arXiv:2512.14946v1 [cs.OS] 16 Dec 2025

EVICPRESS：面向高效大语言模型服务的 KV 缓存压缩与驱逐联合优化

Shaoting Feng*, 1 Yuhan Liu*, 1 Hanchen Li[2] Xiaokun Chen[3] Samuel Shen[3] Kuntai Du[3] ZhuohanGu[4]Rui Zhang[5] Yuyang Huang[1] Yihua Cheng[3] Jiayi Yao[1] Qizheng Zhang[6] Ganesh Ananthanarayanan[7] Junchen Jiang[1,3]芝加哥大学 [2]加州大学伯克利分校 [3]Tensormesh,Inc. [4]麻省理工学院 [5]加州大学圣克鲁兹分校 [6]斯坦福大学 [7]微软

摘要

复用 KV 缓存对于提升大语言模型推理系统的效率至关重要。随着大语言模型用户增多，KV 缓存占用量很容易超过 GPU 内存容量，因此以往的工作提出了将 KV 缓存驱逐 到低层级存储设备，或压缩 KV 缓存，以便在快速内存中容纳更多 KV 缓存。然而，以往工作忽略了一个重要机会：在所有 KV 缓存之间联合 优化驱逐和压缩决策，从而在不损害质量的前提下降低平均生成延迟。

我们提出 EVICPRESS，这是一种 KV 缓存管理系统，可对多个存储层级中的 KV 缓存进行有损压缩和自适应驱逐。具体而言，对于某个上下文的每个 KV 缓存，EVICPRESS 都会综合考虑压缩和驱逐该 KV 缓存对所有上下文整体平均生成质量和延迟的影响。为此，EVICPRESS 提出了一种统一的效用函数，用于量化有损压缩或驱逐对质量和延迟的影响。为此，EVICPRESS 的分析模块会定期更新所有上下文中所有可能的驱逐压缩配置所对应的效用函数分数，并使用快速启发式算法放置 KV 缓存，以重新安排各存储层级中的 KV 缓存，从而最大化每个存储层级上的效用函数分数。与驱逐 KV 缓存或压缩 KV 缓存的基线方法相比，EVICPRESS 在高速设备上实现了更高的 KV 缓存命中率，即更低的延迟；同时，通过对易受压缩误差影响的上下文采用保守压缩，保持了较高的生成质量。在 12 个数据集和 5 个模型上的评估表明，在生成质量相当的情况下，EVICPRESS 可将首令牌生成时间（TTFT）最多提升 2.19× 倍。

1 引言

大语言模型（LLM）广泛应用于聊天机器人、智能体、个人助理及许多其他场景。因此，提高大语言模型推理的效率，以较低的计算成本为用户带来低延迟体验，是工业界和学术界最重要的问题之一。为此，复用 *KV 缓存*——大语言模型推理过程中生成的中间状态——已成为现代大语言模型推理系统中的事实标准优化方法。通过存储并在不同请求中复用上下文的 KV 缓存，可以显著降低推理延迟，大幅提升吞吐量 [22, 29, 53, 54]。

*贡献相同

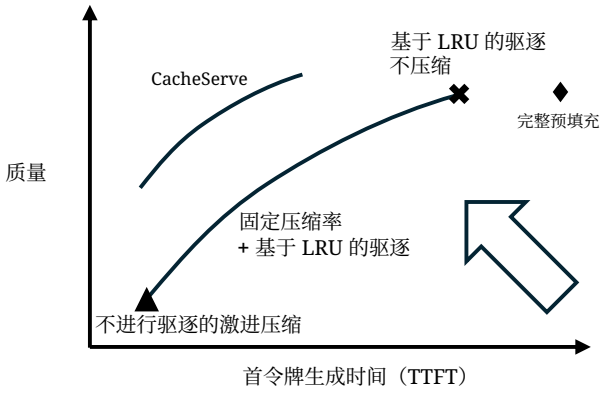


图 1：EVICPRESS 联合优化卸载与压缩，在质量和首个令牌生成时间方面实现了更优的权衡。此图仅用于说明。

然而，要实现这一潜在的性能收益，也带来了新的系统挑战：由于模型上下文窗口限制不断扩大 以及同时服务的用户数量增加，需要存储的 KV 缓存总量持续增长。因此，要将所有 KV 缓存都放入 GPU内存十分困难。为避免重新计算无法放入 GPU内存的 KV 缓存，已有两类研究方向。首先，有研究提出 KV 缓存驱逐[1]，以提高缓存命中率 [6, 38, 59]。这些研究采用经典缓存策略（如最近最少使用（LRU）），将 KV 缓存从 GPU内存驱逐到空间更大、成本更低的低层级存储设备中。这样可以存储更多 KV 缓存，从而避免反复重新计算 KV 缓存，并可能提高缓存命中率。另一类研究则采用 KV 缓存压缩来减少每个 KV 缓存的内存占用，使 GPU内存能够容纳更多 KV 缓存 [17, 18, 31, 32, 63]。

然而，单独使用驱逐或压缩方法都存在局限。应用压缩时，先前的工作只考虑在存储设备达到空间上限时采用更激进的压缩，而没有考虑驱逐的可能性，也没有考虑直接使用传统的驱逐策略，例如 LRU [17, 18, 31, 32]。应用驱逐时，这些工作不会根据驱逐决策调整压缩配置 [6, 38, 59]。这种做法

[1]本文所说的 KV缓存驱逐，是指将 KV 缓存从较快的存储设备移至下一层级的较慢存储设备，而不是将 KV 缓存从系统中删除。

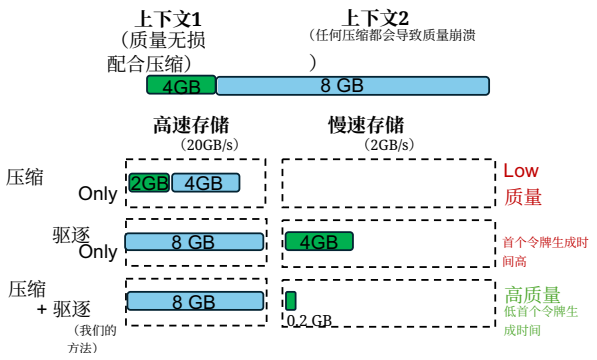


图 2：一个简单示例，说明联合决定压缩和驱逐的优势。考虑两个 *KV* 缓存（一个 *4 GB* 的 *KV* 缓存，在质量不下降的情况下可压缩至 *5%*；另一个 *8 GB* 的 *KV* 缓存，稍作压缩就会使其质量降至 *50%*），它们最初分别存储在快速内存（加载带宽为 *20GB/s*）和慢速设备（*2GB/s*）上。仅压缩方案将两个 *KV* 缓存都压缩一半，从而实现较短的首个令牌生成时间（*0.3* 秒），但质量较低（仅为 *75%*）；仅驱逐方案则将一半 *KV* 缓存驱逐至慢速存储，虽然质量较高（*100%*），但在重新使用被驱逐的 *KV* 缓存时，首个令牌生成时间较长（*2.4* 秒）；通过综合考虑压缩和驱逐，我们能够同时实现较高的质量（*100%*）和较短的首个令牌生成时间（*0.5* 秒）。

通常不是最优的。如图 2所示，如果考虑两个上下文的 KV 缓存（上下文1的大小为 4 GB，上下文2的大小为 8 GB），即使对上下文1进行激进压缩，其质量分数仍可保持在 100%；而即使只进行最低限度的压缩，上下文2的质量也会崩溃至 50%。仅应用压缩时，将两个上下文的 KV 缓存都压缩至原始大小的 50%，上下文2的质量分数会降至 50%，平均质量为 75%，平均首令牌生成时间为 0.3 秒。仅应用驱逐时，上下文1被驱逐至慢速设备，上下文2则保留在高速设备上，质量分数达到 100%，首个令牌生成时间为 2.4 秒。更好的选择是结合压缩和驱逐：具体而言，将上下文2保留在高速存储中，并将上下文1的 KV 缓存压缩至 0.2GB 后驱逐至慢速存储。这样质量分数可达到 100%，首个令牌生成时间为 0.5 秒。

以往的工作并未考虑对所有上下文进行驱逐或压缩的决策，这主要是因为该问题会带来巨大的搜索空间。首先，应用压缩时，可以调整压缩方法和压缩率。其次，压缩需要与驱逐结合使用；例如，压缩率达到一定程度后，应采用驱逐，以避免质量大幅下降。此外，处理任意上下文的 KV 缓存时，仅考虑压缩和驱逐对该上下文自身的影响是不够的，还需要考虑其对其他上下文的影响。将 KV 缓存压缩到适中的压缩率并存放在高速存储设备上，有利于其质量和首个令牌生成时间；然而，这部分空间也可以用来存储更多经过激进压缩的 KV 缓存，从而提升整体首个令牌生成时间和质量。

一个关键洞见是，不同上下文从压缩或驱逐中获益的程度各不相同，因为它们对有损压缩的敏感度各异（§3.1）。有些上下文对压缩误差高度敏感，即使采用保守的压缩率，也会导致质量大幅下降；而另一些上下文则可以大幅压缩，而不影响生成质量。因此，应在全局范围内做出压缩和驱逐决策，以考虑系统中所有上下文的敏感度。例如，当较高存储层级已满时，我们应选择驱逐对有损压缩高度敏感、会因此遭受严重损失的上下文，同时从较低层级调入敏感度较低的 KV 缓存，并对其进行积极压缩，以便将其纳入较高存储层级。

基于所有上下文做出全局缓存管理决策，会产生一个规模随上下文数量呈指数增长的搜索空间。为了使其切实可行，本文提出一种效用函数，以存储层级、压缩方法和压缩率为输入，输出单一分数，定量衡量每项决策对延迟和质量的影响。这样，我们便能针对每项缓存管理决策，根据所有上下文轻松计算延迟与质量之间的整体权衡。

综合上述思路，我们提出 EVICPRESS 系统，它能够针对每个 KV 缓存（每个上下文）自适应地选择最优的压缩-驱逐配置——包括是否驱逐、驱逐到哪个存储设备，以及是否压缩、采用何种方法和压缩比例。EVICPRESS 首先根据一组初始生成的查询，计算每个上下文所有可能配置的效用函数分数。然后，EVICPRESS 将 KV 缓存放置到能够使效用函数分数总和最大的特定存储设备上。由于真实世界查询可能会偏离初始查询集，EVICPRESS 会记录每个上下文新收到的查询，并定期根据最新的查询集重新分析。重新分析后，EVICPRESS 使用贪心算法重新调整不同存储层级中的 KV 缓存，从而使每个层级上的效用函数分数最大化。

我们通过扩展 vLLM 和 LMCache，在现有的大语言模型服务技术栈中实现了 EVICPRESS，新增了 ~3K 行代码，以支持多层 KV 缓存放置、压缩和驱逐控制。具体而言，我们的实现会拦截 KV 缓存的查找、检索和存储操作，针对每个上下文应用压缩/驱逐策略，并与 vLLM 的分页式 GPU内存管理紧密集成，以协调缓存数据在 GPU、CPU 和固态硬盘各层之间的移动。

简而言之，我们的贡献如下：

- 我们设计并实现了 EVICPRESS，这是首个在多层 KV 缓存驱逐系统中同时考虑有损压缩和驱逐的系统。
- 我们提出了一种效用函数，用于量化有损压缩和驱逐对质量和延迟的影响，从而做出缓存管理决策。借助这一效用函数，EVICPRESS 可以依据明确的优化标准，一次性比较所有可行的压缩-

驱逐配置。

• 我们在最先进的推理引擎 vLLM 中实现了 EvicPress，并表明，与在所有上下文中使用相同压缩方法和压缩比例的基线方法相比，在质量分数相同的情况下，EVICPRESS 在五种不同模型上将首个令牌生成时间缩短了 1.43–3. [24], 。与仅采用基于最近最少使用的驱逐方法的基线相比，在质量下降不超过 3% 的情况下，EVICPRESS 将首个令牌生成时间缩短了 1.22 至 1.56×；在质量分数目标为 80% 时，与表现最佳的基线方法相比，推理吞吐量提升了 2.0–3.6×。

需要指出的是，使用能够捕捉延迟和质量影响的效用函数，并根据敏感度调整压缩配置，这些想法并非新颖 [23, 39, 55, 62]。然而，我们是首个将效用函数用于实现驱逐与压缩联合优化的团队，取得的效果远优于固定压缩或驱逐。

2 背景

2.1 大语言模型中的 KV 缓存复用与驱逐

大语言模型推理包含两个阶段：预填充阶段会一次性将整个输入上下文送入大语言模型，以生成 *KV* 缓存；解码阶段则利用 KV 缓存生成连续的输出词元。预填充阶段的计算量很大，尤其是在长上下文场景下，因为其计算复杂度会随输入长度超线性增长。为加速预填充阶段，许多近期系统通过存储和复用先前计算的 KV 缓存来跳过冗余计算 [15, 29, 31]。系统领域的早期研究主要关注分层存储系统内部的 KV 缓存管理。

为减少内存浪费，vLLM [24] 使用 PagedAttention 将 KV 缓存划分为更小的类页面数据块进行管理，而非使用大型连续数据块，从而减少内存碎片。SGLang [60] 采用名为 RadixAttention 的树形缓存机制来优化计算和内存使用：该机制将 KV 缓存存储在基数树中，并通过最近最少使用（LRU）驱逐策略进行管理。一些近期研究进一步拓展了这一方向，将 KV 缓存管理扩展到整个内存层级和存储系统。LM-Cache [29] 会复用任何重复文本对应的 KV 缓存，无论该文本是否为前缀；同时，将未选中的 KV 缓存卸载到 CPU 内存等分层缓存设备中。Mooncake [37],、Attentionstore [13], 和 InstIn- fer [34] 等工作利用分离式资源（即 CPU、动态随机存取存储器和固态硬盘）卸载 KV 缓存，从而加速大语言模型服务。

2.2 KV缓存压缩

由于 KV 缓存的体积相对较大（Qwen3-32B 在 32k token 上下文下为 7.81 GB），此前已有大量工作

数据集	长度（平均值）	上下文类型	上下文数量
SAMSum	11586	少样本	50
TriviaQA	12311	少样本	50
MultiNews	2741	多文档	50
2wikimqa	7526	多文档	50
QASPER	4885	单文档	50
NarrativeQA	40003	单文档	50

表 1： §3中所使用的上下文长度、类型和数量。

致力于压缩 KV 缓存。KV 缓存压缩为大语言模型服务带来两项关键优势。首先，缩小每个缓存条目的体积后，在给定的内存预算内便能存储更多 KV 缓存，从而提高复用的可能性，提升缓存命中率。其次，在基于卸载的推理系统中，KV 缓存存储于 CPU 或固态硬盘内存时，压缩可大幅降低数据传输开销。这一点至关重要，因为通过带宽受限的互连传输大型 KV 缓存可能会成为性能瓶颈。

已有许多 KV 缓存压缩方法，它们在质量保持和系统效率方面各不相同。这些方法大致可分为：(1) 令牌丢弃（例如 H2O、snapkv） [27, 59], , 即有选择地从 KV 缓存中移除不太重要的词元；(2) 量化（例如 CacheGen（自适应量化）、KIVI（均匀量化）） [31, 32], , 即降低 KV 条目的位宽表示；(3) 合并（例如 HOMER、Look-M） [42, 45]；(4) 提示压缩（例如 LLMLingua、LongLLMLingua） [19, 20]。

由于这些类别的性质不同，这些方法的性能也存在很大差异。正如我们将在第 3.2 节中展示的那样，根据具体的模型规模、数据集属性和生成长度，不同方法会呈现不同的权衡。有些方法（如 CacheGen 或 KIVI）具有解压开销，而另一些方法（如 H2O）则可直接用于解码。此外，关于 KV缓存压缩的新研究每月都在以前的工作基础上不断涌现。然而，目前还没有一种有原则的方法，能够在真实系统中选择并采用如此多样的压缩算法。

3 动机

管理 KV 缓存内存最直接的方法，是对所有上下文进行均匀压缩，或采用缓存驱逐方案，例如最近最少使用（LRU）或最不经常使用（LFU）。压缩和驱逐都能显著提高推理吞吐量：压缩 KV 缓存的内存占用，使其能够更快地从远程存储设备读取 [22, 28, 31],；或者将近期更可能需要的 KV 缓存保留在更快的存储设备中，以加快加载速度 [5, 11, 13, 29]。

然而，正如我们将在本节中展示的，仍有进一步优化的空间，

因为 KV 缓存管理决策应取决于特定上下文的特征。

在本节中，我们使用 LongBench 数据集中的六个数据集进行实证研究 [3]。LongBench 是一种广泛使用的数据集，用于验证大语言模型回答长上下文问题的能力。数据集详情见表1。具体而言，我们从 LongBench [3] 数据集中随机抽取了 50 个上下文，涵盖多种任务类型，包括少样本、单文档问答和多文档问答。

3.1 为什么不压缩或驱逐所有上下文?

某些上下文的 KV 缓存对压缩误差高度敏感：即使采用极为保守的压缩比，也会导致质量大幅下降。相比之下，其他上下文受到的影响则微乎其微。由此，我们得出以下第一个洞见。

洞见 1 不同上下文对压缩误差的敏感程度各异，因此对所有上下文简单地采用相同的驱逐或压缩方案并非最优。

图 3 展示了这一点，其中绘制了不同类型数据集的压缩误差敏感度累积分布函数（CDF）。压缩误差敏感度定义为压缩比为 0 时的相对质量下降。9[2],数据集见表1。该图使用 Llama-3.1-8B-Instruct 模型和`keydiff`压缩绘制。如图所示，例如，维基百科文章（2wikimqa）的敏感度较高，中位数为 0.681。我们的直觉是，所有上下文中的关键信息都以较高密度分布。与此同时，故事叙述（NarrativeQA）的敏感度低得多，中位数为 0.340，因为它们基于包含冗余信息的对话。

此外，我们发现，即使两个数据集属于相同类型的上下文，其敏感度分布也可能截然不同。例如，SAMSum 和 TriviaQA 都是少样本数据集，在回答实际问题之前，会先向大语言模型提供若干示例问答对；然而，SAMSum 的敏感度中位数为 0.676，而 TriviaQA 的敏感度中位数为 0.392。此外，MultiNews 样本属于多文档上下文，每个样本由多篇新闻文章拼接而成；QASPER 样本则属于单文档上下文，每个样本仅包含一篇特定主题的文档。然而，MultiNews 和 QASPER 数据集的敏感度中位数非常接近，分别为 0.738 和 0.759。

总体而言，不同上下文的敏感度差异显著。具体来说，变异系数（一种方差度量）介于 0.078 到 0.394 之间，这表明不同上下文的敏感度差异很大，即使它们属于同一数据集也是如此。

[2]压缩比定义为压缩 KV 缓存大小与原始 KV 缓存大小之比。

3.2 为什么不对所有上下文采用相同的压缩方法和压缩比?

另一个自然的问题是，为什么不对所有上下文都采用固定的压缩方法和压缩比？正如 §2.2 中所述，可以调整许多参数，包括不同的压缩方法以及每种方法对应的压缩比。由于不同方法依据不同指标来判断如何从 KV 缓存中丢弃信息，我们推测，不同上下文各自都有最优的压缩方法和压缩比。

洞见 2 最佳压缩方法和压缩比无法仅根据上下文长度或上下文类型简单确定。

为验证这一点，在图4中，我们绘制了三组上下文的长度分布，这三组上下文的最优压缩方法分别为`kvzip`、`knorm`和`snapkv`。这里，最优压缩方法定义为在相同压缩比下质量损失最低的方法[3]。最优方法分别为`kvzip`、`knorm`和`snapkv`的上下文占比分别为 40.3%、40.0% 和 19.7%，这表明不存在适用于所有上下文的主导压缩方法，因此需要针对每个上下文进行方法适配。此外，我们可以看到，这三组上下文的长度分布彼此重叠，说明上下文长度与应选择哪种压缩方法之间没有明确关系。

其次，在图5中，我们绘制了表1中不同数据集的平均最优压缩比及标准差。最优压缩比定义为所有压缩方法中质量-延迟权衡最佳时对应的压缩比[4]。我们可以看到，即使在同一数据集内，不同上下文的最优压缩比也存在很大差异，变异系数为 0.068 至 0.806。这表明，无法仅根据数据集来确定最优压缩比。

此外，我们观察到，即使两个数据集属于相同类型，例如同为多文档上下文的 MultiNews 和 MuSiQue，它们的分布也互不重叠。这表明，仅凭上下文类型也无法轻易确定最优压缩比。

4 EVICPRESS 设计

现在，我们介绍 EvicPress的设计，首先介绍整体工作流程（§4.1）、用于确定每种压缩配置质量与速率权衡的效用函数

[3]生成质量定义为压缩后生成的答案与原始预填充答案之间的相似度。[4]为了衡量首个令牌生成时间与质量之间的相关性，我们使用归一化质量 –归一化首个令牌生成时间，以确保二者在尺度上具有可比性。

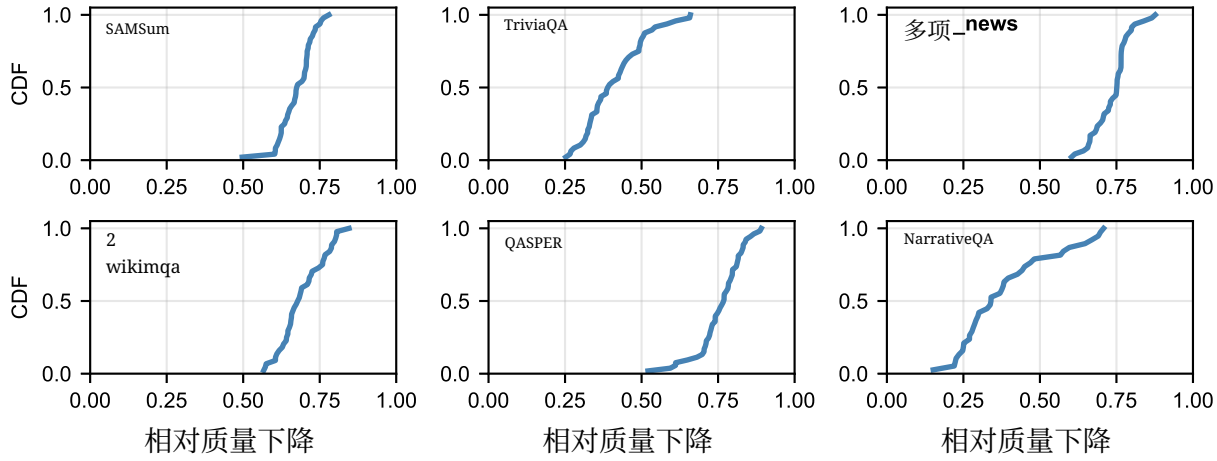


图3： 表*1*.

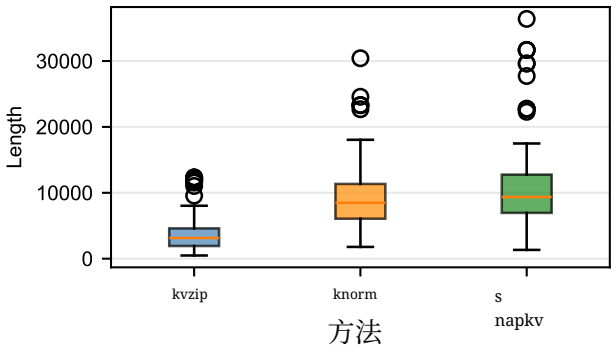


图 4： 在压缩比为 *0.6* 时，使用 *kvzip*、*knorm* 和 *snapkv* 后质量下降最小的上下文的长度分布。

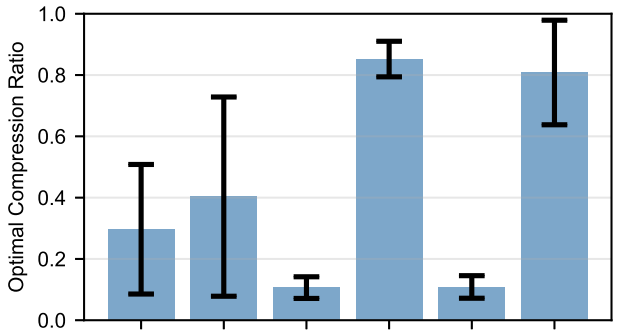


图 5： 不同类型的上下文具有不同的最优压缩比。这里，每个柱状图显示了不同类型上下文的最优压缩比的均值和标准差。

（§4.2）、用于确定每个上下文适用压缩配置的分析机制（§4.3），以及在存储层级已满时管理 KV 缓存的策略（§4.4）。

4.1 总体工作流程

EVICPRESS 包含两个阶段：初始离线阶段和在线推理阶段。

初始离线阶段： 在初始离线阶段，EVIC-PRESS 的分析器（§4.3）会计算每个上下文所有可能的驱逐压缩配置的效用函数分数（§4.2）。为了计算效用函数中的质量和延迟项，我们为每个上下文使用一组初始生成的问题。

在线推理阶段： 在在线推理阶段，当用户请求到来时，查找模块首先确定 EVICPRESS 管理的多层存储系统中是否存在所请求上下文的 KV 缓存。如果存在，EVICPRESS 的检索器会从查找模块返回的位置获取 KV 缓存。如果不存在，则由大语言模型生成 KV 缓存，然后由 EVICPRESS 将其保存到远程磁盘，假设远程磁盘的空间无限。由于在线查询可能与初始分析查询集不同，当达到的质量低于阈值*X*% 且有空闲 GPU 周期时，EVICPRESS 的分析模块还会基于每个上下文更新后的查询集定期重新分析。

请注意，KV 缓存存储操作在 CPU 上运行，不会阻塞其他查询的在线推理计算。此外，重新分析步骤会先将 KV 缓存压缩为多个版本，然后使用不同版本的 KV 缓存运行解码阶段。压缩和

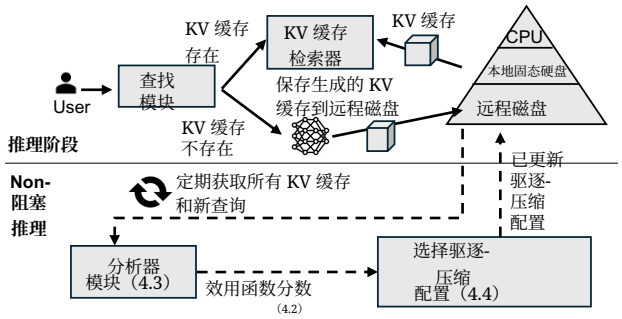


图 6： EVICPRESS 的端到端工作流程。

解码计算的 GPU 计算开销都很低，可以与其他请求的解码计算批处理。正如我们即将展示的（§6.3），重新分析产生的系统开销极小。

4.2 效用函数：平衡质量与加载延迟

每种压缩和驱逐配置都代表了生成质量与加载延迟权衡中的一个点。为了便于衡量各配置对质量和加载延迟的影响，我们设计了一个效用函数，为每种驱逐压缩配置输出一个分数——包括驱逐方案将 KV 缓存卸载到的存储层级，以及所采用的压缩方法和压缩率：

$$\text{Util}(\text{method}, \text{ratio}, \text{device}) = (\alpha \cdot \text{quality} - \text{TTFT}) \cdot \text{frequency},$$

其中， α 是质量与加载延迟之间的权衡参数——较高的 α 会提升 EVICPRESS 的质量，但平均加载延迟也会更高；在评估中，我们将改变 α，以展示 EVICPRESS 在质量与加载延迟之间的权衡。

在上述公式中，函数的输入是驱逐算法将 KV 缓存发送到的存储设备层级，以及所应用的压缩方法/压缩率。质量根据压缩方法和压缩率计算，并对性能分析查询集中的所有数据取平均值；延迟项根据从相应存储设备获取（压缩后的）KV 缓存的估计延迟计算；频率项则表示上下文被访问的次数。

举例来说，以 §3.1中的例子为例，使用 `keydiff` 压缩时，对于敏感度较高的 2wikimQA 数据集中的上下文，当压缩比为 0.6 时，质量分数可达 0.65，首个令牌生成时间为 0.05 秒；对于敏感度较低的 NarrativeQA 数据集中的上下文，当压缩比为 0.9 时，质量分数可达 0.67，首个令牌生成时间同样为 0.05 秒。假设 $\alpha = 1$ 每个上下文出现的频率相同，效用函数的得分分别为 0.607 和 0.621。(keydiff, 0.6, CPU) 配置严格劣于 (keydiff,0.9, CPU)，因为前者的首个令牌生成时间相同，但质量更低。效用函数的得分也反映了这一点：前者的得分低于后者。

该系统通过可调参数α，使用户能够在生成质量和延迟之间进行权衡。较大的 α 更重视生成质量，而较小的α 则更重视降低首个令牌生成时间。例如，在 2wikimQA 数据集中，压缩比为 0.4 时，质量分数为 0.82，首个令牌生成时间为 0.20 秒；而压缩比为 0.2 时，质量提升至 0.92，但首个令牌生成时间增加到 0.24 秒。因此，如果用户设置 $\alpha > 0.3$，效用函数会偏向压缩比为 0.2 所带来的更高质量；相反，较低的 α 则更重视压缩比为 0.4 所带来的延迟降低。

计算效用函数时的另一个重要设计问题是：应以何种合适的粒度来计算该效用函数？EVICPRESS 基于上下文级计算该效用函数，而不是采用块级或令牌级计算。块级计算会将整个上下文的 KV 缓存拆分为多个数据块。这是因为上下文级分析能够从全局视角、结合完整上下文来考察重要的令牌分布。另一方面，对于块级分析，某些词元在某个数据块的范围内可能很重要，但在整个上下文的范围内却并不重要。令牌级分析则更加次优，因为无法进行质量测量。此外，令牌级检索无法使带宽达到饱和；由于内存访问不一致，还会导致速度较低且开销较高。

4.3 分析模块：计算效用函数分数并选择驱逐-压缩配置

基于洞见 1，压缩-驱逐配置必须适应不同的上下文。洞见2 表明，不能仅根据上下文的特征（如长度或类型）来确定配置。因此，EVICPRESS 不再试图找出能够表明各配置优势的上下文特征，而是直接分析所有配置对整体质量和延迟的影响。具体而言，EVICPRESS 使用效用函数（§4.2），为每种（存储设备、压缩方法、压缩比）配置输出一个分数，以同时衡量延迟和质量。

在线推理阶段开始之前，由于系统尚未接收到任何查询，分析模块会参照先前研究，为每个上下文生成一组训练问题 [10,47–49,57],，并据此计算平均质量和延迟。这些问题由 GPT-5 API 针对上下文中的不同段落生成，旨在为每个段落选出能够尽可能消除冗余、从而降低延迟的驱逐-压缩配置。

在线推理期间，由于真实世界查询可能会偏离训练查询

分析模块还会定期使用更新的数据集重新分析。当在线测试数据集上的质量低于分析数据集上的质量，且差距超过X%阈值，同时系统有空闲 GPU 周期时，就会触发重新分析。请注意，分析过程只需采用不同的方法和比例压缩 KV 缓存，开销极低，然后使用这些不同的压缩版本运行解码阶段。众所周知，解码阶段很容易进行批处理（与其他请求一起批处理不会增加解码延迟）。因此，当在线推理请求数少于系统中的最大批大小时，就会有可用的 GPU 周期。在实践中，用户可以根据自己的质量要求自由调整X%。

4.4 驱逐压缩配置选择模块

给定一组初始上下文，其中每个上下文都有一组用于分析的生成的查询，EVICPRESS 会计算每个上下文所有可能的驱逐压缩配置的效用函数分数。随后，EVICPRESS 的配置选择模块会根据效用函数分数最高的配置，将每个上下文的（压缩后的）KV 缓存放置到某个存储层级。

当存储设备 S 已满时，需要更新该设备上现有的 KV 缓存。具体而言，每个上下文的 KV 缓存可以通过更激进的压缩来更新，以便存入 S；也可以将其驱逐到低层级存储，以节省空间。每个上下文都有（#compression methods ×# compression ratios × #lower-tier storage devices）种选项可供选择。在此，我们会过滤掉那些将 KV 缓存保留在存储设备 S 上但不节省空间的选项（例如保持压缩比不变且不将其驱逐到下一层级）。

为每个上下文确定配置，同时最大化设备上的总效用函数分数S 这一问题属于多重选择背包问题，且为 NP 难问题。我们通过贪心方式求解：反复为 S上的上下文寻找效用分数下降最少的更新配置，直到更新后的 KV 缓存能够成功装入。随后，如果低层级存储也已满，则递归地应用相同的更新流程。该流程的具体贪心算法见 §A。

5 实现

我们基于 vLLM v0.11.2 和 LMCache v0.3.9post2 实现了 EVICPRESS，使用 PyTorch v2.9 编写了约 3K 行 Python 代码。

收到新的文本查询时，我们会调用 LMCache 的 KV 缓存查找模块，具体来说是 lookup(tokens)。如果系统中存在所请求的 KV 缓存，则返回命中的词元数量，vLLM 引擎会为命中的词元分配相应大小的 GPU 内存来存储 KV 缓存。随后触发 LMCache 的 retrieve(tokens)->KV Cache 函数，从 EVIC-PRESS 管理的多层存储后端加载 KV 缓存，并将其放入 vLLM 分配的分页式 GPU 内存中。

如果未找到所请求的 KV 缓存，LM- Cache 的 `store(tokens, KV Cache)` 函数便会被触发，将生成的 KV 缓存保存到 CPU DRAM；随后，EVICPRESS 会调用 `manage` 函数，将新的 KV 缓存放置到存储设备上，并在设备已满时更新现有的 KV 缓存条目。具体而言，我们使用类 `KVConfig(KV Cache)` 来记录给定 KV 缓存所有可能配置的效用函数分数。`manage` 函数接收 `[KVConfig(KV Cache 1), ..., KVConfig(KV Cachen)]`，并通过 E`(deviceplacement,compressionmethod, compression ratio)` 的配置选择模块（§4.4）计算每个需要更新的 KV 缓存所采用的设备放置、压缩方法和压缩比。最后，manage 函数会使用新配置更新这些 KV 缓存。

6 评估

评估的主要结论如下：

- 在来自 LongBench 12 个数据集的 555 个上下文和 5 个模型上，[3],与结合固定压缩和经典驱逐的基线相比，EVICPRESS 将首个令牌生成时间缩短了 1.43 至 3.77×，并在首个令牌生成时间相同的情况下将质量提升了 13.58% 至 55.40%。

- 与 smartKV 缓存驱逐系统 IMPRESS 相比，EVICPRESS 将首个令牌生成时间缩短了 1.5 至 5.2 ×，并在首个令牌生成时间相同的情况下将质量提升了 14.29% 至 27.00%。
- 与完整预填充或完全驱逐相比，EVICPRESS 将首个令牌生成时间缩短了 1.29 至 2.19×，同时质量下降幅度不超过 3%。

6.1 评估设置

数据集	Mean	Std	主题
NarrativeQA	108K	55K	长篇故事（文学、电影）。
QASPER	24K	12K	自然语言处理研究论文
多字段问答_英文	29K	15K	法律文件和报告。
HotpotQA	57K	18K	维基百科文章。
2wikimqa	30K	15K	维基百科文章。
MuSiQue	69K	9K	维基百科文章。
政府_报告	54K	34K	美国政府问责局和国会研究服务处报告。
QMSum	57K	27K	会议脚本和记录。
Multi-News_	12K	10K	关于同一事件的新闻文章。
trec	30K	12K	新闻文章和网页。
TriviaQA	47K	25K	知识问答题。
SAMSum	34K	17K	类似 Messenger 的对话历史记录-ries。

表 2：数据集统计信息及描述。

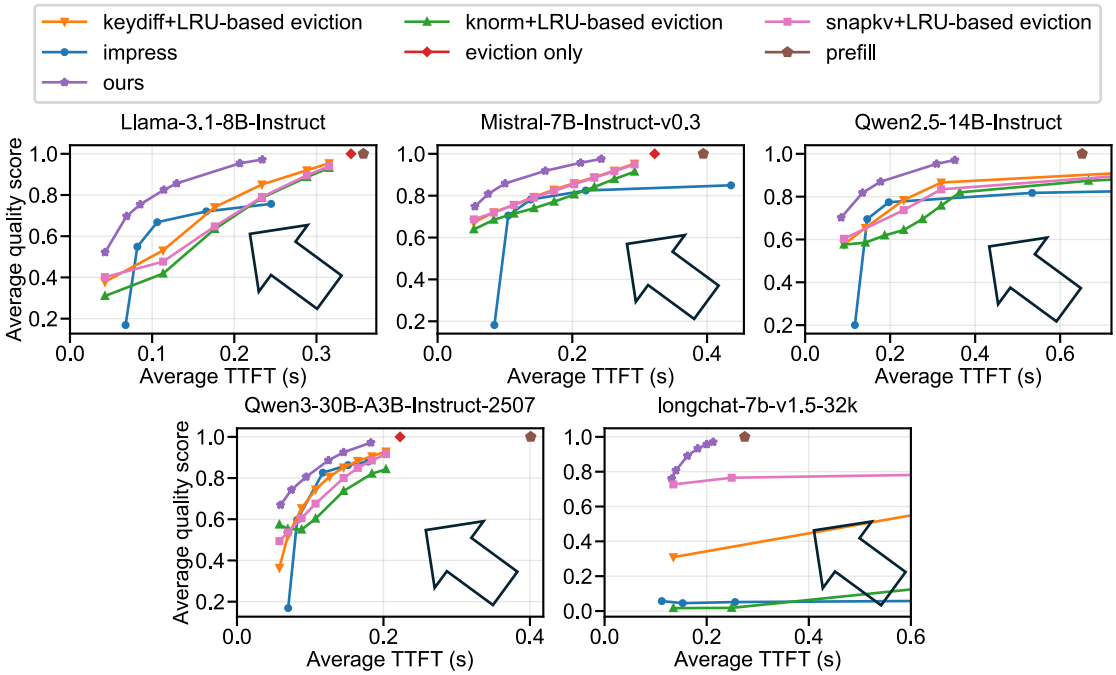


图7：EVICPRESS 与压缩结合基于 *LRU* 的驱逐相比，在质量分数相同的情况下，可将首个令牌生成时间缩短 *1.43* 至 *3.77*× ；与仅进行预填充和驱逐的方案相比，可将首个令牌生成时间缩短 *1.22* 至 *1.56*×。

大语言模型：我们涵盖了多种广受欢迎的开源大语言模型，包括稠密大语言模型，如 Llama（meta-llama/Llama-3.1-8B-Instruct）、Qwen（Qwen/Qwen2.5-14B-Instruct）、LongChat（lmsys/longchat-7b-v1.5-32k）和 Mistral（mistralai/Mistral-7B-Instruct-v0.3），以及混合专家大语言模型（Qwen/Qwen3-30B-A3B-Instruct-2507）。

数据集： 我们使用 LongBench 基准测试 [3],作为评估数据集。LongBench 汇集了许多长上下文数据集，涵盖多种上下文类型（包括政府报告、新闻、对话和故事）。数据集统计信息（包括上下文长度的均值和标准差）以及数据集描述列于表 2。

查询构造： 我们从 LongBench 数据集中随机抽取了 555 个上下文。对于每个上下文，我们提示 GPT-5 生成 100 个问答查询（遵循先前的工作 [10,47–49, 57]），其中 50 个作为训练查询，用于构建 EVICPRESS 分析器，另外 50 个用于测试。

硬件： 我们的实验在一块 80GB H100 GPU 上运行，每个模型的模型权重均由该 GPU 承载。我们分配 80GB CPU DRAM 和 800GB 固态硬盘，并假设远程存储空间无限。

基线方法：我们将 EVICPRESS 与采用 KV 缓存压缩或驱逐的基线方法进行比较。

- 预填充：该基线对每个查询执行完整预填充，不使用前缀缓存。我们使用 vLLMv0.11.2 运行该方法。
- 仅驱逐：当某个存储层级已满时，该系统会使用最近最少使用策略，将 KV 缓存驱逐到下一个存储层级，最近最少使用（LRU）策略，不进行任何压缩。
- *(keydiff/knorm/snapkv)* 压缩 + 基于最近最少使用的驱逐：该基线会对所有上下文应用 keydiff [35], knorm [9]或 snapkv [27]压缩。我们还会调整每种方法的压缩比，从而形成质量与首词生成时间权衡。在某个压缩比下，当存储层级已满时，系统会使用最近最少使用策略，将压缩 KV 缓存驱逐到低层级存储。在这些方法中，keydiff 会计算每个词元与所有其他词元的平均余弦相似度，并丢弃相似度普遍较高的词元；knorm 会计算键缓存的 L2范数，并丢弃 L2范数较低的词元；snapkv 会计算查询附近的词元与较早上下文之间的交叉注意力，并丢弃交叉注意力分数较低的词元。
- *IMPRESS*：IMPRESS [5] 使用八个注意力头中的三个来计算注意力分数，以分析 KV 缓存中的重要令牌。IMPRESS 保留注意力分数位于所有令牌前X%的重要令牌中的X%。随后，它将每个 KV 缓存划分为数据块，并且在推理期间仅加载包含重要令牌的 KV 缓存数据块。我们在实验中调整X%，以实现 IMPRESS 的质量-延迟权衡。

评估指标：

- 质量分数：衡量基于压缩 KV 缓存生成的答案与基于未压缩 KV 缓存生成的答案之间的相似度。按照先前的研究 [2,12,40],，我们使用余弦相似度计算生成

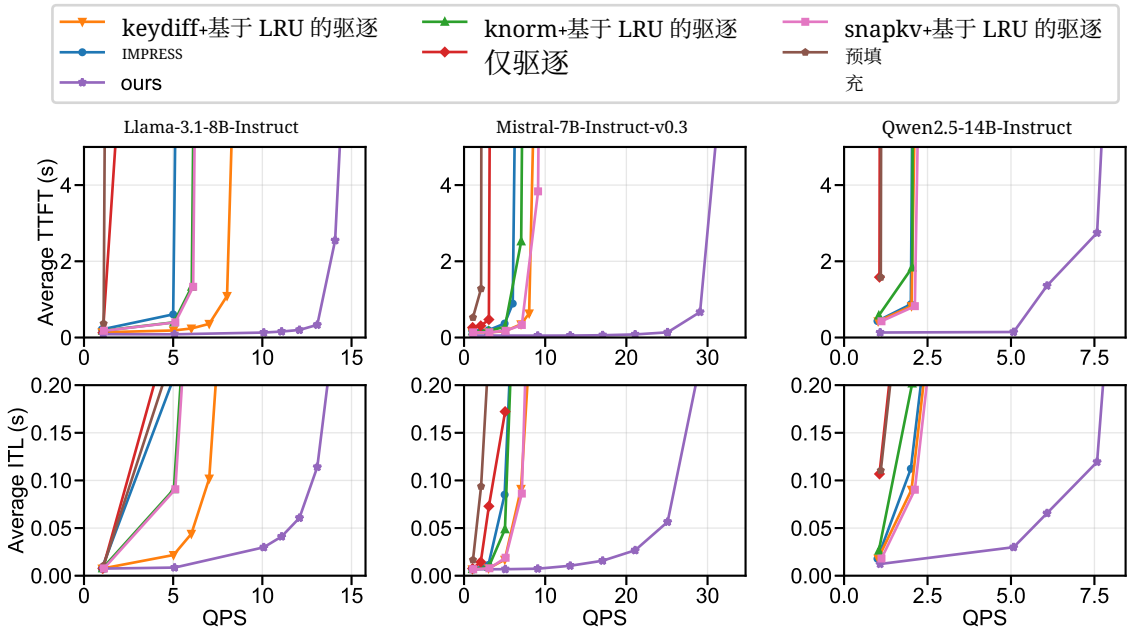


图8：EVICPRESS 在所有每秒查询数水平下，始终保持较低的首个令牌生成时间和令牌间延迟。

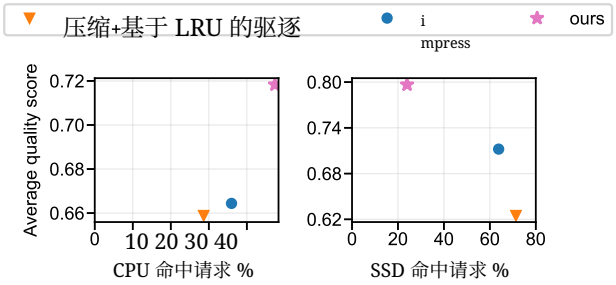


图9：使用 *Qwen2.5-14B-Instruct* 时，EVICPRESS 在保持较高质量的同时，拥有更高的快速设备（*CPU*）命中请求比例。

的答案与基于未压缩 KV 缓存生成的答案的嵌入之间的相似度，所用模型为 *MiniLM-L6-v2* 。该分数越高，生成的答案就越接近基于未压缩 KV 缓存生成的答案，效果也就越好。

- 首令牌生成时间（*TTFT*）是指从查询到达到生成首个词元所需的时间。这包括从存储设备检索 KV 缓存的延迟，以及预填充新查询所需的时间。
- 令牌间延迟（*ITL*）是连续生成词元的平均延迟。ITL 反映编码吞吐量；当推理引擎中运行的请求增多时，ITL 会增加。
- 端到端延迟 是指从收到请求到接收最后一个令牌所需的时间。

6.2 评估结果

首个令牌生成时间与生成质量： 在图 7中，与采用 keydiff、knorm、snapkv 和基于 LRU 的驱逐的基线方法相比，在质量分数相同的情况下，EVICPRESS 将首个令牌生成时间缩短了 1.43 至 3.77× 倍；在首个令牌生成时间相同的情况下，质量分数提高了 13.58% 至 55.40%。与不会影响生成质量的完整预填充相比，在质量下降不超过 3% 的情况下，EVICPRESS 将首个令牌生成时间缩短了 1.29 至 2.19× 倍。与基于 LRU 的驱逐相比，在质量下降小于 3% 的情况下，EVICPRESS 将首个令牌生成时间缩短了 1.22 至 1.56× 倍。最后，与 IMPRESS 相比，EVICPRESS 在不降低生成质量的情况下将首个令牌生成时间缩短了 1.5 至 5.2× 倍；或者在不增加首个令牌生成时间的情况下，将生成质量提高了 14.29% 至 27.00%。

首个令牌生成时间和令牌间延迟与每秒查询数的关系： 图8 展示了在质量分数目标为 80% 时，Llama-3.1-8B-Instruct、Mistral-7B-Instruct- v0.3 和 Qwen2.5-14B-Instruct 在不同每秒查询数（QPS）下的首个令牌生成时间和令牌间延迟。在首个令牌生成时间相同的情况下，与表现最佳的基线相比，EVICPRESS 的请求处理速率可提高 2.0 至 3.6× 倍。同样，在令牌间延迟相同的情况下，与表现最佳的基线相比，EVICPRESS 的每秒查询数可提高 1.9 至 3.4× 倍。

EVICPRESS的改进： EVICPRESS 因不同原因优于不同的基线方法。首先，与对所有上下文采用相同压缩方法和压缩比例的基线方法相比，EVICPRESS 考虑了上下文敏感性差异，并针对不同上下文调整驱逐压缩配置，因此表现更优。其次，与基于 LRU 的驱逐相比，后者会将所有无法容纳在 CPU DRAM 中的上下文驱逐至低层级存储，而 EVICPRESS

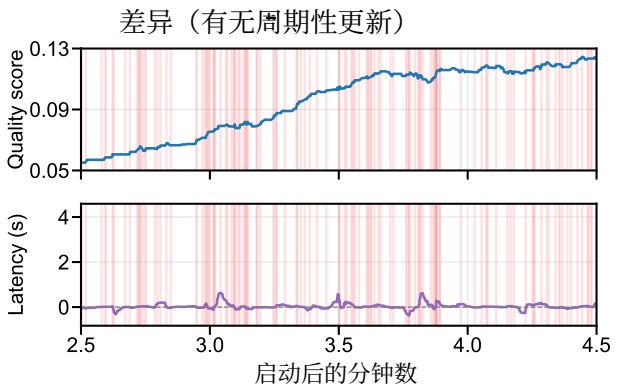


图 10：*Azure* 工作负载下随时间变化的质量和延迟 *[41,46,58]* 每个上下文仅使用一个训练查询。红色竖线表示使用 *Llama-3.1-8B-Instruct* 时的周期性分析事件。在开始阶段以及约 *3.75–4* 分钟附近进行的频繁分析会引入短暂的延迟尖峰，但延迟会迅速恢复至基线水平。随着分析修正过时的压缩决策，模型质量在整个执行过程中稳步提升。

表现更优，因为部分低敏感性上下文可以在不被驱逐至慢速存储的情况下进行大幅压缩。第三，与会产生大量 GPU 计算的完整预填充相比，EVICPRESS 无需执行预填充计算，因此表现更优。最后，与 IMPRESS 相比，后者将 KV 缓存划分为数据块，不仅会将每个数据块中的重要词元加载至高速设备，也会加载不重要的词元；而 EVICPRESS 仅将能够最大化效用函数分数总和的 KV 缓存放置在高速设备上。

总之，EvicPress 在 CPU DRAM 中实现了高得多的缓存命中率，如图 9 所示；相比之下，基线方法对所有上下文和 IMPRESS 应用相同的压缩方法以及基于最近最少使用的驱逐策略。另一方面，如图右侧所示，EVICPRESS 在本地固态硬盘上的缓存命中率低于其他基线方法，这表明大多数缓存命中的词元都位于 CPU DRAM 中，从而比基线方法提供快得多的加载延迟。

我们承认，EVICPRESS 的改进效果取决于不同大语言模型的 KV 缓存大小。当 KV 缓存较小时，动态随机存取存储器可以容纳大部分词元，改进空间有限。例如，MoE 模型 Qwen3-30B-A3B-Instruct-2507 的 KV 缓存较小（每 1K 个词元为 0.0915 GB），而较小的密集模型 Llama-3.1-8B-Instruct 每 1K 个词元的 KV 缓存为 0.12GB，因此即使采用保守的压缩比，大部分 KV 缓存也可以存储在 CPU DRAM 中，从而实现较低的首个令牌生成时间。

6.3 微基准测试

周期性更新的影响： 图 10 使用 Azure 推理跟踪中的真实时间戳评估了我们的周期性性能分析机制（§4.3） [46]。由于 Azure 推理跟踪不包含请求的具体内容，我们改用生成的请求（如 §4.3）进行实验。随后，当使用已存储的 KV 缓存解码查询所得的性能分析质量与预测质量之间的差异超过 0.3 时，EVICPRESS 执行性能分析。

我们展示了使用周期性分析与不使用周期性分析时的时间序列，见图10：

- 上图展示了使用周期性分析与不使用周期性分析时，质量增益如何随时间变化。从图中可以看出，质量增益会随时间逐渐增加，最终稳定在约 11%，这是因为周期性分析使 EVICPRESS 能够使用最新的性能分析结果来做出近乎最优的压缩决策，而不进行周期性更新时，性能分析结果会逐渐过时，导致在相同存储资源预算下准确率降低。

- 下图展示了 EVICPRESS 请求的端到端延迟随时间的变化。可以看出，平均而言，在周期性分析间隔（红色区间）内，EVICPRESS 的端到端延迟较高，因为周期性分析需要 GPU 资源，并会减慢现有请求的执行速度。尽管如此，周期性分析带来的质量提升（约 10%）仍足以抵偿其成本。

不同上下文需要不同的驱逐压缩配置： 图 11 展示了所选驱逐压缩配置的分布。我们观察到，系统确实采用了多种压缩配置，这表明庞大的搜索空间不仅是必要的，也得到了积极利用。我们重点说明两点：

- 磁盘上的 KV 缓存实际上需要比 CPU 上的 KV 缓存更高的平均压缩比。这是因为从磁盘读取 KV 缓存的速度较慢，因此 EVICPRESS 需要进一步压缩 KV 缓存，以缩短其加载时间。
- 我们注意到，尽管 *keydiff* 通常优于 *snapkv*（而 *snapkv*又优于 *knorm*），但仍有相当一部分 KV 缓存采用 *snapkv*，这是因为它在特定上下文中具有优势。

在异构上下文中的改进： 图 12 展示了 EVICPRESS 与基线方法在 LongBench 数据集不同子任务上的首个令牌生成时间—平均分数权衡。我们得出两点结论：

- EVICPRESS 在对应较短上下文的任务上提升更大，例如多项_新闻和 QASPER。这是因为，为了获得更高的 CPU 缓存命中率，EVICPRESS 会将较短的上下文放入 CPU，而不是较长的上下文，这样 CPU 就能容纳更多请求的 KV 缓存

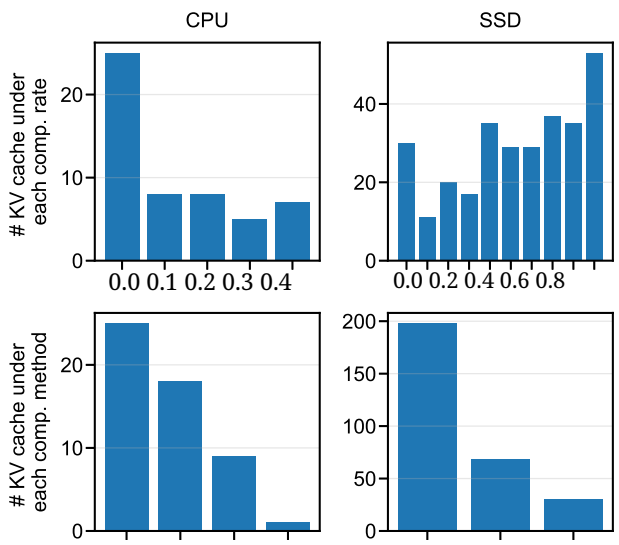


图 11： 在$\alpha = 1$ 条件下，使用 *Mistral-7B-Instruct-v0.3* 时，*CPU* 和固态硬盘存储层级中所选压缩方法及压缩率的分布。EVICPRESS 积极采用了广泛的配置组合，验证了采用大搜索空间的必要性。

。

• EVICPRESS 在 12 项任务中的 11 项上提升了性能。唯一的例外是 MuSiQue；在该任务上，与采用 LRU 驱逐的基线相比，EVIC-PRESS 的表现略差于 keydiff。

7 相关工作

KV 缓存复用与驱逐系统：在多个请求之间存储和复用 KV 缓存，一直是设计高性能大语言模型服务系统的重点。这包括 GPU 上的分页内存管理，例如 vLLM [24] 和 SGLang [60]。然而，当请求到达率较高且并发用户数量较多时，这类工作会面临缓存命中率较低的问题，因为 GPU 内存的容量有限（A100 或 H100 GPU 的显存为 40 至 80 GB）。因此，研究人员提出了更多方案，将无法放入 GPU 内存的 KV 缓存驱逐到低层级存储设备，例如 CPU DRAM [26, 29, 52, 60], 本地磁盘 [22,29,56],甚至远程磁盘 [22,29]。例如，LMCache [21, 29] 复用会将每个 KV 缓存拆分为多个数据块，并利用高性能 CUDA内核优化从 CPU/磁盘加载到 GPU 内存的速度；Mooncake [37] 和 InstInfer [34] 则优化不同 GPU 集群之间的加载带宽，以及预填充与解码分离的性能；IMPRESS 采用细粒度的词元级驱逐策略，通过一部分注意力头从 KV 缓存中识别出重要词元，并将其确定后，将不重要的词元驱逐到低层级存储 [5,50]。然而，这类工作只是在 KV 缓存上采用不同的驱逐方案，甚至没有考虑结合压缩方案来提高缓存命中率。

KV 缓存压缩： 为提高缓存命中率，许多研究探索了正交的方法来压缩 KV 缓存，以减少内存占用。令牌丢弃方法会从 KV 缓存中丢弃不太重要的词元，依据可以是注意力分数或学习得到的重要性信号 [6, 38, 59]。量化方法通过使用更少的比特来表示每个 KV 缓存元素，从而减小 KV 缓存的大小，其中包括自适应量化（对不同层应用不同级别的量化） [17,31] 以及均匀量化 [32]。类似地，这类研究只考虑压缩的可能性，而不将其与驱逐结合，即使质量下降严重也是如此。另一类研究会根据各上下文的误差敏感性采用不同的压缩级别 [7, 51]。然而，与 EVICPRESS 不同，这类研究没有考虑各上下文之间的敏感性差异，也没有将其用于驱逐与压缩的联合优化问题。

更快的大语言模型服务：另一类互补的工作通过改进调度，减少不同推理阶段（例如，预填充、解码或工具调用）之间的空闲时间，从而加速大语言模型服务 [1, 8]；通过预填充-解码解耦，将预填充阶段和解码阶段安排到不同的 GPU 上，以降低服务级别目标违规率 [36, 61]；通过推测解码方法，使用更小、更快的草稿模型提出一系列未来词元，再由更大的目标模型并行验证这些词元 [25, 33]；以及通过分块预填充实现搭载式解码，并降低预填充的内存消耗 [1]。这类工作与 EVICPRESS 互为补充且相互正交，可以与 EVICPRESS 结合，进一步提升大语言模型推理的效率。

其他与 KV 缓存相关的优化：还有许多针对大语言模型推理中 KV 缓存的优化工作，包括带近似的非前缀键值缓存共享 [14, 54]、通过部分层重新计算实现跨大语言模型键值缓存复用 [30]、考虑 KV 缓存位置的请求路由 [43]、改进分页注意力以适配 KV 缓存形状各异的多模态模型 [44]，以及为稀疏注意力模型管理 KV 缓存 [4]。这类工作也让 KV 缓存在大语言模型推理的多种应用场景中更具实用性；EVICPRESS 与这些工作互为补充，因为它们共同优化了驱逐和压缩。

8 局限性与未来工作

部署范围与假设：EVICPRESS 面向一种常见且具有挑战性的部署场景：GPU 和 CPU 内存有限，且采用多级存储层次结构，KV 缓存经常争夺容量，驱逐或压缩决策会切实影响端到端延迟

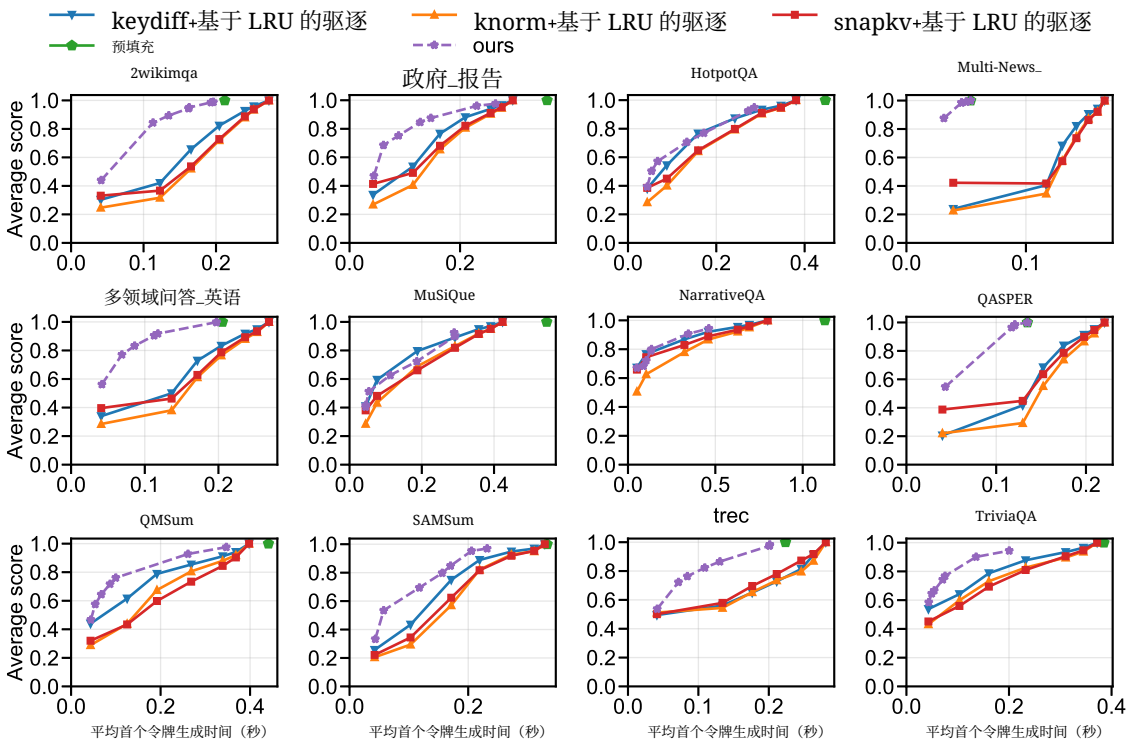


图 12：EVICPRESS 在不同数据集上持续改善首个令牌生成时间—质量权衡。在这些数据集中，EVICPRESS 在较短的上下文（如 *multi_新闻*和 *QASPER*）上提升更显著。

和质量。在动态随机存取存储器充足的情况下（例如，KV 缓存较小的模型，或所有 KV 缓存都能保留在快速内存中的工作负载），复杂的联合优化相较于简单策略的优势自然会减弱。此外，当快速和慢速存储设备的加载带宽相近时，EVICPRESS 联合优化的优势也会降低。我们的原型系统在单节点 GPU 环境中进行了评估。将 EVICPRESS 扩展到多节点部署、跨 GPU 缓存共享、具有严格隔离要求的高度多租户环境，以及其他服务系统和存储层级，是未来工作的重要方向。

适应加载带宽变化：EVICPRESS 目前假设每个存储层级的带宽已知，且在我们分析和重新分析配置的时间段内相对稳定。当可用带宽的变化速度快于我们的重新分析频率，或者带宽波动与突发查询到达高度相关时，预先计算的效用函数分数可能会暂时偏离实际的质量—延迟权衡，并导致次优决策。作为未来工作的一部分，我们计划扩展 EVICPRESS，使其能更快地响应带宽变化，纳入更丰富的在线遥测数据，并共同设计带宽感知分析与调度策略。

扩展到更丰富的压缩方法集合：在评估部分，EVICPRESS 从以下几种压缩方法中选择：这些方法会丢弃 KV 缓存中的重要令牌，以缩减其内存占用。这些方法可以轻松扩展 EVICPRESS，使其能够从 KV 缓存量化方法中进行选择 [16, 17, 32]。类似地，我们可以构建一个配置分析池，结合令牌丢弃和量化，并调整要丢弃的令牌比例以及用于表示每个 KV 元素的比特数。

9 结论

我们介绍了 CacheServe，这是一种用于大语言模型推理工作负载的缓存系统，能够在分层存储设备上协同设计压缩与驱逐策略，从而实现更优的质量-时延权衡。CacheServe 的核心创新在于，它通过定义效用函数并选择能够使效用最大化的压缩-驱逐方案，联合利用驱逐和压缩各自的优势。我们基于生产级大语言模型服务引擎（vLLM）和缓存引擎（LMCache）实现了该系统。针对多种模型和数据集的评估表明，与基线方法质量相当时，CacheServe 可将平均首个令牌生成时间缩短 1.43 至 3.77×；在首个令牌生成时间相同时，则可将质量提升 13.58% 至 55.40%。通过设计兼容未来 KV 缓存压缩与驱逐技术的前向兼容架构，EVICPRESS 为大语言模型推理优化开辟了新维度。

参考文献

[1] Amey Agrawal、Ashish Panwar、Jayashree Mohan、Nipun Kwatra、Bhargav S. Gulavani 和 Ramachandran Ramjee。《Sarathi：通过将解码与分块预填充相结合实现高效的大语言模型推理》，2023年。

[2] Negar Arabzadeh、Amin Bigdeli 和 Charles L. A. Clarke。调整标准检索基准以评估生成答案，2024年。

[3] Yushi Bai、Xin Lv、Jiajie Zhang、Hongchang Lyu、Jiankai Tang、Zhidian Huang、Zhengxiao Du、Xiao Liu、Aohan Zeng、Lei Hou、Yuxiao Dong、Jie Tang 和 Juanzi Li。LongBench：用于长上下文理解的双语多任务基准测试。*arXiv* 预印本 *arXiv:2308.14508*，2023年。

[4] Ziyi Cao、Qingyi Si、Jingbin Zhang 和 Bingquan Liu。跨多个上下文 KV 缓存的稀疏注意力，2025年。

[5] Weijian Chen、Shuibing He、Haoyang Qu、Ruidong Zhang、Siling Yang、Ping Chen、Yi Zheng、Baoxing Huai 和 Gang Chen。 {IMPRESS}：面向大语言模型推理的 {重要性指导}{多层} 前缀 {KV} 存储系统。载于第 *23* 届 *USENIX* 文件与存储技术会议（*FAST 25*），第 187–201 页，2025年。

[6] Yilong Chen、Guoxia Wang、Junyuan Shang、Shiyao Cui、Zhenyu Zhang、Tingwen Liu、Shuohuan Wang、Yu Sun、Dianhai Yu 和 Hua Wu。NACL：一种适用于大语言模型推理时的通用有效 KV 缓存驱逐框架，2024年。

[7] Wen Cheng、Shichen Dong、Jiayu Qin 和 Wei Wang。QAQ：面向大语言模型 KV 缓存的质量自适应量化。载于*IEEE/CVF* 国际计算机视觉会议论文集，第 2542–2550 页，2025年。

[8] Weihao Cui、Yukang Chen、Han Zhao、Ziyi Xu、Quan Chen、Xusheng Chen、Yangjie Zhou、Shixuan Sun 和 Minyi Guo。通过 PD 多路复用优化面向服务级目标的大语言模型服务，2025年。

[9] Alessio Devoto、Yu Zhao、Simone Scardapane 和 Pasquale Minervini。一种简单有效的 l_2 范数策略，用于 KV 缓存压缩，2024年。

[10] Sabri Eyuboglu、Ryan Ehrlich、Simran Arora、Neel Guha、Dylan Zinsley、Emily Liu、Will Tennien、Atri Rudra、James Zou、Azalia Mirhoseini 和 Christopher Re。Cartridges：通过自我学习实现轻量且通用的长上下文表示，2025年。

[11] Shaoting Feng、Hanchen Li、Kuntai Du、Zhuohan Gu、Yuhan Liu、Jiayi Yao、Siddhant Ray、Samuel Shen、Yihua Cheng、Ganesh Ananthanarayanan 和 Junchen Jiang。Adaptcache：面向低延迟、高质量语言模型服务的 KV 缓存原生存储层级，2025年。

[12] Harsha R Gaikwad 和 Arvind W Kiwelekar。用于评估简答题和长答题的生成式人工智能助手。*SN Computer Science*，5(5):633，2024年。

[13] Bin Gao、Zhuomin He、Puru Sharma、Qingxuan Kang、Djordje Jevdjic、Junbo Deng、Xingkun Yang、Zhou Yu 和 Pengfei Zuo。Attentionstore：在大型语言模型服务中跨多轮对话实现经济高效的注意力复用。*arXiv* 预印本 *arXiv:2403.19708*，52:20–38，2024年。

[14] In Gim、Guojun Chen、Seung seob Lee、Nikhil Sarda、Anurag Khandelwal 和 Lin Zhong。提示缓存：用于低延迟推理的模块化注意力复用，2023年。

[15] Zhuohan Gu、Jiayi Yao、Kuntai Du 和 Junchen Jiang。Llmsteer：通过引导复用上下文上的注意力改进长上下文大语言模型推理，2024年。

[16] Yefei He、Luoming Zhang、Weijia Wu、Jing Liu、Hong Zhou 和 Bohan Zhuang。Zipcache：通过显著令牌识别实现准确高效的 KV 缓存量化，2024年。

[17] Coleman Hooper、Sehoon Kim、Hiva Mohammadzadeh、Michael W Mahoney、Yak un Sophia Shao、Kurt Keutzer 和 Amir Gholami。Kvquant：探索通过 KV 缓存量化实现千万级上下文长度的大语言模型推理。*arXiv* 预印本 *arXiv:2401.18079*，2024年。

[18] Bo Jiang、Taolue Yang、Youyuan Liu、Chengming Zhang、Xubin He 和 Sian Jin。Kvcomp：一种面向大语言模型的高性能 KV 缓存有损压缩框架，2025年。

[19] Huiqiang Jiang、Qianhui Wu、Chin-Yew Lin、Yuqing Yang 和 Lili Qiu。LLMLingua：压缩提示以实现大语言模型的加速推理。*arXiv* 预印本 *arXiv:2310.05736*，2023年。

[20] Huiqiang Jiang、Qianhui Wu、Xufang Luo、Dongsheng Li、Chin-Yew Lin、Yuqing Yang 和 Lili Qiu。LongLLMLingua：通过提示压缩加速并增强长上下文场景中的大语言模型，2023年。

[21] Junchen Jiang、Yuhan Liu、Zhuohan Gu、Qizheng Zhang、Chen Wang、Yue Zhu、Ruoyu Qin、Yuwei An、Xiangfeng Zhu、Kuntai Du 等。面向有状态大语言模型推理的网络技术。在 *ACM*数据通信特别兴趣组会议，2025年。

[22] Shuowei Jin、Xueshen Liu、Qingzhao Zhang 和 Z. Morley Mao。计算还是加载 KV 缓存？为什么不两者兼用？，2025年。

[23] Jang-Hyun Kim、Jinuk Kim、Sangwoo Kwon、Jae W. Lee、Sangdoo Yun 和 Hyun Oh Song。Kvzip：与查询无关的 KV 缓存压缩及上下文重构，2025年。

[24] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Jose ph E. Gonza- lez、Hao Zhang 和 Ion Stoica。《通过分页注意力实现大语言模型服务的高效内存管理》，2023年。

[25] Yaniv Leviathan、Matan Kalman 和 Y. Matias。《通过推测解码加速 Transformer 推理》。发表于国际机器学习会议，2022年。

[26] Hanchen Li、Qiuyang Mang、Runyuan He、Qizheng Zhang、Huanzhi Mao、Xiaokun Chen、Alvin Cheung、Joseph Gonzalez 和 Ion Stoica。Continuum：利用 KV 缓存生存时间实现高效、稳健的多轮大语言模型智能体调度，2025年。[27] Yuhong Li、Yingbing Huang、Bowen Yang、Bharat Venkitesh、Acyr Locatelli、Hanchen Ye、Tianle Cai、Patrick Lewis 和 Deming Chen。snapkv：大语言模型在生成前就知道你要找什么，2024年。[28] Akide Liu、Jing Liu、Zizheng Pan、Yefei He、Gholamreza Haffari 和 Bohan Zhuang。MiniCache：面向大语言模型的深度维度 KV 缓存压缩，2024年。[29] Yuhan Liu、Yihua Cheng、Jiayi Yao、Yuwei An、Xiaokun Chen、Shaoting Feng、Yuyang Huang、Samuel Shen、Rui Zhang、Kuntai Du 和 Junchen Jiang。LMCache：用于企业级大语言模型推理的高效 KV缓存层，2025年。[30] Yuhan Liu、Yuyang Huang、Jiayi Yao、Shaoting Feng、Zhuohan Gu、Kuntai Du、Hanchen Li、Yihua Cheng、Junchen Jiang、Shan Lu、Madan Musuvathi 和 Esha Choukse。DroidSpeak：用于跨大语言模型通信和多大语言模型服务的 KV缓存共享，2025年。[31] Yuhan Liu、Hanchen Li、Kuntai Du、Jiayi Yao、Yihua Cheng、Yuyang Huang、Shan Lu、Michael Maire、Henry Hoffmann、Ari Holtzman 等。CacheGen：面向语言模型应用的快速上下文加载。*arXiv* 预印本 *arXiv:2310.07240*，2023年。[32] Zirui Liu、Jiayi Yuan、Hongye Jin、Shaochen Zhong、Zhaozhuo Xu、Vladimir Braverman、Beidi Chen 和 Xia Hu。KIVI：一种无需调优的 KV 缓存非对称 2 位量化方法。*arXiv* 预印本 *arXiv:2402.02750*，2024年。[33] Xupeng Miao、Gabriele Oliaro、Zhihao Zhang、Xinhao Cheng、Zeyu Wang、Rae Ying Yee Wong、Zhuoming Chen、Daiyaan Arfeen、Reyna Abhyankar 和 Zhihao Jia。SpecInfer：通过推测推理和令牌树验证加速生成式大语言模型服务。*arXiv* 预印本 *arXiv:2305.09781*，2023年。

[34] Xiurui Pan、Endian Li、Qiao Li、Shengwen Liang、Yizhou Shan、Ke Zhou、Yingwei Luo、Xiaolin Wang 和 Jie Zhang。《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》，2024年。

[35] Junyoung Park、Dalton Jones、Matthew J Morse、Raghavv Goel、Mingu Lee 和 Chris Lott。《keydiff：面向资源受限环境下长上下文大语言模型推理的基于键相似度的键值缓存淘汰》，2025年。

[36] Pratyush Patel、Esha Choukse、Chaojie Zhang、Aashaka Shah、Íñigo Goiri、Saeed Maleki 和 Ricardo Bianchini。《Splitwise：通过阶段拆分实现高效生成式大语言模型推理》，2024年。[37] Ruoyu Qin、Zheming Li、Weiran He、Mingxing Zhang、Yongwei Wu、Weimin Zheng 和 Xinran Xu。《Moon-cake：以 KV 缓存为核心的解耦式大语言模型服务架构》，2025年。[38] Ziran Qin、Yuchen Cao、Mingbao Lin、Wen Hu、Shixuan Fan、Ke Cheng、Weiyao Lin 和 Jianguo Li。《Cake：结合层偏好的级联式自适应键值缓存淘汰》，2025年。[39] Siddhant Ray、Rui Pan、Zhuohan Gu、Kuntai Du、Shaot- ting Feng、Ganesh Ananthanarayanan、Ravi Netravali 和 Junchen Jiang。《Metis：通过配置自适应实现快速且质量感知的检索增强生成系统》。载于*ACM SIGOPS 第31届操作系统原理研讨会论文集*，第606–622页，2025年。[40] Nils Reimers 和 Iryna Gurevych。《Sentence-BERT：使用孪生 BERT 网络生成句子嵌入》。载于*2019年自然语言处理实证方法会议论文集*。计算语言学协会，2019年11月。[41] Mohammad Shahrad、Rodrigo Fonseca、Inigo Goiri、Go- har Chaudhry、Paul Batum、Jason Cooke、Eduardo Lau- reano、Colby Tresness、Mark Russinovich 和 Ricardo Bianchini。《真实环境中的无服务器：在大型云服务提供商处对无服务器工作负载进行特征分析与优化》。载于*2020年USENIX年度技术会议（USENIX ATC 20）*，第205–218页，2020年。[42] Woomin Song、Seunghyuk Oh、Sangwoo Mo、Jaehyung Kim、Sukmin Yun、Jung-Woo Ha 和 Jinwoo Shin。《层次化上下文合并：提升预训练大语言模型的长上下文理解能力》，2024年。[43] Vikranth Srivatsa、Zijian He、Reyna Abhyankar、Dong- ming Li 和 Yiying Zhang。《Preble：面向大语言模型服务的高效分布式提示调度》，2024年。

[44] Dezhan Tu、Danylo Vashchilenko、Yuzhe Lu 和 Pan- pan Xu。《Vl-cache：用于视觉语言模型推理加速的稀疏性与模态感知键值缓存压缩》，2024年。

[45] Zhongwei Wan、Ziang Wu、Che Liu、Jinfa Huang、Zhi- hong Zhu、Peng Jin、Longyue Wang 和 Li Yuan。Look- m：KV 缓存中的单次查看优化，用于高效多模态长上下文推理，2024年。

[46] Yuxin Wang、Yuhan Chen、Zeyu Li、Xueze Kang、Yuchu Fang、Yeju Zhou、Yang Zheng、Zhenheng Tang、Xin He、Rui Guo、Xin Wang、Qiang Wang、Amelie Chi Zhou 和 Xiaowen Chu。Burstgpt：用于优化大语言模型服务系统的真实世界工作负载数据集，2025年。

[47] Jerry Wei、Da Huang、Yifeng Lu、Denny Zhou 和 Quoc V. Le。简单的合成数据可降低大语言模型的谄媚倾向，2024年。

[48] Jerry Wei、Chengrun Yang、Xinying Song、Yifeng Lu、Nathan Hu、Jie Huang、Dustin Tran、Daiyi Peng、Ruibo Liu、Da Huang、Cosmo Du 和 Quoc V. Le。大语言模型的长篇事实准确性，2024年。

[49] Colin White、Samuel Dooley、Manley Roberts、Arka Pal、Ben Feuer、Siddhartha Jain、Ravid Shwartz-Ziv、Neel Jain、Khalid Saifullah、Sreemanti Dey、Shubh-Agrawal、Sandeep Singh Sandha、Siddartha Naidu、Chinmay Hegde、Yann LeCun、Tom Goldstein、Willie Neiswanger 和 Micah Goldblum。Livebench：一种具有挑战性、污染受限的大语言模型基准测试，2025年。

[50]Zhiqiang Xie、Ziyi Xu、Mark Zhao、Yuwei An、Vikram Sharma Mailthody、Scott Mahlke、Michael Garland 和 Christos Kozyrakis。Strata：面向长上下文语言模型服务的分层上下文缓存。*arXiv* 预印本 *arXiv:2508.18572*，2025年。

[51] Jing Xiong、Jianghan Shen、Fanghua Ye、Chaofan Tao、Zhongwei Wan、Jianqiao Lu、Xun Wu、Chuanyang Zheng、Zhijiang Guo、Min Yang、Lingpeng Kong 和 Ngai Wong。Uncomp：矩阵熵能否揭示稀疏性？——一种源自不确定性感知视角的压缩器设计，2025年。

[52] Yi Xiong、Hao Wu、Changxu Shao、Ziqing Wang、Rui Zhang、Yuhong Guo、Junping Zhao、Ke Zhang 和 Zhenxuan Pan。Layerkv：通过逐层键值缓存管理优化大语言模型服务，2024年。

[53] Jingbo Yang、Bairu Hou、Wei Wei、Yujia Bao 和 Shiyu Chang。Kvlink：通过高效的 KV 缓存复用加速大语言模型，2025年。

[54] Jiayi Yao、Hanchen Li、Yuhan Liu、Siddhant Ray、Yi- hua Cheng、Qizheng Zhang、Kuntai Du、Shan Lu 和 Junchen Jiang。Cacheblend：通过缓存知识融合实现快速的检索增强生成大语言模型服务，2025年。

[55] Bohan Yu 和 Yekun Chai。Evolkv：面向大语言模型推理的进化式 KV 缓存压缩，2025年。

[56]Huawei Zhang、Chunwei Xia 和 Zheng Wang。Kvswap：面向长上下文端侧推理的磁盘感知型KV缓存卸载，2025年。

[57] Xinrong Zhang、Yingfa Chen、Shengding Hu、Zihang Xu、Junhao Chen、Moo Khai Hao、Xu Han、Zhen Leng Thai、Shuo Wang、Zhiyuan Liu 和 Maosong Sun。∞ bench：将长上下文评估扩展至超过 100k 词元，2024年。

[58] Yanqi Zhang、Íñigo Goiri、Gohar Irfan Chaudhry、Ro- drigo Fonseca、Sameh Elnikety、Christina Delimitrou 和 Ricardo Bianchini。利用采集的资源实现更快速、更低成本的无服务器计算。载于*ACM SIGOPS* 操作系统原理第 *28* 届研讨会论文集，第 724–739 页，2021年。

[59] Zhenyu Zhang、Ying Sheng、Tianyi Zhou、Tianlong Chen、Lianmin Zheng、Ruisi Cai、Zhao Song、Yuandong Tian、Christopher Ré、Clark Barrett 等。H2o：面向大语言模型高效生成式推理的重击中项预言机。神经信息处理系统进展，2024年，第36卷。

[60] Lianmin Zheng、Liangsheng Yin、Zhiqiang Xie、Jeff Huang、Chuyue Sun、Cody Hao Yu、Shiyi Cao、Chris- tos Kozyrakis、Ion Stoica、Joseph E Gonzalez 等。使用 SGLang 高效编程大语言模型。*arXiv* 预印本 *arXiv:2312.07104*，2023年。

[61] Yinmin Zhong、Shengyu Liu、Junda Chen、Jianbo Hu、Yibo Zhu、Xuanzhe Liu、Xin Jin 和 Hao Zhang。Dist- serve：为优化有效吞吐量而解耦大语言模型服务中的预填充与解码。*arXiv* 预印本 *arXiv:2401.09670*，2024年。

[62] Xiabin Zhou、Wenbin Wang、Minyan Zeng、Jiaxian Guo、Xuebo Liu、Li Shen、Min Zhang 和 Liang Ding。Dynamickv：面向长上下文大语言模型的任务感知型自适应 KV缓存压缩，2025年。

[63] Jianian Zhu、Hang Wu、Haojie Wang、Yinghui Li、Biao Hou、Ruixuan Li 和 Jidong Zhai。Fastcache：通过轻量级KV缓存压缩框架优化多模态大语言模型服务，2025年。

算法伪代码

算法 1： 策略优化器

输入： 待存储的传入 KV 缓存i（大小为 KV 缓存l_i，质量函数Q_i（压缩比）），存储队列T，当前占用大小S_T，存储内存大小C

输出： KV 缓存上的操作列表：[(KV 缓存i, 压缩比 r_i，操作o_i)]

$r_i \leftarrow$最大化$Q_i(r)$
T追加(i, r_i)
L.append($(i, r_i$,插入$)$)
$S_T = S_T +$大小$of(l_i * r_i)$
当$S_T > C$ **时**
    $j, r, o \leftarrow$在T中效用下降最小的 KV 缓存操作和该操作。
    如果 $o ==$ 压缩**则**
        $S = S_T - T_j * (r_j - r)$
    否则如果$o ==$ 驱逐 **则**
        $r \leftarrow 0$
        $S = S_T - (T_j * r_j)$
    结束 if
    L.追加$((j, r, o))$
结束 while
返回 L 给执行器。