arXiv:2611.33098v1 [cs.AI] 24 Nov 2026

Delegent：一种降低成本且不牺牲性能的多智能体代理框架

Lingwei Gu Jimmy Lin

David R. Cheriton计算机科学学院 滑铁卢大学

摘要

基于前沿语言模型构建的智能体成本高昂。每一步中，前沿模型都会重新读取其完整上下文，并执行每个子任务，不论其难度如何。本文探讨了一个简单的论点：通过将简单子任务委派给运行在自有硬件上的小型语言模型（SLM），可以省去其中大部分成本。为验证这一点，我们构建了 Delegent——一个基于 Claude Code 的简单而有效的智能体框架。在该框架中，前沿模型始终负责所有任务，但会将调查代码、运行测试和搜索语料库等特定子任务交给 SLM 子代理。级联和模型路由器等现有方法通常会选择由哪个模型执行整个任务；而在 Delegent 中，前沿模型始终主导执行过程，并将简单子任务委派给 SLM。由于无需训练路由器或模型，因此可以自由替换任一模型。借助 27B FP8小型语言模型，Delegent 将长周期编码的前沿模型成本降低了 67%，将智能体搜索的成本降低了 59%，且任务性能没有下降；即使是可在笔记本电脑上运行的 20 亿参数小型语言模型，也能将编码成本降低 66%。前沿模型原本需要亲自完成的繁琐工作越多，节省的成本就越大。一条简短的指令——将主要编码任务留给前沿模型——就能实现大部分节省，并避免 20 亿参数小型语言模型丢失任务。代码：
https://github.com/castorini/delegent。

1 引言

智能体本质上就是在循环中使用工具的模型。工具框架会调用模型、执行其工具调用、返回结果，再次调用模型，直到任务完成。这个循环很简单，但成本高昂，因为模型每一步都要重新读取全部上下文，而上下文会随着每次工具调用不断增长。因此，一个需要经过一百步才能解决问题的 Frontier 代码编程智能体，就要为一百份不断变长的提示词付费

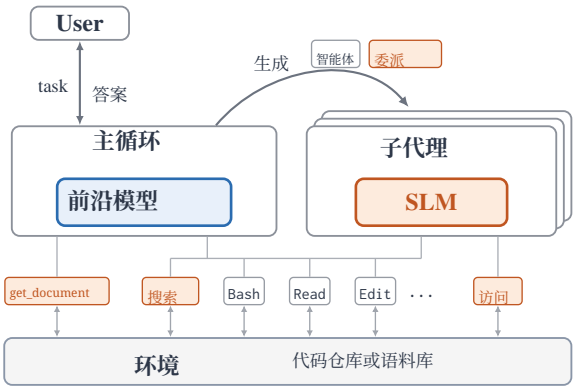

图1：Delegent 工具框架。它以原版 Claude Code 工具框架为基础，橙色部分是我们新增的内容。用户与主循环交互，由前沿模型主导任务并生成子代理。各循环共享大部分工具，并各自增加了少量工具。

。因此，对于按 token 付费的组织而言，长时程智能体成本高昂；而使用订阅计划的个人则会耗尽配额，只能等待下一次重置。

然而，前沿模型在每一步所做的许多事情都是简单的繁重工作，例如定位相关代码、重现错误、运行测试、执行搜索以及阅读文档。这些工作似乎很少需要前沿模型，但现有智能体却用前沿模型处理所有这些工作，甚至包括子代理中的工作，每一步都按前沿模型的价格计费。小型语言模型（SLM）的成本低得多，而且如果已经拥有图形处理器，也无需额外花费。小型语言模型单独处理这些任务时，表现远不及前沿模型，无法取而代之；但它不必与前沿模型竞争，只要前沿模型继续掌控全局，并仅委派简单的子任务即可。由此得出一个简单的论点：

> 在本地硬件上运行的小型语言模型，可以接手前沿模型的简单子任务，大幅降低成本，同时不影响任务表现，也无需训练。

为检验这一论点，我们构建了 Delegent（图1），这是一种让前沿模型与本地小语言模型协同工作的智能体框架。在代码编程方面，Delegent 基于 Claude Code 构建。[1] 它保留了 Claude Code 的智能体循环和子代理，但会按角色分配每次模型调用：主循环的调用交由前沿模型，子代理的调用则交由小语言模型。在智能体搜索方面，它还加入了一种委派工具，其子代理会搜索语料库，并返回附有证据的报告。前沿模型始终负责所有任务。只需一条简短指令告诉它要委派哪些内容，工具框架便会让小语言模型执行委派的子任务，并核查报告中的证据；最终答案则完全由前沿模型生成。由于无需训练，两种模型都可以直接替换使用。

作为核心结果，我们发现，这一论点在长周期编码和智能体搜索两项任务中都成立。使用高推理强度的 GPT-6.1-Sol 作为前沿模型、Qwen3.8-27B-FP8 作为小语言模型时，在 DeepSWE 上，前沿模型的成本降至三分之一，同时解决的任务数量没有减少。在 694 个搜索问题上，其成本降至 41%，且正确回答的问题数量与单独使用该模型时完全相同。即使使用 2B 模型 MiniCPM5-2B，DeepSWE 上的前沿模型成本也能降至三分之一；即便按 API 价格为 Qwen3.8-27B-FP8 付费，每项 DeepSWE 任务的总成本仍仅为 38%。委派会增加一些延迟，而延迟受运行小语言模型的本地硬件性能限制。

与当前结合廉价模型和昂贵模型的主流方法不同，我们不会先让小语言模型给出答案，再决定是否信任它（Chen 等人，2024年；Aggarwal 等人，2024年；Yue 等人，2024年），也不会训练一个路由器，为每个查询挑选一个模型（Ding 等人，2024年；Ong 等人，2025年）。在这两种情况下，模型都会接到完整任务，并必须独立完成；而 2B 小语言模型通常难以胜任。Delegent 中，前沿模型始终负责生成最终答案，小语言模型只处理繁重工作。小语言模型的任何输出都必须先由前沿模型读取，才能被采纳。因此，我们的贡献是一种框架工程：协调两个成本相差悬殊的模型。

为了更准确地界定我们的论断，我们来进一步说明：

• 这里的成本，指的是前沿模型向用户收取的费用，并已将提示缓存纳入考量。小语言模型运行在用户掌控的图形处理器上，而对于代码编程，我们也按 API 价格为 Qwen3.8-27B-FP8 定价。

• 我们所说的本地，是指在用户控制下、由单个图形处理器运行的小型语言模型。27B FP8 模型需要工作站图形处理器，而 2B 模型可在笔记本电脑上运行。

•我们所说的免训练，是指不训练或微调任何模型，因此小型语言模型和前沿模型都可以自由替换。

本文有三项贡献。首先，我们构建了 Delegent——一个开源、免训练的框架，让前沿模型将工作委派给本地小型语言模型。我们在第2节对此进行了介绍。其次，我们使用两个规模迥异的小型语言模型，在三个代码编程数据集和五个搜索数据集上对其进行评估，并在第 4 节表明，在九项评估中的八项里，前沿模型成本降低了一半以上，同时任务性能保持不变。第三，我们在第 5节分析了节省来自何处，以及这些节省如何取决于我们的指令和前沿模型。

2 Delegent 工具框架

Delegent 基于 Claude Code 构建，协调两种成本差异很大的模型，如图 1所示。由前沿模型 主导每项任务。它负责规划、决定委派哪些工作以及是否相信返回的结果，并编写最终补丁或答案。小语言模型子代理在本地图形处理器上承担繁重工作，每个子代理都在各自的工具循环中运行，并使用全新的上下文。Delegent 让前沿模型判断任务类型是代码编程还是搜索，并提供相应的工具集。

按角色路由模型调用。在 Claude Code 中，主循环每一步都会调用模型，而它通过内置 Agent 工具启动的每个子代理也会如此。次要任务交给小型辅助模型。对于每项任务，Delegent 都会在 Claude Code 与模型之间放置一个中继器。该中继器使用 Anthropic Messages API，因此 Claude Code 无需修改即可运行，并根据每次调用所指定的模型来路由调用。主循环发出的调用交给前沿模型，子代理和辅助模型发出的调用交给 SLM。任务中的每次前沿模型调用都携带相同的提示缓存键，因此连续步骤可以命中提供商缓存。SLM 调用通过代理发送至 vLLM（Kwon 等，2023年），代理会将每个子代理固定到一个 GPU 副本上，从而使其不断增长的前缀保持缓存。

[1] https://code.claude.com/docs

运行子任务。对于代码编程，前沿模型通过其 Agent 工具委派一个子任务，并撰写一份简要说明，即给 Claude Code 子代理的指令。子代理在同一个容器中使用 Claude Code 的默认工具工作。对于搜索，Delegent 添加了一个 `Delegate` 工具，用于委派一个聚焦的搜索子任务，例如查找或核实某项事实。我们没有从头设计搜索子代理，而是借鉴了 AREX（Lu 等，2026年），这是一个深度研究代理系列。其最小成员 AREX-Turbo 以 40 亿参数模型为核心构建，因此其提示词和工具已经按照小模型易于遵循的方式编写。AREX-Turbo 智能体通过四种工具循环工作：搜索语料库、访问找到的文档、在历史记录变得过长时将自身历史压缩成笔记，并以答案、支持该答案的证据以及置信度结束。我们的 SLM 子代理只能看到子任务和问题，而它的最终答案、证据和置信度会组成返回给前沿模型的报告。在代码编程和搜索中，同一轮次的子任务会并发运行。

返回报告。 前沿模型的上下文只会接收子代理的最终报告，而不会接收其任何工具输出。对于搜索，工具框架还会在所引用的文档中查找每一条证据，并记录该证据是否以原文形式出现。

引导前沿模型。 一条简短的指令（第 A 节）向前沿模型解释了其中的经济性。由于每一步都会重新读取完整的上下文，成本最低的方案就是让前沿模型自身执行的步骤最少，因此该指令要求它将工具密集型工作委派出去，并将自身的工具使用限制在少数有针对性的检查上。它涵盖了两种委派方式：Claude Code 用于代码编程的子代理，以及用于搜索的 `Delegate` 工具。

3 实验设置

任务。对于代码编程，我们使用从 SWE-bench Verified（Jimenez 等人，2024年）中随机抽取的样本，[2] SWE-bench 专业版（Deng 等人，2025年）以及 DeepSWE v1.1（Huang 等人，2026年），抽样方式见 C 节。对于搜索，我们使用 200 个 BrowseComp-Plus 问题（Chen 等人，2025年），从未用于撰写指令的问题中随机抽取；此外还使用了 CMASS（Sharify 等人，2026年中的四个数据集。CMASS 是一个智能体搜索基准基于 ClimbMix（Diao 等人，2025年）的开放网络语料库包含 4000 亿个词元，智能体只能通过 BM25 搜索访问该语料库（Robertson 和 Zaragoza，2009年；Lin 等人，2021年）。CMASS 数据集包括 BrowseComp-Plus、Total Recall QA（Rafiee 等人，2026年）、HotpotQA（Yang 等人，2018）和 DeepSearchQA（Gupta 等人，2026年），我们将它们分别简称为 TRQA 和 DSQA。DSQA 问题有多个正确答案，CMASS 以两种模式对其进行评估。在找出一个模式下，任一正确答案都计为正确；而找出全部模式则要求答案集合完全一致。

模型。 我们的所有运行均使用 Delegent 工具框架，该框架构建于 Claude Code 之上，并且可以自由更换其调用的前沿模型。我们的主要实验调用 GPT-6.1-Sol，通过 OpenAI API 访问；在 DeepSWE 上，我们使用高强度推理力度的 Claude Opus 5.5 进行消融实验。我们以高推理力度运行 Sol，分别测试有无委派的情况，以控制实验成本。在这一力度下，它单独可以解决略超过一半的 DeepSWE 任务。我们为它配备了两种规模差异很大的小型语言模型，二者都易于在单台设备上运行。Qwen3.8-27B-FP8[3] 可在一块工作站图形处理器上运行，而 MiniCPM5-2B（Hu 等人，2024年；MiniCPM 团队，2025年）小到足以在笔记本电脑上运行。我们使用 vLLM 为二者提供服务，每个副本使用一块图形处理器，并且仅在代码编程任务上测试 MiniCPM5-2B。

基线。 我们在相同任务上，仅使用前沿模型进行对比，采用相同的工具框架，但不使用委派工具和委派指令。对于编码任务，这一设置使用的是 Claude Code 及其默认工具和子代理；我们将每次调用都发送给前沿模型，包括 Claude Code 默认发送给 Haiku 的调用。作为合理性检查，在此设置下，仅 Claude Opus 5.5 就能解决我们 68% 的 DeepSWE 任务，这与 Artificial Analysis 报告的 Claude Code 中 Claude Opus 5.5 达到 68.4% 的结果一致。[4]

指标。 如果最终补丁通过了任务的隐藏测试，则视为该编程任务已解决。搜索答案由 GPT-6.1-Sol 根据标准答案进行评分，采用低投入设置和官方 BrowseComp-Plus 评分提示词（Chen 等人，2025年），且同一评判模型会为每个系统评分。所有代码编程运行，包括仅使用前沿模型的运行，均使用 Claude Code，采用其默认工具（网页访问除外）；前沿模型的上下文上限为 1M 词元

[2] https://openai.com/index/introducing-swe-bench-verified/

[3] https://qwen.ai/blog?id=qwen3.8

[4] https://x.com/ArtificialAnlys/status/2102932119995756613，针对 DeepSWE v1.1 中以最高力度运行的 Claude Opus 5.5。

，小语言模型的上下文上限为 131k 词元，并且轮次和时间限制相同，具体限制见第 B 节。延迟指每项任务的实际经过时间。

成本。 我们按照前沿模型的 API 价格计算前沿模型成本，[5] 并对每个系统启用提示缓存：

$$\text{cost} = \$2\,T_{\text{in}}^{\text{uncached}} + \$0.10\,T_{\text{in}}^{\text{cached}} + \$10\,T_{\text{out}},$$

其中 T 表示以百万为单位的词元数，输出包含推理令牌。我们报告每项任务的成本，以及相对于仅使用前沿模型的成本。对于代码编程，我们还按照 Chutes 的 API 价格为 Qwen3.8-27B-FP8 定价；Chutes 是一家提供同款 FP8 模型的服务商，并且同样启用了提示缓存。输入费用为每百万词元 0.039 美元；如果输入内容与子代理上一次的请求重复，费用则为其十分之一。输出费用为每百万词元 1.25 美元。

4 结果

代码编程。 论点在三个编码数据集上均成立（表 1）。以前沿模型作为子代理，使用 Qwen3.8-27B-FP8 后，前沿模型解决的任务数量不少于单独使用时，成本则降至三分之一到二分之一。在 DeepSWE 上，前沿模型的每项任务成本从 2.11 美元降至 0.70 美元。使用 MiniCPM5-2B 得出的结果也大致相同：任务性能保持不变；在 DeepSWE 上，2B 模型节省的成本与 27B FP8 模型一样多。

为小语言模型付费，对长任务的影响不大。按 API 价格计算，Qwen3.8-27B-FP8 会使每项 DeepSWE 任务增加约 10 美分，因此总成本仍为前沿模型成本的 38%；即使在 SWE-bench Verified 的短任务上，总成本也仅为前沿模型成本的 57%。

搜索。 在全部 694 个搜索问题中（表 2），前沿模型以 Qwen3.8-27B-FP8 作为子代理，正确回答了 663 个，与单独运行时完全相同，成本仅为前沿模型成本的 41%。在任何一组数据中，两个系统的正确回答数相差都不超过三个。不过，节省的成本差异相当大。在 ClimbMix 上，前沿模型成本降至原来的四分之一到三分之一；但在 BrowseComp-Plus 自有语料库上，仅降至原来的 63%。

差异在于前沿模型需要自行阅读多少内容。在 BrowseComp- Plus 上，语料库由一组文件组成，像 rg 这样的命令只会返回匹配的行。在 ClimbMix 上，唯一的访问方式是 BM25 搜索，它会返回整篇文档，供模型随后阅读

[5] https://developers.openai.com/api/docs/pricing，访问于 2026年10月。

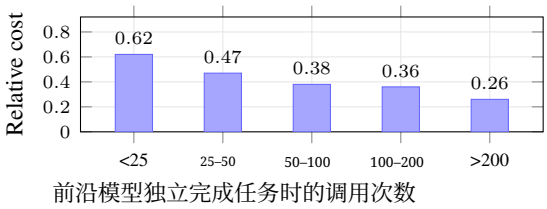

图2：按前沿模型独立完成任务时的调用次数划分的每个编程任务的前沿模型相对成本中位数。汇总了来自三个编码数据集和两种小型语言模型的350组任务对。

一次阅读一页。因此，在 ClimbMix 上回答相同的问题所需的阅读量是前者的两倍多，而委派则将大部分阅读工作交给小语言模型。由于网页代理也会通过搜索引擎查找页面并打开页面，我们认为 ClimbMix 上的节省更具代表性。

提示缓存。 所有系统（包括单独运行的前沿模型）都使用提供商的提示缓存，并为每个任务使用稳定的键，这符合任何实际部署的做法，因此所有节省的成本都是相对于已缓存的基线衡量的。无论是否进行委派，代码编程中的前沿模型输入几乎全部都经过缓存。

延迟。 委派会增加延迟，延迟主要取决于为小语言模型提供服务的本地硬件。使用 Qwen3.8-27B-FP8 时，DeepSWE 任务的中位耗时为38分钟，而非25分钟；短时的 SWE-bench Verified 任务耗时则约为原来的两倍。

5 分析

这些结果留下了两个问题：节省的计算资源从何而来，以及为何任务性能能够保持不变。我们利用根据运行记录计算出的统计数据来回答这两个问题。除非另有说明，我们使用 DeepSWE，这是任务集里最难的一个。其任务流程较长，仅靠前沿模型只能解决略多于一半的任务，因此 DeepSWE 留出了最大的节省空间，也能够揭示任务性能的任何损失。

不同任务的节省幅度有何差异？ 某项任务节省多少，取决于前沿模型独自完成该任务时需要做多少工作。图2 按前沿模型在不采用委派时的调用次数对编码任务进行分组。对于调用次数少于 25 次的任务，委派可将前沿模型成本降至独自完成时的约 60%；其中有 11% 的任务采用委派后成本反而更高。随着任务耗时变长，相对成本会持续下降；对于调用次数超过 200 次的任务，成本约为四分之一，

任务集	系统	已解决	前沿模型成本（每项任务的美元成本）	相对 cost	使用小语言模型的成本按 API 价格计算	前沿模型调用每个任务	小型语言模型调用次数每个任务
已验证（50）	仅使用前沿模型	46	0.244	100%	0.244（100%）	32.4	–
	+ Qwen3.8-27B-FP8	46	0.108	44%	0.139（57%）	12.7	32.7
	+ MiniCPM5-2B	47	0.129	53%	–	15.2	79.0
专业版（75）	仅前沿模型	63	0.617	100%	0.617（100%）	61.2	–
	+ Qwen3.8-27B-FP8	63	0.227	37%	0.266（43%）	19.9	42.7
	+ MiniCPM5-2B	62	0.274	44%	–	23.1	65.5
DeepSWE (50)	仅使用前沿模型	29	2.110	100%	2.110 (100%)	176.1	–
	+ Qwen3.8-27B-FP8	29	0.704	33%	0.806 (38%)	46.6	82.0
	+ MiniCPM5-2B	30	0.712	34%	–	48.1	106.1

表 1：以 GPT-6.1-Sol 为前沿模型的代码编程结果。相对成本是相对于仅使用前沿模型的成本；使用小语言模型时的成本按 Qwen3.8-27B-FP8 的 API 价格计算（第 3 节）；没有 API 提供 MiniCPM5-2B。

任务集	系统	已解决	前沿模型成本（每项任务的美元成本）	相对 cost	前沿模型调用每个任务	小型语言模型调用次数每个任务
BrowseComp-Plus† (200)	单独使用前沿模型	193	0.114	100%	8.8	–
	+ Qwen3.8-27B-FP8	192	0.071	63%	5.7	305.2
BrowseComp-Plus（80）	单独使用前沿模型	72	0.211	100%	13.9	–
	+ Qwen3.8-27B-FP8	71	0.066	31%	6.2	495.7
TRQA（68）	仅使用前沿模型	64	0.048	100%	6.2	–
	+ Qwen3.8-27B-FP8	67	0.015	31%	3.1	69.1
HotpotQA（204）	仅使用前沿模型	198	0.033	100%	5.4	–
	+ Qwen3.8-27B-FP8	198	0.008	25%	2.5	29.3
DSQA，查找一个（71）	单独使用前沿模型	70	0.067	100%	7.7	–
	+ Qwen3.8-27B-FP8	70	0.020	29%	3.6	106.2
DSQA，查找全部（71）	单独使用前沿模型	66	0.121	100%	10.3	–
	+ Qwen3.8-27B-FP8	65	0.032	26%	4.3	223.9
全部（694）	仅使用前沿模型	663	0.091	100%	8.2	–
	+ Qwen3.8-27B-FP8	663	0.037	41%	4.2	194.2

表 2：搜索结果及每个问题的前沿模型成本。†其自身语料库；其他所有数据集均使用 ClimbMix.

而且委派从不会增加成本。在搜索任务中，相应的衡量指标是前沿模型读取的内容量（第4节）。因此，可以根据前沿模型独自处理一项任务所需的时间，预估节省幅度。

节省来自哪里？ Claude Code 默认已经会进行委派，但委派给的是前沿模型本身。在没有我们的工具框架时，GPT-6.1-Sol 会在 98% 的 DeepSWE 任务中启动子代理，而这些同样运行在 GPT-6.1-Sol 上的子代理，其调用次数超过主循环，占其成本的 53%。Delegent 将这部分工作转移给小语言模型，这解释了 80% 的节省。其余节省来自前沿模型本身：现在它每个任务只需进行 46 到 48 个大语言模型轮次，而不是 84 个。它读取和编辑代码的频率与之前大致相同，但测试运行次数从每个任务约 8 次降至 1 到 2 次，而且不再几乎承担每个任务 23 次调用中的大部分，而这些调用原本用于维护任务列表并与其子代理通信。由于只有报告会返回，传达到前沿模型的工具输出也会减少：在 DeepSWE 上减少 11% 到 12%，在 SWE-bench Pro 上减少 13% 到 24%。

小语言模型接手哪些工作？ 小语言模型接手查找和核验这些繁重工作，而前沿模型则自行处理主要的代码编程任务。在代码编程任务中，子代理通常会在代码仓库中搜索与问题相关的代码，阅读相关文件，并运行测试或小型脚本来重现错误或确认修复效果。对于每个 DeepSWE 任务，Qwen3.8-27B-FP8 的子代理会进行约 24 次代码搜索，阅读约 30 个文件或文件片段，运行测试 12 次，并自行运行 5 个脚本。它们编写的文件每个任务约有 4 个，几乎全都是放在代码仓库之外的一次性脚本。MiniCPM5-2B 的工作方式相同，但步骤更多；在 50 个执行轮次中，有 9 次还会编辑源文件。在搜索任务中，子代理会——

Frontier + 子代理	已解决	LLM cost	LLM 轮次
没有我们的指令			
仅 Sol	29	100%	84
Sol + Qwen	28	55%	101
Sol + MiniCPM	22	51%	90
使用我们的指令			
Sol + Sol	34	60%	52
Sol + Qwen	29	33%	46
Sol + MiniCPM	30	34%	48

表 3：在 50 个 Deep-SWE 任务上，采用和未采用我们的指令时的表现。仅使用 Sol 时，其子代理也由 Sol 运行。大语言模型成本是 Sol 收取的费用，包括任何 Sol 子代理产生的费用，相对于仅使用 Sol 的成本；不包括小语言模型。

进行自己的 BM25 搜索，阅读找到的文档，并返回附有引述证据、来源和置信度的答案；随后，执行框架会逐条核对引述内容与其来源。在这两种情况下，前沿模型都会收到易于核验的调查结果，并自行决定答案。

指令是否必要？ 表3 通过改变指令以及运行子代理的模型，展示了指令的必要性，尤其是对于较弱的小语言模型而言。没有指令时，将任务委派给小语言模型可使前沿模型成本降低约一半。使用 Qwen3.8-27B-FP8 时，前沿模型以 55% 的成本解决了 50 个任务中的 28 个；但使用 MiniCPM5-2B 时，它只解决了 22 个，比单独运行时少 7 个。采用指令后，两种小型语言模型的表现都与单独运行前沿模型时持平，而成本仅为其三分之一，因此任务性能不再取决于小语言模型。即使子代理仍由 GPT-6.1-Sol 运行，仅靠指令也能将前沿模型成本降至 60%，且任务性能毫无损失；再将这些子代理换成小语言模型，还能进一步降低 27 个百分点。因此，节省的成本大约五分之三可归因于指令，五分之二可归因于小语言模型。

小语言模型的能力重要吗？ 根据我们的实验，影响不大，尽管这两个小语言模型单独使用时的表现相差甚远：在相同的 30 个 DeepSWE 任务中，Qwen3.8-27B-FP8 能解决 5 个，而 MiniCPM5-2B 一个也解决不了。作为子代理时，它们带来的任务性能相同，前沿模型成本也几乎相同（表 1）。由于前沿模型会分派范围狭窄的子任务，并自行处理主要的编码任务，较弱的小语言模型主要会增加小型语言模型调用次数——每个 DeepSWE 任务 106 次，而不是 82 次——而非降低准确率。

每个 DeepSWE 任务	Sol	Opus
单独运行时的工作方式	漫长的循环：edit, test, fix	先规划，然后执行
单独解决，共 50 个	29	34
仅使用大语言模型轮次	84	18
搭配小语言模型，遵循我们的指令		
已解决，共 50 个	29.5	38
大语言模型轮次	47	13
交由小语言模型处理的轮次	94	21
首次大语言模型编辑前的小语言模型轮次	25%	59%
每份简报中的词数	45	130
前沿模型成本与仅使用大语言模型相比	34%	69%

表4：Delegent 在 DeepSWE 上使用的两个前沿模型。使用小语言模型时，数值取两个小型语言模型的平均值。简报是前沿模型为子代理撰写的内容。

换用不同的前沿模型后会有哪些变化？
节省幅度取决于前沿模型的工作方式，表 4展示了 GPT-6.1-Sol 和 Claude Opus 5.5 的情况。GPT-6.1-Sol 单独运行时，会进行编辑、测试和修复的长循环。它每个 DeepSWE 任务需要 84 个大语言模型轮次，并在第 18 轮才进行首次编辑。相比之下，Claude Opus 5.5 会在行动前先做规划，仅需 18 轮，并在第 7 轮进行首次编辑。委派也遵循这些习惯。GPT-6.1-Sol 会委派循环中的大量工作，而且大多是在开始编辑后才委派，因此每个任务约有 94 个轮次转移给小语言模型，其成本降至 34%。Claude Opus 5.5 则主要在首次编辑前进行委派，以了解即将修改的代码，并撰写更长的说明，约 130 个词，而不是 45 个词。只有约 21 个轮次转移给小语言模型，其成本降至 69%，这与其自身轮次大致成比例；自身轮次从 18 轮降至 13 轮。无论使用哪个小语言模型，Claude Opus 5.5 解决的任务数量也略高于单独运行时。因此，对于大量自行承担繁重工作的前沿模型，Delegent 节省的成本最多；而对于先进行规划的前沿模型，小语言模型主要用于为其规划提供信息。

6 相关工作

FrugalGPT（Chen 等人，2024年）等采用级联的方法，以及 RouteLLM（Ong 等人，2025年）等路由器，会决定由哪个模型作答；而在 Delegent 中，前沿模型始终负责作答。在 Minions（Narayan 等人，2025年）中，云端模型会将长文档任务分解后交给本地模型；我们研究的是长周期的智能体任务，在这些任务中，前沿模型决定交接哪些工作。编排器—工作器设计在智能体中很常见，

[6] 而Belcak 等人（2025年）认为，小型语言模型适用于许多智能体工作。

与我们工作的同时，Samway 等人(2026年) 研究了由编码智能体组成的团队：主智能体将子任务委派给工作智能体。他们发现，低成本模型作为规划器很有用，但作为执行者则不然。[7] 他们的工作智能体是 API 模型，团队会投入更多计算资源以获得更高奖励；而 Delegent 则将任务委派给本地小语言模型，以降低前沿模型成本。我们的消融实验也印证了这一点：如果没有我们的指令，20 亿参数小型语言模型就会漏掉任务。

7 结论

使用 Delegent 时，前沿模型在 DeepSWE 上的花费仅为原来的三分之一，在智能体搜索上的花费仅为原来的 41%，而任务性能毫不逊色；前沿模型原本需要承担的繁重工作越多，节省的成本就越大。Claude Code 已经委派了许多繁重工作，但委派对象仍是前沿模型。只需一条简短指令，将主要编码任务留给前沿模型，这种方法便能奏效；有了这条指令，即使是 20 亿参数小型语言模型也不会漏掉任何任务。节省多少还取决于所用的前沿模型。反复迭代的模型节省最多，而先做规划的模型则主要利用小语言模型来辅助制定计划。这些结果支持了我们的论点：本地小型语言模型无需训练，就能接手前沿模型的大量工作，在不损害任务性能的情况下削减其成本。

局限性

依赖 Claude Code。 在代码编程方面，Delegent 原样复用 Claude Code 的智能体循环和子代理，仅在其外围增加工具框架：根据角色将每次模型调用路由至前沿模型或本地小型语言模型，在两端提供缓存感知服务，并加入引导委派的指令。在搜索方面，它还增加了委派工具和证据核查；其小语言模型子代理使用 AREX-Turbo 的提示词和工具，而 AREX-Turbo 的最终报告会为每项论断注明来源，这正是证据核查所需要的。因此，工具框架工程的工作量并不大，而我们的发现是，这已足以消除大部分前沿模型成本。

小型语言模型的成本。 小语言模型并非免费。我们报告前沿模型成本，因为它是按用量计费且稀缺的资源，而我们的两种小型语言模型都能在用户可以自行拥有的硬件上运行。在 Chutes 上，同一 FP8 模型的 API 价格为每百万输入词元 0.039 美元；对于子代理上一次请求中重复的输入，价格是其十分之一。在我们的运行中，这类输入占其输入量的 86% 到 90%。输出词元价格为每百万 1.25 美元。Qwen3.8-27B-FP8 每项编程任务会增加 3 到 10 美分的成本，而总成本仍为仅使用前沿模型时的 38% 到 57%。没有 API 提供 MiniCPM5-2B。我们只计算小语言模型用于代码编程的成本，因为我们的搜索日志没有记录每次调用的词元数。

任务和模型的覆盖范围。 我们仅研究智能体代码编程和智能体搜索，尽管这两项是当今模型最重要的能力之一。我们的主要结果采用高推理力度的 GPT-6.1-Sol。我们只在 DeepSWE 上测试 Claude Opus 5.5 作为前沿模型，并且只在代码编程任务上测试 20 亿参数小型语言模型。在代码编程方面，Delegent 构建于 Claude Code 之上，而 Claude Opus 5.5 很可能针对 Claude Code 进行了调优，因此在这方面可能更有利于 Claude Opus 5.5。在智能体搜索方面，前沿模型采用各语料库的标准设置；GPT-6.1-Sol 单独就回答了 200 个 BrowseComp-Plus 问题中的 193 个，因此并未受到限制。委派的工作量取决于前沿模型遵循指令的能力，而这一能力在其他前沿模型上可能有所不同。

延迟。 委派会使任务变慢，在较短的编程任务中，中位延迟约为原来的两倍，这可能不适合交互式使用。具体变慢多少取决于为小语言模型提供服务的硬件。我们的延迟是在共享的多种图形处理器上测得的，实际情况会因图形处理器、服务栈和负载而异。

不声称性能有所提升。智能体运行并非确定性的，不同前沿模型采用委派的方式也各不相同。GPT-6.1-Sol 将繁重工作交出去以节省成本，而 Claude Opus 5.5 则利用子代理检查自身工作。因此，并不能保证每个前沿模型或任务集的任务性能都会提升。我们仅论断，委派不会造成任务性能损失，即使委派系统解决的任务比单独使用前沿模型更多也是如此。另一方面，在不同任务集和模型中，前沿模型成本的降低始终如一。

[6] https://www.anthropic.com/research/building-effective-agents [7] 另请参阅 https://cohere.com/blog/lessons-from-scaling-teams-of-frontier-coding-agents

参考资料

Pranjal Aggarwal、Aman Madaan、Ankit Anand、Srividya Pranavi Potharaju、Swaroop Mishra、Pei Zhou、Aditya Gupta、Dheeraj Rajagopal、Karthik Kappaganthu、Yiming Yang、Shyam Upadhyay、Manaal Faruqui 和 Mausam。2024年。AutoMix：自动混合语言模型。载于神经信息处理系统进展。arXiv:2310.12963。

Peter Belcak、Greg Heinrich、Shizhe Diao、Yonggan Fu、XinDong、Saurav Muralidharan、YingyanCeline Lin 和 Pavlo Molchanov。2025年。小型语言模型是智能体人工智能的未来。*arXiv* 预印本 *arXiv:2506.02153*。

Lingjiao Chen、Matei Zaharia 和 James Zou。2024年。FrugalGPT：如何在降低成本并提升性能的同时使用大型语言模型。机器学习研究汇刊。arXiv:2305.05176。

Zijian Chen、Xueguang Ma、Shengyao Zhuang、Ping Nie、Kai Zou、Andrew Liu、Joshua Green、Kshama Patel、Ruoxi Meng、Mingyi Su、Sahel Sharifymoghaddam、Yanxi Li、Haoran Hong、Xinyu Shi、Xuye Liu、Nandan Thakur、Crystina Zhang、Luyu Gao、Wenhu Chen 和 Jimmy Lin。2025年。BrowseComp-Plus：一种更加公平、透明的深度研究智能体评测基准。*arXiv* 预印本 *arXiv:2508.06600*。

Xiang Deng、Jeff Da、Edwin Pan 等。2025年。SWE-Bench Pro：人工智能智能体能否解决长时程软件工程任务？*arXiv* 预印本 *arXiv:2509.16941*。Shizhe Diao、Yu Yang、Yonggan Fu、Xin Dong、Dan Su、Markus Kliegl、Zijia Chen、Peter Belcak、Yoshi Suhara、Hongxu Yin、Mostofa Patwary、Yingyan Celine Lin、Jan Kautz 和 Pavlo Molchanov。2025年。Nemotron-CLIMB：用于语言模型预训练的基于聚类的迭代式数据混合自举。发表于神经信息处理系统进展。arXiv:2504.13161。Dujian Ding、Ankur Mallick、Chi Wang、Robert Sim、Subhabrata Mukherjee、Victor Rühle、Laks V. S. Lakshmanan 和 Ahmed Hassan Awadallah。2024年。混合大语言模型：兼顾成本效益与质量的查询路由。发表于国际学习表征会议。arXiv:2404.14618。Nikita Gupta、Riju Chatterjee、Lukas Haas、Connie Tao、Andrew Wang、Chang Liu、Hidekazu Oiwa、Elena Gribovskaya、Jan Ackermann、John Blitzer、Sasha Goldshtein 和 Dipanjan Das。2026年。DeepSearchQA：弥合深度研究代理的全面性差距。*arXiv* 预印本 *arXiv:2601.20975*。Shengding Hu、Yuge Tu、Xu Han 等。2024年。MiniCPM：借助可扩展训练策略，揭示小型语言模型的潜力。发表于语言建模会议。arXiv:2404.06395。

Wenqi Huang、Charley Lee、Leonard Tng 和 Serena Ge。2026年。DeepSWE：在真实的长时程工程任务上衡量前沿编码代理。*arXiv* 预印本 *arXiv:2607.07946*。

Carlos E. Jimenez、John Yang、Alexander Wettig、Shunyu Yao、Kexin Pei、Ofir Press 和 Karthik Narasimhan。2024年。SWE-bench：语言模型能否解决现实世界中的 GitHub问题？发表于国际学习表征会议。

Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph E. Gonzalez、Hao Zhang 和 Ion Stoica。2023年。使用 PagedAttention 高效管理大型语言模型服务的内存。发表于第二十九届操作系统原理研讨会论文集。

Jimmy Lin、Xueguang Ma、Sheng-Chieh Lin、Jheng-Hong Yang、Ronak Pradeep 和 Rodrigo Nogueira。2021年。Pyserini：用于可复现信息检索研究的 Python工具包，支持稀疏和稠密表示。发表于第四十四届 *ACM* 国际信息检索研究与发展会议论文集，第 2356–2362 页。

Shuqi Lu、Chaofan Li、Kun Luo、Zhang Zhang、Hui Wang、Hongwang Xiao、Lei Xiong、JiahaoWang、Sen Wang、Xiyan Jiang、Wanli Li、Yuyang Hu、Hongjin Qian、Bingyu Yan、Jianlyu Chen、Ziyi Xia、Yingxia Shao、Kang Liu、Zhicheng Dou及其他5位作者。2026年。AREX：迈向用于深度研究的递归自我改进代理。*arXiv* 预印本 *arXiv:2607.21461*。

MiniCPM 团队。2025年。MiniCPM4：面向终端设备的超高效大语言模型。*arXiv* 预印本 *arXiv:2506.07900*。Avanika Narayan、Dan Biderman、Sabri Eyuboglu、Avner May、Scott Linderman、James Zou 和 Christopher Ré。2025年。端侧与云端语言模型之间的高效协作。载于第*42*届国际机器学习会议论文集，PMLR 第267卷，第45682–45719页。arXiv:2502.15964。Isaac Ong、Amjad Almahairi、Vincent Wu、Wei-Lin Chiang、Tianhao Wu、Joseph E. Gonzalez、M. Waleed Kadous 和 Ion Stoica。2025年。RouteLLM：利用偏好数据学习大语言模型路由。载于国际学习表征会议。arXiv:2406.18665。Mahta Rafiee、Heydar Soudani、Zahra Abbasiantaeb、Mohammad Aliannejadi、Faegheh Hasibi 和 Hamed Zamani。2026年。Total Recall QA：面向深度研究代理的可验证评估套件。*arXiv* 预印本 *arXiv:2603.18516*。Stephen Robertson 和 Hugo Zaragoza。2009年。概率相关性框架：BM25及其进阶。信息检索基础与趋势，3(4)：333–389。

Keenan Samway、Daniel D’souza、Omar Darwiche Domingues、Marzieh Fadaee、Tan Yi-Chern 和 Nouha Dziri。2026年。通过代码编程智能体团队扩展推理时计算。*arXiv* 预印本 *arXiv:2610.17471*。

Sahel Sharifymoghaddam、Lingwei Gu、Yijun Ge 和 Jimmy Lin。2026年。CMASS：用于评估搜索智能体在异构任务中的表现的 ClimbMix 智能体搜索套件。*arXiv* 预印本 *arXiv:2608.20317*。

Zhilin Yang、Peng Qi、Saizheng Zhang、Yoshua Ben-gio、William W. Cohen、Ruslan Salakhutdinov 和 Christopher D. Manning。2018。HotpotQA：一个用于多样化、可解释的多跳问答的数据集。载于*2018*年自然语言处理实证方法会议论文集，第2369–2380页。

Murong Yue、Jie Zhao、Min Zhang、Liang Du 和 Ziyu Yao。2024年。采用思维表征混合的大语言模型级联实现高效推理。载于国际学习表征会议。arXiv:2310.03094。

A 提示词

代码编程指令。 我们将此指令作为用户级 CLAUDE.md 文件提供给 Claude Code。

```
# 使用子代理 子代理（Agent 工具）使用小型快速模型运行，成本远低于你。你采取的每一步都会重新读取整个对话，因此成本最低的方案就是尽量减少你自己采取的步骤。- 委派调查工作（找出问题涉及的代码并说明其工作原理；编写一个最精简的脚本来复现错误并报告运行结果）和验证工作（运行复现脚本和相关测试；只报告失败项）。代码修改由你自己完成。- 在前台运行子代理（不要使用 run_in_background），并在同一轮中启动彼此独立的子代理，这样它们的报告就能一并返回，无需轮询。- 要求每个子代理提交一份简短报告，最多 150 个词：只包含你需要的事实，并注明文件路径、行号和准确的错误消息。- 不要为这项工作维护任务列表（TaskCreate / TaskUpdate / TaskList / TaskGet）；在你自己的消息中记录计划即可。- 子代理的报告可能有误：在依据某项事实进行修复之前，先亲自核实。
```

搜索指令。 我们将此指令附加到前沿模型的系统提示中。

```
你有一位研究助手：Delegate 工具会交给它一个明确的子任务；它会用自己的搜索引擎搜索同一语料库，并返回一份简短报告，其中包含证据、文档 ID 和置信度。它速度很快，而且高置信度报告通常是正确的。请按以下方式使用它：- 将所有搜索工作都委派出去，包括核查候选答案（“语料库是否证实了 X……？”）。并行运行彼此独立的子任务。- 不要自行进行宽泛搜索或通读整份文档。要确认某一项事实，只进行一次有针对性的搜索，或者只阅读文档中相关段落（使用带偏移量的 get_document）。- 每份报告结尾都会自动在引用的文档中查找其证据：找到的段落与原文完全一致。- 只要证据涵盖问题的每个条件，就立即给出最终答案。
```

示例任务说明。下面是每个前沿模型为同一 DeepSWE 任务中的 Qwen3.8-27B-FP8 子代理撰写的第一份任务说明。该任务要求为 ABS 编程语言解释器添加模块加载标志。GPT-6.1-Sol 的任务说明：

```
检查 ABS 模块的 require 加载器、缓存、环境、测试和项目指令。仅进行只读调查；不要编辑。指出需要修改的文件/函数及相关测试命令，包括环境 stderr 和运行时值的工作方式。报告不超过 150 词，并注明路径和行号。
```

Claude Opus 5.5 的简要说明，精简版：

```
在 /app 的 Go 代码仓库（ABS 编程语言解释器）中，仅报告事实（最多 200 个单词，并注明文件路径和行号；不要修改文件）：

- 内置函数 require(path) 目前是如何实现的：函数名称/文件、它如何解析路径（相对于哪个目录？如何跟踪当前文件所在目录 […]）、是否存在模块缓存（变量名称、键类型），以及如何对模块文件进行词法分析、解析和求值（调用哪个辅助函数）。- 内置函数如何注册（映射的名称/文件、函数签名、如何访问环境 […]）。- CLI 入口点：main.go 和 repl/repl.go 中的 BeginRepl(args []string, version string) […]
```

Claude Opus 5.5 后续的测试简报要求子代理针对新测试文件之外的每个失败测试，检查它在基础分支上是否也会失败，并报告该失败是否早已存在。

B 运行设置

设置项	值
Claude Code 版本	2.1.232
工具	默认启用，但网络搜索和网络抓取除外
前沿模型上下文	最多 1M 个词元
小语言模型上下文	131k 个词元
轮次限制	100 轮
工具时间限制	60分钟（DeepSWE 为90分钟）

表5：所有代码编程运行共用的设置。

代码编程。 表5列出了所有代码编程运行共用的设置。两种小型语言模型在代码编程操作前都会先进行思考，而在搜索时，Qwen3.8-27B-FP8 则会直接作答，不进行思考。

搜索子代理。 在搜索任务中，小语言模型子代理会在我们自有的循环中运行，最多执行80步，并使用 AREX-Turbo 提供的四种工具：

•search 最多接受五条查询，并为每条查询返回排名前10的 BM25 命中结果及其最匹配的段落；这些结果通过 ClimbMix 搜索 API 获取，或者在 BrowseComp-Plus 上通过语料库文件的 BM25 索引获取，索引采用基准测试的默认参数。

• visit 会返回与指定目标最匹配的文档段落。

• update_context 会用自己的笔记替换子代理的历史记录。

• finish 会返回答案、附带来源的证据以及置信度。

我们采用这些提示词和工具的原始版本，但去掉了 AREX-Turbo 的 Google Scholar 工具，也没有对它们进行调优。

用于搜索的前沿模型。前沿模型在 BrowseComp-Plus 语料库上运行时，使用取自一段 Claude Code 录制会话的系统提示词以及 Bash 和 Read 工具，并在我们自行构建的循环中运行，该循环加入了 Delegate 工具。在 ClimbMix 上，它使用官方 BrowseComp-Plus 智能体的搜索和 get_document 工具。启用委派后，前沿模型在 BrowseComp-Plus 上自身工具调用的输出会被截断至 12,000 个字符，而不是 Claude Code 的 30,000 个字符。它的 395 次自身调用中，有 18 次应用了截断。

硬件。 EachSLM 副本会处理多个并发任务，几乎总是由单个图形处理器运行。在代码编程任务中，Qwen3.8-27B-FP8 运行于 NVIDIA H200 和 RTX PRO 6000 Blackwell 图形处理器上，MiniCPM5-2B 则运行于 H200 、 RTX PRO 6000 Blackwell 、 RTX 6000 Ada 和 L40S 图形处理器上。Claude Opus 5.5 的运行也使用了 H100 和 RTX A6000 图形处理器，其中一个 Qwen3.8-27B-FP8 副本分布在两块 A6000 上。在搜索任务中，Qwen3.8-27B-FP8 主要运行于 RTX PRO 6000 和 H100 图形处理器上。由于预算限制，我们通过 Claude 子智能体提供 Claude Opus 5.5，每次调用都使用干净的上下文。

选择工具。 在我们的实验中，工具由任务集决定。之后，我们让 GPT-6.1-Sol 在低投入程度下，通过以下提示在两套工具集之间进行选择：

```
这里有一个任务。 <任务> ..... </任务> 不要执行该任务。判断它属于哪种任务，并且只用一个单词回答：如果要求你在软件仓库中修改代码，则回答代码编程；如果要求你找出某个问题的答案，则回答搜索。
```

对于全部 869 个任务和问题，它选择了我们实验中使用的工具。每个任务的问题成本约为 0.0005 美元，在每个任务集上都低于一个相对成本点，而我们的成本数据未计入这一点。

重新运行与后续运行。Claude Code 会压缩较长的对话，以缩短其上下文。在所有系统中，前沿模型的某次运行若经压缩后失败，便会使用完整的 100 万 token 上下文重新运行。因基础设施故障而失败的搜索问题（694 个中有 8 个）会在相同设置下重新运行；没有任何问题因其答案而被重新运行。主实验之后新增的运行，即压缩后的重新运行和第 5 节中的运行，均通过同一代理，在共享的 vLLM 集群上运行两个小型语言模型；该代理执行的请求转换与我们自己的服务器相同，且使用相同的权重。

C 任务采样

使用 Python 的 random.Random(20261020)，从所有参与比较的系统均已完成且未发生基础设施故障的任务中抽取编程任务：50 个 SWE-bench Verified 任务（在未用于撰写指令的任务中，所有符合条件的任务）、75 个 SWE-bench Pro 任务，以及 50 个 DeepSWE v1.1 任务。对两个小型语言模型进行单独比较时，则使用相同的随机种子，从双方均已完成且未发生基础设施故障的任务中抽取 30 个 DeepSWE 任务。200 个 BrowseComp-Plus 问题是使用种子 20261019 从留出问题中抽取的。抽取的任务列表随论文一并发布。