arXiv:2611.31118v1 [cs.OS] 23 Nov 2026

TideSand：面向大语言模型智能体的高性能保真沙箱多路复用

Shulu Li [*1] **Hanchen Li** [*1] **Xiaokun Chen** [2] **Qiuyang Mang** [1] **Hanfei Chen** [3] **Jacky Kwok** [4] **Hanfei Yu** [1] **Alvin Cheung** [1] **Edward A. Lee** [1] **Joseph E. Gonzalez** [1] **Ion Stoica** [1]

摘要

随着大语言模型（大语言模型）智能体被部署用于软件工程、性能调优和自动化研究等越来越长周期且复杂的任务，对能够为其提供隔离执行环境的沙箱的需求迅速增长。典型沙箱会在每个智能体的整个生命周期内分配固定数量的资源，即使大语言模型智能体正在等待推理，此时沙箱大多处于空闲状态。这会导致资源利用率较低。以往工作通过超额分配资源来利用统计复用，使资源使用量超过主机容量，因为并非所有沙箱都会同时处于活跃状态。然而，当主机资源被多个沙箱复用时，会出现若干挑战。最重要的是，智能体沙箱具有突发性；当多个沙箱的突发负载发生重叠时，资源争用可能导致沙箱获得的资源少于承诺值，从而减慢工具调用速度。这会损害工具调用保真度，我们将其定义为沙箱保持工具调用执行时间和结果不变的能力。对于依赖准确性能测量的新兴智能体工作负载，沙箱系统需要在保持工具调用保真度的同时最大化资源利用率，从而更好地服务于该工作负载。

我们提出了 TideSand，这是一种沙箱系统，可为性能敏感型大语言模型智能体（例如执行系统测量任务的自动化研究智能体）智能地复用资源。TideSand 融合了三项关键理念：代理感知调度，在工具调用层面智能调度沙箱执行，以避免资源争用；推测式换出，将等待推理的沙箱内存卸载，并在其下次工具调用前恢复；以及选择性回滚，在发生资源争用时为选定的沙箱创建检查点，并在机器空闲时恢复这些沙箱。我们在 AlgoTune、Terminal-Bench 4.0 和 Terminal-Bench Science 上，对六组模型-工作负载对进行了 TideSand 评估。与严格预留相比，TideSand 在保持工具调用保真度的同时，吞吐量提高了 3.8–10×；与超额分配基线相比，在吞吐量相近的情况下，保真度提高了 7–44%。此外，TideSand 将 AlgoTune 上的评估误差降低了 19

× 通过提升工具调用保真度。我们希望本研究能激励未来的工作，在保持较高资源利用率的同时，通过统计复用维持工具调用保真度。我们的代码已开源，地址为 https://github.com/TideSand/tidelet。

1 引言

提供大语言模型（LLM）智能体服务的成本不止于推理。智能体还需要执行环境，以便编辑代码、运行测试、分析数据和开展实验（Yang 等人，2024年；Wang 等人，2025年；Chen 等人，2025年）。这些环境，即沙箱，可隔离工具执行，并在连续调用之间保留文件、进程和内存状态（Huang 等人，2026b；Dong 等人，2026年）。与单次工具调用不同，沙箱可以在整个智能体任务期间持续分配，包括等待模型推理的时间段（Huang 等人，2026b；Li 等人，2026b）。近年来，随着相同能力水平下的推理成本不断下降（Maslej 等人，2025年），中央处理器和这些沙箱所需的内存可能会逐渐占据总成本中越来越大的比例。要降低这项成本，就需要根据智能体的工作流调整沙箱管理。

智能体沙箱的工作负载特征独具一格：工具执行通常是突发性的。例如，编译任务可能需要多个 CPU 来运行，随后等待一次大语言模型推理调用来处理其输出。为满足编译阶段的需求，沙箱在其整个生命周期中通常会被分配一组固定资源。然而，这会导致资源利用率偏低，因为智能体等待推理时，资源仍被分配给它，却处于闲置状态。提高利用率的自然方法是通过资源超额订阅来实现统计复用（即，并非所有环境都会同时处于活跃状态）。假设一个简单的例子：有两个智能体沙箱。每个沙箱都每隔一秒需要一个 CPU，

[*]共同贡献。[1]加州大学伯克利分校。[2] HarborFramework。[3]佐治亚理工学院。[4]斯坦福大学。

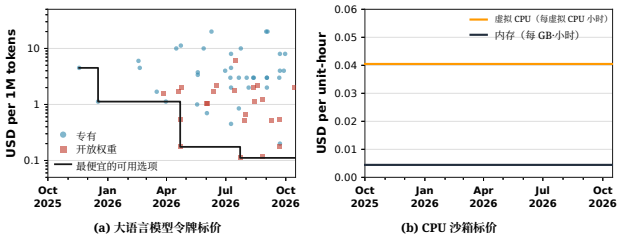


图 *1*。在过去一年中，等智能大语言模型令牌价格大幅下降，而 *CPU* 沙箱成本则相对稳定。

并用闲置的一秒等待远程大语言模型推理。如果每个沙箱分配一个 CPU，利用率就是 50%。相反，我们可以复用一个 CPU 来运行两个沙箱，将资源需求减半。DeepSeek Elastic Compute (DSec) (Huang 等人，2026b) 和 AgentZip (Li 等人，2026b) 等现有系统通过这种复用方式提高了沙箱密度。

然而，资源复用带来了诸多挑战。首先是准入控制。如果没有足够的资源可供沙箱恢复，我们可能需要延迟其执行，直到部分资源被释放，以防止抖动。其次，每个沙箱都具有与其关联的状态。为了让另一个沙箱利用同一资源，我们必须先卸载其状态，并在恢复它之前恢复该状态。然而，卸载和恢复所产生的开销可能会抵消复用的优势，因此我们需要谨慎选择执行这些操作的时机。

最重要的是，在许多场景中，智能体旨在提升系统性能。在这种情况下，准确的资源分配和测量至关重要，因为这是智能体依赖其进行迭代的信号。考虑一项请求十个 CPU核心、但在大部分执行过程中只获得六个核心的性能分析任务。工具调用响应可能反映的是来自其他沙箱的干扰，而非该实现本身的行为。基于此报告采取行动的智能体会将资源可用性的变化误认为程序性能的变化。

为衡量这一效果，我们使用**工具调用保真度**来描述在复用沙箱中执行时，执行时序和结果的保留程度。具体而言，我们在资源有保障的沙箱中运行同一工具调用，再在复用沙箱中运行一次，并比较两者的工具调用时序和结果。工具调用保真度高的沙箱系统，在这两种情况下应能产生相似的结果。

这里有一个独特的机会，可以智能地调度智能体的工具调用。稍微延迟大语言模型响应不会对智能体产生影响，因为这与大语言模型推理速度较慢的情况无法区分。这使调度器能够延迟执行工具调用，直到有足够的资源可用，同时对沙箱完全透明。这样会增加排队延迟，但能够保留工具调用保真度。

这一要求引出了最大化 **沙箱有效吞吐量** 的目标：提高成功完成的智能体任务速率，同时确保其工具调用满足指定的保真度要求。我们通过将工具结果和执行时间分布与参考执行进行比较来评估保真度；参考执行使用经过配置的资源，且不受其他沙箱的资源争用影响。使用分布可以体现即使在不存在资源争用的执行过程中也会出现的变化。因此，有效吞吐量既体现了同时服务更多智能体所带来的收益，也体现了重试、失败和失真观测所产生的代价。

为最大限度地提高有效吞吐量，我们提出了 TideSand：在保护工具调用保真度的同时，超额分配沙箱资源。它结合智能体循环信号与资源观测数据，采用三种机制：

- **代理感知调度** 仅在中央处理器、内存和带宽有足够资源余量时，才准入额外工作，从而减少资源抖动，保留工具调用保真度。
- **推测式换出**会在没有正在执行的工具时，于推理时间间隔内将空闲沙箱的内存移至磁盘。当工具调用即将执行时，它会先从交换空间恢复内存，避免回收内存只是将恢复开销转移到工具执行阶段。
- **选择性回滚**通过在工具调用执行前有选择地为沙箱创建检查点，并在资源饱和时关闭沙箱以释放余量，从而应对资源需求的突然峰值和残余资源争用。资源可用于执行后，这些工具调用将重新执行。

我们使用 Sonnet 5.5 和 DeepSeek V4.1 Flash，在 AlgoTune、Terminal-Bench 4.0 和 Terminal-Bench Science 上的六个模型-工作负载对上评估了 TideSand。TideSand 在保持工具调用保真度的同时，实现了比严格预留高 3.8–10× 的吞吐量；或者，在吞吐量相近的情况下，其保真度比包括超额承诺在内的基线高 7–44%。此外，通过提升工具调用保真度，TideSand 将 AlgoTune 上的评估误差降低了 19×；该误差是相对于严格预留条件下智能体的任务奖励进行测量的。

本文做出了三项贡献：

- 我们将**工具调用保真度**定义为衡量智能体沙箱中资源公平性的指标，并将**有效吞吐量**表述为沙箱系统为智能体提供服务的总体目标。
- 我们设计了 TideSand，这是一种结合代理感知调度、推测式换出和选择性回滚的沙箱系统，能够利用空闲资源，同时保障工具调用保真度。
- 我们展示了 TideSand 的吞吐量比严格预留高 3.8–10× 倍，或在吞吐量相近的情况下，比基线的保真度高 7–44%。

它还通过提升保真度，将任务奖励误差降低了 19×。

2 背景

2.1 新兴的智能体工作负载

大语言模型智能体会执行软件工程（Yang 等人，2024年；Wang 等人，2025年）、科学数据分析（Chen 等人，2025年）和网页自动化（Zhou 等人，2024年）等任务。OpenHands 等平台会在称为沙箱（Wang 等人，2025年）的隔离执行环境中运行工具，以编辑文件、运行测试和操作浏览器。由于智能体任务会持续多个轮次，后续操作会建立在先前操作的基础上，因此有状态的沙箱会在整个执行过程中保留文件系统、进程和内存状态（Huang 等人，2026b；Dong 等人，2026年）。

近来，人们开始探索利用智能体开展自主研究（Mang 等人，2025年；2026年；Press 等人，2025年）。在这类工作流中，通常会为智能体提供一个研究环境和一个最终目标。智能体会进行性能分析、分析和调优，以针对更实际且开放式的任务提出创新解决方案。值得注意的是，由于这些任务面向数据库等实际系统，因此需要的资源要多得多。

2.2 沙箱成本增加

为了服务这些智能体，系统需要同时考虑大语言模型推理成本和沙箱成本。而沙箱成本正出人意料地变得越来越重要。图 1比较了在 Terminal-Bench 2.1（Terminal-Bench 团队，2026b）中得分至少达到 50% 的模型，在 2026年10月列出的令牌价格，并按发布日期分组。绘制出的最低价格从 2025 年 11 月发布批次的 4.50 美元降至 2026 年七月发布版本的每百万令牌 0.111 美元，差距大约为 40×（假设输入/输出比例为 3:1，与表2中的设定相同）。作为参考，AWS Fargate 在美国东部（弗吉尼亚北部）的 Linux/x86 按需费率在右侧保持稳定。

此外，沙箱已经占据了智能体服务成本中相当大的份额。图 2 显示，当我们以最大工作量运行 GPT-6 Sol 、 GPT-6.1 Sol 和 GPT-6 Luna 来执行 Terminal-Bench 4.0 时，即使排除需要 GPU 的任务，中央处理器和内存成本仍分别占总成本的 16.4%、33.5% 和 74.2%。这一惊人的对比，尤其是即使对于相同的模型类型，模型成本也在不断降低，表明沙箱很快可能成为智能体服务中最大的瓶颈。

2.3 智能体沙箱的相关工作

系统机制：Firecracker（Agache 等人，2020年）和 gVisor（gVisor 作者，2026年）分别通过轻量级微型虚拟机和用户空间应用内核隔离执行，

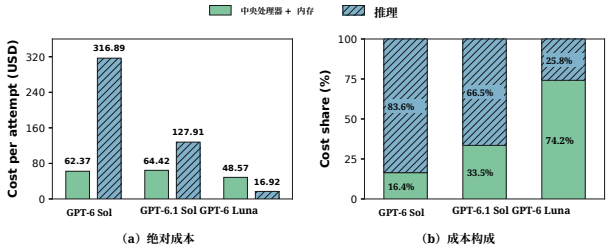


图 *2. Terminal-Bench 4.0* 上各模型在最大工作量下的端到端智能体成本细分，不包括 *GPU* 任务

表*1.* 沙箱机制与目标： ✓ 表示明确关注；–表示其他情况。

系统	超额订阅 scription	内存回收	推理感知 a ware	保真度-保持
DSec	✓	✓	–	–
SpecBox	–	–	✓	–
AgentZip	✓	✓	✓	–
AgentCgroup	✓	–	✓	–
TideSand	✓	✓	✓	✓

分别而言，TrEnv-X（Huang 等人，2026a）通过由 CXL/RDMA 内存池支持的可复用沙箱和内存模板，降低启动和内存成本。DeltaBox（Dong 等人，2026年）通过增量状态捕获和冻结进程模板加速检查点与回滚，并将检查点工作与推理重叠。这些机制为高效的沙箱调度提供了构建模块。

工作负载感知优化：近期系统日益针对智能体工作负载定制这些机制，通常通过超额订阅来共享工具调用间隙中闲置的资源。DeepSeek Elastic Compute（DSec）（Huang 等人，2026b）结合了超额订阅、内存回收和强化学习协调的沙箱生命周期，同时利用中央处理器优先级和核心调度保护延迟敏感工作。SpecBox（Zhang 等人，2026）根据流式模型输出和工具转换预测沙箱请求，以便在推理或执行进行期间并行完成准备工作。AgentZip（Li 等人，2026b）利用模板和跨沙箱冗余压缩内存，在推理期间安排压缩，并预取页面以减少恢复停顿。AgentCgroup（Zheng 等人，2026）通过对相互竞争的工具调用进行限流来处理超额订阅中的资源争用，但代价是工具调用速度会被拖慢过久。

DSec、SpecBox、AgentZip 和 AgentCgroup 针对智能体工作负载调整了资源共享和超额订阅，而以往的工作为沙箱调度提供了有价值的系统原语。TideSand 在这些系统原语的基础上，设计了新的工作负载感知调度策略，以更好地支持新兴的性能敏感型智能体工作负载。

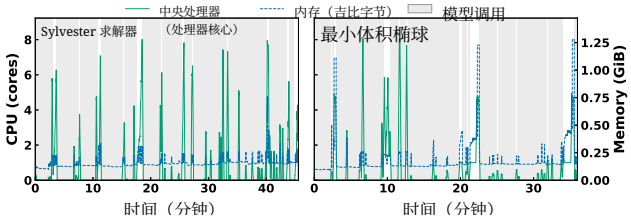


图*3*。两个任务随时间变化的 *CPU* 使用量（处理器核心）和内存用量（吉比字节）。阴影区间表示模型调用。*CPU* 使用率呈突发性，而内存在各执行突发之间保持驻留。

3 面向性能敏感型智能体的沙箱

尽管以往在沙箱中进行超额分配的工作提高了资源利用率，但仅靠多路复用不足以提供性能准确的沙箱。

3.1 工作负载特征与多路复用

智能体沙箱对 CPU 和内存的需求具有突发性。工具调用可能需要多个 CPU 核心，而推理期间这些核心大多处于空闲状态，内存则保持驻留（图 3）。若始终按照峰值需求为每个沙箱配置资源，会浪费本可供其他智能体使用的容量。当不同智能体的工具执行错开时，主机可以进行超额承诺：容纳已配置资源总量超过其物理容量的沙箱。

例如，DSec（Huang 等人，2026b）会在节点仍有已测量的中央处理器和内存容量时准入一个沙箱，从而进行超额承诺，使各沙箱配置的限制总和超过主机的物理容量。

3.2 多路复用下的资源争用

多路复用在预测未来需求方面面临挑战。多个已准入的智能体可能同时完成推理并启动资源密集型工具，即使主机平均利用率适中，也会造成资源争用。例如，DSec 会在创建沙箱前检查本地资源容量，并结合 CPU 优先级与核心调度来限制干扰（Huang 等人，2026b）。这些控制以沙箱级粒度运行，无法确保资源能够满足此类突发需求。

另一方面，智能体循环为推测性准备提供了机会。在工具调用流式生成的过程中，调度器可以预判即将到来的执行，并在模型生成剩余参数时开始恢复已回收的内存。将恢复操作移到这一时间段内，可以降低工具调用的开销。反之，若保留正在推理的智能体所用空闲沙箱中的非活动页面，可能会迫使活动工具将数据换出到交换空间，如图4所示。

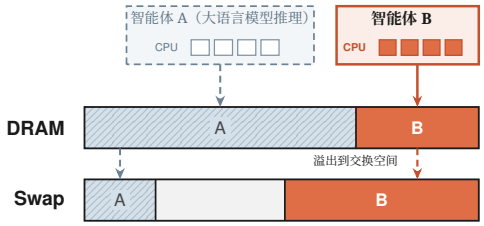
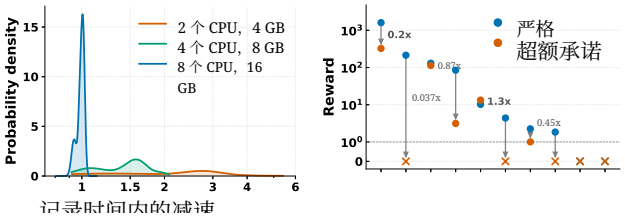


图*4*. 在朴素多路复用下，尚未进行工具调用的智能体 *A* 正在使用活动智能体*B*的沙箱资源。



（a）工具调用减速。　　（b）最终基准奖励。

图*5*. 资源争用会导致工具执行时间偏离参考执行，并影响最终基准奖励。

3.3 工具调用保真度

最重要的是，多路复用造成的资源争用，可能会对新兴的性能敏感型沙箱（例如自动化研究智能体）产生更严重的影响。自动研究智能体会进行实验，并利用观测结果指导后续调查（Lu 等人，2024年）。例如，AlgoTune 的智能体会分析候选程序，并根据测量得到的运行时间选择更快且正确的实现（Press 等人，2025年）。如果沙箱获得的 CPU 核心数或常驻内存少于实验所假定的资源量，资源争用就可能扭曲观测结果。为了确保工具调用的结果和执行时间与在其配置的 CPU 和内存上进行的参考运行保持一致，我们提出了沙箱执行的一项属性，并将其称为工具调用保真度。换言之，即使调用返回了正确的输出，只要 CPU 共享或内存压力改变了其运行时间，保真度仍可能下降。

我们在图5中展示了资源争用对工具执行时间和最终基准奖励的影响。左图比较了三种 CPU 和内存分配下 10 个工具调用各重复 10 次的执行时间分布，并将其归一化为使用 8 个 CPU 和 16 GB 内存时记录的时间。降低资源分配会同时增加执行时间及其波动。右图比较了严格预留策略和基于资源使用情况的朴素超额分配策略下的最终基准奖励。奖励可能朝两个方向变化，这表明即使执行相同任务，资源争用也会改变智能体的结果。超额分配下获得更高奖励并不意味着更高的保真度：无论朝哪个方向变化，该结果都不同于参考执行。

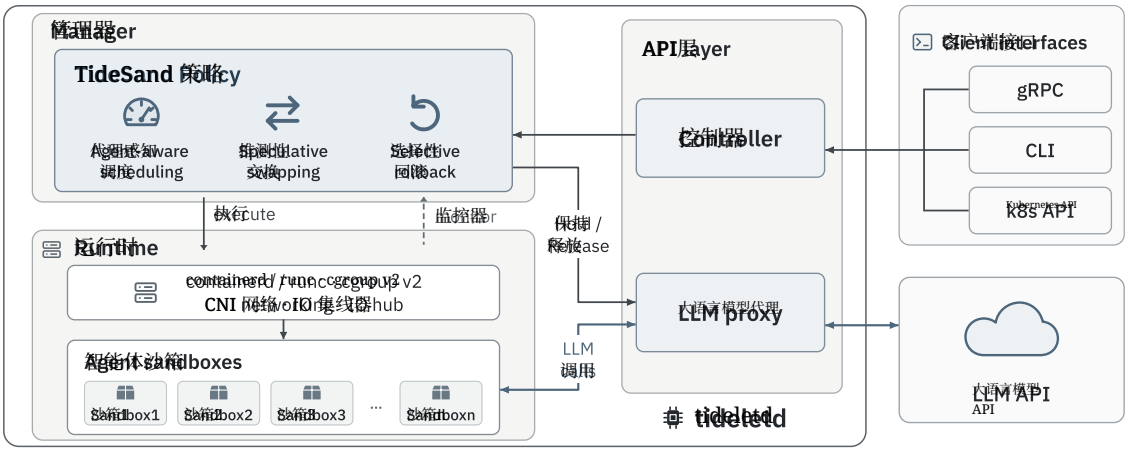


图 *6. TideSand* 的系统概览。

鉴于资源争用对实验结果的影响，对于自主研究智能体的沙箱系统而言，工具调用保真度是首要考虑因素。因此，与其立即运行资源不足的工具调用并扭曲其结果，不如延迟该工具调用，因为对于大语言模型推理而言，排队延迟与大语言模型推理延迟无法区分。另一方面，执行期间由资源争用造成的减速会成为测量结果的一部分，并可能误导智能体接下来的步骤。

3.4 改进沙箱有效吞吐量

因此，统计复用的机会伴随着一项要求：这些工具调用运行时必须保持执行条件。只有在不损害工具调用保真度的情况下，提高沙箱并发度才有意义。我们将这一目标定义为 沙箱有效吞吐量：工具调用满足指定保真度要求的已成功完成智能体任务的速率。提高有效吞吐量需要在推理期间回收资源、在下一次工具调用前恢复资源，并在需求超过主机容量时控制并发执行。

4 系统设计

4.1 系统概览

TideSand 是一个沙箱控制平面，可在多个智能体沙箱之间进行主机资源的统计复用。这种共享提高了利用率，因为沙箱在工具执行期间的需求具有突发性，而在智能体等待大语言模型响应时，资源需求通常会降至所请求资源的一小部分。TideSand 的关键洞见是调度单个工具调用，而不仅仅是在沙箱准入时进行调度。

图 6 展示了我们系统的架构。TideSand 作为一项策略运行于 `tideletd`中，这是一个由四个组件构成的主机本地守护进程。控制器 提供通过 gRPC、命令行界面和 Kuber- 管理沙箱的接口 netes。该系统对沙箱透明运行。唯一需要进行的配置更改是将其大语言模型端点指向由 tideletd 管理的大语言模型代理 ，这样 TideSand 无需修改代理框架，即可观察并调度工具调用。管理器 承载调度策略，负责做出调度决策并控制运行时。我们的运行时 使用通过控制组管理的容器，但也可以替换为其他沙箱技术，例如微型虚拟机，前提是该技术支持管理器所需的操作。

管理器由三个组件组成。资源监视器 与运行时实现通信，并报告每个沙箱的 CPU 使用率、内存流量和工具调用活动。调度器 决定何时准入沙箱、执行工具调用、交换内存以及创建沙箱检查点。它将这些决策传递给动作执行器，后者向大语言模型代理和运行时发出相应命令。

4.2 代理感知调度

大语言模型代理： TideSand 观察智能体的推理—执行循环，并通过大语言模型代理控制工具调用的传递。智能体将模型 API 请求发送给代理，代理在控制响应时机的同时保留请求和响应内容；除了配置端点外，无需更改代理框架。模型请求表示进入推理阶段，而包含工具的响应则标识下一个执行区间。在传递该响应之前，代理会将工具名称报告给调度器。仅包含已识别轻量级工具（例如文件读取和编辑）的响应无需经过中央处理器准入；包含外部执行工具（例如 Shell 命令）的响应则需等待调度器准入。对于流式响应，代理会转发第一个工具调用事件之前的文本，并缓冲该事件及响应的剩余部分，从而防止智能体在准入前启动工具，同时允许模型继续生成

内容。不包含工具调用的响应不会因资源准入而被暂缓。

统计资源预测：工具调用的资源需求在执行前是未知的。高保真度执行要求主机上有充足的资源可用，但如果需求较低，仍为每次调用预留沙箱完整的配置资源配额，就会浪费容量。

为兼顾保真度和利用率，TideSand 会在执行前预测每次调用的资源使用情况，并预留预测的资源量。预测依据沙箱近期的调用记录进行学习。设 $u_i(t)$ 为测得的沙箱 i 的中央处理器使用量，L_i 为其配置的中央处理器限额。在每次重负载调用 k 期间，调度器会跟踪 $u_i(t)$ 的指数平均值，时间常数为 10 秒，并在调用结束时记录其峰值 $p_{i,\,k}$。下一次调用的预测需求 d_i 是最近三次调用中的最大峰值；没有历史记录的沙箱使用主机上观测到的峰值中位数，若尚无调用完成，则使用 L_i。预留量为

$$r_i^c = \begin{cases} L_i & \text{if } \hat{d}_i \geq L_i/2, \\ \max(\hat{d}_i, g_i) & \text{otherwise}, \end{cases} \tag{1}$$

其中 g_i 是沙箱在调用间隔期间的后台使用量。预测使用量达到限额一半或以上的调用会预留整个限额，以保留参考运行中供中央处理器密集型任务使用的并行度。内存流量也以相同方式根据每个调用的持续流量 b_i进行预测，预留量 $r_i^b = \min(\hat{b}_i,\ BL_i/C)$受限于沙箱根据其中央处理器限额所占的主机带宽份额 B。预计新沙箱将使用 $\hat{m}$ 字节内存，即现有沙箱常驻内存的中位数。

细粒度准入：TideSand 以两种粒度准入工作：当工具调用的预留量符合主机余量时，即释放该调用；当沙箱的预期内存用量符合要求时，即启动该沙箱。设 C 为主机的处理器核心数，B 为其可持续内存带宽，U_c 和 U_b 分别为所有沙箱测得的中央处理器使用量和内存流量。测量结果可能无法及时反映新释放的调用。因此，调度器会为最近 10 秒内释放的调用添加一个待处理项，记为 $\mathcal{R}$，用于计入尚未反映在测量使用量中的预留量部分：

$$P_c = \sum_{j \in \mathcal{R}} \max(0,\, r_j^c - u_j), \tag{2}$$

对流量也同样如此。当沙箱i 的高负载响应 P_b 时，

$$\begin{aligned} U_c + P_c + r_i^c &\leq \alpha C, \\ U_b + P_b + r_i^b &\leq \beta B. \end{aligned} \tag{3}$$

预算阈值为 $\alpha = 0.7$ 和 $\beta = 0.8$。如果没有待处理项，在任何响应尚未.....之前，一批响应就会根据相同的表面余量被释放出现在样本中。无法容纳的响应会按到达时间顺序排队。调度器优先考虑最早到达的响应，但如果较早的响应无法容纳，而较年轻的响应可以容纳，则允许后者先行，从而避免因大型预留而造成不必要的空闲。若新沙箱的启动配额满足公式3，且其预期内存符合准入目标，

$$M + \hat{m} + M_{\text{swapped}} \leq \mu M_{\max}, \tag{4}$$

其中，M 为正在使用的内存，M_{swapped}为等待响应的已换出沙箱将取回的内存。

4.3 推测式换出

内存换页：内存不像中央处理器容量那样容易复用，因为它保存着持久的沙箱状态。若要复用这部分容量，就必须先卸载状态，并在沙箱再次访问它之前将其恢复。将状态卸载到磁盘会占用带宽，而活动的工具调用也可能需要带宽。对于中央处理器密集型工作负载，这种开销可能并无必要，因为瓶颈在于中央处理器容量而非内存。因此，TideSand 仅在内存压力下才会主动启动换出。准入会在内存容量达到 80% 时停止（公式 4），但已准入的沙箱仍会继续增长。当它们的增长使正在使用的内存超过 90% 时，调度器会选取正在等待模型响应的沙箱，并优先选择常驻工作集最大的沙箱。对于每个选中的沙箱，调度器都会要求内核回收该沙箱 80% 的常驻内存并将其换出，同时不停止其中的进程。

推测性预取：已换出的沙箱必须在下一次工具调用前恢复其工作集，以避免执行期间发生交换空间换入导致的页面错误。大语言模型代理会在时机到来时通知调度器：下一条包含工具调用的模型响应表明智能体即将恢复执行。当该响应到达代理时，调度器会保持该响应，并为沙箱发起预取。预取会请求内核将沙箱已换出的页面读回内存，随后即可继续处理该响应；但较大的响应仍需遵循工具调用准入机制，而内存换入现在可与等待准入的过程重叠进行。

4.4 选择性回滚

资源预测基于统计：它预留的是沙箱近期调用所显示的资源需求，而不是其下一次调用可能产生的需求。一次工具调用所使用的资源可能超过预测的资源分配，而且多个获准执行的调用也可能同时突发，超出各自的预测值。如果突发需求超过余量，主机就会达到饱和，从而降低工具调用保真度。降低此类峰值出现概率的简单方法是在容量以下留出较大的余量，但这样会降低资源利用率。TideSand 则选择尽可能接近容量运行，并将饱和视为需要恢复的罕见事件，而不是必须避免的状态。

选择性回滚会将选定的调用从执行中移除，使其余调用保有充足资源；待峰值过去后，再重试已停止的调用。

沙箱检查点： 只有在能够将沙箱恢复到调用开始前的状态时，才能撤销该调用。TideSand 在智能体发送模型请求的时刻创建整个沙箱的检查点，该请求先于工具调用，并捕获其中进程和文件系统的状态。在请求而非响应时创建检查点，可让检查点过程与模型推理重叠，从而减少智能体感知到的延迟。回滚时，系统会停止沙箱、恢复检查点，并让智能体从发送请求处继续执行；智能体会重新发送请求，响应中携带相同的工具调用，此时调度器会像准入其他调用一样准入它。然而，创建检查点会占用本可用于运行其他任务的宝贵资源。因此，TideSand 只为选定的调用创建检查点。

选择检查点与恢复： 只有当调用可能导致峰值，且估算的检查点开销相对于其持续时间较低时，才会为其创建检查点：该调用必须预留至少一半的中央处理器限额，主机负载必须足够高，以至于峰值会造成影响；并且根据先前检查点估算的预期检查点时间，必须只占沙箱典型调用持续时间的一小部分。当测得的中央处理器使用量或内存流量连续三秒超过主机容量的 85% 时，就会触发回滚。随后，调度器会停止那些测得的使用量超出其预留量最多的已创建检查点调用，直至预计移除这些调用后，需求降至阈值以下。一旦预留量允许，停止的调用就会恢复并排在队列首位。在 §6中的各项工作负载中，创建检查点的调用不到 3%，回滚的调用不到 0.5%，中央处理器开销低于 2%（图 14）。

5 实现

沙箱守护进程： TideSand 是 tideletd 的一种策略，由一个约 2.1 万行 Go 代码编写、运行于主机本地的沙箱守护进程实现。该守护进程负责管理沙箱表，在 cgroup v2 下通过 containerd 和 runc 运行容器，提供 gRPC API 和命令行客户端；它还可通过 virtual-kubelet 提供程序将主机注册为 Kubernetes 节点，从而使 Pod 映射到一个沙箱。每个沙箱创建时都会设置其任务指定的 CPU 和内存限制；这些限制约束沙箱，但不会在主机上预留资源。守护进程每隔 250 毫秒对每个沙箱采样一次，记录其累计 CPU 时间、常驻工作集、交换空间和压力停滞信息，并通过硬件末级缓存未命中计数器读取主机的内存流量。策略通过一个简洁的事件接口实现：它接收资源采样数据、大语言模型代理发来的模型请求和模型响应事件，以及沙箱生命周期事件，并返回诸如释放或保持响应、准入或排队沙箱、回收或预取内存、创建检查点、回滚以及恢复等操作。严格预留、阈值超额分配和 TideSand 都是该接口的策略；TideSand 策略约有 1,500 行代码，并通过守护进程标志选择启用。

大语言模型代理： 守护进程为模型 API 运行一个透传代理，智能体通过其基础 URL 设置访问该代理，因此 Claude Code 无需修改即可在沙箱中运行。代理转发流式令牌，直到响应中出现工具调用，然后缓冲余下的内容；随后，它将工具名称告知策略，并在策略释放缓冲内容时将其交付。代理还会记录每次请求和响应，使调度器无需对智能体进行插桩即可获取每个沙箱的执行和推理时间间隔。

内存和检查点： 推测式换出通过内核的回收和 process_madvise 接口对沙箱中的进程进行操作，因此换出和预取都在智能体之外进行。选择性回滚使用整个容器的检查点：守护进程冻结沙箱的控制组，使用 CRIU 转储其进程，并为其文件系统的可写层创建快照；回滚时恢复进程和可写层，智能体重新发出创建检查点之前的模型请求。检查点镜像存放在内存后备文件系统中，并在调用完成后丢弃。

重放框架： 为了在完全相同的智能体行为下比较策略，我们构建了 TideBox，包含约 6K 行 Python 代码。它会记录一次智能体运行过程，捕获每次模型请求和响应及其时间信息，以及沙箱遥测数据；随后在全新的沙箱中重放该记录：每个沙箱的控制器通过代理从记录带中响应智能体的模型请求，而工具调用则在沙箱中实际执行。重放驱动程序控制沙箱到达时间，按可配置的次数重复每条记录，并将每次工具调用的输出和执行时间与记录及参考重放进行比较。

6 评估

6.1 实验设置

模型与智能体： 我们使用两个模型：Claude Sonnet 5.5 和 DeepSeek V4.1 Flash。框架为 Claude Code（版本 2.1.300–2.1.309），其未经修改地运行在每个沙箱中，模型则通过 TideSand 代理访问。每项任务均使用该基准测试的智能体默认设置运行（AlgoTune 和 Terminal- Bench 科学任务使用 最大强度，Terminal-Bench 4.0 在 Sonnet 上使用 中等强度），并且每项实验都会按照第 §5节所述重放这些记录，以便在完全相同的智能体行为下进行比较。

硬件： 每项实验均在 AWS c7g.metal

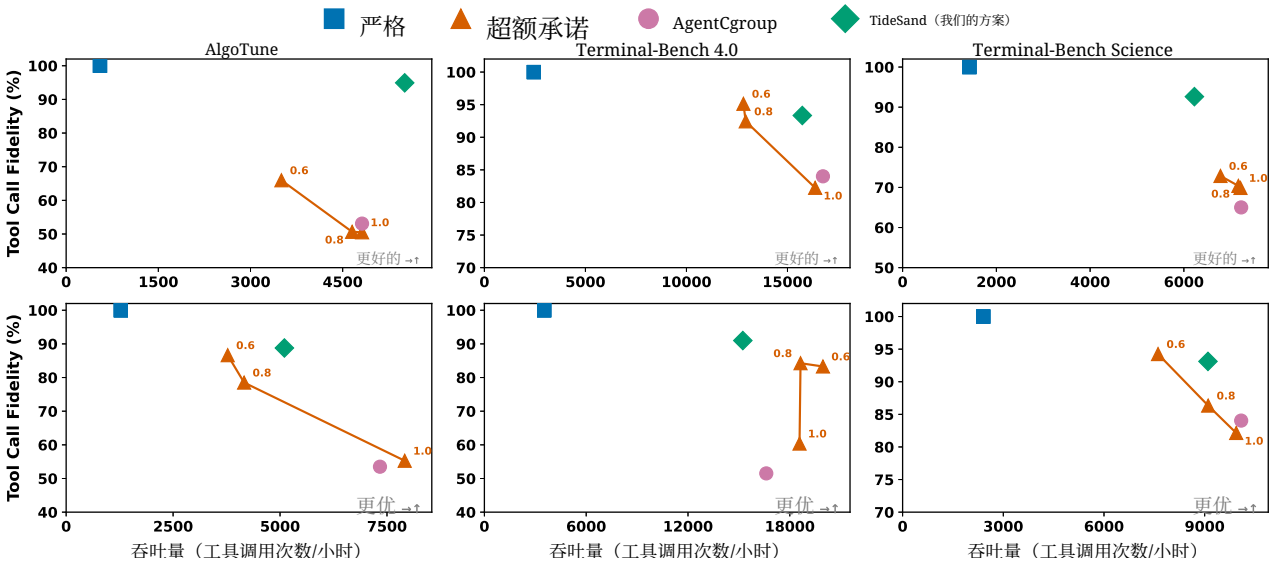


图 7. 在六组模型-工作负载对上，*TideSand* 在吞吐量与保真度之间实现了更优的权衡。吞吐量和保真度越高越好。

裸金属实例上运行：配备 64 个 Arm Neoverse-V1 处理器核心（Graviton3）、128 GB 动态随机存取存储器、gp3 存储，以及搭载 Linux 6.17 的 Ubuntu。沙箱运行于一个包含 56 个处理器核心和 112 吉比字节的控制组中；其余 8 个处理器核心用于运行重放驱动程序和守护进程。一个 64 GB 的交换文件为基线的推测式换出和内存超额承诺提供支持。

工作负载： 我们使用三个配备资源密集型工具的智能体基准测试。

- AlgoTune（Press 等人，2025年）：智能体在保持正确性的同时优化算法的运行时，并在沙箱中对候选实现进行性能分析；共 50 个任务，每个任务配备 8 个 CPU 和 16 GB 内存。

- Terminal-Bench 4.0（Merrill 等人，2026年；Terminal-Bench 团队，2026c）：由任务专用测试验证的终端任务；共 50 个录制任务，每个任务配备 2–16 个 CPU 和 4–16 GB 内存。

- Terminal-Bench Science（Terminal-Bench-Science 团队，2026年）：源自已发表研究的数据分析和仿真任务；从 62 个录制任务中随机抽取 50 个，每个任务配备 1–4 个 CPU 和 2–16 GB 内存。

每个任务的智能体运行时间上限为两小时。为了给主机施加足够的负载，驱动程序以固定到达率提供录制任务，并重复执行每条记录：AlgoTune 为 $\times 2$（100 个沙箱，每 15 秒一个），Terminal-Bench 为$\times 8$（400 个沙箱，每 3 秒一个），Terminal-Bench Science 为 $\times 2$（100 个沙箱，每 15 秒一个）。

基线： 我们将 TideSand 与在同一守护进程中实现的三种策略进行比较：

- 严格预留：只有当沙箱配置的中央处理器和内存需求均不超过物理容量时，才允许其准入。
- 超额承诺-θ：当主机的实测中央处理器和内存使用量均低于容量的一小部分 θ，且 $\theta \in \{0.6, 0.8, 1.0\}$；一旦准入，沙箱便无需进一步控制即可运行。这类似于 DeepSeek 的 DSec 所采用的基于阈值的超额承诺（Huang 等人，2026b）。
- AgentCgroup（Zheng 等人，2026年）：AgentCgroup 的工具调用资源控制器，移植为其 cgroup-v2 控制循环（附录 B）。每次工具调用根据提示进行分类，并分配内存配额和 CPU 权重；在内存压力下，优先保护高类别沙箱，并对其余沙箱进行节流。

指标： 我们根据以下指标进行评估：

- 吞吐量：每主机小时完成的工具调用数。我们测量至运行中 80% 的工具调用完成为止，以避免未完成任务对延迟产生主导影响。
- 工具调用保真度：延迟膨胀低于其参考时间 10% 的工具调用所占比例。参考时间是在严格预留的主机上，对同一记录进行五次严格回放所得时间中的最慢值。我们仅测量参考时间至少为一秒的调用的保真度。
- 重放奖励误差：在新沙箱系统中运行的任务验证器得分，相对于同一记录的严格回放所得得分的百分比变化。

6.2 端到端结果

吞吐量与保真度： 图 7 展示了四种策略在六个模型-工作负载对上的吞吐量和保真度。严格策略保持了保真度，但吞吐量显著较低。另一方面，TideSand 在保持与严格策略相近保真度的同时实现了更高的吞吐量，与基线相比取得了更好的权衡。AgentC 组在每个

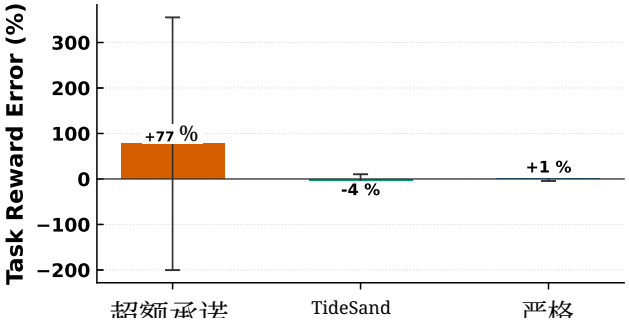


图8. 与在实际评估中采用超额承诺策略相比，*TideSand* 降低了 *AlgoTune* 上的实际奖励误差。

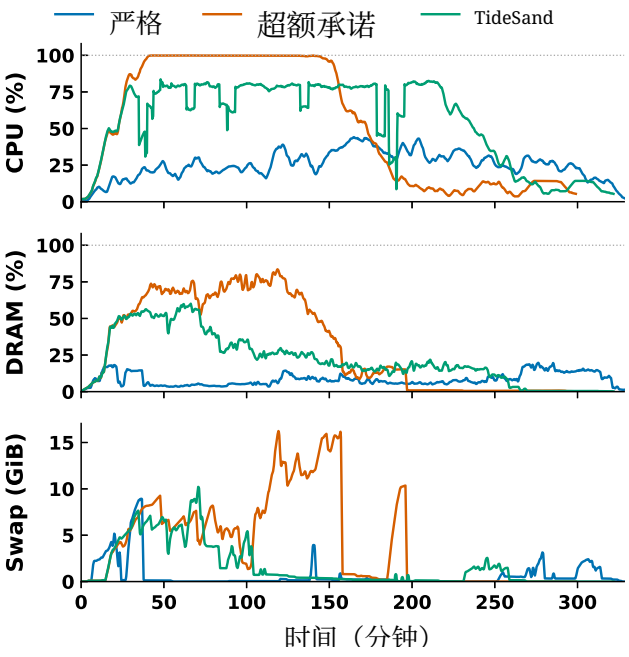


图9. 在严格预留、*Overcommit-0.8* 和 *TideSand* 策略下，使用 *Claude Sonnet 5.5* 运行 *Terminal-Bench Science* 时主机的中央处理器、动态随机存取存储器和交换空间使用量随时间的变化。

工作负载上的表现都接近超额承诺-1.0：其吞吐量与超额承诺-1.0 相差不超过 11%，保真度介于 52% 至 84% 之间，比 TideSand 低 9–42%。值得注意的是，尽管权衡曲线较差，超额承诺有时仍能取得比 TideSand 更高的吞吐量。我们没有将 TideSand 扩展到那个权衡点，因为在实际应用中，较低的保真度（例如低于 80%）可能导致智能体运行失败。

工作负载突发性： TideSand 在使用 Sonnet 运行 AlgoTune 时的优势最大，图 10 解释了原因：其沙箱的突发性最强（峰值 25× 平均中央处理器使用量，而其他工作负载为 4–16×），且分散程度最高。分散程度是指沙箱中央处理器使用量相对于其平均值的方差，用于衡量每个沙箱的中央处理器需求波动幅度。基线在准入沙箱时不会预测突发情况，因此当多个沙箱同时出现突发负载时，它们会使

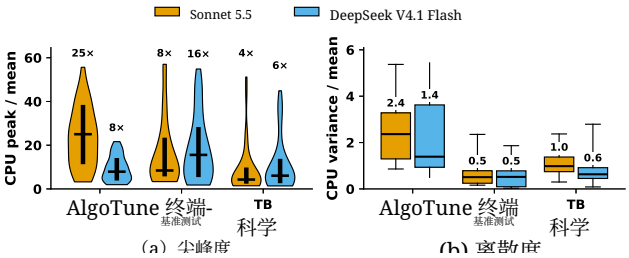


图10. 严格预留下各工作负载的中央处理器突发性：（*a*）每个沙箱的中央处理器峰均比；（*b*）每个沙箱的中央处理器方差均值比，单位为处理器核心。*AlgoTune Sonnet* 的尖峰度和离散度最高。

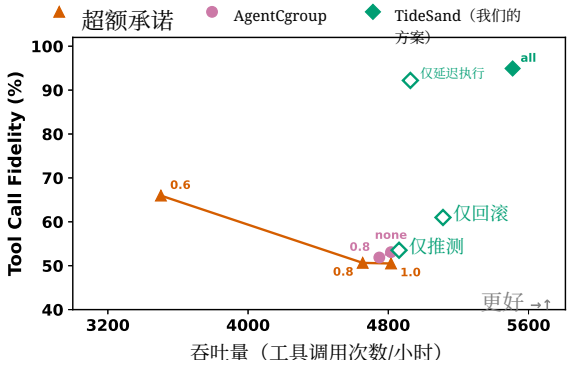


图 11。*TideSand* 的消融实验（*AlgoTune*，×2）。我们将 *TideSand* 的各个机制分别与完整策略进行比较。

主机饱和。TideSand 仅在工具调用符合测得的余量时才予以准入，因此工作负载越分散，相较于基线获得的收益就越大。

任务得分误差： 我们在图 8 中评估 TideSand 保真度提升的显著性。对于每项任务，我们在 Sonnet 下重放 AlgoTune 的同一条记录轨迹，并使用基准测试的验证器，在不同沙箱策略下对完全相同的最终解决方案重新评分。与严格回放相比，超额承诺在 32 项评分任务中平均改变奖励 $+77\%$，标准差为 278%；而 TideSand 使其改变了 $-4\% \pm 14\%$，显著降低了误差。请注意，第二次严格运行与第一次的差异为 $+1\% \pm 5\%$，这与 TideSand 的误差几乎无法区分。

资源利用率： 图9 展示了在 Terminal-Bench Science 的 Sonnet 运行期间，主机的 CPU、DRAM 和交换空间使用情况如何随时间变化。严格预留使主机在六小时内的平均 CPU 利用率保持在 23%，且 DRAM 使用量从未超过五分之一。Overcommit-0.8 在第一个小时内准入该工作负载，并在接下来的一小时内使主机 CPU 利用率保持在 100%，同时常驻内存和交换空间用量不断上升。这损害了这一时段内运行任务的保真度。TideSand 运行相同的工作负载时，CPU 利用率低于 80%，DRAM 利用率低于 60%，从而保持了工具调用保真度。

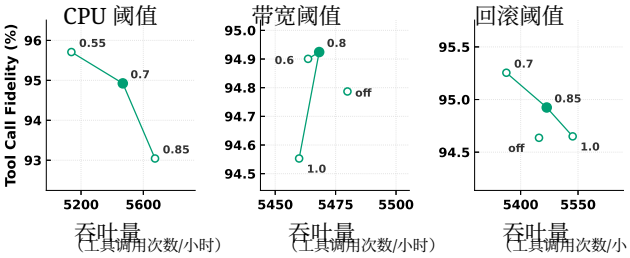


图 *12*：在 *Sonnet 5.5* 上运行 *AlgoTune* 数据集时，*TideSand* 在不同超参数设置下依然稳健。

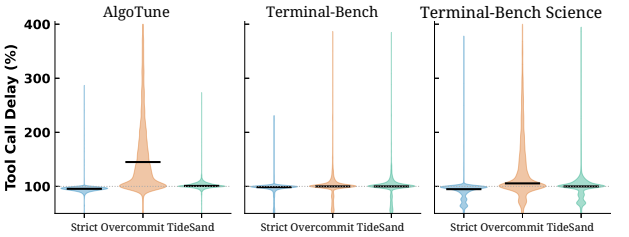


图 *13*：在不同工作负载下，*TideSand* 的工具调用分布与严格模式相似。

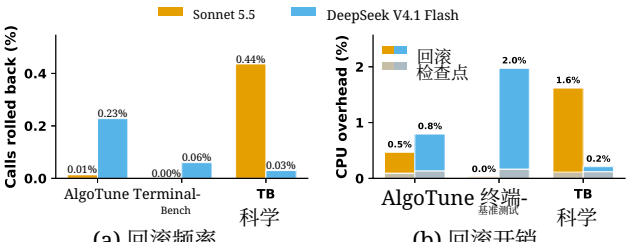


图*14*。*TideSand* 回滚的工具调用很少，在各项工作负载中的中央处理器开销均低于 *2%*。

6.3 扩展评估

工具调用分布： 图 13 展示了三种 Sonnet 工作负载中，每次工具调用的执行时间相对于其参考执行的情况。严格模式下，执行时间紧密集中在参考执行值附近；而超额承诺会形成长尾，许多调用显著变慢。TideSand 的表现接近严格模式，并消除了大部分长尾。Terminal-Bench 的差距最小，因为其调用时间较短，且很少使主机达到饱和。

机制消融： 图 11展示了 TideSand 在 AlgoTune 和 Sonnet 上逐一启用单个机制时的表现。每种机制单独使用时的表现都高于过度承诺曲线；只有延迟执行的表现接近完整策略，但吞吐量有所牺牲。推测式换出和回滚的表现略高于过度承诺曲线，因为它们能腾出更多空间以减少争用。不过，没有任何单一机制能像 TideSand 那样实现最佳权衡。

参数敏感性： 图 12使用 Sonnet 在 AlgoTune 上，以默认配置为基准，每次只改变一个参数。在所有这些设置下，保真度均保持在 93% 到 96% 之间，且没有任何设置导致任一指标大幅下降。改变这些参数只会使 TideSand 沿着吞吐量—保真度权衡曲线略有移动，但不会削弱该设计相较于基线的优势。

微基准测试： 图 14衡量了在六次端到端运行中，选择性回滚的触发频率及其开销。只有 0.6–2.9% 的调用会创建检查点，回滚也很少见：在使用 Sonnet 的 Terminal-Bench Science 上，回滚最多涉及 0.44% 的调用。将检查点冻结和丢弃执行结果的开销都计算在内，每项工作负载的中央处理器开销均低于 2%。这表明 TideSand 很少需要回滚，且代价极小。

7 相关工作

大语言模型推理：大语言模型服务系统通过内存管理和注意力内核（Kwon 等，2023年；Zheng 等，2024年；Ye 等，2025年）、预填充-解码解耦（Zhong 等，2024年；Qin 等，2026年），以及 KV 缓存压缩、复用和卸载（Liu 等，2024b；Yao 等，2026年；Gao 等，2024年；Liu 等，2025年）来提升推理性能。对于智能体工作负载，Agentix（Luo 等，2026年）利用程序级历史记录调度调用，Continuum（Li 等，2026a）在工具调用期间保留具有有界生存时间的 KV 缓存，而 MORI（Xia 等，2026年）则利用相对空闲度在 GPU 和 CPU 内存之间放置缓存。ThunderAgent（Kang 等，2026年）协调程序感知的推理调度与工具资源管理。TideSand 解决了为沙箱共享 CPU 和内存这一互补性挑战。

无服务器容器与沙箱： 无服务器系统通过缓存、预取和回收降低环境启动开销与内存成本。Catalyzer（Du 等人，2020年）复用初始化状态以加快沙箱启动；MITOSIS（Wei 等人，2023年）支持基于 RDMA 的远程派生，无需在每台主机上配置缓存实例。RainbowCake（Yu 等人，2024年）共享容器层；REAP（Ustiugov 等人，2021年）预取快照工作集；FaaSMem（Xu 等人，2024年）和 Squeezy（Lagkas Nikolos 等人，2026年）则通过卸载和回收提升内存效率。Jiagu（Liu 等人，2024a）结合性能感知调度与分阶段扩缩容，在维持服务质量的同时提升利用率。TideSand 通过高效共享来利用智能体特定的优化，并控制资源争用，以保持保真度并保留自主研究智能体的测量结果。

商业沙箱： 我们介绍两家领先商业沙箱提供商的超额订阅策略：E2B 通过基于 userfaultfd 的惰性内存加载支持内存超额订阅，仅在访问沙箱页面时分配物理内存（E2B，2026b；a）。Modal 通过将有保障的 CPU 预留与最大使用限制分离来支持 CPU 超额订阅，使沙箱能够在有

可用空闲资源时突破其预留容量（Modal，2026b；a）。然而，这两种策略的设计目标都是改善执行时间，而非保真度。我们的重点是在共享空闲资源的同时，为自动化研究测量保留这一标准。

8 未来工作

扩展至更多沙箱类型： 将 TideSand 扩展到 GPU 或 TPU 沙箱，需要在 CPU 内存之外跟踪并回收加速器状态。工具可能会加载模型权重、分配设备缓冲区或启动内核，因此恢复沙箱时必须保持主机与设备状态之间的一致性，同时权衡传输成本与预期空闲时间。尽管 NVIDIA 的 CUDA 检查点（英伟达，2026年）等工具支持在保存和恢复 CPU 检查点的同时保存和恢复 GPU 状态，但任意加速器工作负载以及其他后端（例如 TPU）仍需评估特定后端的运行时和驱动能力。

与推理系统协同设计： 沙箱与推理系统可以双向协调：推理进度可以指导沙箱恢复，而工具执行信号则可以为 KV 缓存保留提供信息。例如，Continuum 使用感知工具调用的生存时间策略，使智能体的 KV 缓存能够在短时工具调用期间保留在 GPU 内存中（Li 等人，2026a）。联合调度器可以利用实时信号进一步优化这些决策，例如在沙箱服务器已经出现资源争用时驱逐 KV 缓存，或在长时间推理等待期间回收沙箱内存。

更好的机制支持： TideSand 的主要贡献是提出了实现高性能保真沙箱的三个关键理念。然而，实现这些理念的机制仍可进一步提高效率，这有待未来研究。例如，我们发现没有现成的优质即读即用方法来实现沙箱之间的内存带宽隔离，这使得实现完美的工具调用保真度变得困难。借助更高效的机制（如 Dong 等人，2026年），还可以进一步降低检查点与恢复开销。

9 结论

智能体沙箱带来了沙箱管理的新机遇：突发式工具执行支持资源多路复用，从而实现更高的资源利用率。然而，资源争用可能扭曲指导后续智能体决策的测量结果，损害工具调用保真度和任务质量。我们提出了 TideSand，它结合代理感知调度、推测式换出和选择性回滚，实现高性能保真的沙箱多路复用。在六个模型-工作负载对上进行的评估表明，TideSand 在保持工具调用保真度的同时，吞吐量达到严格预留的 3.8–10× 倍；在吞吐量相近的情况下，其保真度比超额分配高 7–44%；并将智能体任务奖励的误差从 77% 降至 4%。我们希望这项工作能够推动开发出更多面向大语言模型智能体、具有更高性能保真的沙箱系统，以实现高资源利用率。

参考文献

Agache、A.、Brooker、M.、Iordache、A.、Liguori、A.、Neuge-bauer、R.、Piwonka、P. 和 Popa、D.-M. Firecracker：面向无服务器应用的轻量级虚拟化。在 第 *17* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 20*），第 419–434 页。USENIX协会，2020年。网址：https://www.usenix.org/conference/nsdi20/presentation/agache。

Artificial Analysis. Terminal-Bench 2.1 基准测试，2026年。网址 https://artificialanalysis.ai/evaluations/terminalbench-2-1。访问于2026年10月25日。Chen, Z., Chen, S., Ning, Y., Zhang, Q., Wang, B., Yu, B., Li, Y., Liao, Z., Wei, C., Lu, Z., Dey, V., Xue, M., Baker, F. N., Burns, B., Adu-Ampratwum, D., Huang, X., Ning, X., Gao, S., Su, Y., 和 Sun, H. ScienceAgentBench：迈向对数据驱动的科学发现领域语言智能体的严格评估。载于 国际学习表征会议，第 2025 卷，第 96934–96990 页，2025年。网址 https://proceedings.iclr.cc/paper_files/paper/2025/file/f12b4df26344f3be803c06b555252efe-Paper-Conference.pdf。

Dong, Y., He, J., Liu, S., Hou, Y., Du, D., Xu, Z., Yu, S., Yang, B., Xia, Y., 和 Chen, H. DeltaBox：通过毫秒级沙箱检查点/回滚扩展有状态人工智能智能体。arXiv 预印本 arXiv:2605.22781，2026年。网址 https://arxiv.org/abs/2605.22781v2。Du, D., Yu, T., Xia, Y., Zang, B., Yan, G., Qin, C., Wu, Q., 和 Chen, H. Catalyzer：通过无初始化启动实现无服务器计算的亚毫秒级启动。载于第二十五届程序设计语言与操作系统架构支持国际会议论文集，第 467–481 页。ACM，2020年。doi：10.1145/3373376.3378512。网址https://doi.org/10.1145/3373376.3378512。E2B。E2B 基础设施架构，2026a。网址 https://github.com/e2b-dev/runtime/blob/main/docs/ARCHITECTURE.md。访问于2026年10月25日。

E2B. 使用 Docker Compose 嵌入 E2B，2026b。网址 https://github.com/e2b-dev/runtime/blob/main/embed/compose/README.md。访问日期：2026年10月25日。

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., 和 Zuo, P. 使用 CachedAttention 为多轮对话提供经济高效的大语言模型服务。载于 *2024*年 *USENIX*年度技术会议（*USENIX ATC 24*），第 111–126 页。USENIX协会，2024年。网址 https://www.usenix.org/conference/atc24/presentation/gao-bin-cost。

Huang, J., T. Ma, Liu, Z., Lin, S., Chen, K., Jiang, J., Liao, X., Shan, Y., Y. Wu, Zhang, N., Lu, M., T. Ma, Gong, H., and M. Zhang. TrEnv-X：在不同函数和节点之间透明地共享无服务器执行环境。*ACM*计算机系统汇刊，44(3): 1–39，2026a。doi: 10.1145/3805475。URLhttps://doi.org/10.1145/3805475。

Huang, J., Tang, H., Chen, J., Liu, Y., Chen, Y., Cheng, Y., Tao, Y., Zhou, J., Chen, Y., Chen, H. 等。DeepSeek Elastic Compute（DSec）：面向大规模高效智能体训练的沙箱基础设施。arXiv 预印本 arXiv:2609.22978，2026年b。网址https://arxiv.org/abs/2609.22978v1。Kang, H., Li, Z., Xu, W., Yang, X., Chen, Y., Wang, J., Chen, B., Krishna, T., Xu, C. 和 Arora, S. ThunderAgent：简单、快速且程序感知型的智能体推理系统。arXiv 预印本 arXiv:2602.13692，2026年。网址 https://arxiv.org/abs/2602.13692v3。Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H. 和 Stoica, I. 使用 PagedAttention 高效管理大语言模型服务的内存。载于第*29*届操作系统原理研讨会论文集，第611–626页。ACM，2023年。doi：10.1145/3600006.3613165。网址 https://doi.org/10.1145/3600006.3613165。Lagkas Nikolos, O., Alverti, C., Psomadakis, S., Goumas, G. 和 Koziris, N. Squeezy：无服务器函数的快速虚拟机内存回收。载于第*21*届欧洲计算机系统会议论文集，第1467–1482页。ACM，2026年。doi：10.1145/3767295.3769357。网址 https://doi.org/10.1145/3767295.3769357。Li, H., He, R., Mang, Q., Zhang, Q., Mao, H., Chen, X., Z hou, H., Zhang, H., Cheung, A., Gonzalez, J. 和 Stoica, I. Continuum：借助 KV 缓存生存时间实现高效且稳健的多轮大语言模型智能体调度。arXiv 预印本 arXiv:2511.02230，2026年a。网址 https://arxiv.org/abs/2511.02230v7。Li, M., XU, C., Zhang, Q., Yu, J., Sun, X., Mai, H. 和 Xie, Z. 面向高扇出智能体沙箱的内存压缩。arXiv 预印本 arXiv:2609.11294，2026年b。URL https://arxiv.org/abs/2609.11294v1。

Liu, Q., Yang, Y., Du, D., Xia, Y., Zhang, P., Feng, J., Larus, J. R., and Chen, H. 协调效率与实用性：利用 Jiagu 优化无服务器计算中的资源利用率。载于*2024*年 *USENIX*年度技术会议（*USENIX ATC 24*），第 1–17 页。USENIX协会，2024a。URL https://www.usenix.org/conference/atc24/presentation/liu-qingyuan。

Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., Maire, M., Hoffmann, H., Holtzman, A. 和 Jiang, J. CacheGen：用于快速大语言模型服务的 KV 缓存压缩与流式传输。载于 *2024*年*ACM SIGCOMM*会议论文集，第 38–56 页。ACM，2024年b。doi：10.1145/3651890.3672274。URLhttps://doi.org/10.1145/3651890.3672274。Liu, Y., Cheng, Y., Yao, J., An, Y., Chen, X., Feng, S., Huang, Y., S hen, S., Zhang, R., Du, K. 和 Jiang, J. LMCache：用于企业级大语言模型推理的高效 KV 缓存层。arXiv 预印本 arXiv:2510.09665，2025年。URLhttps://arxiv.org/abs/2510.09665v2。Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J. 和 Ha, D. The AI Scientist：迈向全自动开放式科学发现。arXiv 预印本 arXiv:2408.06292，2024年。URLhttps://arxiv.org/abs/2408.06292v3。Luo, M., Shi, X., Cai, C., Zhang, T., Wong, J., Wang, Y., Wang, C., Huang, Y., Chen, Z., Gonzalez, J. E. 和 Sto- ica, I. Agentix：面向大语言模型智能体（作为通用程序）的高效服务引擎。载于 第 *23* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 26*），第 2443–2459 页。USENIX协会，2026年。URL https://www.usenix.org/conference/nsdi26/presentation/luo。Mang, Q., Chai, W., Li, Z., Mao, H., Zhou, S., Du, A., Li, H., Liu, S., Chen, E., Wang, Y., Chu, X., Cheng, Z., Xu, Y., Xia, T., Wang, Z., Shi, T., Yao, J., Zhao, Y., Zhang, Q., Ruan, C., Shen, Z., Liu, K., He, R., Xing, D., Li, Z., Zeng, Z., Jiang, Y., Cheng, L., Zhao, Z., Sun, Y., Zheng, W., Zhang, M., Ji, R., Tu, X., Zheng, Z., Chen, Z., Zhou, K., Wang, Z., Chen, J., Korolova, A., Henderson, P., Viswanath, P., Ganesh, V., Xie, S., Liu, Z., Song, D., Min, S., Stoica, I., Gonzalez, J. E., Shang, J. 和 Cheung, A. Frontiercs：不断演化的智能所面临的不断演化的挑战，2025年。URLhttps://arxiv.org/abs/2512.15699。

Mang, Q., Chai, W., He, R., Li, D., Zheng, Z., He, Y., Zhou, X., Wang, H., Ray, S., Peng, B., Ji, R., Yang, S., Xi, H., Chen, B., Han, X. D., Pan, M., Mishra, M., Yu, B., Liu, S., Akillioglu, K., Pan, Y., Wang, Z., Cemri, M., Antunes, D., Wang, Y., Zhang, A. I., Chen, Z., Zhang, Q., Liu, K., Huang, W., Ruan, S., Liu, S., Li, D., Cheng, Z., Shen, Z., Zhou, S., Chen, H., Zhang, Y., Lyu, B., Ao, Q., Zheng, H., Jiang, C., Hu, J., Chen, X., Li, K., Zhong, Z., Tian, H., Zhao, T., He, Y., Li, H., O. Press, Jiao, J., Olukotun, K., Narasimhan, K. R., Shang, J., Sen, K., Zaharia, M., Dimakis, A., Stoica, I., Gonzalez, J. E. 和 Cheung, A. Frontiercs 2：面向计算机科学研发的基准测试，2026年。URLhttps://arxiv.org/abs/2610.29932。

Maslej, N.、Fattorini, L.、Perrault, R.、Gil, Y.、Parli, V.、Kariuki, N.、Capstick, E.、Reuel, A.、Brynjolfsson, E.、Etchemendy, J. 等。《2025年人工智能指数报告》。arXiv 预印本 arXiv:2504.07139，2025年。网址 https://arxiv.org/abs/2504.07139v3。

Merrill, M.、Shaw, A.、Carlini, N.、Li, B.、Raj, H.、Bercovich, I.、Shi, L.、Shin, J.、Walshe, T.、Buchanan, E. K. 等。《Terminal-Bench：在命令行界面中对智能体执行困难且真实任务的基准测试》。载于国际学习表征会议，第 2026 卷，第 40903–40986 页，2026年。网址 https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf。

Modal。配置中央处理器、内存和磁盘，2026a。网址 https://modal.com/docs/guide/resources。访问日期：2026年10月25日。

Modal。沙箱，2026b。网址 https://modal.com/docs/sdk/py/latest/Sandbox。访问日期：2026年10月25日。

英伟达。CUDA 检查点与恢复工具，2026年。网址 https://github.com/NVIDIA/cuda-checkpoint。访问日期：2026年10月25日。

O. Press、B. Amos、H. Zhao、Y. Wu、S. Ainsworth、D. Krupke、P. Kidger、T. Sajed、B. Stellato、J. Park 等。《AlgoTune：语言模型能加速通用数值程序吗？》载于 神经信息处理系统进展，第 38 卷，主会。Curran Associates, Inc.，2025年。doi：10.52202/085713-3585。网址https://doi.org/10.52202/085713-3585。

R. Qin、Li, Z.、W. He、J. Cui、H. Tang、F. Ren、T. Ma、S. Cai、Y. Zhang、M. Zhang、Y. Wu、W. Zheng 和 X. Xu。《Mooncake：面向大语言模型服务的以 KVCache 为中心的解耦架构》。*ACMTransactions on* 存储，22(4)：1–38，2026年。doi：10.1145/3773772。网址：https://doi.org/10.1145/3773772。

Terminal-Bench-Science 团队。Terminal-Bench-Science：评估科学领域研究工作流中的 AI 智能体。计算机软件，2026年。网址：https://doi.org/10.5281/zenodo.22110253。gVisor 作者。什么是 gVisor？，2026年。网址：https://gvisor.dev/docs/。访问日期：2026年10月25日。

Terminal-Bench 团队。Terminal-Bench 2.1 排行榜，2026a。网址：https://www.tbench.ai/leaderboard/terminal-bench/2.1。访问日期：2026年10月25日。Terminal-Bench 团队。Terminal-Bench 2.1，2026b。网址：https://www.tbench.ai/news/terminal-bench-2-1。访问日期：2026年10月25日。

Terminal-Bench 团队。Terminal-Bench 4.0，2026年c。网址 https://www.tbench.ai/news/terminal-bench-4-0。访问于2026年10月25日。

D. Ustiugov、P. Petrov、M. Kogias、E. Bugnion 和 B. Grot。无服务器函数快照的基准测试、分析与优化。载于第*26*届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第559–572页。ACM，2021年。doi：10.1145/3445814.3446714。网址 https://doi.org/10.1145/3445814.3446714。

Vals AI。Terminal-Bench 2.1 排行榜与方法论，2026年。网址 https://vals-ai.com/benchmarks/terminal-bench-2-1。访问于2026年10月25日。

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J. 等。OpenHands：面向人工智能软件开发者的通用智能体开放平台。在 *International Conferenceon LearningRepre-sentations*，第 2025 卷，第 65882–65919 页，2025年。网址：https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf。

Wei, X., Lu, F., Wang, T., Gu, J., Yang, Y., Chen, R., and Chen, H. 无需预置并发：为无服务器计算快速设计的 RDMA 远程分叉。载于第 *17* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 23*），第 497–517 页。USENIX协会，2023年。网址：https://www.usenix.org/conference/osdi23/presentation/wei-rdma。

Xia, T., Li, H., Li, Z., Chen, X., Kang, H., Qiao, Y., Xu, Y., and Stoica, I. 闲置是相对的：利用工具调用空闲窗口，在智能体系统中通过MORI 实现卸载。arXiv 预印本 arXiv:2606.00866，2026年。网址：https://arxiv.org/abs/2606.00866v1。Xu, C., Liu, Y., Li, Z., Chen, Q., Zhao, H., Zeng, D., Peng, Q., Wu, X., Zhao, H., Fu, S., and Guo, M. FaaSMem：利用内存池架构提升无服务器计算的内存效率。载于第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *3* 卷，第 331–348 页。ACM，2024年。doi：10.1145/3620666.3651355。网址：https://doi.org/10.1145/3620666.3651355。Yang, J., Jimenez, C., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and O. Press. SWE-agent：智能体—计算机接口实现自动化软件工程。载于神经信息处理系统进展，第 37 卷，第 50528–50652 页。Curran Associates, Inc.，2024年。doi：10.52202/079017-1601。网址：https://doi.org/10.52202/079017-1601。Yao, J., Li, H., Liu, Y., Ray, S., Cheng, Y., Zhang, Q., Du, K., Lu, S., and Jiang, J. CacheBlend：通过缓存知识融合实现快速检索增强生成大语言模型服务。*ACM*计算机系统汇刊，2026年。doi：10.1145/3790254。网址：https://doi.org/10.1145/3790254。Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., and Ceze, L. FlashInfer：面向大语言模型推理服务的高效、可定制注意力引擎。载于机器学习与系统论文集，第 7 卷。MLSys，2025年。网址：https://proceedings.mlsys.org/paper_files/paper/2025/file/db f02b21d77409a2db30e56866a8ab3a-Paper-Conference.pdf。Yu, H., Basu Roy, R., Fontenot, C., Tiwari, D., Li, J., Zhang, H., Wang, H., and Park, S.-J. RainbowCake：通过分层容器缓存与共享缓解无服务器计算中的冷启动。载于第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *1* 卷，第 335–350 页。ACM，2024年。doi：10.1145/3617232.3624871。网址：https://doi.org/10.1145/3617232.3624871。Zhang, Y., Wo, T., Wang, J., Sun, X., M. Zhang, Yuan, C., Li, L., Hu, C., Zomaya, A. Y., and Yang, R. SpecBox：面向高效大语言模型智能体服务的推测式沙箱调度。arXiv 预印本 arXiv:2607.23933，2026年。网址：https://arxiv.org/abs/2607.23933v2。Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Bar- rett, C., and Sheng, Y. SGLang：高效执行结构化语言模型程序。载于神经信息处理系统进展，第 37 卷，第 62557–62583 页。Curran Associates, Inc.，2024年。doi：10.52202/079017-2000。网址https://doi.org/10.52202/079017-2000。Zheng, Y., Fan, J., Fu, Q., Yang, Y., Zhang, W., and Quinn, A. AgentCgroup：理解并控制人工智能智能体的操作系统资源。arXiv:2602.09345，2026年。Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. DistServe：将预填充与解码分离，以优化大语言模型服务的有效吞吐量。载于第 *18* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 24*），第 193–210 页。USENIX协会，2024年。网址https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin。Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Srid- har, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. WebArena：用于构建自主智能体的真实网络环境。载于国际学习表征会议，第 2024 卷，第 15585–15606 页，2024年。网址https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf。

10 人工智能使用披露

我们主要使用代码智能体协助生成代码。同时，我们也使用人工智能工具进行校对、语法纠正和句子润色。

大语言模型价格数据

表 2。 如图 1(a) 所示，在 Terminal-Bench 2.1 上得分至少达到 50% 的模型。得分取我们数据中记录的最佳独立结果：T 为官方 tbench.ai 排行榜（Terminal-Bench 团队，2026a），V 为 Vals AI（Terminus 2 测试框架）（Vals AI，2026年），A 为 Artificial Analysis（Artificial Analysis，2026年）。价格为 2026年10月 汇总的每 100 万个 token 的美元输入/输出费率；混合价格按输入与输出 3:1 加权。粗体行表示创下新的最低混合价格。

模型	开发者	释放	Open	TB2.1 (%)	Src	输入	输出	混合
Gemini 3 Pro	谷歌	2025年11月18日		73.9	T	2.00	12.00	4.500
Gemini 3 Flash	谷歌	2025年12月17日		53.9	V	0.50	3.00	1.125
Claude Sonnet 4.6	Anthropic	2026年2月17日		57.3	V	3.00	15.00	6.000
Gemini 3.1 Pro	谷歌	2026-02-19		70.8	V	2.00	12.00	4.500
GPT-5.4 mini	OpenAI	2026-03-17		54.7	V	0.75	4.50	1.688
GLM-5.1	智谱	2026-03-27	✓	58.6	T	1.00	3.20	1.550
通义千问3.6 Plus	阿里巴巴	2026-03-31		53.2	V	0.50	3.00	1.125
Claude Opus 4.7	Anthropic	2026年4月16日		68.9	T	5.00	25.00	10.000
Kimi K2.6	Moonshot	2026年4月20日	✓	53.6	V	0.95	4.00	1.712
MiMo V2.5	小米	2026-04-22	✓	60.7	V	0.14	0.28	0.175
MiMo V2.5 Pro	小米	2026-04-22	✓	57.3	V	0.435	0.87	0.544
GPT-5.5	OpenAI	2026-04-23		83.2	T	5.00	30.00	11.250
DeepSeek V4	DeepSeek	2026-04-24	✓	50.2	V	1.32	3.96	1.980
Composer 2.5	Cursor	2026-05-18		58.4	V	0.50	2.50	1.000
Gemini 3.5 Flash	谷歌	2026-05-19		74.2	V	1.50	9.00	3.375
通义千问3.7 Max	阿里巴巴	2026-05-19		61.0	V	2.50	7.50	3.750
Claude Opus 4.8	Anthropic	2026年5月28日		78.9	T	5.00	25.00	10.000
MiniMax-M3	MiniMax	2026年6月1日	✓	65.2	A	0.60	2.40	1.050
通义千问3.7 Plus	阿里巴巴	2026年6月2日		52.8	V	0.40	1.60	0.700
Nemotron 3 Ultra	英伟达	2026年6月3日	✓	50.9	V	0.60	2.40	1.050
Claude Fable 5	Anthropic	2026年6月9日		83.8	T	10.00	50.00	20.000
Kimi K2.7 Code	Moonshot	2026年6月12日	✓	67.0	V	0.95	4.00	1.712
GLM-5.2	智谱	2026年6月16日	✓	67.8	V	1.40	4.40	2.150
Claude Sonnet 5	Anthropic	2026-06-30		74.6	T	2.00	10.00	4.000
Grok 4.5	xAI	2026-07-08		79.3	T	2.00	6.00	3.000
GPT-5.6 Luna	OpenAI	2026-07-09		79.0	V	0.20	1.20	0.450
GPT-5.6 Sol	OpenAI	2026-07-09		89.5	A	4.00	20.00	8.000
GPT-5.6 Terra	OpenAI	2026-07-09		78.4	T	2.00	12.00	4.500
Muse Spark 1.1	Meta	2026-07-09		76.2	T	1.25	4.25	2.000
Inkling	Thinking Machines	2026年7月15日	✓	55.1	A	1.00	4.05	1.762
Kimi K3	Moonshot	2026年7月16日	✓	85.0	A	3.00	15.00	6.000
Gemini 3.5 Flash-Lite	谷歌	2026年7月21日		50.2	V	0.30	2.50	0.850
Gemini 3.6 Flash	谷歌	2026年7月21日		73.8	V	1.50	7.50	3.000
Ling 3.0 Flash	InclusionAI	2026年7月23日	✓	50.2	V	0.075	0.22	0.111
Claude Opus 5	Anthropic	2026年7月24日		84.6	V	5.00	25.00	10.000
Inkling Small	Thinking Machines	2026年7月30日	✓	55.1	V	0.30	1.20	0.525
DeepSeek V4 Flash 0731	DeepSeek	2026年7月31日	✓	67.0	V	0.44	1.32	0.660
Qwen3.8 Max	阿里巴巴	2026年8月2日		67.4	V	2.00	6.00	3.000
Muse Spark 1.2	Meta	2026年8月5日		69.7	V	1.25	4.25	2.000
Grok 4.6	xAI	2026-08-12		78.3	V	2.00	6.00	3.000
DeepSeek V4 Pro 0813	DeepSeek	2026-08-13	✓	54.7	V	1.32	3.96	1.980
Gemini 3.7 Flash	谷歌	2026-08-13		77.5	V	1.50	7.50	3.000
Qwen3.8 27B	阿里巴巴	2026-08-14	✓	79.8	A	0.50	3.00	1.125
GLM-5.3	智谱	2026-08-18	✓	83.9	A	1.40	4.40	2.150
GLM-5.3-Flash	智谱	2026-08-26	✓	84.3	A	0.075	0.25	0.119
Hy4 预览版	腾讯	2026-08-27	✓	55.1	V	0.834	2.501	1.251
Claude Fable 5.1	Anthropic	2026年9月1日		91.4	A	10.00	50.00	20.000
Gemini 3.8 Flash	谷歌	2026年9月2日		87.6	A	1.50	7.50	3.000
Muse Spark 1.3	Meta	2026年9月2日		79.0	V	1.25	4.25	2.000
通义千问3.8 Max 0902	阿里巴巴	2026年9月2日		88.8	A	2.00	6.00	3.000
GPT-6 Astra	OpenAI	2026年9月3日		89.9	A	10.00	50.00	20.000
DeepSeek V4.1 Flash	DeepSeek	2026年9月9日	✓	74.5	V	0.30	1.20	0.525
Grok 4.7	xAI	2026-09-21		73.4	V	2.00	6.00	3.000
MiMo V2.6 Flash	小米	2026-09-21	✓	76.4	V	0.14	0.28	0.175
MiMo V2.6 Pro	小米	2026-09-21	✓	67.8	V	0.435	0.87	0.544
Claude Opus 5.5	Anthropic	2026-09-22		87.6	V	4.00	20.00	8.000
GPT-6 Luna	OpenAI	2026-09-22		73.0	V	0.10	0.50	0.200
GPT-6 Sol	OpenAI	2026-09-22		83.2	V	2.00	10.00	4.000
Claude Sonnet 5.5	Anthropic	2026-09-28		83.2	V	2.00	10.00	4.000
Gemini 4 Argon	谷歌	2026-09-30		57.7	V	4.00	20.00	8.000
Muse Spark Open	Meta	2026-10-15	✓	76.4	A	1.25	4.25	2.000

B AGENTCGROUP 基线

§ 中的 AgentCgroup 基线6 是 AgentCgroup 工具调用资源控制器的移植版本（Zheng 等人，2026年），作为其 cgroup-v2 控制循环，与其他策略在同一守护进程中运行。

每次由沙箱的模型请求界定的工具调用都会开启一个资源域。根据调用推导出的提示（文件工具和轻量 Shell 命令为低级，软件包获取为中级，其他调用均为高级）会将沙箱的 memory.high 设为该调用的内存配额加上调用开始时沙箱的内存用量（256 MiB、1 GiB 或上限），并将其 CPU 权重设为对应类别的权重（高级为 150，低级为 50）。当主机内存使用率超过 85%、内存压力失速信息超过 10%，或高级调用因内存而停滞时，就会出现内存压力，并开启持续一秒的保护窗口。在此期间，高级沙箱通过 memory.low 获得保护；低级沙箱则被限制在当前用量加 256 MiB，并在超过其上限三秒后被冻结。由于无法在容器的执行路径上创建控制器的每调用子控制组，因此资源域就是沙箱的控制组；而智能体对控制器限流反馈的响应——以更高提示重新发起调用——则会在限流 30 秒后被重放为提示提升。