Mayfly：面向递归自我改进的生命周期感知型高带宽闪存

Chunlin Tian 新加坡国立大学 新加坡

Mingjia Fan 新加坡国立大学 新加坡

Feng Yu 新加坡国立大学 新加坡

Qingyun Zou 新加坡国立大学 新加坡

Wenju Zhao 华中科技大学 中国

Weng-Fai Wong 新加坡国立大学 新加坡

Bingsheng He*新加坡国立大学 新加坡

arXiv:2611.32264v1 [cs.AR] 24 Nov 2026

摘要

递归自我改进（RSI）使大语言模型能够通过针对各项任务提出并评估候选变体来改进其智能体系统。同一代中的会话会共同扩展各自的 KV 缓存，并在完成时释放缓存，从而形成先突增、后下降的负载。在高并发情况下，即使是就绪会话的 KV 也会超出图形处理器高带宽内存的容量。将这些 KV 卸载到主机动态随机存取存储器或固态硬盘可以增加容量，但就绪会话在执行前仍需等待高带宽内存腾出空间。高带宽闪存（HBF）可以就地承接溢出数据，但过多的读取会延长解码步，而重复写入则会消耗闪存耐久性。我们的关键洞见是，就绪 KV 的读取位置应随突增及其后的下降而变化：在突增导致数据溢出高带宽内存时，就地从 HBF 读取，读取量以每一步的带宽所允许的上限为准；待负载下降腾出空间后，再从高带宽内存读取。就绪与停放状态之间的切换只决定是否读取某个块，因此该块的闪存副本仍保持有效；而会话结束（由编排器通知）则决定其专属 KV 何时失效。我们提出 Mayfly，这是一种面向 RSI 的生命周期引导型 HBF 管理机制。Mayfly 在溢出期间限制每一步的 HBF 读取量，并在会话结束、释放空间时将 KV 移回高带宽内存。对于从高带宽内存逐出的每个块，它都会保留一个有效的后备副本，以避免状态转换期间重复写入；同时将会话结束信号转发给控制器，避免垃圾回收时复制已失效的 KV。实验结果表明，在每代 128 个会话、增加四个 HBF 站点的情况下，Mayfly 比将相同 KV 停放在 HBF 上快 34%，比在高带宽内存已满时始终从 HBF 读取快 23%；随着并发度提高，Mayfly 相较于停放的优势也随之扩大，在八个站点、256 个会话时达到 55%。每块保留一份闪存副本，并在收到通知的会话结束时进行修剪，可避免写入放大。

关键词

高带宽闪存、KV缓存、递归自我改进、大语言模型服务

1 引言

递归自我改进（RSI）循环让大语言模型改进其所运行的智能体系统。我们研究的是测试框架 RSI，它保持

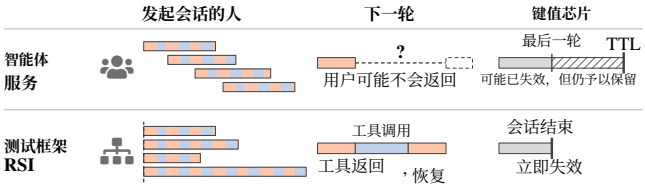


图1：RSI 和智能体服务中的键值生命周期。柱形表示会话（桃色：大语言模型调用，蓝色：工具调用）；灰色：键值生命周期。

模型不变，重写其测试框架、提示词、工具和控制流，并通过运行每个候选项 [12, 56, 64, 67]来为其评分：每一代都会在一组任务上评估一批候选项，每次评估都是一个智能体会话，交替进行大语言模型推理和工具调用 [2, 19]。选择会等待所有会话结束，因此完成一代所需的时间决定了循环的节奏。服务引擎会在会话的各次调用之间保留其 KV 缓存，而各会话则争用模型权重占用后剩余的高带宽内存 [24, 68]。在我们研究的循环中，编排器会同时启动一代中的所有会话，并且不会替换已结束的会话，因此它们的 KV 会先突发增长，随后逐渐减少（图3a）。在突发期间，会话在等待工具时处于停放状态；在大语言模型调用正在运行或排队时，则处于就绪状态，无论其 KV 位于何处。图形处理器及其高带宽内存上的争用会延长大语言模型调用，使更多 KV 处于就绪状态：在一项轨迹驱动模拟中，对于单个仅在高带宽内存中保存 KV 的 H200 上每代 32 个 RSI [67] 会话的情形，仅就绪 KV 就有 86% 的运行时间超过高带宽内存容量；相比之下，当没有调用等待高带宽内存时，所有实时 KV 超出该容量的时间为 55%（图3a），因为排队中的调用会使其会话保持就绪（§3.1）。

最新进展。智能体服务系统会在会话运行前，从主机内存或驱动器中恢复这类键值数据 [1, 9, 26, 32, 42, 48]，，以增加容量，却无法增加 HBM 中的可用空间。高带宽闪存（HBF）将 NAND 堆叠置于 GPU 封装上 [35, 53]，，使超出 HBM 容量的键值数据也能在其所在位置参与步骤，但持续读取带宽和写入耐久性有限。近期研究将其用于智能体和大语言模型服务，存放暂停会话的键值数据并在需要时恢复 [3]，或让解码直接读取原位键值数据 [20, 43, 58, 65, 69]。闪存是否适用于键值数据仍有争议：如果服务器不知道会话何时结束，就无法判断其键值数据何时失效；而在生产环境服务轨迹中，写入键值池的每个字节平均只会被读回 0.20–0.88 次 [28]。

*通讯作者。

生命周期感知放置更进一步：它在分配时为每个上下文片段选择存储层，并在测试框架报告的上下文边界处释放块 [63]。然而，在写入键值数据时做出的放置决策，既无法决定后续步骤中如何提供这些键值数据，也无法利用会话结束后释放的 HBM：即使已知某个块何时结束，究竟应当原位读取还是移回 HBM，仍取决于当前就绪集合、空闲 HBM 以及该步骤的带宽。 那么，当 *RSI* 的就绪键值数据增长到超出 *HBM* 容量、随后又缩减时，*HBF* 应如何为其提供服务，才能让就绪会话不再等待 *HBM* 空间，同时避免闪存读取降低步吞吐量，并且闪存写入量不超过键值数据本身所需？

挑战。 突发的每个阶段都与其中一个目标相悖。随着突发流量上升，一代中的会话会一同溢出：如果每个就绪会话在每个步骤都从 HBF 读取其溢出数据，读取量就会超过堆栈持续带宽，并延长每个步骤。随着突发流量下降，已完成的会话会释放 HBM；而当 HBM 再次有空间后，KV 仍从闪存读取，这会带来时间和能耗成本。在整个过程中，一个会话会在就绪和停放之间切换数十次，而且大多不到一秒，因此，在每次切换时都迁移 KV 的层每次都会重写闪存，而将 KV 保留固定时间的层则会让垃圾回收复制已结束会话的 KV。

我们的贡献。 据我们所知，这是首项针对递归自我改进管理 HBF 的工作。在递归自我改进中，键值突发会超出高带宽内存容量，随后逐渐回落。 RSI 会暴露这些决策所需的状态和事件。我们的核心洞见是，就绪KV的读取位置应随突发及其回落而变化：当突发超出高带宽内存容量时，尽可能按照每个步骤的带宽限制直接从 HBF 读取；当突发回落、释放出高带宽内存后，再从高带宽内存读取。就绪与停放之间的切换只决定是否读取一个块，因此其闪存副本仍然有效；而会话结束则由编排器发出通知，决定何时释放该会话的私有键值。与交互式智能体服务不同，交互式服务中的键值会在计时器上等待可能再也不会返回的用户；编排器会主动结束每个会话（图 1）：当智能体不调用工具而直接回复时，RSI 会话即告结束，且不会恢复，因此其私有键值会立即失效；共享任务前缀则会一直保留，直至最后一位用户结束会话。我们提出 `Mayfly`，将这些信号转化为生命周期引导的高带宽闪存管理。逐步门控仅在不会降低该步骤预估令牌吞吐量时，才允许请求直接读取键值；每个获准的请求都会对其完整上下文执行注意力计算。随着会话结束并释放高带宽内存，`Mayfly` 会将键值移回高带宽内存，使后续步骤不再从闪存读取。 `Mayfly` 会为每个从高带宽内存驱逐的块保留有效的 HBF 副本，因此后续驱逐无需写入任何内容；它还会将会话结束事件转发给高带宽闪存控制器，使垃圾回收无需复制已结束会话的私有键值。它涵盖服务运行时、双层注意力内核，以及位于 HBF 基础裸片上的键值感知控制器。我们在离散事件模拟器中，基于真实智能体轨迹和记录的递归自我改进运行评估 `Mayfly`；该模拟器的图形处理器模型经过 H200 校准，并对 HBF 路径进行了建模。在新增相同的四个 HBF 站点、每代 128 个会话时，`Mayfly` 比将相同键值停放在 HBF 上的方案快 34%，首令牌中位时间低 1.9×；比每当高带宽内存占满就从 HBF 读取的方案快 23%。在 256 个会话、八个站点时，6× 比停放方案快 55%，首令牌中位时间低 2.。即使键值停放在 HBF 上，RSI 每写入一个字节，仍会读回 23–28 次；相比之下，服务键值缓存池的读回次数为 0.20–0.88。我们的贡献如下：

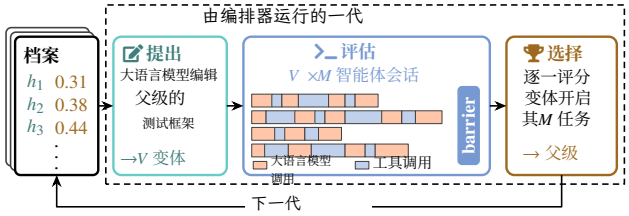


图 2：harness RSI 循环的一代。选择会等待其所有会话结束。

- 一项特征分析：在负载状态下，就绪会话持有 RSI 大部分键值，而仅这些会话的键值就大多数时候超过高带宽内存容量（§3）。
- `Mayfly`，它利用会话状态、空闲 HBM 和会话结束信息，决定每个步骤将哪个就绪KV就地读取或返回 HBM，并从主机将每个块写入闪存一次（§4）。
- 磨损分析表明，每个块保留一份副本，并在会话结束时进行修剪，可避免写入放大：一个堆栈的寿命可达到1.7–2。8× 倍于采用30分钟到期时间或在层已满时释放的情况，并且可达到9–37× 倍于每次返回高带宽内存时都丢弃其副本的情况（§5）。

2 背景与相关工作 k

2.1 递归自我改进

在递归自我改进（RSI）中，人工智能系统会改变自身、评估变化，并保留能够带来改进的变化 [10, 51]；对于大语言模型而言，模型写入变化，基准测试对其进行评判 [12, 66]。循环可以改变模型的权重 [14], 、某项任务的提示词或解题程序 [7, 37, 50, 52], ，也可以改变围绕固定模型构建的测试框架：其提示词、工具、内存和控制流 [8, 25, 29, 56, 60, 61, 64, 67]。下文研究测试框架循环，并将其称为 RSI。权重固定时，图形处理器负责提供推理服务，而它们增加的内存则是并发会话在多次调用之间保留的键值。每一代运行三个步骤（图 2）：首先提出 V 候选运行框架，评估每个候选项在 M 任务上的表现，然后选择为下一代提供种子的候选项。每次评估运行一个会话，其中基于某个候选项构建的智能体经过多轮交互解决一项任务，例如 GitHub issue。每一轮都会发出一个请求，即一次大语言模型调用；其输出通常会在沙箱中调用工具，而工具的结果会进入下一次请求。会话在有请求正在运行或排队时处于就绪状态，在等待工具时处于停放状态，其键值也会被放置在相应位置。 在所研究的循环中，一代会话同时开始，选择会等待所有会话结束。服务引擎会在请求之间保留会话的键值，因此每个请求只需预填充自上一次请求以来新增的词元 [24, 68]。以下三个属性对后文至关重要。请求结束并不会使其失效，因为会话的下一次请求会再次读取其历史记录；就绪与停放之间的切换只会改变读取方式；而编排器结束会话后，会话的私有键值即可回收。已存储的副本只有在其编码的上下文未发生变化时才能继续复用：追加内容的测试框架会使其保持有效，而修剪或压缩上下文会使从第一个发生变化的词元开始的键值失效。

2.2 高带宽闪存

高带宽闪存（HBF）将 NAND 裸片堆叠在基础裸片上，并通过宽接口连接，使堆栈能够与高带宽内存并排安装在图形处理器上

封装 [35, 53]。我们将堆栈在封装上的位置称为 站点：H200 有六个高带宽内存站点，每个 HBF 堆栈各占一个站点。容量来自 NAND 的高密度：OCP 规范的目标是在 16 个芯片中实现 550GB，而 HBM4 堆叠的容量为 48–64 GB [53]。带宽则来自阵列并行性：页读取需要数微秒，因此只有同时让许多平面保持忙碌，堆栈才能实现高带宽 [20, 69], ，而其在功耗与热限制下可持续达到的速率低于接口峰值 [28]。HBF 沿袭了 NAND 的管理约束。一个页只能编程一次，且只能与其所在的整个块一起擦除，因此垃圾回收（GC）会在擦除前复制块中仍然有效的页；写入放大系数（WAF）用于统计这些复制操作。存储单元可承受的编程/擦除（P/E）周期次数有限。若控制器不知道某数据已失效，那么每次垃圾回收涉及该数据时，都会将其复制。

2.3 相关工作

传统的键值卸载。 智能体服务会在多次调用之间保留会话的键值，因为每轮都重新预填充长历史所消耗的 GPU 时间，比存储这些历史更高 [1, 9]。当高带宽内存无法容纳这些键值时，系统会将其移至主机动态随机存取存储器、本地 SSD 或远程内存池。它们会在每轮开始时将历史恢复到高带宽内存中 [4, 9, 21, 32, 48, 49], ，根据智能体的调度保留、交换或预取键值 [1, 26, 42], ，并限制写入固态硬盘的内容 [16, 27]。在这些基于恢复的方案中，较远的层增加了容量，但会话只有在键值返回高带宽内存后才能运行，因此就绪会话的数量受高带宽内存容量限制。另一类方案则在键值所在的位置计算注意力，例如在主机 CPU 上或存储设备内部 [17, 18, 41]；这种方案跳过恢复步骤，但受主机或存储设备带宽限制。

基于 HBF 的大语言模型与智能体服务。 HBF 将闪存容量引入封装，近期研究以两种方式将其用于键值缓存。热冷分层将暂停的智能体会话的键值缓存存放在 HBF 中，同时将热 KV 保留在 HBM 中，并在会话恢复时还原其 KV [3]。执行层设计允许解码直接从 HBF 原位读取 KV，可以读取全部 KV [43, 58, 65, 69] ，也可以读取按预算选取的较旧页面子集 [20], ，从而让长上下文和大批次运行超出 HBM 的容量。两种设计都会在某个块同时位于 HBM 中时，保持其闪存副本有效 [3, 58]。研究还探索了将 HBF 用于通用大语言模型推理 [54] ，以及采用写入感知型 KV 策略的生成式推荐 [46]；MemExplorer 的设计搜索将其用于长上下文 KV 溢出，但未对写入耐久性建模 [59]。

HBF 适合用于键值吗？ 键值是否应放在闪存上，取决于三个问题。第一，更快的远端访问是否能消除瓶颈：如果就绪会话在等待高带宽内存空间，更快的恢复只会让等待依旧存在。第二，写入闪存的键值是否有足够的读取次数来抵消写入成本：键值会随着每个生成的令牌增长 [35], ，因此容纳它的闪存层会持续接收追加写入。第三，在功耗、热限制和耐久性限制下，带宽能否维持；在这种情况下，封装内低功耗双倍数据速率内存可能更适合键值 [47]。Li 等人 [28] 将 SSD 风格键值池的后备层替换为 HBF，并发现该池吸收的写入字节数多于其返回高带宽内存的字节数，同时热限制会限制堆栈的带宽；他们质疑的是 HBF 能否直接替代该层，而不是其他用途。生命周期感知放置 [63] 通过在分配时放置每个上下文段来减少 HBF 写入，但这一决策既不涵盖后续读取，也不涵盖高带宽内存释放空间后的迁移。因此，HBF 的影响取决于哪些键值会到达闪存、之后如何读取这些键值，以及设备是否能获知自身何时失效。

3 特征分析与动机

3.1 就绪键值缓存超出高带宽内存容量

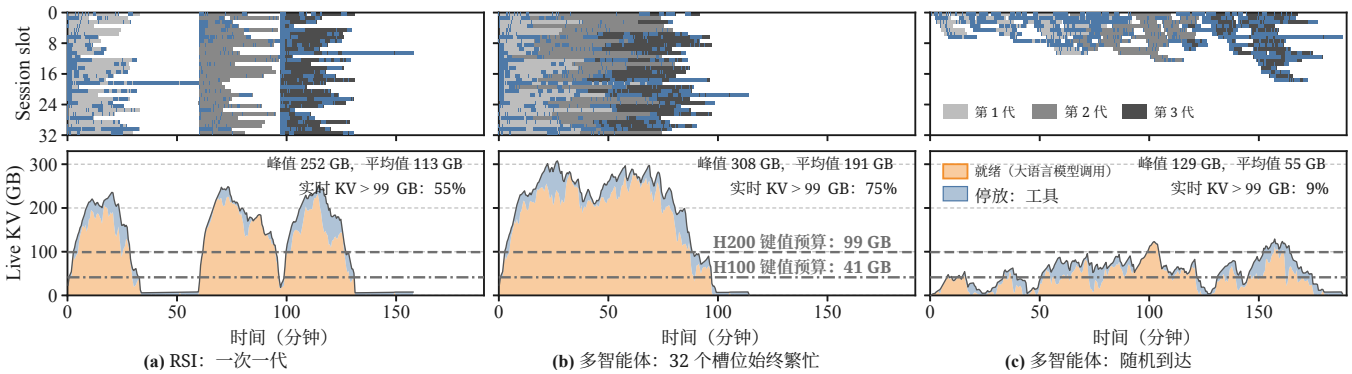
我们通过两种方式研究 RSI 在一块 H200 上的键值：使用 vLLM 运行会话，并在 vLLM 的离散事件模拟器中重放记录的会话；该模拟器的步骤时间根据 H200 重放结果拟合，以改变负载、到达情况和内存。

突增与下降。 交互式智能体服务运行着彼此无关的用户会话，这些会话会在任意时间开始；RSI 编排器会同时启动某一代的会话，并执行每次工具调用，因此工具一返回，就会立即开始下一轮。图3 模拟了一次真实 RSI 运行中的同一批 96 个会话 [67]在一块 H200 上采用三种到达模式运行的情况。作为 RSI 循环（图 3a），一代中的 32 个会话会共同扩展其上下文，三代中的 KV 峰值达到 236–252GB。已完成的会话不会被替换，因此随后 KV 会减少，而该代则等待最慢的会话；在第 1 代和第 3 代中，最后一个会话有 27 分钟（共 60 分钟）是独自运行的。因此，溢出情况会在一代内部发生变化：每当一个会话结束，就会释放 HBM，所以在峰值时溢出 HBM 的 KV，之后又能放回其中。若采用始终占满 32 个槽位的多智能体模式（图 3b），相同的会话会让 KV 保持在接近其 191 GB 平均值的水平；若随机到达（图 3c），则有 9% 的时间超出 H200 的 KV 预算，而 RSI 模式下这一比例为 55%。真实循环呈现了这两个阶段：真实 DGM 运行在每一代开始评估时，峰值达到 3。1× 其平均 KV；而我们加载的循环中，真实 Open- Hands [57] 在 SWE-rebench 问题上运行 [2] ，并遵循 DGM 的结构，峰值达到 239 GB，是 HBM 可容纳 KV 的 2。4× 倍。

没有用户时，停放的键值只需等待工具。在交互式智能体服务中，会话也会停放，等待用户响应：在 TraceLab 的 8,058 个 Claude Code 和 Codex 会话中， [70],用户回复的中位时间为 130 秒，且 25% 的间隔超过 5 分钟。在最繁忙的一天，若为托管提示缓存保留 5 分钟的键值，会话平均占用 155GB，其中 20% 处于就绪状态，17% 在等待工具，63% 在等待用户（图4a）；其就绪键值有 98% 的时间都能装进一块 H200 的高带宽内存中，因此智能体服务可以将大部分键值停放在主机内存或驱动器上 [1, 26, 32, 42]。Harness RSI 没有用户：在第一个柱状条中，其 55% 的键值处于就绪状态，其余 45% 在等待工具返回。

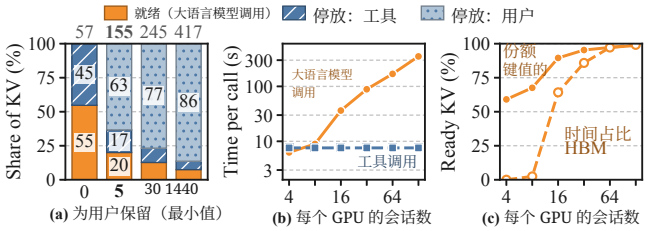
负载使大部分键值处于就绪状态。RSI操作员会提高负载，即每个循环同时运行的会话数（槽位），以及共享一块图形处理器的循环数，以便每小时完成更多会话。图4b、c 模拟了图3a 中的会话，其中键值仅存放在高带宽内存中，vLLM 会重新计算无法装入的部分；运行一个循环、槽位数从 4 到 32，或运行两个和四个这样的循环（启动时间相隔 900 秒）。吞吐量在每个变体采用两个循环、每个循环 8 个槽位时达到峰值，即每小时 27 个会话；从 32 个槽位起则降至 14–15 个。工具的耗时平均保持在记录的 7.6 秒，而大语言模型调用的耗时则从 4 个槽位时的 6.3 秒，延长到 32 个槽位时的 89 秒，以及四个循环时的 352 秒，因为每个步骤要服务更多会话，被抢占的会话则需等待高带宽内存（图4b）。就绪份额从 59% 增至 95% 和 99%；仅就绪键值就有 86% 和 99% 的时间超过高带宽内存容量（图4c），平均占用 166GB 和 560GB，而 H200 的键值预算为 99GB。

> **要点1**。 RSI 的 KV 突发会超出高带宽内存（HBM），并随着会话结束而减少。在负载下，仅其就绪KV就已超过高带宽内存（HBM）容量，而智能体保留的 KV 大多在等待用户。



(a) RSI：一次一代　(b) 多智能体：32 个槽位始终繁忙　(c) 多智能体：随机到达

图 3：三种到达模式下三代会话（4 种变体 ×8 任务）。上：按时隙显示的会话，蓝色为工具调用。下：实时 KV；在无限内存条件下，与 H200 和 H100 的 KV 容量预算对比。



(a) 为用户保留（最小值）　(b) 每个 GPU 的会话数　(c) 每个 GPU 的会话数

图 4：（a）最繁忙一天中 TraceLab 智能体的键值，按状态和用户等待时间划分；上方为平均 GB 数。（b、c）图 3a 中的会话在单张 H200 上仅使用高带宽内存、负载逐渐上升时的情况。

表 1：按每代会话数统计，智能体键值超出高带宽内存、存放于单一远端内存时的运行时间，相对于非易失性内存 express 交换的 174、282、490 和 940 分钟的倍数。

远端内存	读取带宽	32	64	128	256	结论
无（仅高带宽内存）	2.8 TB/s	2.25	2.87	3.49	3.69	太小
交换：停放 *KV* 在其会话运行前移入高带宽内存						
主机动态随机存取存储器 [45]	25 GB/s	0.99	1.00	0.99	0.99	无增益
NVMe 固态硬盘 [36]	14 GB/s	1.00	1.00	1.00	1.00	**Fits**
HBF [53]	1.5 TB/s	0.99	0.99	0.98	0.98	没有收益
就地、无界：图形处理器读取原处的就绪*KV*						
主机动态随机存取存储器	64 GB/s	7.0	15	22	25	太慢
NVMe 固态硬盘	14 GB/s	32	66	100	114	太慢
HBF	467 GB/s	1.29	1.16	0.83	0.50	**Fits**

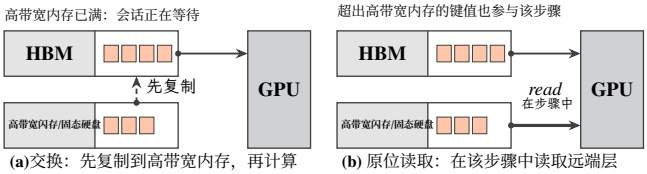
3.2 恢复与无界读取会耗费时间

我们使用 vLLM 在一台 H200 上运行 32 个仓库级会话的一代。它最多保留 261 GB 的 KV。仅使用高带宽内存（HBM）时，vLLM 会重新计算 3730 万个词元的预填充，并耗时 122 分钟、消耗 4.1 MJ；而使用 384 GB 主机内存进行卸载时，则耗时 31 分钟、消耗 0.9 MJ。更大的高带宽内存（HBM）只会移动这一阈值：在模拟器中，B300 可将 32 个会话的一代维持在 0。95× 其 KV 容量，而 40 个会话需要 1。3×。超出高带宽内存（ HBM ）的 KV 会通过两种方式到达图形处理器（GPU）（图 5），而这两种方式都会耗费时间。我们在该模拟器中对它们进行比较，并在 §3.3 比较两种写入策略。

恢复会让就绪会话等待高带宽内存。恢复层会在会话运行前将其键值移回高带宽内存（图 5a）。对于停放 KV，这项成本很低：恢复一个恢复运行的会话所需的 7.5GB 数据，从 Gen5 硬盘读取仅需 0.53 秒，从接口速率为 1.5 TB/s 的 HBF 读取仅需 5 毫秒；最多只占运行时间的 1.7%。在每代 32 到 256 个会话的情况下，主机动态随机存取存储器、Gen5 硬盘和 HBF 的完成时间彼此相差不超过 2.3%（表 1）。因此，硬盘的交换性能不逊于 HBF，但仅靠交换并不能消除 RSI 等待高带宽内存的问题。首先，只有部分键值会被停放：负载下，就绪会话持有其中 95–99%（§3.1），而交换必须将所有就绪 KV 保留在高带宽内存中。其次，会话会在就绪和停放状态之间切换数十次，而且大多在不到一秒后就切换（图 6a），因此交换移出的键值几乎立刻就要重新放回高带宽内存。于是，就绪会话会排队等待高带宽内存；更快的层只能缩短每次恢复的时间，却无法缩短队列：使用 HBF 或硬盘时，在 32 个会话的情况下，调用仍需等待 11–12 秒才能获得高带宽内存空间，在 128 个会话时则为 95–97 秒。按访问分层可将热数据集的解码读取保留在高带宽内存中，[3]；但在这里，仅就绪KV这一热数据集就已超过高带宽内存容量。

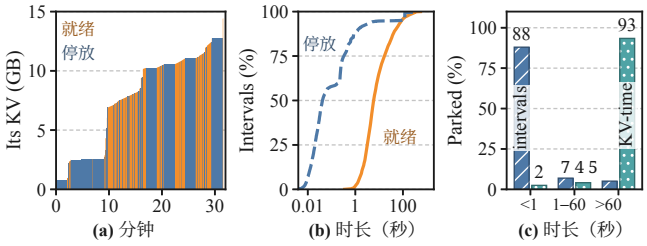
无界原地读取会延长每个步骤。原地读取（图 5b）时，高带宽内存之外的键值会在其会话运行的步骤中加入该步骤，因此没有会话需要等待腾出空间，但远端内存的带宽也会影响该步骤。读取 1 GB 键值，若从高带宽内存读取，分页解码注意力可达到 2.8 TB/s，耗时 0.36 毫秒；若从一个 HBF站点读取，速度为 467 GB/s，耗时 2.1 毫秒；经由 PCIe 从主机动态随机存取存储器读取，耗时 15.6 毫秒；从 NVMe 驱动器读取则耗时 71 毫秒。一个 HBF站点的容量相当于 22× 个 H200 高带宽内存堆栈。高带宽内存满时进行原地读取，主机动态随机存取存储器耗时为 NVMe 暂存的 7–25× 倍，而驱动器则为 32–114× 倍，甚至比仅使用高带宽内存还要久（表 1）。高带宽内存和远端内存使用独立通道，因此图形处理器会并发读取这两个层级；当一个步骤从远端内存读取的数据不超过 B_F 时，带宽为 $B_F\,t_H$ 的远端内存不会增加耗时，t_H 即该步骤从高带宽内存读取所需的时间：一个耗时 49 毫秒、处理 99 GB 键值和 34 GB 权重的步骤，可隐藏从一个 HBF站点读取的 20 GB 数据，或以 202 GB/s 读取的 8.7 GB 数据；此后每多读取 1 GB，都会延长该步骤中每个请求的耗时。因此，高带宽内存满时从 HBF 读取溢出数据，在 32 和 64 个会话时分别会使耗时增加 30% 和 17%，只有从 128 个会话开始才有收益（表1）。会话结束释放高带宽内存后，闪存读取只会浪费时间和能耗。

> **要点 2。**如果封装内原地读取能隐藏在高带宽内存读取之后，并在高带宽内存释放后立即停止，就能结束对高带宽内存的等待。



(a)交换：先复制到高带宽内存，再计算　(b) 原位读取：在该步骤中读取远端层

图5：超出高带宽内存容量的键值数据到达图形处理器的方式：(a) 在会话运行前交换到高带宽内存，或 (b) 在原位置读取。



(a) 分钟　(b) 时长（秒）　(c) 时长（秒）

图6：(a) 单个会话的键值数据。(b) 就绪和停放区间的长度。(c) 按长度划分的停放区间和键值数据时间。

3.3 按状态或计时器进行分层会产生写入成本

只有对于读取频率远高于写入频率的数据，闪存才有价值 [28]。堆栈的寿命会随着主机写入和垃圾回收复制而缩短（§2.2），而两种常见策略都会增加这两者：按会话状态分层，以及在固定时间内保持数据。

按状态分层会在每次切换时移动键值数据。 停放和就绪的键值数据是同一批块，只是所处时刻不同。一个真实的 DGM 会话要运行 39 轮，因此状态切换约 80 次，而且大多间隔很短（图 6）：88% 的停放区间会在一秒内结束，而超过一分钟的区间仅占 5%，却主要由测试运行构成，并占停放键值数据时间的 93%。因此，按状态选择的层会在放置块后不久就将其移动。将每个停放的块都降级到 HBF 以下的 NVMe 驱动器并不能节省时间，因为大多数降级的块会在一秒内返回（运行时间反而延长 0.4–3.4%）。这会使 HBF 承受 13–16× 次写入；每日驱动器写入次数为 170–280，而不是 11–21（每日整盘写入次数）。仅用于停放数据的 HBF 在每代 32 个会话时会与驱动器一同耗尽，同时承受 18 次每日整盘写入次数；而 7.5× 更大容量的驱动器则为 2.4。读取次数远多于切换次数：每当调度某个会话时，该会话的块都会在每个步骤中被读取。将停放的键值数据取出并放回的层，会在每轮中写入每个块约一次；而跨切换仍保持有效的副本只需写入一次。

定时数据保持会保留已失效的键值。 智能体服务器无法知道哪条响应是某个会话的最后一条：在 TraceLab 中，用户在间隔超过一小时后才回复的情况占 3.4%。因此，服务器会在固定时长内保留键值（图 1）；加上托管提示缓存的 5 分钟数据保持时间，为用户保留的键值中有 13% 属于已结束的会话。固定时长同样不适用于 RSI：5 分钟的时限会使等待长时间工具调用的会话过期，导致额外重新计算 9–17% 的预填充令牌；而 30 分钟的时限在每代 128 个会话时会使多达 1.5 TB 的数据保持活动状态。DGM 会话的键值中位数为 9.9 GB，并会在编排器宣布会话结束时失效。若闪存控制器未收到通知，那么每次垃圾回收涉及这些已失效但仍有效的键值时，都会将其复制；每代 128 个会话时，写入放大系数达到 1.13–1.90。

> **要点 3。** 读取次数远多于状态切换，且失效会收到通知；按状态或计时器进行分层会浪费写入。

3.4 设计要求

这三项成本源于同一个原因：每种方法都让交换机在就绪与停放之间的切换决定了超出其实际职责的事项。将 KV 所在位置的决定权交给交换机；无界的原地读取将步骤读取的内容绑定于交换机；按状态分层则将闪存副本的生命周期绑定于交换机，并根据空闲状态推断块的结束。我们的关键洞见是：交换机只决定是否读取某个块，因此其闪存副本保持有效（**D1**）；空闲 HBM 和步骤的带宽决定从何处读取该块，因此步骤会在有界范围内原地读取就绪 KV（**D2**），并且随着会话结束，KV 会返回 HBM（**D3**）；而会话的结束时间由编排器宣布，因此决定其副本何时失效（**D4**）。将这些决定彼此分离后，就能让就绪会话无需等待 HBM 即可运行，使每个步骤的闪存读取保持在有界范围内，并且每个块只写入一次，在其失效时将其回收。

4 Mayfly 设计

`Mayfly` 在主机上增加服务运行时，在图形处理器上增加双路径注意力内核，并在 HBF 基础裸片上增加 KV 控制器（图 7），通过围绕每块一份后备副本的三种机制满足 §3.4 的设计要求，并使该副本在状态切换期间保持有效（**D1**）。1 迁移或读取。在每个步骤开始前，运行时中的门控会决定哪些就绪请求运行，以及每个请求持有的、超出 HBM 范围的 KV 是被恢复，还是在原位读取（**D2**）；随着会话结束，KV 会返回 HBM（**D3**）。2 在一个步骤内读取两个层。内核接收基础裸片上以条带方式读取的数据，计算请求在两个层中的 KV 的精确注意力。3 写入控制。运行时仅写入没有副本的活动块；当编排器宣布会话结束时，控制器会使该会话的 KV 失效（**D4**）。既有 HBF 层级大多根据热度、上下文长度、KV 类型或年龄来决定这些事项，或在分配 KV 时将其固定 [63]；`Mayfly` 会在决定这些事项的 RSI 事件处重新作出决策（§3.4）：状态切换会保留块的闪存副本，空闲 HBM 和步骤的带宽决定从何处读取就绪 KV，而已宣布的会话结束则使其 KV 退役。`Mayfly` 在高带宽内存堆栈旁增加 HBF 节点，同时保持 GPU 核心、高带宽内存裸片和 NAND 裸片不变；编排器保留其控制流，并增加两条通知，分别用于其已结束会话和代。表 2 比较了既有 HBF 设计：没有任何一种设计完整满足超过一项要求，也没有任何一种设计能在会话结束时将 KV 返回 HBM。

4.1 一个备份副本，两种状态

键值数据存放在 vLLM 的 16 令牌块中，每块 4 MiB，`Mayfly` 分别追踪三件事（图 8a）：会话状态是就绪还是停放直至结束，此状态仅由运行时持有；每个块所在位置，是在高带宽内存中还是在 HBF 上；以及该块在 HBF 上是否有有效备份副本。一个块在填满 16 个词元后便不可更改，且只有完整的块才会离开高带宽内存；请求仍在填充的块会留在其中。副本遵循一条规则：块首次离开高带宽内存时写入副本，并且在该块所属的最后一个会话结束前，副本始终有效。将块恢复到高带宽内存时会保留副本，再次将其逐出时则不会写入任何内容。因此，状态切换本身不会移动任何键值数据。当会话调用工具时，它会转为停放状态，其块仅在高带宽内存承压时才会离开；

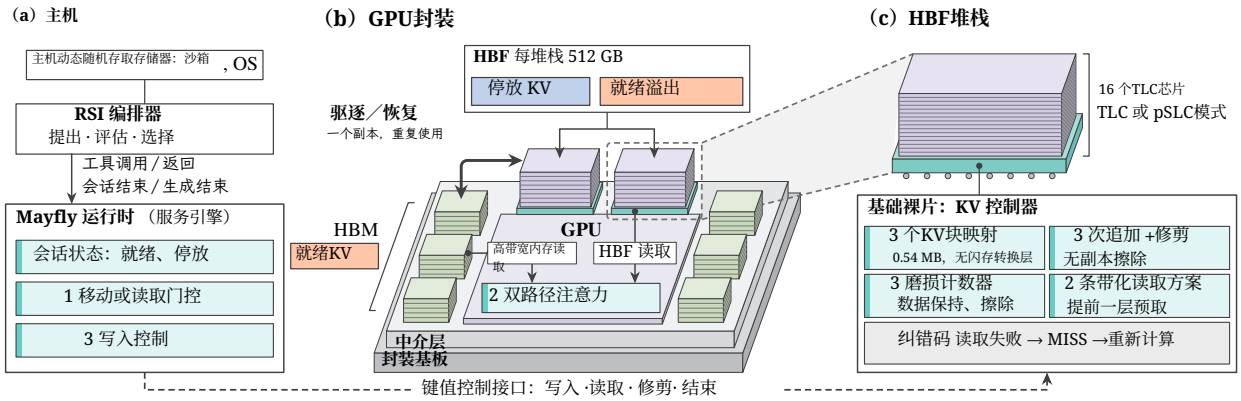


图7：Mayfly架构：(a) 主机，(b) GPU封装，(c) HBF堆栈。青绿色：Mayfly新增内容；1–3：三种机制。一个堆栈在TLC模式下可容纳512GB，在pSLC模式下则可容纳其三分之一，但编程/擦除周期数是前者的四倍（30K 对 7.5K）。

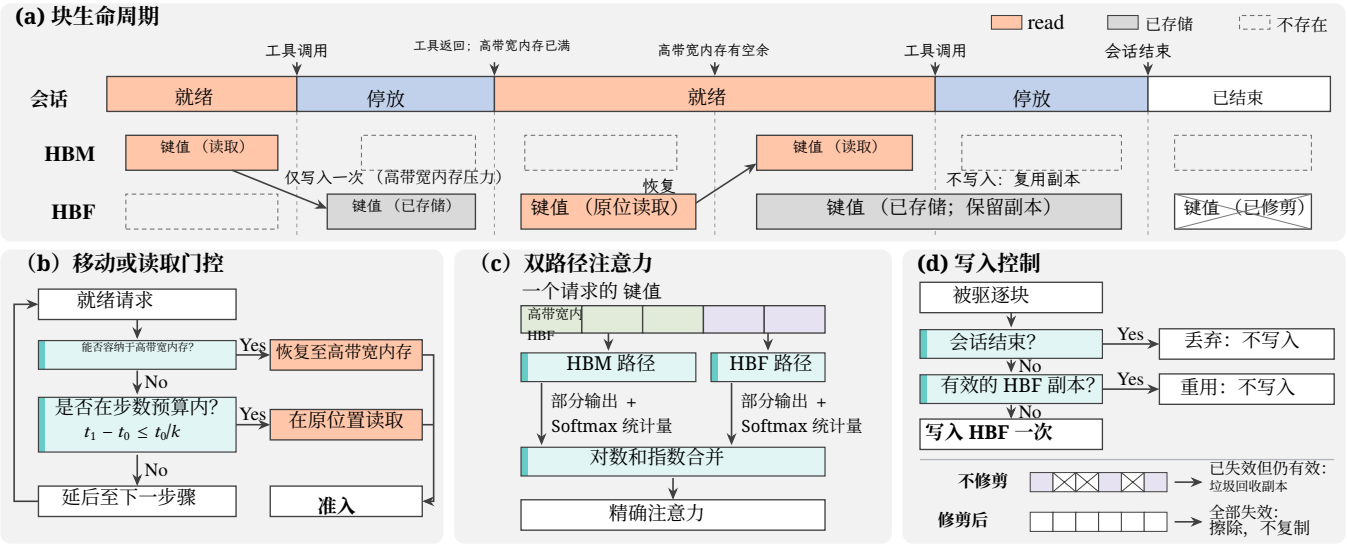


图 8：Mayfly 的运行时表现：(a) 跨越不同会话状态的完整 KV 块；(b) 每一步中针对每个就绪请求的门控；(c) 单步中对高带宽内存和 HBF 的注意力；(d) 有无修剪时的驱逐写入和擦除单元。

表 2：先前的 HBF 设计与设计要求的对比。✓ 是，部分符合，✗ 否或未说明，– 不适用。

设计	In 位置	精确 注意力	D1 Copy kept	D2 Step 有界	D3 返回 HBM	D4 释放 末尾
Baek 等人 [3]	✗	✓	✓	–	–	✗
SETA [43]	✓	✓	✗	✗	✗	✗
FlashAccel [58]	✓	✓	✓	✗	✗	✗
HBFlex [69]	✓	✓	–	✗	–	◐
Yu 等人 [65]	✓	✓	–	✗	✗	✗
SPLASH [20]	✓	✗	–	✓	✗	◐
Lachesis [63]	✓	✓	–	✗	✗	✓
Mayfly	✓	✓	✓	✓	✓	✓

工具返回时，会话转为就绪状态，门控会恢复其块，或让图形处理器直接读取块所在位置（§4.2）。高带宽内存按以下顺序保留内容，活动会话共享的任务前缀、就绪KV 和停放 KV，按各自上下文的尾部优先移出。停放 KV 保留在 HBF 上：降级到驱动器的块很快又会写入 HBF，而且这样一层可为每个图形处理器腾出一个驱动器，并省去第二次放置决策。

信号与命令。 运行时需要从循环中获取三个信号。工具调用和工具返回，服务引擎已经将其视为一个请求结束、下一个请求到达，它们标记会话处于停放或就绪状态；由于就绪KV会在其所在位置读取，因此无法预测工具何时返回。编排器会添加另外两个信号：会话_代结束 会修剪会话的私有块，而代_代结束会在某一代的所有会话结束后发送，修剪它们共享的前缀，并备份丢失的会话结束位置。运行时通过四条命令驱动控制器（图 7）： 写入(块, 会话, 代, 类别)，其中类别为共享 时表示任务前缀，为会话 时表示私有键值；读取(块)，返回数据或

MISS；修剪(会话)；以及代结束。这些信号只影响性能和磨损：丢失的会话_代结束会使无效键值留在闪存中，直到其代结束或保持目标到期。

4.2 迁移或读取：逐步门控

每个步骤开始前，门控会选择要运行的请求，并确定读取每个请求键值的位置（图 8b）。它从 §3.2 的有界条件出发：若 HBF 读取能在该步骤的高带宽内存读取完成前完成，就不会增加耗时。它会估算请求集合 A 的一个步骤所需时间： $T(A)$ $=$ $c_0 +$ t_H t_F c_0 max(,)。其中，涵盖调度、启动和采样；$t_H = t_W + K_H/B_H$ 以高带宽内存的带宽 B_H 读取权重（位于 t_W）以及该集合在高带宽内存中的键值（K_H）； $t_F = K_F/B_F + K_W/B_W$ 则读取该集合在 HBF 上的键值（ K_F），并写入离开高带宽内存的键值（ K_W）。高带宽内存路径与 HBF 路径并行运行，因此取两者的最大值；而 HBF 的读取和写入共用其半双工接口，因此耗时相加。B_F 和 B_W 分别对封装中的 HBF 堆栈求和；运行时会将每个会话的块均匀分布到这些堆栈上。门控按照服务引擎的顺序构建该步骤的请求集合 A，其中有 $k = |A|$：先按运行中请求的先后顺序排列（最早的在前），再按到达顺序排列等待中的请求。它会直接接纳第一个请求；如果某个请求的键值与 A 的键值合并后能放入高带宽内存，也会接纳该请求。其他请求 r 会选择原地读取；门控只有在这不会降低该集合的估算令牌吞吐量时才接纳它。由于每个请求每步解码一个令牌，判定条件为 $(k+1)/t_1 \geq k/t_0$，其中 $t_0 = T(A)$ 和 $t_1 = T(A \cup \{r\})$，即 $t_1 - t_0 \leq t_0/k$。该判定比有界条件更宽松：如果新增耗时由该请求自身的令牌承担， r可能会比该步骤的高带宽内存读取持续更久。未被接纳的请求会跳过该步骤、保留原有位置，并将其键值留在 HBF 上；只有在没有运行中请求被跳过时，等待中的请求才会获准加入，因此新会话不会越过被跳过的请求。该判定保护的是吞吐量，而非任何单个请求的延迟。

返回高带宽内存。KV 在读取它的步骤中返回高带宽内存。每个步骤结束后，运行时会按步骤顺序重新分配高带宽内存：先分配该步骤请求的 KV，且最早的请求优先，停放 KV 则让出空间。某个块若从 HBF 读取后有了空间，就会留在高带宽内存中，只需写入一次高带宽内存，无需额外读取 HBF；在该步骤结束后、且该块的每一层都已读取完毕时，块表会切换到高带宽内存中的副本，而 HBF 中的副本仍然有效。若恢复的会话所需 KV 能容纳下，就会在其第一个步骤中以这种方式恢复。某一代中的会话会逐个结束并释放高带宽内存，而最慢的会话仍会继续解码，并决定其运行时间（§3.1），因此每个步骤都会使就绪 KV 超出高带宽内存的部分缩小；一旦全部就绪 KV 都能容纳，所有请求都无需读取 HBF 即可运行，HBF 便会闲置。

4.3 在一个步骤中读取两个层级

Mayfly与 SPLASH [20], HBFlex [69] 和 FlashAccel [58]一样，假设图形处理器访问 HBF 堆栈的方式与访问其旁边的高带宽内存堆栈相同（图 7b）：堆栈中的块映射到图形处理器的物理地址空间，注意力内核像加载高带宽内存块一样加载这些块。vLLM 的块表中的条目可以指向任一层，内核会按层拆分请求的键值（图 8c），就像 Flash-Decoding 将长上下文拆分到多个线程块中一样 [6]：一组线程块对高带宽内存块执行注意力计算，另一组对 HBF 块执行注意力计算；每组都会生成部分输出及其 Softmax 最大值和总和，再通过对数和指数归约将两者合并，从而对整个上下文进行精确注意力计算，不会丢弃任何键值，以节省读取开销。两组并行运行，因此 T 耗时取决于较慢的路径，而合并操作会在每层增加一次归约。

条带化读取方案。在步骤开始前，运行时会向控制器发送该步骤的读取命令：每个请求在 HBF 中持有的块对应一条命令。这样，每个地址在需要之前都已知晓，无需进行任何预测，这与 HBF 上的 MoE 专家预取不同 [11]。控制器会将每个块的页条带化分布到堆叠中的各个裸片和平面上，使某一层所需的页能让每个平面都保持繁忙；同时，当图形处理器运行第 ℓ 层时，控制器会发出第 ℓ+1 层的 NAND 读取。容量为传输延迟与带宽乘积两倍的缓冲区，可以掩盖数据从裸片的页寄存器传输到图形处理器的耗时，因此，读取速度受阵列每次页读取返回的字节数限制，而非单次读取的延迟： D 个裸片、每个裸片有 P 个平面，每次读取可返回 DP 个 4 KiB 页 [53]，每次读取耗时为 1.68 TB/s；在 20 μs 下，16 个裸片、每个裸片 512 个平面可达到这一速度，3.6×，满足一个站点的需求。额定接口意味着这种并行度是可行的：若页读取耗时为 20 μs，16 个裸片只有在每个裸片约有 460 个 4 KiB 平面时，才能达到 1.5TB/s 级别。

4.4 写入控制

传统层或牺牲缓存会将从高带宽内存驱逐的每个块都写入。 Mayfly 在每次驱逐时按以下顺序做出决定（图 8d）： 不再被任何活动会话引用的块将被丢弃，此类块首先离开高带宽内存；拥有有效备份副本的块在高带宽内存中释放，无需写入；只有没有副本的活动块才会被写入，且只写入一次。不同于经重用验证的准入 [27], 这不需要重用证据，因为会话运行时会读取活动会话的键值。随后由死亡而非垃圾回收释放闪存。在 会话_结束时，运行时会修剪该会话的私有块；共享前缀最退会在其最后一个用户于 代_结束时死亡。修剪只会将块标记为死亡：垃圾回收不再复制这些块，并且当擦除单元中的所有块都已死亡后，该擦除单元会在无需复制的情况下被擦除。一个会话的块会被写入得彼此接近，也会同时死亡，因此在同一条追加流中，大多数擦除单元会在被回收前就变空，无需进行逐会话放置。为应对丢失的会话结束信号，控制器会按每一代和类别保留打开的擦除单元，并由 代结束 将其整体擦除，包括共享前缀。

4.5 KV 原生 HBF 控制器

传统闪存控制器处理的 4 KiB 写入必须在掉电后保留，并且可能在数年后才被读取。因此，对于 512 GB 堆叠，其页映射表需要约 500MB 的日志式 DRAM；它只有在主机覆盖或丢弃数据时，才会得知数据已失效。RSI KV 以大块写入，可通过预填充重新生成，会主动声明数据已失效，且会在数小时内被读取，因此 Mayfly 控制器利用图 7c. 所示的所有四项特性。

映射与分配。控制器映射整个 KV块，而非单个页，并将它们追加到开放的擦除单元中。擦除单元即超级块，是条带一起擦除的 NAND 块，大小为 64 MB 至 1 GB。一个堆叠中的 122K 个块，以及用于记录最小单元的表，仅占用 0.54 MB 的 SRAM，不到 121–176 mm^2 基础裸片面积的 0.1%；9.6 MB 的读取缓冲区则占 1.1–1.6%，而此前的 HBF 设计会配置 32–40 MB 的缓冲区 SRAM [58, 69]。KV 可以重新计算，因此控制器不保留日志；掉电后，它会报告所有块均已丢失，并按需重新计算。

数据保持。 NAND 的规格规定，在寿命终止时可保持数据三个月至一年，而 RSI KV 会在写入后的 94 分钟内被读取。控制器记录每个擦除单元的编程时间和数据保持

目标；一旦某个单元的使用时间超过其目标期限，其块读取结果为MISS，并重新计算，而不会刷新，因为刷新那些可能永远不会被读取的 KV 会增加写入次数。低于规格规定的 24 小时通电保持时间的目标 [53] 可以用数据保持换取耐久性。

缺失块与纠错。 当一个块的存储单元超过其保持目标、某个码字未通过其 CRC，或发生掉电后，该块会被读为 MISS。内核会标记该块的请求，而步骤中的其他请求则提交其词元。运行时会保留每个块的词元，并通过在其之前的 KV 上预填充该块的 16 个词元来将其重新计算到高带宽内存中，较早缺失的块优先处理；随后修剪失败的副本，并使用精确注意力重新运行该请求的步骤。因此，若某个缺失块之前的块均已存在，该缺失会为其所属请求额外增加一次该块的预填充和一个步骤。任何 HBF 基础裸片都需要纠错 [53], 而此前的 HBF 研究既未计入任何成本 [13, 28]或将其省略 [58, 69]。由于错误数量超过纠错能力的码字会导致重新计算且不会产生数据，因此目标应由 KV 的读取量，而非存储级不可纠正误码率决定。随着存储单元磨损，越来越多的块会被读为 MISS，磨损的单元也会退役，因此该层损失的是容量，而不是数据。

5 评估

5.1 实验设置

工作负载。 我们采用三个测试框架-RSI系统。DGM [67]在 SWE-bench 上演化一个编码代理；我们的运行由 gpt-oss-120b 提供服务， [40],包含 5 代、147 个会话和 7,008 个回合；另一次由 GPT-5 nano 提供服务的运行 [39] 包含 197 个会话和 6,814 个回合。RRSI [60] 在基线和三轮迭代的基础上，针对 EngDesign 改进一个代理测试框架；我们的运行由 gpt-oss-120b 提供服务，包含 4,807 个会话和 24,740 个回合。Harness² [61],是一个独立的框架，用于改进 OpenCode 智能体的测试框架；我们在其 25 个 LAB任务上的运行由 H200 上的 Qwen3-32B 提供服务，包含 664 个会话和 8,584 次调用。每次重放都保留每个会话的回合、上下文和工具时间，并使用 FP8 权重在 Qwen3-32B 上运行。我们根据首次 DGM 运行的评估会话构建图3a 所示的循环：在三个代中的每一代中， V 变体都尝试相同的八个任务， $8V$会话同时开始，并在最慢的会话结束后进入下一代；额外尝试会使用自身的键值重复一次已记录的尝试。 V=4，每代会话数为 8、16、32、32、256。记录的运行包含十个有多次尝试的任务；除非标记为三次或十次样本的平均值，否则结果使用其中八个任务的一次样本，速度提升均为运行时间之比。离散事件模拟器

重新实现了 vLLM 的块池、调度器和抢占机制，并根据 H200 重放数据拟合步骤时间和能耗，其中一次重放将 KV 溢出到主机动态随机存取存储器（误差在 0.5% 以内）；在一个留出的重放测试中，使用 20 GB 的 KV 池时，其运行时间与实测值相差不超过 0.9%，能耗相差不超过 1.6%，抢占次数相差不超过 5%。读取 HBF 的步骤耗时为门控的 T（§4.2），其计算时间取最大值，并将 c_0 在 H200 重放数据上拟合为 6.0 ms。我们分部分检查 HBF 路径：一个基于主机内存的 H200 双路径内核、一个周期近似的读取流水线，以及一个错误估计带宽的门控。

硬件配置。 图形处理器为 H200，其六个高带宽内存位点共提供 141 GB，其中 99 GB 用于键值。HBF 位于新增的封装位点上，与这些位点并排；每 32 个会话配备一个 512 GB 的三层单元堆栈，低于 OCP 规范目标的 550 GB。一个位点的读取速度为 467 GB/s，即 H200 高带宽内存位点处理分页解码注意力时的速度，低于 OCP 规定的 1.5 和 3.1 TB/s [53], , 其写入速度为

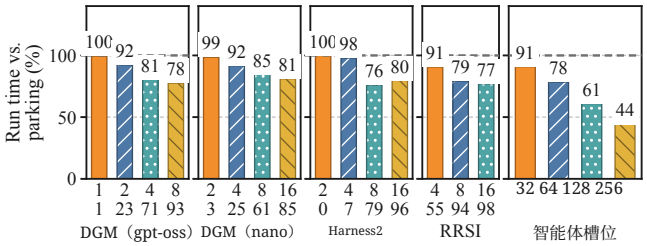


图9：Mayfly 的运行时间占 HBF-Park（每个 H200 的完整运行次数）和 NVMe-Park（智能体槽位）的百分比。标签：每个 H200 的运行次数，以及其下方带有就绪 KV 的时间占比（相对于高带宽内存，%）。

其十分之一 [43, 69]；我们还采用 202 GB/s，这是堆栈在热限制下的速度 [28]。该规范未规定能耗和阵列时序，因此我们分别采用读取和写入时每比特 20 和 100 pJ，并将页读取的 20 μs 取自先前估算的高端值 [13, 28, 58, 65]。每次层写入和修剪都会通过采用贪心垃圾回收的闪存模型进行重放，我们报告磨损情况：假设磨损均匀分布，每个块每天消耗其编程/擦除周期预算的每日整盘写入次数 × WAF；编程/擦除周期预算依据 3D TLC 测定 [5, 23]。

基线。 每个基线都保留六个 HBM 位点，并将 HBM 之外的键值（KV）放置在一种远端内存上：HBF；通过 PCIe 连接的主机动态随机存取存储器，以 25 GB/s 作为停放层、以 64 GB/s 原位读取（25 pJ/bit）；通过 NVLink-C2C 连接的 Grace LPDDR5X（480 GB、450 GB/s、8 pJ/bit）；或第五代NVMe驱动器（3.84 TB，读取速度 14 GB/s、写入速度 10 GB/s，分别为 300 和 600 pJ/bit）。在驱动器或 HBF 上停放（NVMe-Park、HBF-Park）时，只有当就绪会话的 KV 能够容纳在 HBM 中并已恢复到其中后，才会接纳该会话。HBF-Park 采用 Mayfly 的 HBF 容量和写入控制，在会话恢复期间保持块的闪存副本有效，并以 OCP接口的 1.5 TB/s 速度进行恢复。在没有读取上限的情况下，HBF-All 在每个步骤都从 HBF 读取所有就绪 KV，而 HBF-Greedy 会在 HBM 已满时原位读取就绪 KV。所有 HBF 设计都运行在相同的新增位点上，因此相对于 HBF 基线的所有增益都建立在相同的封装资源之上。

5.2 整体性能

Mayfly的优势随就绪KV溢出而显现，并随着负载增加而扩大。**完整运行。**我们重放完整运行， R每个 H200 在两处 HBF 站点上启动一次，启动间隔为 900 秒（图 9）：图 3中的 DGM 运行、对应的 GPT-5 nano 运行、RRSI 运行以及测试框架² 运行。一次 DGM 运行最多保持 10 个会话处于打开状态，因此每个 H200 运行一次时，其就绪键值超出高带宽内存的时间占 1.4%。就绪键值保持在高带宽内存容量以内时，所有设计的完成时间相差不超过 3.1%。当就绪键值大部分时间都超出高带宽内存容量时，Mayfly 在每个 H200 上运行 4 次和 8 次 DGM 时，比 HBF-Park 分别快 19% 和 22%；运行 8 次和 16 次测试框架² 时，分别快 24% 和 20%。GPT-5 nano 运行的会话较短，只有在运行次数翻倍时才会像其他运行那样频繁地溢出高带宽内存；在运行 8 次和 16 次时，分别快 15% 和 19%。RRSI 在运行 4 次、8 次和 16 次时，分别快 9%、21% 和 23%。仅使用高带宽内存时，耗时最多可达其 5倍。

负载上升。 在 DGM 循环中，就地读取就绪溢出比将其停放更快完成，而且负载越高，收益越大。使用 HBF 进行停放时，键值位于高带宽内存之外的就绪会话需要等待高带宽内存腾出空间。一次调用平均等待 11–203 秒，且有 71–735 次调用被抢占（表 3）。就地读取时，等待时间降至 6.6–75 秒，且没有调用被抢占。 Mayfly 分别提前 8、16、34

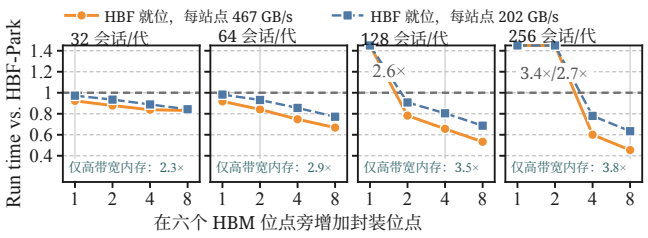


图 10：随着添加 HBF 站点，HBF-Park 的运行时间变化；每个站点的带宽为 467 或 202GB/s。竖线：每 32 个会话配置一个站点；顶部标签：超出坐标范围的数据点。

表 3：按每代会话数划分的性能提升来源。写入TB数：写入 HBF 或 NVMe-Park 驱动器的数据量。粗体：最佳值。

	HBF-Park				NVMe-Park				Mayfly			
会话/代	32	64	128	256	32	64	128	256	32	64		
运行时间（分钟）	172	279	482	919	174	282	490	940	**159**			
每次调用等待时间（秒）	11	37	95	203	12	38	97	208	**6.6**	**22**	**47**	**75**
中位数首词元时间（秒）	7.8	34	89	187	9.3	37	94	188	**2.8**	**22**	**46**	**73**
抢占次数	71	170	375	735	63	155	338	706	**0**	**0**	**0**	**0**
每个 HBF站点的 GB/s	–				–				123			
写入TB数	**1.09**	2.28	**4.70**	**9.56**	**1.09**	**2.27**	**4.70**	**9.56**	1.12			
能耗（MJ）	4.1	7.3	14.0	27.4	4.2	7.6	14.6	28.7	**3.8**	**6.8**		

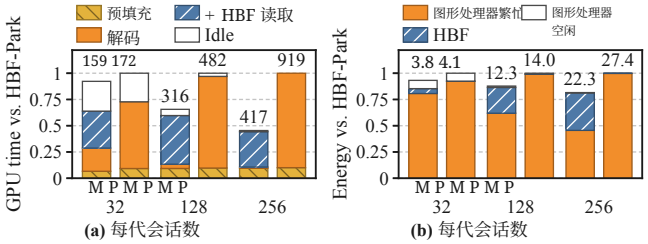


图11：Mayfly（M）和 HBF-Park（P）的 (a) GPU时间和 (b) 能耗分别用于何处，每种负载均按 HBF-Park 的总量进行缩放；数字：总计时间（分钟）和能耗（兆焦耳）。

和 55%（图10）；在十个任务样本中，收益平均为 8.0%、17.5%、36.3% 和 53.9%，95% 置信区间为 ±1.5、1.8、0.6 和 0.3 个百分点。节省的是解码时间（图11）：在 128 个会话时，Mayfly 解码耗时 243 分钟，其中 223 分钟使用 HBF 读取；相比之下，HBF-Park 耗时 421 分钟。在这两种设计中，准入排队均占平均首 token 时间的 97–100%。Mayfly 消耗 0.19 至 9.8 MJ 的 HBF 能耗，以节省 0.46 至 14.8 MJ 的图形处理器能耗；与 NVMe-Park 相比，能耗降低 9–22%。只有当 HBF 读取的成本为 33–61 pJ/bit，而不是我们建模采用的 20 pJ/bit 时，才会达到能耗平衡。计入 HBF 待机功耗后，Mayfly 在任何数值下都比 HBF-Park 能耗更低；当 HBF 待机功耗不超过 55–82 mW/GB（每堆栈 28–42 W）时，也比 NVMe-Park 能耗更低。用 NVMe 驱动器而非 HBF 进行停放，结果变化不大（表 1），因此 Mayfly 比 NVMe-Park 分别提前 9%、17%、35% 和 56% 完成。新增站点必须能够容纳其所服务的溢出（图 10）。一个 HBF 站点无法容纳 128 个会话的就绪溢出，耗时是 HBF-Park 的 2。6× 倍，因此 Mayfly 为每 32 个会话配置一个站点。在每个站点 202 GB/s 的热限制下 [28], Mayfly 仍比 HBF-Park 分别提前 1%、7%、19% 和 36% 完成。

表 4：Mayfly 移除一种机制后，与Mayfly（每块写入一次，写入放大系数 1.00）相比，按每代会话数统计。[a]128 / 256 个会话，单级单元站点。[b]没有会话结束；Mayfly（§5.4）的 LRU 变体，与启用这些机制的自身版本相比。

	运行时间（%）				写入与磨损			
会话/代	32	64	128	256	32	64	128	256
无界（HBF-Greedy）	+41	+39	+29	+13	–			
永不返回高带宽内存	+61	+60	+50	+28	–			
既不绑定也不返回	+55	+43	+30	+12	–			
返回时丢弃副本	+0.5–6				每块写入次数为 9–37 次			
5 分钟到期时间[b]	+0.2–2.5				+5–24% 键值已写入			
仅在满载时释放[b]	–				WAF[a]		2.1	2.8
不修剪，2%备用空间	–				WAF	1.11	1.42	1.57 1.77
保守门控预算	在0.5以内				–			

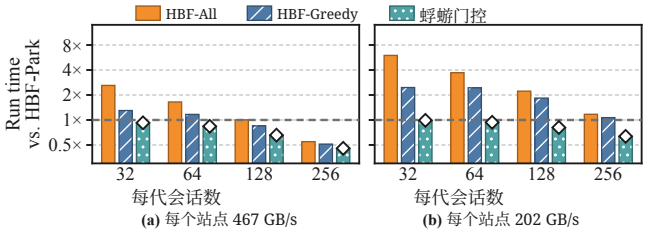


图12：在每个站点带宽为（a）467 GB/s 和（b）202 GB/s 时，HBF-Park 上按门控策略划分的运行时间。菱形：保守预算。

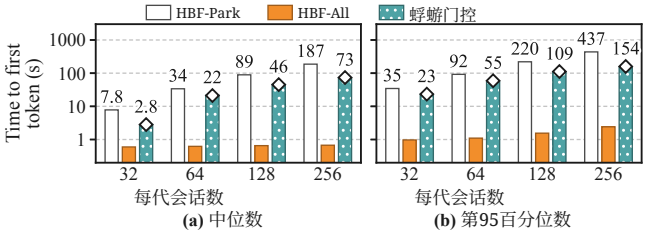


图13：不同门控策略下的首 token 时间：（a）中位数，（b）第95百分位数。菱形：保守预算。

5.3 消融研究

执行路径的两个部分都能缩短运行时间，且只有二者结合时（表4）效果才最好。若不返回高带宽内存，有界方案反而比完全无界更慢，并且每个此类变体都会读取1.5–4。7× 个HBF字节数。该收益对门控的具体规则具有稳健性（图12）：保守预算的完成时间与其相差不超过0.5%，而HBF-Greedy需要1.7–2。6× 倍的时间，在202GB/s带宽下。会话数不超过128时，HBF-All比HBF-Park耗时更长；在256个会话时，其耗时为0。55× 倍，与HBF-Park相同，而45× 对于Mayfly而言。与HBF-Greedy相比，在32至256个会话下，该有界方案分别节省29%、28%、23%和11%的运行时间；在十个任务样本下则分别节省35%、35%、27%和11%：带宽越稀缺，其作用越明显。

首令牌时间。 与无界原地读取相比，该上限以首令牌时间换取运行时间（图 12a 和 13）。Mayfly仅当新调用的原地读取不会降低该步骤的令牌吞吐量时才予以接纳，其后的调用则按顺序等待。在所有负载下，它仍比 HBF-Park 更快完成，且第 95 百分位首令牌时间更低：128 个会话时为 109 秒，对比 220 秒；而在 256 个会话时为 154 秒，对比

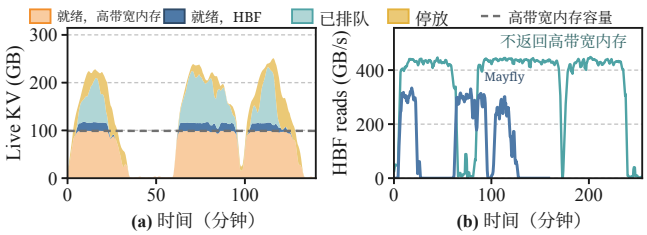


图 14：每代 32 个会话时的突增与下降：(a) Mayfly 按状态和位置划分的实时 KV；(b) 返回高带宽内存与不返回时的 HBF 读取。

437 秒。对于最多 128 个会话的情况，不设读取上限的设计（HBF-All、HBF-Greedy 和每个块仅放置一次，§5.8）能将第 95 百分位首令牌时间保持在 2 秒以内，但完成时间晚 1.1–2。8×，因为读取会延长每个步骤。在 256 个会话时，HBF-All 的第 95 百分位首令牌时间为 2.4 秒，完成时间晚 1。2×。运行时间本身也是一种尾部度量，因为一代要等最慢的会话结束才算完成；按会话计算，Mayfly的第 99 百分位持续时间比 HBF-Park 低 4–54%，比 HBF-Greedy 低 14–30%。与 HBF-Park 相比，97–100% 的会话完成时间不晚于它。门控始终接纳最早的运行中请求，因此不会有请求挨饿：没有任何运行中调用被总共延迟超过 93 秒。

跳过。 当门控拒绝其原地读取时，运行中的调用会跳过一个步骤。在所有调用中，14–19% 至少跳过一个步骤，这会使整个运行期间第 99 百分位调用延迟 2.3–2.9 秒；而在首个令牌生成前跳过的调用不到 1%。平均而言，每个步骤会评估该规则 1.6–8.9 次，每次评估包含两个闭式估计。若门控对 HBF 带宽的估计误差为 20% 或 40%，运行时间最多会延长 8% 或 21%。所有这类门控的完成时间都短于 HBF-Park，且第 95 百分位首令牌时间更低。

随着会话结束，将 KV 归还至 HBM。 这样一来，只有当实时 KV 超出 HBM 容量时才需要读取 HBF（图 14a）。每代 32 个会话时，Mayfly 读取 HBF 的时间占图形处理器繁忙时间的 49–62%（图 14b）；会话数为 64 和 128 时，这一比例分别为 83% 和 92%。一种从不归还 KV 的变体，在整个繁忙时间内都需要读取 HBF。它耗时为原来的 1.28–1。61×倍；在 32 个会话时，首 token 时间第 95 百分位从 23 秒升至 82 秒。时间损耗发生在每代仍有超过四分之一的会话运行期间，而非最后的收尾阶段，因为突发期间会话不断结束并释放 HBM。哪种归还方式能带来收益取决于负载：在 32 个会话时，应整体恢复停放的会话，因为仅将 KV 归还至原位置读取会使耗时从 159 分钟增至 256 分钟；在 64 和 128 个会话时，应让 KV 保留在 HBM 中并在原位置读取，因为仅恢复停放的会话会使耗时从 235 和 316 分钟分别增至 273 和 447 分钟。归还本身代价很小：在 32 、 128 和 256 个会话时，从 HBF 搬移的数据量分别为 24 、 49 和 60 TB，耗时分别增加 8、18 和 22 秒，最多仅占图形处理器繁忙时间的 0.13%。

5.4 写入放大

保留副本并在会话结束时修剪，可使 HBF 保持每块写入一次，平均写入放大系数为 1.00。Mayfly 会在块首次离开高带宽内存时将其写入 HBF，并在状态切换期间保留副本；它绝不会写入已失效的键值，因此在 32、64 和 128 个会话下，每个写入字节分别会被读取 1,040、2,520 和 4,380 次；相比之下，HBF 仅用于停放时，每个写入字节只会被读取 23–28 次。每次返回高带宽内存时释放副本，会耗费运行时间的 0.5–6%，但会使每块被写入 9–37 次。如果没有结束信号修剪，在预留空间为 2%、擦除单元为 1 GB 时，垃圾回收副本占物理写入量的 10–44%。写入放大系数随负载增加而上升，随预留空间增大而下降；预留空间为 7%、2% 和 1% 时，写入放大系数最高分别达到 1.16、1.77 和 2.32；无论在会话结束还是代结束时修剪，都能使其降至 1.00。在 HBF-Park 上，只有当放置依据会话生命周期时，放置策略才有帮助。每代一个单元可将写入放大系数降至 1.00，而按键值类型划分的流 [69] 在 128 个会话时仅将写入放大系数从 1.60 降至 1.54，每会话单元 [38] 则可将其降至 1.05；64 MB 的单元可使单个流的写入放大系数保持在 1.01 以下。在停放路径上，修剪和存活性准入可将写入量减少 15–41%；而仅在块被逐出两次后才写入 [27]，会导致恢复的会话重新计算键值，使运行速度降低 5–9%，Gandalf [16]对基于历史的门控也得出了这一结论。

不等待会话结束就释放 KV。 无法感知会话结束的引擎必须根据年龄或在分层填满时释放 KV。我们在采用 LRU分层替代状态顺序和保守预算的 Mayfly 变体上，分别测量了每条此类规则。给定会话结束信号后，该变体的完成时间与 Mayfly 相差不超过 0.5%，且 WAF 保持在 1.00，因此将其作为参照；每条规则产生的重新计算开销均计入其中。没有任何固定规则能在所有负载下匹配会话结束信号。过早释放会丢失实时 KV：5 分钟后到期还会使正在等待长时间工具调用的会话到期，这些会话会多写入 5–24% 的 KV，并使运行时间延长 0.2–2.5%；在 256 个会话下，改为使块到期，在六次运行中的三次里耗时 2.2–3.4× 倍，因为排队等待高带宽内存的会话会丢失其 KV，并重新计算 6–13× 次预填充；而在 1 分钟后到期会使运行时间延长 3–13%。过晚释放会保留无效 KV：像 SPLASH 一样在 30 分钟后到期时，在 128 个会话下，它保留的 KV 是六个 SLC 站点所容纳 KV 的一半以上，因此 WAF 达到 1.7；仅在分层填满时释放 KV，则在 128 个会话下达到 2.1 的 WAF，在 256 个会话下达到 2.8。与事后针对每种负载选择的最佳到期策略相比，未经调优的会话结束策略的运行时间相差不超过 1%。

5.5 敏感性分析

除非阵列每个裸片的平面数少于 256，或页读取耗时为 $100\,\mu$s，否则读取路径可持续提供 467 GB/s 的站点带宽。我们逐层重放读取 HBF 所涉及的步骤：阵列轮次、裸片到基座传输、LDPC 和 9.6 MB 缓冲区，每个堆叠包含 16 个裸片。对于所有裸片数、平面数和页读取时间，HBFSim [13] 报告的带宽均与我们的阵列上限相同，误差在 0.2% 以内。每个裸片有 512 个平面、每页 4 KiB 时，阵列可提供 1.68 TB/s，且不会增加步骤时间。平面数为 256 时仍不会增加；而 128 和 64 个平面分别会增加 2–7% 和 50–83%，因此 202 GB/s 的热限制对应于满速运行时约 64 个平面。页读取时间不超过 $50\,\mu$s 时不会增加耗时，而在 $100\,\mu$s 时会增加 13–29 %。每层以 15–33 μs 合并两条路径，最多增加 3.6%；在每层都出现 20 或 $100\,\mu$s 的停顿，最多分别增加 1.0% 或 4.3%。在 H200 上，针对主机内存的双路径内核运行速度为较慢路径的 0.993–1.020× 倍，与全注意力相比差异不超过 2×10^{-4}。以读取速率的 0.05–0.5× 倍写入 HBF，对运行时间、能耗和每日驱动器写入次数的影响均低于 1.5%。**纠错。**一个 4 MiB 块包含 1,024 个 4 KiB 码字，因此 10^{-6} 的单码字故障率会导致 0.1 % 的块读取失败，并使运行的预填充时间增加 3.7–10×。若要将新增的预填充时间控制在 1% 以内，故障率需约为 10^{-9}。键值数据会在写入后 94 分钟内读取。在如此短的数据保持时间内，经过 2K 次擦写循环的 3D TLC 页每 KiB 最多出现 19 个比特错误 [44], 约为 2.3×10^{-3}，或平均每个码字 85 个，因此必须能够解码含有最多 146 个错误的码字。针对 4 KiB 码字的比特翻转 LDPC 译码器测得，在 190 个错误时帧错误率为 3×10^{-8}，且未出现错误平层 [30]。随着

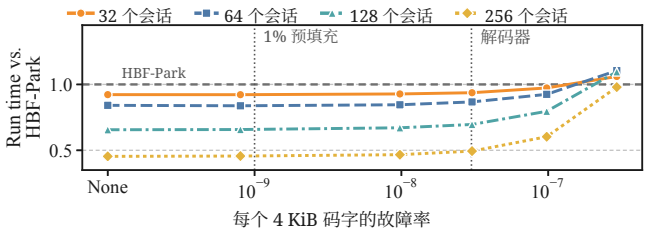


图 15：注入码字故障后的运行时间。

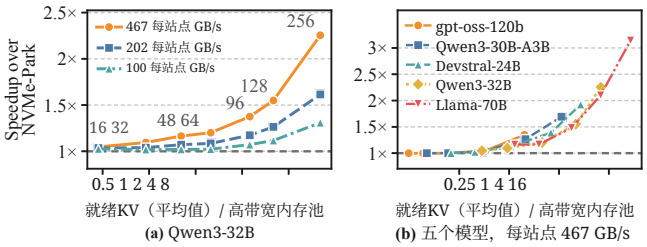


图16：就绪KV超出高带宽内存的程度所带来的相对NVMe-Park的增益：（a）Qwen3-32B在不同站点带宽下的表现（标签：会话数）；（b）在467GB/s下的五种模型，按每个令牌的KV量排序。

在每次原地读取中注入故障，并恢复 §4.5 运行（图 15）时，8×10^{-10} 运行时间最多变化 0.5%，+1% 时需要3×10^{-9} 达到 2×10^{-8}，而在 3×10^{-8} Mayfly 仍比 HBF-Park 提前 6–51% 完成。在 16 纳米工艺下，每个解码器以 19.3 Gb/s 的速率、占用 0.12 mm^2 时，解码站点的 467 GB/s 数据需要 23 mm^2, 13，占基础裸片面积的 19%；而额定的 1.5 TB/s 需要 76mm^2。

其他远端内存。 封装外内存有两个方面的不足。主机动态随机存取存储器和 NVMe 驱动器的带宽不足：在 128 个会话下原地读取，它们所需时间分别是 22 倍和 100× NVMe-Park 的时间（表 1）。Grace 内存的容量不足：其 480 GB 无法容纳 128 个会话下的就绪溢出，因此运行过程中有 71–72% 的部分需要重新计算，耗时是 4 倍。1× 在 64 个会话下，它可以容纳溢出数据，但速度仍慢 9–10%。容量决定门槛，带宽决定斜率。在固定为 450 GB/s 的带宽下，480 GB 的内存耗时是 4 倍。29× 的时间，而 Mayfly 耗时更短，1 太字节便已足够。在固定为 2 TB 的容量下，450 GB/s 的带宽最多耗时 1 倍。38× 的时间。

5.6 耐久性分析

堆栈的寿命取决于其写入控制（§5.4）。采用写入控制时，TLC 堆栈在 32 至 256 个会话下，每天可承受 20、14、11 和 8.4 次驱动器写入，写入放大系数为 1.00，因此每个块每天会消耗其 7.5K 个编程/擦除周期中的相应次数 [23]。保留副本可使堆栈的使用寿命达到每次返回高带宽内存时释放副本的 9–37× 倍。会话结束后再释放，可使其寿命达到采用 30 分钟到期时间或层已满时释放的 1.7–2。8× 倍。键值数据读取时较新，最旧不超过 94 分钟；放宽未用到的数据保持要求 [31, 33] 可根据 Luo 等人的模型额外获得 2.0–3。8× 个周期 [34]；我们的磨损数据未计入这一因素。

SLC 模式以牺牲三分之二的容量换取耐久性，增加站点则可提升带宽：在 32 至 256 个会话下，4、4、6 和 12 个 SLC 站点比三层单元快 9–13%，并且无论是停放还是就绪的键值数据都能容纳，且没有驱动器低于该值。它们每天可承受 17–24 次驱动器写入，因此每一代消耗一个块 0.6–1.7 个编程/擦除周期；相比之下，3D TLC 以伪单层单元模式运行、寿命终止保留时间为一年时为 30K 个周期 [55],若未注明数据保持时间，则为 100K [15], 原生 3D SLC 则超过 30K [22]；在 30K 个周期下，伪单层单元堆栈在持续负载下可使用 3.4–4.8 年。磨损失效不会导致数据丢失：磨损块会退出该层，其中保存的键值数据则根据其上下文重新计算，就像发生读取失败后一样。保守的工作点采用 202 GB/s 的伪单层单元站点带宽，这一带宽可由每个裸片含 128 个平面的阵列持续提供；周期数设为 30K 时，4、4、6 和 12 个站点分别用时 1.12、1.17、1.39 和 1.75 × 在 32 至 256 个会话时比 NVMe-Park 快，且中位数首词元时间降低 1.5–6.8×，能耗减少 9–15%，并且没有发生抢占。

5.7 通用性

收益取决于就绪 KV 超出高带宽内存容量的程度，而非会话数量（图 16）。每个站点带宽为 467 GB/s 时，Mayfly 的运行速度比 NVMe-Park 快 1。05× 当就绪 KV 平均占高带宽内存池的 0.47× 时，速度快于 NVMe-Park；占 2.5× 时，快 1.20×；占 6.4× 时，快 1.55×；占 12.8× 时，快 2.25×。在 100 GB/s 带宽下，就绪 KV 平均占池容量约 5× 之前，收益始终低于 10%。哪种负载会达到这一临界点取决于模型：gpt-oss-120b [40]，是一种混合专家模型，每个令牌保留 36 KiB 的 KV，因此在最多 64 个会话时，其就绪 KV 都能放入高带宽内存，此时 Mayfly 与所有基线性能持平；在 128 个会话时，Mayfly 比 NVMe-Park 快 1.05×，在 256 个会话时快 1.35×。Llama-3.3-70B 每个令牌保留 320 KiB，因此在 16 个会话时，Mayfly 快 1.16×，在 256 个会话时快 3.14×。对照相同的比例，五种模型的性能从低于池容量五分之一时的持平，提升到接近池容量时的 1.10–1.35×，以及超过池容量三倍时的 1.39–3.14×（图 16b）。低负载时，最强基线是 HBF-Park；其他情况下则是 HBF-Greedy。与其相比，Mayfly 在 Qwen3-32B 上最终快 1.08–1.29×，在 Llama-3.3-70B 上快 1.12–1.41×，在 Devstral-24B 上快 1.12–1.14×，在 Qwen3-30B-A3B 上快 1.03×（最多 128 个会话）；在 256 个会话时快 1.00–1.13×，此时八个站点的总读取速度超过高带宽内存。就绪 KV 平均占池容量不到一半时，HBF-Greedy 的完成时间仅比 Mayfly 长 0.4%。从约等于池容量开始，最多 128 个会话时，其完成时间要长 3–71%。FP8 KV 可将每个令牌的字节数减半，并可与 Mayfly 结合使用；在 128 个会话时，Mayfly 的完成速度比最佳 FP8 基线快 1.32×（三次采样的平均值）。我们仍使用 BF16 KV，因为在我们的测试中，FP8 KV 导致 530 道 MATH 题中有 41 道答案发生变化，并使 LongBench 得分从 43.4 降至 39.6。作为并发自主代理，在有 $8V$ 个忙碌槽位且不生成后续代的情况下，32 至 256 个槽位时，使用 Mayfly 的会话比使用 NVMe-Park 的会话分别提前 9%、22%、39% 和 56% 完成（图 9）；这是因为它们始终将就绪 KV 保持在高带宽内存容量之外，而 RSI 只会突发出现。对这些智能体而言，固定到期时间就足够了：设置为 5 或 30 分钟时，128 个槽位下的完成时间与 Mayfly 相差不超过 1%，写入 HBF 的数据量多 1–6%。在 RSI 场景中，没有一种固定到期时间适用于所有负载（§5.4），因此 Mayfly 依赖编排器通知的会话结束信号。

5.8 与先前工作的比较

对于Qwen3-32B，在每32个会话配置一个站点时，所有先前的HBF层在32至256个会话下都比Mayfly耗时更长。

既有 HBF 层级。 我们在相同的站点、跟踪记录和步骤模型上运行最接近的 HBF 层级策略的改编版本（§2.3）；表3和4之外的层级数值为三个任务样本的平均值。HBF-Park 遵循 Baek 等人 [3]，在 32 至 256 个会话时，耗时分别是 Mayfly 的 1.08、1.19、1.52 和 2.20× 倍。FlashAccel 式层级 [58] 会写穿每个被驱逐块，在原位读取所有就绪键值，并在空间不足时释放最早会话的键值。若不将热页复制到高带宽内存，其耗时是 Mayfly 的 1.20–2.90× 倍；若高带宽内存可免费填满，则该方案即为 HBF-Greedy，正如 Yu 等人 [65] 不带前缀

重新加载，耗时长 13–41%（表 4）。Lachesis式放置 [63] 会在分配时固定每个块所在的层，并在会话结束时释放该块。采用高带宽内存优先时，耗时为其 1.13–1.57× 倍；采用预知生命周期时，耗时为其 1.16–2.08× 倍。不进行复制时，这两种方案都会持续从 HBF 读取相同的块，吞吐量达到站点 467 GB/s 的 374–448 GB/s；在最多 128 个会话时，能耗为 Mayfly 的 1.6–3.9× 倍。SETA [43], 会从 HBF 为超过 32K 词元的会话提供服务，并在负载下将其调回高带宽内存；HBFlex [69], 会从 HBF 读取所有就绪键值。两者在四个 HBF 站点、 128 个会话时的耗时分别为 Mayfly 的 1.33 和 1.59× 倍。Mayfly在四个 HBF 站点、128 个会话时。

并发限制。 仅限制并发量无法解决高带宽内存的问题：在最佳上限（8 个会话）下，其耗时仍是 32 至 256 个会话时 Mayfly 耗时的 1.20–3.19× 倍。在四个站点运行 128 个会话时，将 HBF-Park 的并发上限设为 32 至 128 个会话，只会使其运行时间变化 3%；原本等待高带宽内存空间的调用，现在改为等待可用槽位。在相同的第 95 百分位首令牌时间下，Mayfly 比任一停放设计快 1.27–1.64× 倍。将并发限制设为 32 个会话（即其最佳上限）时，HBF-Greedy 需要 363 分钟；而不设上限时，Mayfly 无需调节上限，速度快 12%。

6 结论

Harness RSI 的键值会以突发形式溢出高带宽内存，在高负载下主要表现为就绪会话的键值。停放 KV 只需要容量，任何层都能提供这种容量；而超出高带宽内存容量的就绪 KV 需要执行时间带宽，这由 GPU封装上的 HBF 提供。Mayfly 在状态切换期间保持一个有效的 HBF 副本，在会话结束时返回高带宽内存，从而限制每个步骤的 HBF 读取次数，并在已宣布的会话结束时使 KV 失效。在模拟的 H200 上，128 个会话时，它比在 HBF 上进行停放提前 34% 完成（八个站点、256 个会话时提前 55%），比在高带宽内存已满时读取 HBF 提前 23% 完成。这些机制适用于任何就绪 KV 超出高带宽内存容量的场景，例如自主智能体并发运行时。

参考文献

[1] Reyna Abhyankar、Zijian He、Vikranth Srivatsa、Hao Zhang 和 Yiying Zhang。2024 年。InferCept：面向增强型大语言模型推理的高效拦截支持。载于国际机器学习会议（*ICML*）（*PMLR*，第 *235* 卷），第 81–95 页。[2] Ibragim Badertdinov、Alexander Golubev、Maksim Nekrashevich、Anton Shevtsov、Simon Karasik、Andrei Andriushchenko、Maria Trofimova、Daria Litvintseva 和 Boris Yangel。2025 年。SWE-rebench：用于软件工程智能体任务收集与去污染评估的自动化流水线。载于 神经信息处理系统进展（*NeurIPS*），数据集与基准赛道。轨迹：nebius/SWE-rebench-openhands-trajectories。[3] Jongjin Baek、Won Ji、Seungjae Yoo 和 Joo-Young Kim。2026 年。面向智能体大语言模型服务的高带宽内存与高带宽闪存冷热分层。*IEEE* 计算机体系结构快报25，2（2026），355–358。doi:10.1109/LCA.2026.3729099 arXiv:2609.25782。[4] Weijian Chen、Shuibing He、Haoyang Qu、Ruidong Zhang、Siling Yang、Ping Chen、Yi Zheng、Baoxing Huai 和 Gang Chen。2025 年。IMPRESS：面向大语言模型推理的基于重要性感知的多层级前缀键值存储系统。载于 *USENIX*文件与存储技术会议（*FAST*）。[5] Sungjun Cho、Beomjun Kim、Hyunuk Cho、Gyeongseob Seo、Onur Mutlu、Myung-suk Kim 和 Jisung Park。2024 年。AERO：通过自适应擦除操作提升现代基于NAND闪存的固态硬盘的寿命与性能。载于 第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议（*ASPLOS*）论文集。[6] Tri Dao、Daniel Haziza、Francisco Massa 和 Grigory Sizov。2023 年。面向长上下文推理的 Flash-Decoding。https://pytorch.org/blog/flash-decoding/。[7] Chrisantha Fernando、Dylan Banarse、Henryk Michalewski、Simon Osindero 和 Tim Rocktäschel。2024 年。Promptbreeder：通过提示词演化实现自指式自我改进。载于 国际机器学习会议（*ICML*）。[8] Xinghong Fu、Aravinth Kulanthaivelu 和 Yutaro Yamada。2026 年。通过快速树搜索实现自我改进。*arXiv* 预印本 *arXiv:2609.19526*（2026）。[9] Bin Gao、Zhuomin He、Puru Sharma、Qingxuan Kang、Djordje Jevdjic、Junbo Deng、Xingkun Yang、Zhou Yu 和 Pengfei Zuo。2024 年。借助 CachedAttention，为多轮对话提供高性价比大语言模型服务。

发表于USENIX年度技术会议（ATC）。[10] Irving John Good。1966。《关于首台超智能机器的推测》。载于《计算机进展》，Franz L. Alt和Morris Rubinoff（编）。第6卷。Elsevier，第31–88页。[11] Minho Ha、Myunghyun Rhee、Euiseok Kim和Hoshik Kim。2026。HyPAIC：采用高带宽闪存的个人人工智能计算机，用于高性能万亿参数大语言模型推理。arXiv预印本arXiv:2611.04144（2026）。[12] Shengran Hu、Cong Lu和Jeff Clune。2025。智能体系统自动化设计。载于国际学习表征会议（ICLR）。[13] Yanpeng Hu、Yiwei Yang、Yuanwu Zhu、Yusheng Zheng、Andi Quinn和Wei Zhang。2026。HBFSim：真实GPU执行下高带宽闪存的快速高保真模拟。arXiv预印本arXiv:2609.09800（2026）。[14] Chengsong Huang、Wenhao Yu、Xiaoyang Wang、Hongming Zhang、Zongxia Li、Ruosen Li、Jiaxin Huang、Haitao Mi和Dong Yu。2025。R-Zero：从零数据开始的自进化推理大语言模型。arXiv预印本arXiv:2508.05004（2025）。[15] Innodisk。 [n. d.]。M.2（S80）3IE7工业固态硬盘数据表。https://www.innodisk. com/upload/file/Innodisk_M2_S80_3IE7_Datasheet.pdf。基于112层3D TLC NAND的iSLC，100K编程/擦除周期。[16] Maharani A. P. Irawan、Kuntai Du、Jiayi Yao、Dongjoo Seo、Yuyang Huang、Haryadi S. Gunawi和Junchen Jiang。2026。Gandalf：闪存永不消亡！延长大语言模型服务中用于KV缓存卸载的闪存存储寿命。arXiv预印本arXiv:2610.30791（2026）。[17] Hongsun Jang、Jaeyong Song、Changmin Shin、Si Ung Noh、Jaewon Jung、Jisung Park和Jinho Lee。2026。面向长上下文大语言模型离线推理的高性价比近存储处理解决方案。载于ACM操作系统原理研讨会（ASPLOS）。[18] Xuanlin Jiang、Yang Zhou、Shiyi Cao、Ion Stoica和Minlan Yu。2025。NEO：通过CPU卸载缓解GPU内存危机，实现在线大语言模型推理。载于机器学习与系统会议（MLSys）。[19] Carlos E. Jimenez、John Yang、Alexander Wettig、Shunyu Yao、Kexin Pei、Ofir Press和Karthik Narasimhan。2024。SWE-bench：语言模型能否解决真实世界的GitHub问题？载于国际学习表征会议（ICLR）。[20] Aditya Anirudh Jonnalagadda、Agasthi Haputhanthri、Pranav Dangi、Rohan Juneja、Wenshuo Yue、Aritra Bagchi、Bin Gao和Tulika Mitra。2026。SPLASH：协同设计稀疏注意力与高带宽闪存，实现高效的长上下文推理。arXiv预印本 arXiv:2609.23816（2026）。[21] Joseph Kanichai、Tiziano De Matteis和 Animesh Trivedi。2026。构建py-kvcache：使用NVMe固态硬盘对vLLM外部KV缓存进行性能特性分析。arXiv预印本arXiv:2609.11744（2026）。[22] KIOXIA。2023。FL6系列企业级NVMe固态硬盘产品简介。https: // americas.kioxia.com/content/dam/kioxia/shared/business/ssd/enterprise-ssd/asset/productbrief/eSSD-FL6-product-brief.pdf。XL-FLASH，96层BiCS 3D SLC；五年内每日整盘写入次数为60。[23] Omin Kwon、Kyungjun Oh、Jaeyong Lee、Myungsuk Kim和Jihong Kim。2025。使用状态感知随机化器提升高容量现代固态硬盘的寿命与性能。载于IEEE/ACM计算机辅助设计国际会议论文集（ICCAD）。1–9。[24] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph E. Gonzalez、Hao Zhang和Ion Stoica。2023。借助PagedAttention实现大语言模型服务的高效内存管理。载于ACM操作系统原理研讨会（SOSP）。[25] Yoonho Lee、Roshen Nair、Qizheng Zhang、Kangwook Lee、Omar Khattab和Chelsea Finn。2026。Meta-Harness：模型工具框架的端到端优化。arXiv预印本arXiv:2603.28052（2026）。[26] Hanchen Li、Runyuan He、Qiuyang Mang、Qizheng Zhang、Huanzhi Mao、Xi-aokun Chen、Hangrui Zhou、Huanchen Zhang、Alvin Cheung、Joseph Gonzalez和Ion Stoica。2025。Continuum：借助KV缓存生存时间实现高效而稳健的多轮大语言模型智能体调度。arXiv预印本arXiv:2511.02230（2025）。[27] Hui Li、Qihui Zhou、Xiao Yan、Shumeng Zhang、Zijie Dai、Zequn Gong、Mingcong Song、Guoping Long、Hongjie Si和James Cheng。2026。SARIA：面向持久性的KV缓存，以及协调的DRAM–SSD重新加载，用于大语言模型智能体。arXiv预印本arXiv:2610.31169（2026）。[28] Zhuoran Li、Zhuohang Bian、Xin Huang、Yibo Zhao、Guangyu Sun和Youwei Zhuo。2026。高带宽闪存很糟？面向以KV为中心的大语言模型服务的高带宽闪存全栈特性分析。arXiv预印本arXiv:2608.11668（2026）。[29] Jiahang Lin、Shichun Liu、Chengjun Pan、Lizhi Lin、Shihan Dou、Zhiheng Xi、Xuanjing Huang、Hang Yan、Zhenhua Han、Tao Gui和Yu-Gang Jiang。2026。智能体工具框架工程：可观测性驱动的编码智能体工具框架自动演化。arXiv预印本 arXiv:2604.25850（2026）。[30] Li-Wei Liu、Yen-Chin Liao和Hsie-Chia Chang。2022。UP-GDBF：用于NAND闪存应用的19.3 Gbps无误码底限4KB LDPC解码器。IEEE开放电路与系统期刊

3 2022 228 236 10 1109 2022 3209152（），–。[31] 2012 doi：。/OJCAS。。Ren-Shuo Liu、Chia-Lin Yang和Wei Wu。。通过放宽保持时间优化基于NAND闪存的固态硬盘。载于USENIX文件与存储技术会议（FAST）。

[32] Yuhan Liu、Yihua Cheng、Jiayi Yao、Yuwei An、Xiaokun Chen、Shaoting Feng、Yuyang Huang、Samuel Shen、Rui Zhang、Kuntai Du和Junchen Jiang。2025。LMCache：面向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）。[33] Yixin Luo、Yu Cai、Saugata Ghose、Jongmoo Choi和Onur Mutlu。2015。WARM：通过写入热度感知的数据保持管理延长NAND闪存寿命。载于IEEE海量存储系统与技术会议（MSST）。doi：10 1109 2015 7208284 。/MSST。。Yixin Luo、Saugata Ghose、[34] Yu Cai、Erich F. Haratsch和Onur Mutlu。2018。通过容忍早期数据保持损失和工艺偏差延长3D NAND闪存寿命。载于ACM SIGMETRICS。[35] Xiaoyu Ma和David Patterson。2026。大语言模型推理硬件的挑战与研究方向。《计算机》59，5（2026），55–64。doi:10.1109/MC.2026.3652916 arXiv:2601.05047。[36] 美光科技。2024。美光9550 NVMe固态硬盘产品简介。https://www. micron.com/9550。[37] 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 Ba-log。2025。AlphaEvolve：用于科学与算法发现的编码智能体。arXiv预印本 arXiv:2506.13131（2025）。[38] NVM Express。2022。TP4146灵活数据放置。NVM Express批准的技术提案；已纳入NVM Express基础规范2.1。[39] OpenAI。2025。GPT-5 nano。https://platform.openai.com/docs/models/gpt-5-nano。[40] OpenAI。2025。gpt-oss-120b和gpt-oss-20b模型卡。arXiv预印本arXiv:2508.10925（2025）。[41] Xiurui Pan、Endian Li、Qiao Li、Shengwen Liang、Yizhou S han、Ke Zhou、Ying-wei Luo、Xiaolin Wang和Jie Zhang。2024。InstInfer：面向高性价比长上下文大语言模型推理的存储内注意力卸载。arXiv预印本 arXiv:2409.04992（2024）。[42] Zaifeng Pan、Ajjkumar Dahyalal Patel、Yipeng Shen、Zhengding Hu、Yue Guan、Wan-Lu Li、Lianhui Qin、Yida Wang和 Yufei Ding。2025。KVFlow：高效前缀缓存，加速基于大语言模型的多智能体工作流。载于神经信息处理系统进展（NeurIPS）。[43] Haneul Park、Jiwon Chang、Wonkyoo Lee、Dongsop Lee、Chunsung Kim和Nam Sung Kim。2026。SETA：采用高带宽闪存的智能体大语言模型服务中的生命周期感知型会话执行分层。《IEEE计算机体系结构快报》（提前在线发表）（2026）。[44] Jisung Park、Myungsuk Kim、Myoungjun Chun、Lois Orosa、Jihong Kim和Onur Mutlu。2021。通过优化读重试降低固态硬盘读取延迟。载于第26届ACM计算机系统结构支持编程语言与操作系统国际会议论文集（ASPLOS）。[45] PCI-SIG。2019。PCI Express基础规范，第5.0版。PCI-SIG。[46] Danni Peng、Kai Wu、Tianyu Zuo、Pengfei Xia和Hui Zang。2026。通过写入感知型KV缓存策略支持生成式推荐服务使用高带宽闪存。arXiv预印本 arXiv:2609.07175（2026）。[47] Vinicius Petrucci、Felippe Zacarias和Vishal Tanna。2026。高带宽闪存是你所需要的一切吗？载于系统基础设施热点研讨会（HotInfra）。[48] Ruoyu Qin、Zheming Li、Weiran He、Jialei Cui、Feng Ren、Mingxing Zhang、Yongwei Wu、Weimin Zheng和Xinran Xu。2025。Mooncake：以更多存储换取更少计算——面向大语言模型聊天机器人的以KV缓存为中心的架构。载于USENIX文件与存储技术会议（FAST）。[49] Shi Qiu、Yifan Hu、Xintao Wang、Wenhao Zhu、Jianqin Yan、Hao Chen、Kaiqiang Xu、Kai Chen和Yiming Zhang。2026。Tutti：让基于固态硬盘的KV缓存适用于长上下文大语言模型服务。arXiv预印本arXiv:2605.03375（2026）。[50] Bernardino Romera-Paredes、Mohammadamin Barekatain、Alexander Novikov、Matej Balog、M. Pawan Kumar、Emilien Dupont、Francisco J. R. Ruiz、Jordan S. Ellenberg、Peng ming Wang、Omar Fawzi、Pushmeet Kohli和Alhussein Fawzi。2024。通过使用大语言模型进行程序搜索发现数学规律。《自然》625，7995（2024），468–475。[51] Jürgen Schmidhuber。2007。哥德尔机：完全自指的最优通用自我改进系统。载于《通用人工智能》，Ben Goertzel和Cassio Pennachin（编）。施普林格，第199–226页。[52] Asankhaya Sharma。2025。OpenEvolve：开源进化式编程智能体。https://github.com/algorithmicsuperintelligence/openevolve。[53] SK海力士和闪迪。2026。OCP高带宽闪存（HBF）架构规范，第0.7.0版。开放计算项目（OCP）。https://www.opencompute. org/documents/ocp-hbf-architecture-specification-v0-7-0-final-pdf/ 于FMS 2026发布。[54] Dowon Son、Yonggon Park、Hyunuk Cho、Hyungkyu Ham、Onur Mutlu、Sungjin Lee、Gwangsun Kim和Jisung Park。2026。探索面向现代大语言模型推理的高带宽闪存：机遇与挑战。《IEEE计算机体系结构快报》（2026）。doi:10.1109/LCA.2026.3705817 arXiv:2608.13868。[55] Swissbit。2023。X-76m2工业级M.2固态硬盘数据手册。采用伪SLC模式的3D NAND，30K编程/擦除周期。[56] Wenyi Wang、Piotr Piękos、Nanbo Li、Firas Laakom、Yimeng Chen、Mateusz Ostaszewski、Mingchen Zhuge 和 Jürgen Schmidhuber。2026。《Huxley-Gödel Machine：通过逼近最优自我改进机器实现人类水平的编码代理开发》。载于国际学习表征会议（ICLR）。[57] 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。2025。《OpenHands：面向人工智能软件开发者的通用型智能体开放平台》。载于国际学习表征会议（ICLR）。[58] Xinyu Wang、Yalong Xue、Xiaotian Sun、Xiaoyu Zhang、Xinjiang Zhang、Chunmeng Dou、Xueqi Li 和 Xiaoming Chen。2026。《FlashAccel：利用高带宽闪存（HBF）实现高吞吐量大语言模型推理》。arXiv预印本 arXiv:2607.10186（2026）。[59] Haoran Wu、Zeyu Cao、Yao Lai、Binglei Lou、Jiayi Nie、Can Xiao、Timi Adeniran、Kevin Lau、Przemyslaw Forys、Kauser Johar、Catriona Wright、Junyi Liu、Kai Shi、Nicholas D. Lane、Rika Antonova、Jianyi Cheng、Timothy Jones、Aaron Zhao 和 Robert Mullins。2026。《MemExplorer：探索面向智能体推理神经网络处理器的异构内存设计空间》。arXiv预印本 arXiv:2604.16007（2026）。[60] Peng Xia、Rujun Han、Zifeng Wang、Yanfei Chen、Yufan Zhuang、Yoonho Lee、Chengsong Huang、Han Yu、Zhongying CuiZhu、Yifei Ming、Huaxiu Yao、Burak Gokturk、Tomas Pfister 和 Chen-Yu Lee。2026。《RRSI：智能体框架的正则化递归自我改进》。arXiv预印本 arXiv:2609.24972（2026）。[61] Ruiyao Xu、Yanfei Chen、Zhongying CuiZhu、Bhavana Dalvi Mishra、Yifei Ming、Han Yu、Rujun Han、Chen-Yu Lee 和 Tomas Pfister。2026。《Harness^2：面向开放世界的递归式智能体框架构建》。arXiv:2610.14224。[62] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv、Chujie Zheng、Dayiheng Liu、Fan Zhou、Fei Huang、Feng Hu、Hao Ge、Haoran Wei、Huan Lin、Jialong Tang、Jian Yang、Jianhong Tu、Jianwei Zhang、Jianxin Yang、Jiaxi Yang、Jing Zhou、Jingren Zhou、Junyang Lin、Kai Dang、Keqin Bao、Kexin Yang、Le Yu、Lianghao Deng、Mei Li、Mingfeng Xue、Mingze Li、Pei Zhang、Peng Wang、Qin Zhu、Rui Men、Ruize Gao、Shixuan Liu、Shuang Luo、Tianhao Li、Tianyi Tang、Wenbiao Yin、Xingzhang Ren、Xinyu Wang、Xinyu Zhang、Xuancheng Ren、Yang Fan、Yang Su、Yichang Zhang、Yinger Zhang、Yu Wan、Yuqiong Liu、Zekun Wang、Zeyu Cui、Zhenru Zhang、Zhipeng Zhou 和 Zihan Qiu。2025。《Qwen3技术报告》。arXiv预印本 arXiv:2505.09388（2025）。[63] Jaehoon Yang、Jeongmin Lee、Haneul Park、Seung Yul Lee、Nam Sung Kim 和 Jae W. Lee。2026。《Lachesis：跨高带宽内存与高带宽闪存的面向智能体服务的生命周期感知型键值缓存放置》。arXiv预印本 arXiv:2610.08378（2026）。[64] Xunjian Yin、Xinyi Wang、Liangming Pan、Li Lin、Xiaojun Wan 和 William Yang Wang。2025。《Gödel Agent：用于递归自我改进的自指智能体框架》。载于第63届计算语言学协会年会论文集（ACL）。27890–27913。doi:10.18653/v1/2025.acl-long.1354[65] Zack Yu、Chloe Wong、Coleman Hooper、Minjae Lee、Wonjun Kang、Youngjin Cho、Michael W. Mahoney、Yakun Sophia Shao、Kurt Keutzer 和 Amir Gholami。2026。《高带宽闪存用于大语言模型服务的特性分析》。arXiv预印本 arXiv:2609.39131（2026）。[66] Eric Zelikman、Eliana Lorch、Lester Mackey 和 Adam Tauman Kalai。2024。《自学优化器（STOP）：递归式自我改进代码生成》。载于语言建模会议（COLM）。[67] Jenny Zhang、Shengran Hu、Cong Lu、Robert Tjarko Lange 和 Jeff Clune。2026。《Darwin Gödel Machine：自我改进智能体的开放式进化》。载于国际学习表征会议（ICLR）。[68] Lianmin Zheng、Liangsheng Yin、Zhiqiang Xie、Chuyue Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica、Joseph E. Gonzalez、Clark Barrett 和 Ying Sheng。2024。《SGLang：结构化语言模型程序的高效执行》。载于神经信息处理系统进展（NeurIPS）。[69] Shuzhang Zhong、Weikai Xu、Yifan Zhou、Tongbin Zhao、Tenghao Zhao、Yifei Kang、Cunyin Chang、Shu Li、Guangyu Sun 和 Meng Li。2026。《HBFlex：连接细粒度大语言模型状态与粗粒度HBF并行执行的灵活内存系统》。arXiv预印本 arXiv:2609.18675（2026）。[70] Kan Zhu、Mathew Jacob、Chenxi Ma、Yi Pan、Stephanie Wang、Arvind Krishnamurthy 和 Baris Kasikci。2026。《TraceLab：大语言模型服务中编程智能体工作负载的特性分析》。arXiv预印本 arXiv:2606.30560（2026）。SyFI编程轨迹（syfi_coding_trace），知识共享署名4.0国际许可协议，https://github.com/uw-syfi/TraceLab。