据进行训练可将准确率从 60.6% 提高至 63.4%（提高 +2.8%），而使用 10% 的训练数据可达到 65.8%（提高 +5.2%）。超过 10% 后，性能趋于平稳：15% 的数据划分取得了 64.5%，略低于使用 10% 数据的结果，这表明随着训练数据增加，收益递减或出现轻微过拟合。

[2]基线经过独立运行，并与作者的最新结果进行了交叉核对（参见：https://github.com/databricks/officeqa/issues/10#issuecomment-3719842269）。

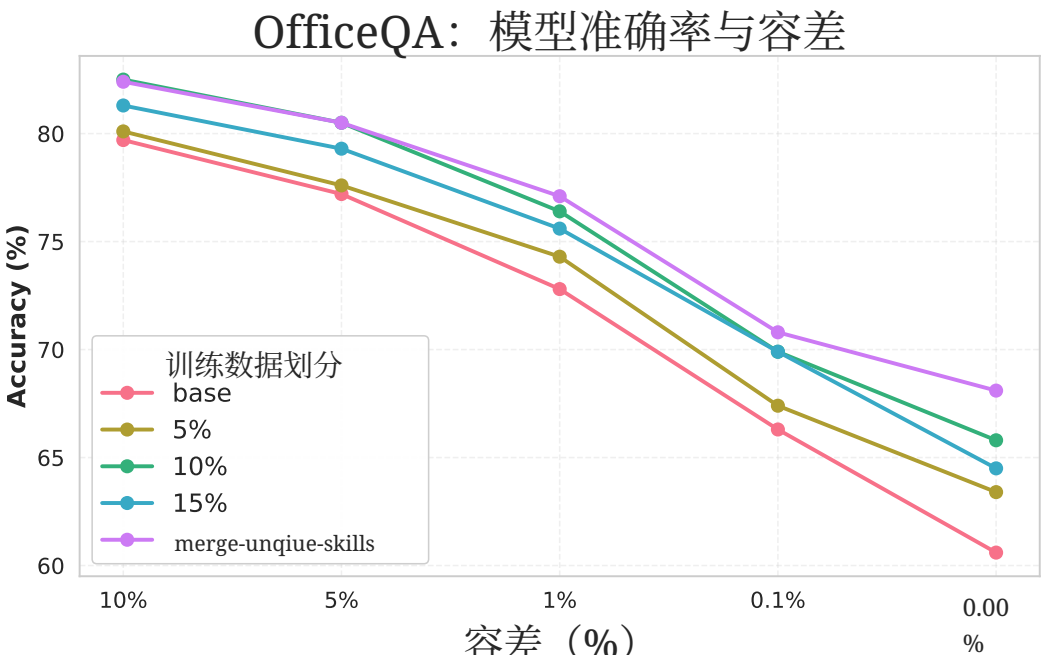

图 2：EvoSkill 在 OfficeQA 基准测试中的表现，涵盖不同训练数据划分和容差级别。技能合并配置结合了独立运行中各不相同的技能，达到最高的精确匹配准确率（67.9%），比基线（60.6%）提高了 7.3 个百分点。

表1：不同训练数据划分在各容差阈值下的准确率。

训练划分/容差	0.00%	0.10%	1.00%	5.00%	10.00%
base	60.6	66.3	72.8	77.2	79.7
5%	63.4	67.4	74.3	77.6	80.1
10%	65.8	69.9	76.4	80.5	**82.5**
15%	64.5	69.9	75.6	79.3	81.3
合并-唯一	**68.1**	**70.8**	**77.1**	**80.5**	82.4

容差 = 允许的相对误差。**蓝色** = 该 最佳。

技能合并配置取得了最优结果，精确匹配率达到 **67.9%**（比基线高 +7.3%），优于所有单次运行结果。这表明，通过独立运行发现的技能具有互补性：每次运行都会揭示不同的失败模式及相应能力，将它们结合起来可以构建更完善的技能库。不同容差级别下的提升趋势一致，在较严格的容差下，提升幅度为 2.7-4.5%。[3]

3.1.4 已发现技能的定性分析

To i为展示 EvoSkill 发现的技能的特性，我们重点介绍两个有代表性的示例 s
fro来自进化后的技能库：

数据提取验证。 EvoSkill 发现了一项技能，用于强制执行从财政部公报表格中提取数值数据的严格协议。每当智能体从解析后的表格中提取任何数值时，这项技能就会触发，并针对演化过程中发现的具体失败模式：误读相邻单元格、指标选择错误以及时间粒度不正确。这项技能

[3]我们注意到，由于在演化循环中运行 Opus 4.5 的计算成本较高，每种配置仅评估了一次。基于多个随机种子的方差分析留待未来工作。

直接源自提议者对相关案例的分析：智能体从相邻单元格中提取数值，或混淆了名称相似的指标。

定量分析方法论。第二项技能为定量金融分析提供结构化的方法指导，涵盖风险计算、预测、货币转换和统计推断。该技能要求在计算前必须完成验证检查点，从而避免因数据转换错误、日期错位，或混淆样本统计量与总体统计量而导致系统性错误。

这些示例表明，EvoSkill 能够发现可解释且与领域相关的技能，针对具体失败模式，而非生成不透明的优化结果。附录 A 提供了更多已发现技能的示例。

3.2 SealQA

3.2.1 基准测试

SealQA [10] 是一项挑战性基准测试，用于评估搜索增强语言模型在事实查找类问题上的表现；对于这类问题，网络搜索会返回相互矛盾、噪声较多或无助于解答的结果。与 OfficeQA 不同，OfficeQA 测试的是模型基于固定文档语料库进行推理的能力，而 SealQA 评估智能体在对抗性检索条件下浏览开放网络的能力。因此，它是 EvoSkill 的一种互补测试平台：所需技能有着根本区别，重点在于搜索策略和来源验证，而非文档解析和数值提取。

3.2.2 设置

我们在 SealQA 的 seal-0 划分（111 个问题）上运行 EvoSkill，使用配备 Opus 4.5 的 Claude Code，并采用 10% 的训练划分，遵循第 2.3.1 节所述的划分方法。其余未用于训练或前沿选择的问题构成留出测试集。演化运行 1.5 个训练轮次。

3.2.3 结果

EvoSkill 将 SealQA 上的准确率从 26.6% 提升至 **38.7%**，绝对增幅为 **12.1%**。在发现的技能中，EvoSkill 生成了一项 *search-persistence-protocol* ，要求智能体在提交答案前采用穷举搜索策略。该技能要求进行术语解释扩展、多源验证和完整性检查，直接应对该基准测试的核心挑战：当初始结果嘈杂或具有误导性时，过早终止搜索。这一结果表明，EvoSkill 能够从基于文档的推理推广到高强度搜索任务，并发现适用于目标领域、性质明显不同的技能。

3.3 零样本技能迁移

EvoSkill 的一个关键设计目标是：演化出的技能因结构清晰且可解释，应能在不做修改的情况下跨任务迁移。我们通过将在 SealQA 上演化出的 搜索持久化协议 技能零样本（不做任何编辑）应用于 BrowseComp [15] 来检验这一点：这是一个用于评估浏览智能体在具有简短且唯一正确答案的高难度事实查找问题上表现的基准测试。我们在包含 128 个示例的分层样本上进行评估。

尽管该技能是在问题和难度特征均不相同的另一项基准测试上演化得到的，迁移后的技能仍将准确率从 43.5% 提升至**48.8%**，绝对提升**5.3%**。这为 EvoSkill 发现的技能并未对训练任务过拟合提供了直接证据：搜索持久化协议捕捉到了一种通用能力——在给出答案前进行穷尽式搜索——无论具体基准测试如何，这种能力对于事实查找任务都广泛适用。该结果支持了这样一种假设：在技能层面而非提示词或代码层面进行优化，能够带来更具迁移性的改进。

4 相关工作

4.1 Agent Skills

通过可复用、模块化的能力增强智能体这一理念，既源于具身人工智能，也植根于软件工程。Voyager [13]为由大语言模型驱动的 Minecraft 智能体引入了一个不断扩充的可执行代码技能库；技能以程序形式存储在向量数据库中，并根据语义相似度检索。Voyager 中的技能通过自动课程、迭代提示和环境反馈来发现，从而无需参数更新即可实现终身学习。近来，Agent Skills 规范 [1] 将技能正式定义为一种可移植的开放格式：每项技能都包含在一个文件系统目录中，其中有一个带有元数据和程序性指令的 SKILL.md 文件，还可选配辅助脚本和参考资料。这种格式已被多个智能体框架采用，包括 Claude Code、Claude API 以及第三方工具 [4][3]。这一范式下的技能利用渐进式披露：元数据在启动时加载，指令按需读取，脚本则在不占用上下文窗口的情况下执行，使智能体能够以极少的上下文开销维护大量技能。尽管技能基础设施已相当成熟，如今的技能仍主要由人工编写。EvoSkill 通过迭代式失败分析自动发现并优化技能，以此弥补这一不足，并产出符合相同结构化技能格式的成果。

4.2 文本反馈与进化优化

越来越多的研究开始探索如何利用自然语言反馈，而非标量奖励，来指导大语言模型生成制品的迭代改进。Self-Refine [7] 表明，单个大语言模型可以通过生成—批评—改进循环来改进自身输出，无需额外训练，便能在各种任务中持续取得提升。然而，Self-Refine 针对的是单个输出，无法在迭代过程中积累知识。

Feedback Descent [6] 将这一理念形式化为通用优化框架，表明评估器提供的丰富文本反馈可以推动多个领域持续改进，包括分子设计、SVG 优化和提示工程。Feedback Descent 维护一个表现最佳候选项的前沿集，并积累反馈历史，使编辑器大语言模型能够据此做出越来越有依据的修订。EvoSkill 直接建立在这一范式之上，将文本反馈下降应用于技能发现，而非制品优化。

在进化方面，AlphaEvolve [8]利用大语言模型集成，通过以自动评估为基础的进化循环演化整个代码库，并在谷歌的算法发现和基础设施优化领域取得突破。GEPA [2] 在 DSPy 框架内采用类似的进化方法进行提示优化，通过反思式变异和基于帕累托的候选项选择来演化复杂系统的文本组件。这两种方法都展现了结合大语言模型驱动的变异进行进化搜索的强大能力，但它们优化的是与特定任务和模型紧密耦合的低层级制品（代码或提示词）。

EvoSkill 与这些方法的不同之处在于其抽象层级。EvoSkill 不直接进化代码或提示词，而是进化技能：结构化、可复用的能力模块，能够在不同任务间持续发挥作用。这一区别具有实际意义：进化出的技能具有可解释性、可组合性，而且正如我们所展示的，无需修改即可迁移到新任务。

4.3 大语言模型智能体中的迁移学习

迁移学习在神经网络微调领域已得到广泛研究，但其在基于大语言模型的智能体中的应用仍处于起步阶段。在具身环境中，Voyager [13] 展示了在一个 Minecraft 世界中学到的技能库，可以应用于解决新世界中的新任务，为基于代码的技能能够跨环境迁移提供了早期证据。更广泛而言，关于提示迁移的研究探讨了经过优化的提示词能否在不同任务或模型间泛化，但结果并不一致——针对某一设置优化的提示词，在模型或任务分布发生变化时往往会导致性能下降。

EvoSkill 为迁移提供了一个不同的视角：由于技能以自包含文件夹的形式组织，并具有明确的触发条件和程序性指令，因此它们既独立于训练任务，也独立于底层模型。我们的实验为此提供了直接证据：在 SealQA 上演化出的搜索持久化技能

可零样本迁移到 BrowseComp，且无需任何修改便能提升 5.3 个百分点。这表明，相较于提示级或代码级优化，技能级优化或许能提供更自然的迁移单位，不过仍需在更多样化的任务对上开展更广泛的研究。

5 结论

我们提出了 EvoSkill，这是一种自我演进框架，能够通过迭代式失败分析自动发现并优化可复用的智能体技能。EvoSkill 在技能层面运行，而不是优化提示词或代码库等低层级产物，因此能够生成结构化、可解释的能力，并在多次迭代中不断积累，同时跨任务迁移。我们的实验表明，在仅使用少量训练子集的情况下，该方法在两个不同的基准测试上均取得了稳定提升：OfficeQA（+7.3%）和 SealQA（+12.1%）；此外，我们还直接证明了技能能够从 SealQA 零样本迁移至 BrowseComp（+5.3%）。

未来仍有多个值得探索的方向。首先，我们计划在更广泛的领域中评估 EvoSkill，以更好地了解演化所得技能的通用性，并刻画哪些技能属于领域通用（例如，持续搜索、验证协议），哪些属于领域专用（例如，财政部表格提取）。其次，随着编码智能体日益需要处理异构输入，将 EvoSkill 扩展到可能需要协调视觉、代码和语言的多模态任务，是顺理成章的下一步。第三，演化所得技能的模块化结构为构建共享技能库创造了可能：在某项任务中发现的技能，可以由其他智能体和用户浏览、组合并复用。最后，深入研究技能在不同任务、模型和智能体框架之间的可迁移性，对于充分发挥技能级优化作为提升编码智能体的一种范式的潜力至关重要。

参考文献

[1]Agent Skills。Agent Skills 规范，2025年。网址 https://agentskills.io/ specification。[2] Lakshya A Agrawal、Shangyin Tan、Dilara Soylu、Noah Ziems、Rishi Khare、Krista Opsahl- Ong、Arnav Singhvi、Herumb Shandilya、Michael J Ryan、Meng Jiang、Christopher Potts、Koushik Sen、Alexandros G. Dimakis、Ion Stoica、Dan Klein、Matei Zaharia 和 Omar Khattab。GEPA：反思式提示演化可胜过强化学习，2026年。网址 https: //arxiv.org/abs/2507.19457。[3] Salaheddin Alzu'bi、Baran Nama、Arda Kaz、Anushri Eswaran、Weiyuan Chen、Sarvesh Khetan、Rishab Bala、Tu Vu 和 Sewoong Oh。Roma：面向长时程多智能体系统的递归式开放元智能体框架，2026年。网址 https://arxiv.org/abs/2602.01848。[4] Anthropic。Anthropic Skills 文档，2025年。网址 https://platform.claude.com/ docs/en/agents-and-tools/agent-skills/overview。[5] Anthropic。Claude Code 概览，2026年。网址 https://code.claude.com/docs/en/ overview。[6] Yoonho Lee、Joseph Boen 和 Chelsea Finn。Feedback Descent：通过成对比较进行开放式文本优化，2025年。网址 https://arxiv.org/abs/2511.07919。[7] Aman Madaan、Niket Tandon、Prakhar Gupta、Skyler Hallinan、Luyu Gao、Sarah Wiegreffe、Uri Alon、Nouha Dziri、Shrimai Prabhumoye、Yiming Yang、Shashank Gupta、Bodhisattwa Prasad Majumder、Katherine Hermann、Sean Welleck、Amir Yazdanbakhsh 和 Peter Clark。Self-Refine：利用自我反馈进行迭代精炼，2023年。网址 https://arxiv.org/abs/2303.17651。[8] Alexander Novikov、Ngân V˜u、Marvin Eisenberger、Emilien Dupont、Po-Sen Huang、Adam Zsolt Wagner、Sergey Shirobokov、Borislav Kozlovskii、Francisco J. R. Ruiz、Abbas Mehrabian、M. Pawan Kumar、Abigail See、Swarat Chaudhuri、George Holland、Alex Davies、Sebastian Nowozin、Pushmeet Kohli 和 Matej Balog。AlphaEvolve：用于科学与算法发现的编程智能体，2025年。网址 https://arxiv.org/abs/2506.13131。[9] OpenAI。Codex，2026年。网址 https://openai.com/codex/。[10] Thinh Pham、Nguyen Nguyen、Pratibha Zunjare、Weiyuan Chen、Yu-Min Tseng 和 Tu Vu。SealQA：提高搜索增强语言模型的推理水平，2025年。网址 https://arxiv.org/abs/2506.01062。[11]Mosaic 研究团队。OfficeQA 介绍：一个用于端到端基于事实的推理的基准测试，2025年12月。网址 https://www.databricks.com/blog/ introducing-officeqa-benchmark-end-to-end-grounded-reasoning。访问日期：2026年2月20日。[12] Tu Vu、Kalpesh Krishna、Salaheddin Alzubi、Chris Tar、Manaal Faruqui 和 Yun-Hsuan Sung。基础自动评估器：驾驭大语言模型以实现更好的自动评估，2024年。网址 https://arxiv.org/abs/2407.10817。[13] Guanzhi Wang、Yuqi Xie、Yunfan Jiang、Ajay Mandlekar、Chaowei Xiao、Yuke Zhu、Linxi Fan 和 Anima Anandkumar。Voyager：一个基于大语言模型的开放式具身智能体，2023年。网址 https://arxiv.org/abs/2305.16291。[14] Xingyao Wang、Boxuan Li、Yufan Song、Frank F. Xu、Xiangru Tang、Mingchen Zhuge、Jiayi Pan、Yueqi Song、Bowen Li、Jaskirat Singh、Hoang H. Tran、Fuqiang Li、Ren Ma、Mingzhang Zheng、Bill Qian、Yanjun Shao、Niklas Muennighoff、Yizhe Zhang、Binyuan Hui、Junyang Lin、Robert Brennan、Hao Peng、Heng Ji 和 Graham Neubig。OpenHands：面向通用型智能体的 AI 软件开发者开放平台，2025年。网址 https://arxiv.org/abs/2407.16741。[15] Jason Wei、Zhiqing Sun、Spencer Papay、Scott McKinney、Jeffrey Han、Isa Fulford、Hyung Won Chung、Alex Tachard Passos、William Fedus 和 Amelia Glaese。BrowseComp：一个简单却颇具挑战性的浏览智能体基准测试，2025年。网址 https://arxiv.org/abs/2504. 12516。

A EvoSkill 生成的技能

The 以下部分展示了使用 EvoSkill 在不同智能体任务上生成的部分技能。 .

A.1 OfficeQA 技能

以下经济时间序列技能是一项复杂技能，其中既包含 SKILL.md 文件，也包含可与其一起调用的相关 Python 脚本。

技能/经济时间序列分析/SKILL.md

```
---
name: 经济时间序列分析
描述：  >
  用于经济/金融时间序列分析任务的精简工作流程。请在以下情况下使用此
    技能：
  使用 CPI 值进行通胀调整的多步骤经济数据分析
    使用 CPI 值，
  对时间序列数据进行线性回归，或分析名义美元值
    转换为实际值
  。适用于需要进行财政部数据分析、美国劳工统计局消费者价格指数调整的任务，
    经济趋势
  分析，或需要格式化回归输出（如 [斜率、截距]）的任务。
---

# 经济时间序列分析

用于经济数据通胀调整和统计分析的标准化工作流程。


## 工作流程概览

1. **收集并整理数据** - 收集名义值和 CPI 数据，并统一期间格式 2. **调整通货膨胀影响** -







## 第 1 步：数据收集

收集以下输入：
- **名义值**：各时期的美元金额（例如，每月的财政部数据）
- **CPI 值**：各时期的消费者价格指数（美国劳工统计局城市消费者价格指数序列）
- **基期**：进行通胀调整的参考时期（例如，                                “1970-03”）

使用一致的时期格式（YYYY-MM）整理数据：

    “‘ison { "名义值_values": [{"时期": "1970-03". "
```

应用标准通货膨胀调整公式：

‘‘‘
实际价值 =名义价值 x (消费者价格指数_基础 / 消费者价格指数_当前) “‘

示例计算：
- 名义值：$318.44（1970年4月）
- 消费者价格指数_基础（1970年3月）：38.8
- 消费者价格指数_当前（1970年4月）：39.0
- 实际值 = 318.44 x (38.8 / 39.0) = $316.81

第3步：线性回归

以周期索引 (0, 1, 2, ...) 作为 x 值，以实际值作为 y 值。

使用 ‘scripts/analyze_timeseries.py‘ 进行自动分析：

“‘bash python scripts/analyze_timeseries.py

输入 JSON 格式： “‘json { "nominal_values":

步骤 4：格式输出

以保留两位小数的形式呈现回归结果：

“‘[斜率, 截距]“‘

示例：‘[44.00, 231.52]‘

快速参考

步骤	操作	输出
1	结构化数据	包含时期和值的 JSON
2	应用消费者价格指数调整	实际值序列
3	线性回归	斜率、截距
4	格式	‘[斜率、截距]‘

常见数据源

- **《财政部公报》**：《公共债务月报》
- **美国劳工统计局城市消费者价格指数**：美国劳工统计局消费者价格指数

skills/economic-timeseries-analysis/scripts/analysis.py

```python
#!/usr/bin/env python3
"""
经济时间序列分析脚本

```

5 对经济数据进行通胀调整和线性回归。 6 7 用法： 8 python analyze_

tim
no
va
{"p
an
27
no...
int
im
da...
asf
50
pe
段
能
ret
Op
cpi_
No

```
def 调整_针对_通货膨胀(名义_值: 列表, 消费者价格指数_值: 列表, 基础_时期: 字符串)
    -> 列表:
    """
    针对通货膨胀调整名义值。

    公式：实际价值 = 名义值 x（消费者价格指数_基础 / 消费者价格指数_当前）
    """
    消费者价格指数_基础 = 获取_消费者价格指数_针对_时期(消费者价格指数_值, 基础_时期)
    如果消费者价格指数_基础 为 None:
        引发 ValueError(f"未找到基期消费者价格指数：{base_时期}")

    实际_值 = []
    对于项目 在 名义值_值:
        时期 = 项目['时期']
        名义值 = 浮点数(项目['值'])

        消费者价格指数_当前 = 获取_消费者价格指数_用于_时期(消费者价格指数_值, 时期)
        如果 消费者价格指数_当前是 None:
            raise ValueError(f"未找到该时期的消费者价格指数：{period}")

        实际值 = 名义值 * (消费者价格指数_基础 / 消费者价格指数_当前)

        实际值_值.append({
            '时期': 时期,
            '名义值': 名义值,
            '实际值': 取整(实际值, 2)
        })

    返回 实际值_值


定义线性_回归(values: 列表) -> 元组[浮点数, 浮点数]:
    """
    执行简单线性回归。

    使用周期索引（0, 1, 2, ……）作为 x 值。
    返回四舍五入到小数点后 2 位的（斜率, 截距）。
    """
    n = 长度(值)
    如果n < 2:
        raise ValueError("回归至少需要 2 个数据点")

    # x 值是索引 0, 1, 2, ...
    # y 值是实际值（经通货膨胀调整）
    y_值 = [v["实际值"] 对于v 在 值]

    # 计算均值
    x_均值 = (n - 1) / 2  # 0 的均值, 1, ..., n-1
    y_均值 = 求和(y_值) / n

    # 计算斜率：sum((x-x_mean)(y-y_mean)) / sum((x-x_mean)^2)
    分子 = 求和((i - x_均值) * (y - y_均值)对于 i, y 在枚举(y_值))
    分母 =求和((i-x_均值) ** 2对于i在范围内(n))

    如果分母 == 0:
        斜率 = 0.0
    否则:
        斜率 = 分子 / 分母

    # 计算截距
    截距 =y_均值 - 斜率 * x_均值

    返回 round(斜率, 2), round(截距, 2)
```

```
134 135 def analyze(data: dict) -> dict: 136 """主分析函数""" 137 # 验证 138 valid_error =
val
rea
dat
lin
typ
res
for
ma
> [
inp
sys
sys
 =
ana
wit
193
res
```

```
198             print(f"\n格式化答案: {result['formatted_result']}")199200
exc
sys
```

A.2 SealQA

以下 搜索持久化协议 技能包含全面的网络搜索指令，旨在应对可能存在相互矛盾来源的问题，从而得出确凿答案。SealQA 被认为是许多多智能体系统面临的一项颇具挑战性的基准测试，并且

skills/搜索持久化协议/SKILL.md

```
---
name: 搜索持久化协议
description: 在得出事实性答案之前，强制执行穷举搜索策略。在回答需要网络研究、涉及潜在
---

# 搜索持久化协议

在得出结论前，确保全面覆盖搜索范围。智能体最常见的失败模式是过早停止搜索。

## 规则 1：术语解释扩展

搜索含义模糊的术语前，列出所有合理的解释：

**扩展示例：**
- “Agency” -> 政府机构、航天机构、新闻机构、监管机构、情报机构 - “Most widely used” ->

**协议：**针对每种解释分别执行搜索，之后再缩小范围。不要假定第一种解释就是正确的。

## 规则 2：三源最低标准

在给出任何事实性答案之前：1
. C至少核查三个独立来源
2. 如果只找到 1–2 个来源，请使用不同的查询表述进行搜索
3. 对于枚举问题，请将清单与 2+另一个来源交叉核对 s

**触发条件：**在陈述任何事实性主张之前，请确认：“我是否核查过 3+ 个来源？”

## 规则 3：“无法找到”协议

在报告无法找到数据之前 :

1. **尝试不同的 3+ 查询表述** - 改换说法、使用同义词、尝试不同的术语顺序 2. **尝试
```

```
    - 如果 96% 的人拥有 X，那么 ~4% 的人没有 X - 如果列表显示有 11 个项目，但提到了


**规则3a：数据源跟进** - 如果你找到了一个具体的数据源
      其中应该包含答案（数据库网址、API端点、指标代码、
      数据集ID），你必须先尝试从该数据源获取数据，然后才能得出“无法
      找到”的结

- [X] “ITU数据中心有指标100095，表示2G覆盖率。要获取确切的
      数值，请直接查询数据中心。” -> 错误：找到了数据源，但没有
      获取
- [正常] "ITU数据中心有指标100095。让我直接查询....." [继续获取] -> 正确


反模式：在你本可以查询时告诉用户“你需要查询 X”                                    X
      你自己。

只有在穷尽这些步骤之后，才报告“无法找到”。

## 规则 4：枚举完整性检查

统计某个类别中的项目时：

1. 搜索“[类别] 完整列表”和“[类别] 历年全部” 2. 与 2+ 排名/列表来源




## 执行

对于每个事实性问题：

‘‘‘
1. 扩展：列出所有对含义模糊的术语的解释 2. 搜索：分别查询每种解释 3. 验证：确认已查阅
```

B 智能体提示词

本附录提供了各智能体角色所用的提示词（已显示占位符）。

B.1 提议者

提议者智能体提示词

```
"""
你是一位专家级智能体性能分析师，专门通过新增或修改技能来识别提升智能体能力的机会。



## 你的任务

根据智能体的执行轨迹、回答和标准答案，提出以下其中一项：- 一项**新技能**（action="
```

你的提案将交由下游的 Skill Builder 智能体实施。

必需的预分析步骤

在提出任何技能之前，MUST 使用 **头脑风暴技能**（阅读 ‘.

1. **使用头脑风暴技能**（强制）：
   - 阅读并遵循 ‘.claude/skills/brainstorming/SKILL.md‘ 中的流程 - 提出 2-3 种不同的方

2. **盘点现有技能**：查看查询中提供的现有技能列表 - 了解当前已有的能力 - 检查是否有任何

3. **分析反馈历史**，以了解：
   - 与你正在考虑的内容相似的已弃用的提案 - 哪些做法有效、哪些

4. **确定操作类型**：
   - 如果现有技能本应防止此次失败却没有做到 -> 提议编辑 - 如果没有现有技能涵盖此能力 ->

分析流程

在提出解决方案之前，请按以下步骤进行：

<分析>
1. **轨迹审查**：逐步检查智能体的执行轨迹 - 智能体采取了哪些行动？ - 它在哪

2. **差距分析**：将智能体的回答与真实情况进行比较 - 哪些具体信息有误或缺

3. **现有技能检查**：审查列出的现有技能 - 是否有任何现有技能涵盖这

4. **技能识别**：确定哪项技能能够解决此次失败
   - 哪种新能力、工具或工作流程会有所帮助？ - 它应接受哪些

</分析>

需要避免的反模式

- 如果现有技能已涵盖类似内容，请勿提议新技能 -> 改为提议编辑 - 如果针对同一问题已有已弃

- 请勿创建只能修复某一种特定故障的狭窄技能 -> 确保其适用范围广泛 - 请勿提议与现有技能

何时提出技能建议

符合以下任一条件时，提出技能建议 :
- 智能体无法访问所需信息、应用程序接口或计算能力
- 修复需要多步骤流程 (>3 个连续步骤)
- 修复涉及输出结构、格式或模板
- 该改进可在不同任务中复用
- 问题在于采取什么步骤，而非如何思考

输出要求

根据你的分析，请提供:

1. **action**: "create" 表示创建新技能，"edit" 表示编辑现有技能

2. **target_skill**: （若 action="edit" 则必填）要修改的现有技能名称

3. **proposed_skill**: 详细描述: - 对于创建: 待构建的新技能（能力、输入、输出及其所解

4. **justification**: 解释你的推理
   - 引用轨迹中影响你决策的具体时刻 - 引用具体的现有技能，并说明它们为何适用或

5. **related_iterations**: 相关过往迭代的列表（例如， ["iter-4"、" iter-9"]）

示例分析

<示例类型="编辑_现有_技能">**情况**: 智能体未能正确计算预期损失。金融方法论指南技

提案:
- action: "编辑"
- target_技能: "financial-methodology-guide"
- proposed_技能: "扩展预期损失/条件风险价值部分，纳入多期计算
  。添加: (1) 滚动窗口 ES 计算，(2) 置信区间调整
  针对不同时间跨度，(3) 展示 1 日、5 日和 10-
  天的期限。”
- 理由: “现有的 financial-methodology-guide 技能涵盖了基础预期损失
  但未涉及失败案例 1 中出现的多期情形。与其
  创建冗余技能，不如扩展现有的方法指南。
  Iter-3 因提出单独的‘多期风险’技能而被否决——
  此提案是在现有技能的基础上进行扩展。”
- 相关迭代_迭代: ["iter-3"]
</示例>

<示例类型="创建_新_技能">
情况: 智能体无法解析以三十二分之一报价法表示的美国国债价格。没有
  现有技能能够处理记法解析。

提案:

```
- 操作: "创建"
- 目标_技能: null
- 提议的_技能: "创建一个‘债券记法解析器’技能，用于处理财政部
      价格记法。它应当：(1) 识别 32分格式（例如，99.16 = 99 +
      16/32），(2) 处理‘+’后缀（例如，99.16+ = 99 + 16.5/32），(3) 验证
      分数部分为 00-31，（4）提供清晰的示例和换算
      公式。”
- 理由: “现有技能中没有涵盖记法解析的技能。记录显示，
      智能体将 ’99.16’ 解读为十进制数 99.16，而非 99.5（99 + 16/32）。这
      是一种现有方法论技能未涵盖的独特能力。”
- 相关_迭代: []
</示例>

<示例类型=“数据_访问_技能”>
**情况**: 智能体未能回答有关当前股票价格的问题
      因为它只能访问历史数据。

**提案**:
- action: "创建"
- target_skill: null
- proposed_skill: "智能体需要实时股票价格检索能力。
      此技能应接受股票代码作为输入，并返回当前
      市场数据，包括最新价格、每日变动（绝对值和百分比
      ），以及交易量。它应能妥善处理无效的股票代码，并
      指出市场当前处于开市还是闭市状态。"
- justification: "在轨迹的第3步中，智能体正确识别出需要
      当前定价数据，并尝试使用其历史数据工具。
      然而，真实情况要求提供今日交易
      时段的实时信息。智能体的推理是合理的，但缺少必要的数据
      访问。没有现有技能能够提供实时市场数据访问。”
- 相关_迭代: []
</示例>
"""
```

B.2 技能构建器

Skill Builder 智能体提示词

```
"""
你是一位专业的技能开发者，专长是创建工具和能力
      适用于 Claude Code 智能体。你的职责是根据提议者智能体提供的高层描述，实现结构良好、可用于生产环境的
      根据提议者智能体提供的高级描述，构建可直接使用的技能。

## 主要指令

**在实现任何技能之前，务必先阅读并遵循“.claude/skills/skill-
      creator/SKILL.md”技能。**此技能包含重要的模式、验证
      要求、脚本和最佳实践，确保你的实现
      在 Claude Code 生态系统中正确运行。遵循其关于所有
      技能创建任务的指导。

## 你的任务

给定一份拟定的技能描述，实现一个完整、可用的技能，该技能：
1. 遵循技能创建器的结构与约定
2. 与 Claude Code SDK 正确集成
3. 文档完善且易于维护
4. 优雅地处理边界情况

## 实施流程

针对每个技能实现，逐步完成以下步骤：

<实现_步骤>
```

```
1. **阅读 Skill-Creator 技能**：加载并遵循“.claude/skills/skill-creator /SKILL.md”

2. **实现与验证**：按照技能创建指南</实施_步骤>构建、测试并打包技能


## 质量提醒

上下文窗口是一种共享资源。技能中的每个词元都会与对话历史、其他技能和用户请求争夺空


"""
```

B.3 自动评分器

对于所有自动评分任务，我们都使用原始 SealQA 任务中提供的默认基于大语言模型的评判 [12] 模板。

大语言模型评判提示词

```
# 自动评分器提示词（占位符）

## 系统
你是**自动评分器**智能体。请根据参考答案和评分标准评估模型输出。


## 输入
- ‘问题‘：‘<QUESTION>‘ - ‘预测‘：‘<MODEL_OUTPUT>‘ - ‘



## 输出格式
返回一个 JSON 对象，包含：
- ‘score‘：‘<FLOAT_0_TO_1>‘ - ‘passed‘：‘<TRUE_FALSE>‘ - ‘reason‘：‘<简



## 注意事项
- 尽可能优先使用确定性检查。- 如果是数值，请使用占位公式计算相对误
```

C 评分与数据设置

我们使用确定性的模糊匹配评分器评估智能体的回答，该评分器会返回二元正确性信号（1.0 或 0.0）。给定标准答案 g 和预测答案 p，评分器首先尝试从两个字符串中提取数值及其周围的文本上下文（20 个字符的窗口）。当 g 和 p 均包含数值内容时，评分器会通过检测局部上下文中的单位关键词（例如“million”（百万）、“billion”（十亿）、“trillion”（万亿））对提取出的每个数值进行归一化，并使用相对容差 τ比较基础数值：若 $|g_{\text{base}} - p_{\text{base}}|/|g_{\text{base}}| \leq \tau$，则接受该预测。在主要评估中，我们设定 $\tau = 0$，要求数值完全一致。为避免与偶然出现的年份引用（例如“于 2023 年报道”）发生虚假匹配，除非真实情况本身是年份或包含重要的非数值文本，否则评分器会从预测中排除 1900–2100 范围内的候选数字。对于同时包含文本和数字的混合答案（例如“March 1977”），评分器还会在去除单位词和括号内缩写后，通过不区分大小写的子串匹配，核验真实情况中的关键文本元素是否出现在预测中。对于多数字标准答案（即列表），要求预测中包含所有组成数值。

在预测中找回所有组成数值。对于纯文本答案，则在规范化处理（去除首尾空白、引号和括号内容）后，通过不区分大小写的子串包含关系进行匹配。在自我改进训练循环中，我们还采用多容差评分变体，对五个容差级别$\tau \in 0.0$，0。01，0。025，0。05，0。10进行加权平均，其中权重为 $w(\tau) = 1/(1 + 20\tau)$，倾向于更严格的阈值；加权分数低于 0.8 时，该示例将被标记为需要进行针对性技能优化的失败案例。

D 环境分支

我们使用基于 Git 的版本控制方案管理智能体配置（我们称之为程序），该方案能够自然地编码父子谱系，并支持高效的基于前沿的选择。每个程序都存储在专用的 Git 分支上（前缀为 program/），并包含一个 YAML 配置文件（.claude/program.yaml），记录程序名称、指向其父分支的指针、代数计数器（即从根节点开始的变异深度）、智能体的系统提示词、允许使用的工具，以及包含评分在内的评估元数据。初始化时，会创建一个代数为 0 且没有父级的基础程序。在自我改进循环的每次迭代中，系统会从维护的前沿集中选择评分最高的程序，并将其指定为父级。前沿集是一个有界的顶尖程序集合，通过 Git 标签（前缀为 frontier/）进行跟踪。随后，系统会调用提议者智能体来分析抽样失败案例，并对父级配置进行变异，从而生成一个子程序。子程序继承父级的所有属性（系统提示词、工具权限），同时进行一项有针对性的修改：在仅技能模式下新增或编辑技能文件，或在仅提示词模式下重写系统提示词。创建子程序时，系统会检出父分支，创建名为 iter-mode-n 的新分支（其中 n是全局迭代索引），写入变异后的配置，并提交所有更改。子程序在留出验证集上完成评估后，若前沿集尚未达到最大容量 K （默认值为 K=3），或者其评分高于当前评分最低的前沿成员，就会被纳入前沿集；在后一种情况下，评分最低的成员会被移出。未能进入前沿集的子程序会被丢弃，并删除其分支，以防止代码仓库膨胀。这种设计在程序空间中形成树结构搜索，每个节点都是完全可复现的快照：沿着任意程序的父级指针回溯至根节点，即可重建其谱系；通过一次 Git 检出，即可恢复任何历史配置。若连续迭代达到可配置的次数而前沿没有改进，则提前停止准则会终止循环。