ContiguousKV：通过粒度对齐的键值缓存管理加速大语言模型预填充

Jing Zou * *ShangyuWu** *Hancong Duan Qiao Li*† *Chun Jason Xue* 电子科技大学 穆罕默德·本·扎耶德人工智能大学 电子科技大学 穆罕默德·本·扎耶德人工智能大学 穆罕默德·本·扎耶德人工智能大学

arXiv:2601.13631v1 [cs.OS] 20 Jan 2026

摘要

使用持久化前缀键值（KV）缓存高效服务大语言模型（LLM），对于对话式搜索和多轮对话等应用至关重要。处理请求需要加载预先计算的前缀 KV 缓存并生成首个词元，这一过程称为重新预填充阶段。将此共享前缀缓存卸载到二级存储，对于实现内存扩展性至关重要。采用卸载的重新预填充会在两个方面遭遇严重的输入/输出瓶颈。首先，语义感知键值缓存剪枝算法以细粒度选择重要词元，而系统以粗粒度的固定大小块管理输入/输出，导致严重的读取放大。其次，识别重要词元与加载 KV 缓存之间的顺序依赖会造成输入/输出空闲和计算气泡，使系统资源利用不足。

本文提出了*ContiguousKV*，一种高性能前缀键值缓存卸载系统，通过连接算法语义与 I/O效率，加速重新预填充阶段。我们首先介绍连续块，这是一种统一的数据管理粒度，可使键值缓存剪枝与 I/O 操作相协调。所有对 I/O 性能至关重要的机制均以连续块为粒度执行，从而消除读放大。通过利用不同层之间重要连续块索引的高度相似性，我们提出了周期内和跨周期异步预取，以打破 I/O 与计算之间的顺序依赖，有效消除空闲气泡。最后，我们提出注意力引导的缓存管理，将语义上至关重要的前缀数据保留在内存中。在 Qwen2.5 系列模型上的评估表明，与最先进的卸载系统 IMPRESS 相比，ContiguousKV 在重新预填充阶段实现了 3.85 倍加速，同时保持较高的输出质量。

1 引言

日益增多的大语言模型（LLM）服务工作负载，包括对话助手 [10, 46]，检索增强生成（RAG） [15, 40], 和交互式文档分析 [2],都具有共享上下文这一特点。多个用户请求（例如，针对检索到的文档提出后续问题）会共享一个很长且完全相同的前缀（文档文本）。为避免重复计算，系统可以持久存储该共享前缀的预计算键值（KV）缓存，并在多个请求间复用 [6,11,43]。这种复用引入了一个至关重要但成本高昂的阶段：重新预填充阶段。重新预填充阶段通过从较慢的存储层级（例如 CPU 内存或固态硬盘）加载共享前缀的预计算 KV 缓存、计算新请求中非共享后缀词元的 KV 缓存，并对预计算 KV 缓存执行注意力计算来生成首个词元。

为提升重新预填充阶段的性能，现有最先进的基于卸载的推理系统 [6, 11]会识别出主导模型性能的关键令牌，并从固态硬盘加载其预计算 KV 缓存以执行推理。然而，算法决策与系统操作之间存在根本性的粒度失配。KV 缓存选择算法以语义单元为操作对象——单个令牌或保留语义的较小连续令牌组（数据块）。相比之下，IMPRESS 等卸载系统 [6] 会以粗粒度的固定大小块（例如，包含 64 个令牌、大小为 1.8 MB 的数据块）管理数据，以分摊输入/输出开销。这种失配迫使系统为访问仅有几千字节的相关剪枝 KV 缓存而加载整个数兆字节的数据块，导致严重的读放大和输入/输出带宽浪费。系统的操作粒度无法感知算法语义。此外，重新预填充计算在每一层中本质上都是串行的。计算前必须先加载前缀键，因此剪枝决策与 KV 缓存加载之间存在严格的顺序依赖。这会产生大量空闲“气泡”：图形处理器可能因等待输入/输出而停滞，输入/输出子系统也可能处于空闲状态，表明资源利用率不足，限制了重新预填充的性能。此外，使用传统策略将有限的、更快的内存作为卸载前缀的缓存，忽视了数据的语义价值，导致与当前请求最相关的令牌命中率低。

*这些作者贡献相同。†通讯作者。

我们认为，要加速重新预填充阶段，需要进行协同设计：使系统的输入/输出粒度与算法的 KV 缓存剪枝粒度对齐，以消除输入/输出浪费；并协调异步预测式输入/输出，从而打破顺序依赖并消除空闲气泡。我们提出**ContiguousKV**，这是一个专为优化重新预填充阶段而设计的系统。ContiguousKV 围绕一种核心抽象连续块构建，该抽象作为重要性评分、存储、传输和缓存的统一粒度。这种对齐确保每次输入/输出操作都能准确获取算法所需的语义连贯单元，从而消除读放大。在此基础上，我们发现，ContiguousChunk 中的重要令牌索引集在各个 Transformer 层之间具有很强的相似性。我们利用这一特性，设计了周期内与周期间异步预取引擎。当前层进行计算时，该引擎会推测性地获取后续层的 KV 缓存，打破顺序依赖，并将重新预填充阶段转变为流水线化的高吞吐量过程。为优化更快内存层级的使用，我们提出了一种缓存策略，可动态优先缓存注意力分数较高的连续块；注意力分数是衡量当前请求上下文中语义重要性的直接指标。这样可确保最有价值的前缀数据始终易于访问，进一步降低慢速输入/输出的深度和频率。

我们在具有大型、可复用前缀 KV 缓存的 Qwen2.5 模型上进行了全面实验。实验表明，与最先进的卸载型推理系统相比，ContiguousKV 在重新预填充阶段实现了 3.85 倍加速，同时保持了较高的准确率。

本文的贡献如下：

- 我们提出了连续块这一统一粒度，将系统输入/输出与算法剪枝对齐，从而消除重新预填充期间的读取放大。
- 我们设计了一种两级异步预取机制，利用跨层相似性，将输入/输出与模型计算流水线化，从而消除空闲气泡。
- 我们开发了一种注意力引导的缓存策略，利用语义重要性提升缓存效率。

本文其余部分安排如下。§2 介绍背景，§3 对重新预填充瓶颈进行详细的动机分析。§4 介绍了所提出的连续 KV 设计。实验结果见§5。§6展示了相关工作，§7总结了本文。

2 背景

2.1 现代大语言模型基础

大语言模型架构与计算流程。如图1所示，典型的大语言模型 [36,42,49] 由输入层、输出层和多个 Transformer 层堆叠而成。每个 Transformer 层主要由一个自

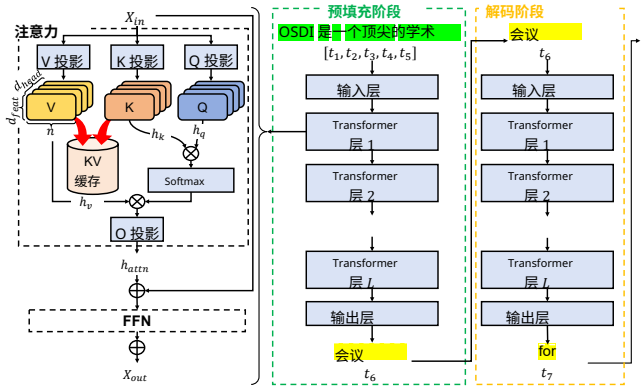


图 1：现代大语言模型的标准架构及其推理阶段（即预填充阶段和解码阶段）。

注意力模块和一个前馈网络（FFN）模块组成。自注意力模块将每个词元的隐藏状态投影为查询（Q）、键（K）和值（V）张量，使模型能够评估每个词元的重要性。随后，前馈网络（FFN）通常采用多层感知机，对每个词元的表示进行独立的非线性变换。这种架构使大语言模型能够捕捉输入中复杂的上下文关系。

形式化地，给定由 n 个词元组成的输入序列 T $= [t_1, t_2, \ldots, t_n]$，模型会通过遍历全部 L 层的前向传播来计算第一个输出词元 t_{n+1}。首先，将输入词元 T 转换为输入张量 $X_{in} \in R^{n \times d_{feat}}$，其中 d feat 是大语言模型的特征维度。然后，将 X_{in} 传入 Transformer 层和输出层，以获得最终输出 X_{out}。对于第 L_l 层，假设输入张量为 X_{in}^l，则将其分别与三个权重矩阵（即 W_q^l、W_k^l 和 W_v^l）相乘，生成注意力模块中的 Q、K 和 V 张量（即 h_q^l、h_k^l 和 h_v^l）。这些张量的形状为 n $\times d_{head} \times$ d feat，其中 d_{head} 维度用于捕捉不同的句法依赖、语义角色、实体关系等 [8, 31, 53]。随后，利用 h_q^l、h_k^l 和 h_v^l 计算注意力输出 $h_{attn}^l = (\text{softmax}(h_q^l \cdot h_k^l) \cdot h_v^l) \cdot W_o^l$，其中 W lo 是输出投影的权重矩阵。经过残差连接后，注意力输出作为前馈神经网络模块的输入。前馈神经网络模块主要利用两个线性投影，并在两者之间应用一个激活函数，从而得到 Transformer 层的输出。由于前馈神经网络模块并非本文重点，我们省略其细节。

推理阶段。如图 1 所示，大语言模型的推理过程可分解为两个不同的阶段，即预填充阶段和解码阶段。**(1) 预填充阶段**：推理从此阶段开始，该阶段将整个输入序列作为大语言模型的提示输入进行处理。对于每一层，生成的键和值张量都会存储为键值（KV）缓存，以提供上下文信息

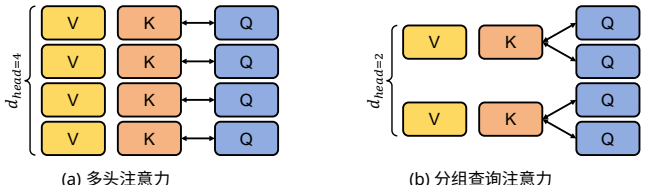


图2：多头注意力与分组查询注意力的比较。

供解码阶段使用。此阶段的输出是首个生成令牌，其主要性能指标是首令牌生成时间（TTFT）。(2) **解码阶段**：模型基于预填充阶段缓存的键值对，开始自回归地生成新词元。在每一步中，模型都会将缓存的键值对与新生成的键值对拼接起来，然后执行注意力运算以生成下一个词元。尽管已有若干研究 [6, 11, 13, 47, 48] 通过将 KV 缓存卸载到主存或磁盘，解决了内存受限挑战，但新的输入/输出受限挑战随之出现。

新兴大语言模型架构：分组查询注意力（GQA）。如图1所示，Transformer 中的基本注意力机制是多头注意力（MHA），它为每个注意力头使用独立的键和值向量。MHA 中键、值和查询的头数相同。为缓解 MHA 导致的显著内存占用，Llama- 等现代顶尖大语言模型3 [36] 和 Qwen2。5 [42],已广泛采用分组查询注意力 [1]。GQA 是一种混合方法，在 MHA 与极端形式的多查询注意力（MQA）之间取得平衡。如图2所示，在 GQA 中，多个查询头分组后共用一个键和值头。这种设计大幅减少了不同 K 和 V 头的数量。

这种架构修改显著减少了键值缓存。例如，基于 MHA 的模型 OPT-66B [49] 使用 64 个头，而基于 GQA 的模型 Qwen2.5-70B [42] 仅使用 8 个头。这使每个令牌的键值数据量减少了八倍。

2.2 使用前缀键值缓存重新预填充

大语言模型应用的兴起，例如检索增强生成 [15, 39, 40, 44], 多轮对话 [5, 7, 9, 33], 长上下文问答（QA） [2,17], 代码补全 [4, 22, 24, 38], 和多智能体系统 [10,21,46],，从根本上改变了服务工作负载。对于检索增强生成或长上下文问答，一个或多个文档可能作为数十个用户查询的上下文。在对话中，冗长的系统提示词或共享聊天历史为多个用户会话提供共同基础。代码补全通常需要读取一个或多个源代码文件，以理解代码逻辑并生成新代码。多智能体

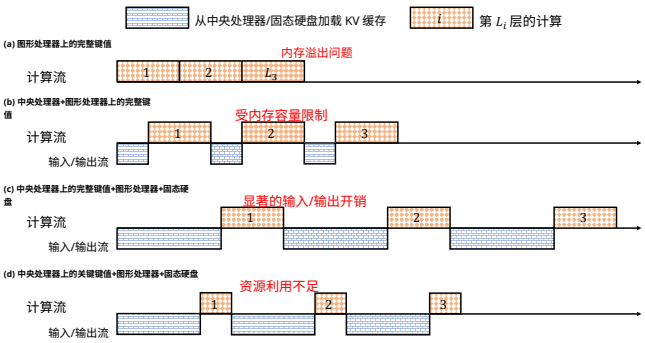


图 3：基于卸载的推理系统范式。

系统会复用中间状态，例如内部代理的输出。在这些应用中，一个关键特征逐渐显现：它们都会产生带有较长共享前缀的请求。

对每个请求都从头开始处理这些共享前缀，计算开销极高，而且通常会导致首令牌生成时间显著变差 [16, 18, 29, 51]。为缓解这一问题，现有基于卸载的推理系统 [6, 11, 16, 19, 43]提出复用预计算的键值缓存（称为共享前缀 KV 缓存），以减少后续请求中的冗余计算。我们可以通过加载预先存储的共享前缀 KV 缓存来处理新请求，然后为新的独有后缀计算键值缓存。与第 2 节介绍的典型预填充或解码阶段不同，这一过程构成了一个独立且关键的阶段，我们将其正式称为 **重新预填充阶段**。

定义 2.1 重新预填充阶段同时受计算和输入/输出限制，在处理新请求时需要：

1. 从较慢的存储层（例如 *CPU* 内存或固态硬盘）加载共享前缀的预计算 *KV* 缓存；*2.* 计算新请求中非共享后缀词元的 *KV* 缓存，并对预计算 *KV* 缓存执行注意力运算，以生成第一个词元。

RePrefill 阶段具有以下区别于传统大语言模型推理阶段的关键特性：**(1)阶段内顺序依赖**。在单次推理过程中，只有先从较慢的存储中加载相应的前缀键值缓存，才能执行每一层的计算；**(2) 资源利用不足**。输入/输出操作与计算并未并行执行，尽管二者并不冲突。这会导致输入/输出资源或计算资源在另一方忙碌时处于空闲状态，严重阻碍系统效率。这些固有特性带来了现有服务系统尚未充分解决的根本性挑战。

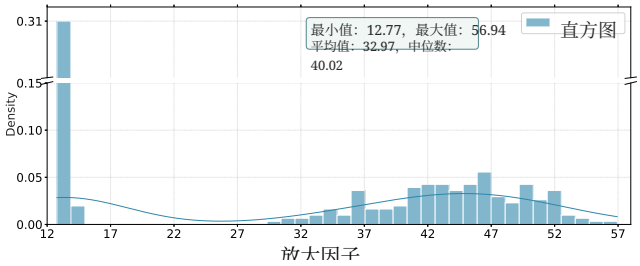


图 4：现有最先进推理系统（例如 IMPRESS [6]）在 RTE 数据集上服务 Qwen2.5-7B 时的读放大。该系统采用 KV 缓存剪枝方法来识别重要的 KV 缓存，KV 缓存预算比例为 25%。

2.3 基于卸载的推理系统

为了容纳长前缀产生的海量键值缓存，现代大语言模型推理系统采用了由 GPU 内存、CPU 主存和固态硬盘组成的多层存储层次结构。我们将其归纳为四种范式（图 3），

- **GPU 上的完整 KV**。 默认的推理系统 [18,51]通常会将完整的 KV 缓存专门存储在图形处理器内存中。该系统通常受图形处理器内存容量限制，因此不适用于具有较长共享前缀的工作负载。
- **GPU 上的完整 KV + CPU**。 为了解决内存溢出问题，一些推理系统 [13,47] 会将 KV 缓存扩展到 CPU 主存，将其作为容量更大的第二级缓存。虽然这缓解了内存容量压力，但仍无法适应 KV 缓存在解码过程中不断增长的情况。
- **GPU 上的完整 KV + CPU + 固态硬盘**。对于超出 DRAM 总容量的长上下文，一些系统 [11,16]会将大部分 KV 缓存卸载到固态硬盘。尽管这提供了近乎无限的存储空间，却会引入显著的输入/输出开销。
- **图形处理器上的关键键值 + 中央处理器，以及固态硬盘上的完整键值**。最新的最先进（SOTA）系统 [6,19] 试图通过有选择地仅加载重要或关键的键值对（或数据块），同时将完整缓存保留在固态硬盘上，来缓解 I/O 瓶颈。这样可以减少需要传输的数据量。然而，这些系统缺乏精心设计的资源管理方法，导致资源利用不足。

这一演进凸显了一个持续存在的矛盾：扩展容量会引入速度较慢的存储层级，其 I/O 开销将主导推理延迟。这些局限在混合 I/O-计算 Re-Prefill 阶段尤为突出，因此需要构建一种与现代选择性 KV 缓存访问模式协同设计的存储系统。

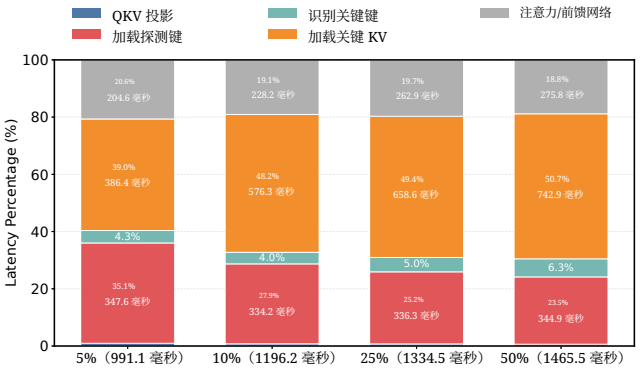


图 5：不同 KV 缓存预算比例下，现有最先进推理系统（IMPRESS [6]）的延迟分解。

3 动机与挑战

3.1 动机

重新预填充阶段已成为共享前缀场景中的关键步骤。然而，现有针对共享前缀键值缓存的大语言模型推理系统（如 Atten- tionStore [11] 和 IMPRESS [6],）未能应对重新预填充阶段的独特特性，尤其是在服务现代大语言模型时。它们的局限源于过时的架构假设和低效的数据管理策略。

动机 1：严重的读放大。最先进的基于卸载的推理系统 [6, 11] 通常以数据块为粒度管理 KV 缓存：将多个词元逻辑分组到一个数据块中，以便与 I/O 接口对齐。为减少内存开销，IMPRESS 采用稀疏 KV 缓存算法，例如 H2O [50],，以减少注意力运算所需的 KV 缓存数量。这意味着 IMPRESS 只能加载包含所需 KV 缓存的数据块。然而，这种基于数据块的方法与稀疏 KV 缓存算法配合使用时，会导致严重的读放大。

我们进行了一项初步实验，以展示 IMPRESS [6] 在 RTE 数据集上服务 Qwen2.5-7B 时的读放大情况。如图 4所示，IMPRESS 在服务大多数请求时（约 31%）会产生约 12 倍的读放大。这些读放大是因为只需要数据块中一小部分重要词元，却必须读取整个数据块。值得注意的是，读放大值可能极高（从 25 到 56）。当这些请求的大部分 KV 缓存前缀数据已经缓存，而其余 KV 缓存分散在多个大型数据块中时，就会出现这种情况。例如，某个请求需要从磁盘额外加载 11 个词元。然而，这些词元分布在 9 个数据块中，导致读放大

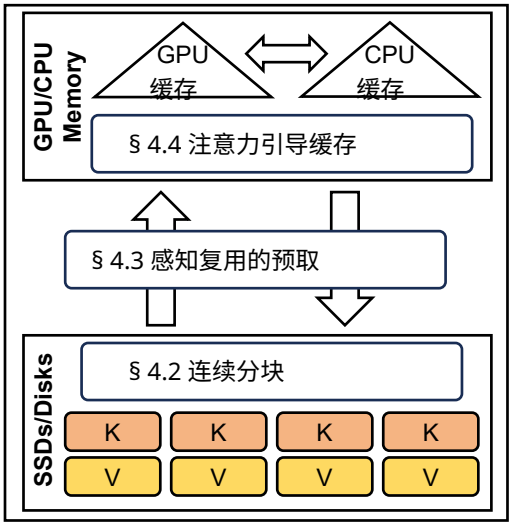


图 6：ContiguousKV 设计概览。

52 的分类。

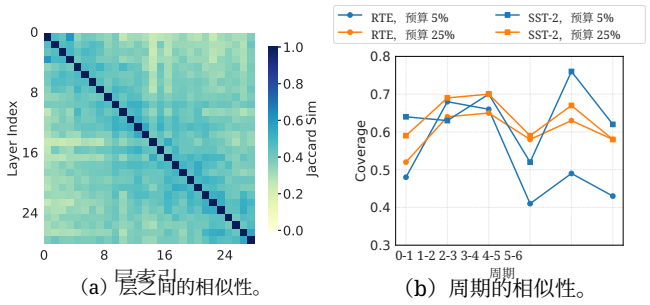
动机 2：资源管理效率低下与利用不足。如第 2.2 节所述，重新预填充阶段存在强烈的阶段内顺序依赖：必须先加载共享前缀每一层的 KV 缓存，才能进行该层的计算。例如，识别关键令牌需要加载前缀键向量，而执行注意力运算则需要加载前缀 KV 缓存。尽管现有系统 [6, 11] 能够在有限程度上重叠这些输入/输出与计算步骤，但图形处理器或 I/O 子系统仍会有相当长的时间处于空闲状态。

我们在图 5 中开展了初步实验，对 IMPRESS 的延迟进行分解，并量化严重的资源利用不足。在各种 KV 缓存预算比例（5%-50%）下，I/O密集型阶段（包括加载探测键和加载关键 KV）占重新预填充阶段的 65% 以上，而计算密集型阶段占比不足 35%。这种不平衡表明，图形处理器在大部分时间都处于空闲状态，等待加载键值数据。即使启用了一定程度的重叠，根本性的顺序依赖仍使 I/O 成为关键路径。

3.2 挑战

上述动机转化为构建高效的基于卸载的推理系统、以支持重新预填充阶段时面临的两个具体设计挑战。

挑战 1：利用硬件能力消除读放大，同时不牺牲带宽。现代存储设备（如 NVMe固态硬盘）支持细粒度随机读取，原则上可以消除因加载包含稀疏重要数据的大块数据而造成的读放大。然而，若不加区分地发出大量



（a）层之间的相似性。　（b）周期的相似性。

图7：不同层与不同周期的重要连续块索引之间的相似性。

细粒度、非顺序的读取请求，会导致 I/O 操作串行化，无法利用底层设备的高顺序带宽。关键挑战在于重新组织数据并设计访问机制，既能细粒度地加载重要数据，又能保留大规模连续读取流，从而维持较高的 I/O 吞吐量。

挑战2：突破顺序依赖，实现有效预取。资源严重利用不足的根本原因在于重新预填充工作负载中计算与输入/输出之间的阶段内顺序依赖。针对当前推理流程、对固定输入/输出和计算阶段进行流水线化的系统级优化已达到极限。因此，关键挑战在于利用大语言模型的内部特征打破这种依赖，从而提前实现有效预取，并最大限度地实现输入/输出与计算重叠。

4 ContiguousKV

4.1 概述

我们提出 ContiguousKV，通过协同的系统设计解决重新预填充瓶颈。图6展示了 ContiguousKV 的概览，其核心包括三项创新：ContiguousChunk 抽象、两级预取引擎和注意力引导缓存控制器。ContiguousChunk 抽象通过创建一个协调一致的单元，使数据管理粒度与算法粒度相匹配，从而解决粒度不匹配问题，并消除仅需特定词元时造成的无效数据传输（§4.2）。我们的两级预取机制利用相似性，将顺序执行的重新预填充过程转变为流水线执行，有效地以计算掩盖输入/输出延迟（§4.3）。注意力引导缓存控制器进一步依据注意力分数，优先处理语义上重要的数据块，从而优化内存利用率（§4.4）。

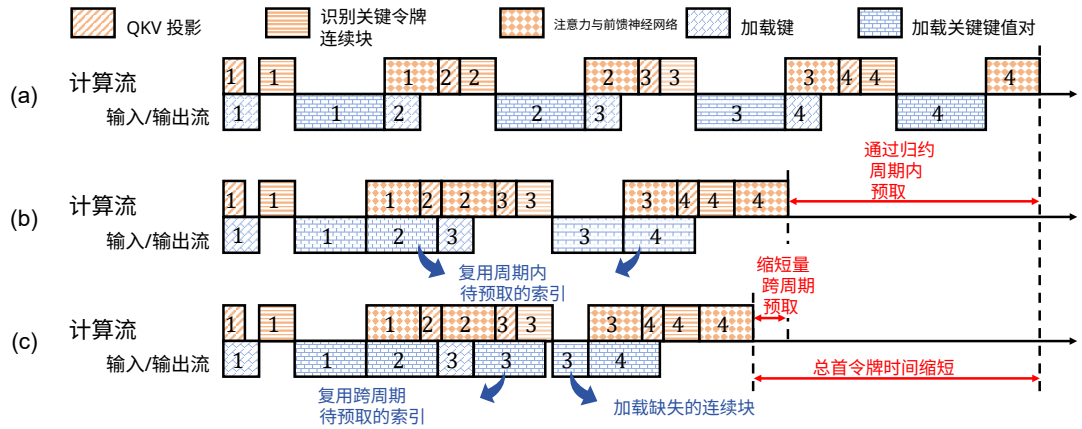


图8：现有基于卸载的推理系统中的默认预取机制与我们的两级预取机制对比。

4.2 连续分块：使系统粒度与算法粒度对齐

基于卸载的大语言模型推理中，KV 缓存管理面临着算法决策与系统操作之间根本性的粒度不匹配。在算法层面，令牌级剪枝方法 [20, 26, 50] 根据注意力分数选择单个词元，却会破坏自然语言中的局部语义连贯性。近期研究 [25, 45]表明，将连续词元组合成组能更好地保留语义上下文，从而在提升输出质量的同时，更有效地压缩 KV 缓存。在系统层面，IM-PRESS 等分块级卸载框架 [6]以较粗的数据块（例如数兆字节的块）管理 KV 缓存，以摊销输入/输出开销；但当每个数据块中只需少量内容时，这会造成严重的读放大（图4）。我们引入**ContiguousChunk**这一统一粒度，使系统输入/输出与算法剪枝相匹配，以解决性能瓶颈。

定义 4.1 连续块 是一组连续的词元。形式上，对于长度为 n 的输入前缀，我们将其划分为 m 个 $= \lceil n/c \rceil$ 连续块，每个连续块最多包含 c 个词元（例如，$c = 16$）。

连续块建立了统一的粒度，用于规范所有关键操作：存储组织、缓存管理和 I/O 预取。这种一致的粒度消除了不同层采用不同粒度的系统所固有的低效问题。与 IMPRESS 等系统不同，后者以粗粒度数据块管理存储，却必须以更细的粒度（例如词元粒度）处理驱逐和选择；我们的设计确保驱逐单元、存储单元和预取单元均采用连续块粒度。这种对齐是实现高效性的根本。

考虑一种典型配置：对于 Qwen2.5-7B 模型，每个词元的键值缓存约占 28 KB（3584 个隐藏维度 $\times$ 4 个注意力头 $\times$ 2每个浮点数所占字节数）。这已经超过了典型的 SSD 页大小（4KB）。当词元被分组成较大的数据块时，例如先前系统中每块包含 64 个词元 [6, 11], 每个数据块约包含 1,800 KB 的键值数据。如果重要性选择机制只识别出块内少数几个关键词元（例如 16 个词元），或者采用较低的预算比例（如 5%），那么加载整个数据块可能会导致读放大率达到 4。相比之下，连续块粒度将块大小设为 16 个词元，使系统输入/输出与算法剪枝保持一致。因此，采用基于数据块的键值缓存剪枝算法后，读取一个重要的连续块不会产生读放大。这种对齐确保从存储中读取的几乎每个字节都能用于计算，从而消除带宽浪费并最大限度地提高输入/输出效率。

4.3 复用感知异步预取

本节将介绍复用感知异步预取技术。我们首先展示在采用固定的周期性复用策略时，不同层之间的相似性。然后，我们介绍周期内或周期之间的预取策略。

4.3.1 关键连续块索引的相似性

与先前的工作类似 [25, 45], ，我们发现，重要连续块索引的集合在不同 Transformer 层之间保持高度相似。因此，我们采用相同的策略，将一组连续的层简单地分组，称为 **周期**，以共享相同的连续块索引。

定义 4.2 周期 是指一组连续的层，它们共享相同的重要连续块索引。每个周期包含 p 层（例如 $p = 4$）。

每个周期使用的重要连续块索引在该周期的第一层确定，识别过程详见第 4.2 节。同一周期内其余层复用相同的连续块索引。

周期之间的相似性。图7(a) 展示了各层之间的相似性，可以观察到，相邻层的索引往往相似，而相距较远的层则有所不同。我们在图7(b) 中对 Qwen2.5-7B 进行了观察实验，以证明不同周期中的重要连续块索引也具有很高的相似性。我们使用覆盖率来表示相似性，该指标通过计算相邻周期之间相同连续块索引的数量得出。结果显示，周期之间的平均相似性为 52% 至 64%。这表明，在周期之间复用部分连续块索引也具有很大潜力。利用这种高度相似性，我们将介绍预取机制，以在计算期间隐藏输入/输出开销。

4.3.2 周期内/周期间预取机制

图8(a) 以一个四层大语言模型为例，展示了现有系统中重新预填充阶段的执行时间线。每一层的计算都遵循严格的顺序依赖关系。首先，系统从固态硬盘加载前缀的键向量；与此同时，系统执行 QKV 投影，以获取当前层的查询向量。在查询和前缀键都加载到 GPU 内存后，系统会识别该层的关键词元。最后，系统加载这些词元对应的键值缓存，以完成注意力计算。尽管在识别出关键词元后可以预取下一层的前缀键 [6, 11],但一个基本依赖关系——在识别出第l层的关键词元之前，无法加载第l 层的键值缓存——会导致图形处理器和输入/输出资源都出现显著的空闲时段。这些空闲的“气泡”使输入/输出与计算串行执行，导致系统资源利用不足并限制吞吐量。

为从根本上打破这种顺序依赖并消除空闲气泡，我们设计了一种结合周期内预取与跨周期预取的策略，利用各层间可复用的关键连续块索引，如第3节所述。其核心思想是协调异步输入/输出操作，使其以两种不同粒度领先于计算执行：在一个周期内（周期内预取）以及在连续周期之间（跨周期预取）。该流水线能够有效隐藏输入/输出延迟、减少空闲气泡，并最大限度地提高输入/输出与计算资源的并行利用率。

周期内预取：将输入/输出与计算重叠。通过复用同一组连续块，我们的周期内预取机制会预取关键连续块，以便将输入/输出操作与层计算流水线化，如图8（b）所示。以一个包含 4 层的大语言模型为例，周期为 2 层时的流程如下：

1. 在一个周期内的第一层（L_1），系统执行标准步骤：加载前缀键、计算查询，并确定关键连续块索引。重要的是，这组索引只需确定一次，便会在同一周期内的所有后续层中复用。2. 确定索引后，系统立即发出异步I/O请求，为同一周期内所有层的所有关键连续块加载相应的键值缓存。值得注意的是，不同层的加载过程依次进行，可以与每一层的计算流水线化。3. 当正在从存储中获取层L_1 的键值缓存时，层L_1 的计算继续进行。

4. 对于该周期中的每个后续层（L_2），其所需的 KV 缓存很可能已预取至 GPU 内存中，从而消除了逐层等待输入/输出的时间。后续层 KV 缓存的预取与前面层的计算以流水线方式并行进行。

跨周期预取：预热周期边界。虽然周期内预取消除了同一周期内各层之间的气泡，但每个周期开始时仍存在一个明显的气泡：第一层必须先加载其前缀键并完成计算，之后才能进行任何识别。为缓解这一问题，我们利用了连续周期之间关键连续块索引的相似性（见图7）。我们采用一种轻量级的推测式预取策略，如图8(c)所示：

1. 在一个周期（例如周期 i，但第一个周期除外）正式开始计算之前，我们会异步预取前一个周期（$i-1$）中关键连续块的 KV 缓存，供周期 i 中后续的层使用。2. 周期 i 开始后，一旦其第一层明确确定了自身的关键连续块集合，系统只需加载差集，即尚未预取的缺失连续块。这样，最终的加载步骤就会大幅缩短。

这种跨周期预取起到了“预热”作用，显著减少了初始输入/输出空隙。结合使用周期内和跨周期预取机制，可确保图形处理器计算和 I/O 带宽近乎持续得到利用，从而将重新预填充阶段从顺序执行的瓶颈转变为高度流水化、高效的过程。

4.4 基于注意力的缓存管理

尽管周期内/周期间预取机制有效降低了输入/输出延迟和空闲时间，但图形处理器和 CPU 内存的容量有限，因此必须采用智能缓存管理，以最大限度地提高关键键值数据的命中率。基于卸载的系统中现有的缓存策略，例如最近最少使用、LFU，甚至 IMPRESS 中基于分数的策略 [6],都依赖访问频率和静态重要性比率等粗粒度统计数据，无法捕捉缓存数据细微的、依赖查询的

语义重要性，而这种重要性正是模型自身的注意力机制所反映的。

为解决这一问题，我们提出一种注意力感知的缓存策略，根据注意力感知缓存分数来管理连续块。该策略根据连续块对模型推理的贡献（以其注意力分数 A_j 量化）及其访问频率 F_j，计算连续块 j 的缓存分数 S_j。这一缓存分数有助于缓存以整体且语义感知的方式管理连续块。

与现有的基于数据块的 KV 缓存选择方法类似 [25],，我们通过累加每个连续块内所有词元的重要性分数，计算各连续块的重要性分数。例如，给定一个输入，我们首先使用标准注意力机制计算词元级注意力分数 $h_{qk} = \mathrm{softmax}(h_q \cdot h_k)$，其中 $h_{qk} \in R^{n \times n}$。然后，将 h_{qk} 沿第二个维度求和，得到第 i 个词元的重要性分数 a_i。第 j 个连续块的重要性分数 A_j，即第 $c \cdot (j-1)$ 个至第 $c \cdot j$ 个词元的注意力分数之和：

$$A_j = a_{c\cdot(j-1)} + a_{c\cdot(j-1)+1} + \cdots + a_{c\cdot j-1} \tag{1}$$

对于缓存中存储的每个连续块 j，我们维护两个动态值：

- **基于累积注意力的重要性**I_j。初始时，I_j 的值为零。处理完一个请求后，我们记录计算中涉及的每个关键连续块 A_j 的注意力分数，该分数根据公式 *1* 计算。然后，将新计算出的连续块注意力分数 A_j 加到 I_j 上，以更新其值，即：$I_j = I_j + A_j$。

- **访问频率**（F_j）。我们记录连续块 j 被访问的次数（即加载到图形处理器内存或 CPU 内存中的次数）。

然后，缓存分数是其重要性与频率的乘积：

$$S_j = I_j \times F_j \tag{2}$$

基于缓存分数，我们使用两个最小堆来管理图形处理器和中央处理器上的连续块。具体而言，在推理系统初始化阶段，我们会在中央处理器内存中创建两个空的最小堆。推理过程中，每次预取过程开始前，我们都会使用已识别出的重要连续块索引，检查相应的数据块是否已缓存。对于尚未缓存的数据块，我们会调度输入/输出线程从固态硬盘加载，如第 4.3.2 节所示。加载完成后，这些连续块会被传输到图形处理器进行计算，同时更新其缓存分数。随后，根据分数将它们插入图形处理器堆或中央处理器堆。两个堆都会淘汰分数较低的连续块。值得注意的是，从图形处理器堆中淘汰的连续块，要么被插入中央处理器堆（如果其分数仍相对较高），要么被彻底从内存中淘汰。我们将所有缓存分数（包括那些被从内存中淘汰的连续块的分数）都存储在一个内存表中。

表 1：前缀的详细配置。

数据集	# 示例	长度	大小（GB）		
			7B	14B	32B
SST-2	100	3.8k	67	226	301
SubJ	110	4.4k	67	230	305
TREC	120	5k	75	257	343
RTE	80	6k	68	233	311

4.5 实现细节

由于现有最先进的卸载推理系统（例如 IMPRESS、AttentionStore）均未开源，我们基于一个知名的公开可用大语言模型卸载推理框架 FlexGen [32]构建了 ContiguousKV。我们对其核心实现进行了大量修改，以支持现代大语言模型架构（例如 Qwen2.5 系列），并集成新颖的 KV 缓存选择算法（例如 H2O [50],ChunkKV [25]）。具体来说，我们实现了两个关键类：*InferenceEngine*，用于系统粒度对齐和两级预取机制；以及 *CacheManager*，用于注意力引导的缓存管理。

我们的实现精心管理系统资源，在不引入显著开销的情况下最大限度地提高效率。首先，异步预取的数据不会占用超出预设内存的额外空间。相反，我们将给定的内存预算（例如 10 GB）划分为专用缓存区域和较小的预取缓冲区。在我们的实验中，在总内存限制为 10 GB 的情况下，我们仅在图形处理器和中央处理器上分别分配 0.2 GB 和 0.4 GB 作为预取缓冲区。其次，为了控制输入/输出与计算之间的重叠并最大限度地提高资源利用率，我们引入了一个关键参数，即子周期_大小，该参数决定了在当前周期的计算开始之前必须加载多少层的键值缓存。

5 评估

5.1 实验设置。

模型与系统配置。我们使用 Qwen2.5 系列模型评估 ContiguousKV [42]，涵盖三个规模：Qwen2.5-7B、Qwen2.5-14B 和 Qwen2.5-32B。所有实验均在一台服务器上进行，该服务器配备 2 颗 Intel Xeon Platinum 8370C 中央处理器（共 64 个核心）、128 GB DRAM、一块配备 80 GB HBM2e 内存的 NVIDIA A800 图形处理器，以及一块 4 TB Samsung 990 Pro NVMe 固态硬盘。测得该固态硬盘的峰值读取带宽约为 7.45 GB/s。该图形处理器

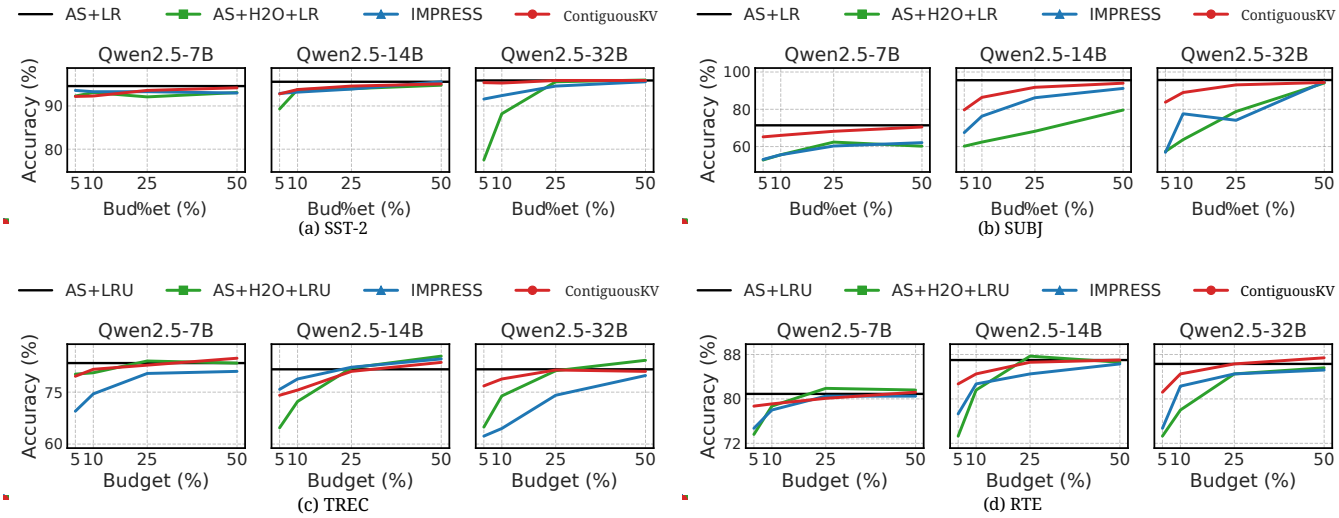


图9：四个数据集和三种大语言模型上不同推理系统的准确率。

通过 PCIe 4.0 ×16 与主机相连，测得的双向带宽约为 32 GB/s。

数据集和指标。 我们选择四种具有代表性的自然语言理解任务来模拟重新预填充阶段：用于情感分析的 SST-2 [34]、用于主观性分类的 Subj [28]、用于问题分类的 TREC [37]，以及用于文本蕴含的 RTE [3]。我们以基于提示的少样本方式评估大语言模型在这些任务上的表现 [12, 39]。这些数据集非常适合评估重新预填充阶段，因为它们被设计为基于提示的生成任务，通常只需模型生成一个令牌或很短的序列（例如标签词）。这一特性最大限度地降低了解码阶段对端到端延迟的影响，从而确保我们的性能测量（首令牌生成时间）主要反映共享前缀处理及相关输入/输出-计算编排的效率，也就是本文的核心关注点。我们按照先前工作 [12, 39]中的做法，以准确率、平均首令牌生成时间和尾延迟（P95 首令牌生成时间）衡量结果。我们在四种不同的 KV 缓存预算比例下开展实验，即 5%、10%、25% 和 50%。

基线推理系统。 我们将 ContiguousKV 与为重新预填充阶段设计的最先进基于卸载的推理系统进行比较：

- *AS*+最近最少使用 [11]: AttentionStore（AS）通过多级存储系统，使用完整的前缀键值缓存为大语言模型提供推理服务。

- *AS+H2O*+最近最少使用：我们沿用先前工作中使用的基线 [6],，该基线将 AS+最近最少使用与重要键值选择算法（例如 H2O [50]）相结合。不同于 AS+ 最近最少使用，该基线仅将重要词元加载到内存中，而不是加载完整的前缀键值缓存。

- *IMPRESS* [6]: 当前最先进的基于卸载的推理系统，可选择性地加载部分用于重要性识别的键，并采用基于分数的缓存策略。

由于这些基线方法均未开源，我们根据其论文，在能力范围内对其进行了重新实现。对于 AS+最近最少使用和 AS+H2O+LFU，我们禁用了其面向调度器的优化，使其更适用于一般场景，例如抢占式调度环境，这与 IMPRESS 的论文 [6]一致。

配置。 我们从训练数据集的每个类别中采样数据，将其作为少样本示例，以构造共享前缀。采样过程符合正态分布。提示模板与先前工作相同 [12, 39]。前缀的详细配置列于表 1 中。SemChunk 的块大小 c 设置为 16，连续 KV 预取的周期大小 p 设置为 8 层，子周期大小 sp 设置为 4 层。AS+LRU、AS+H2O+LRU 和 IMPRESS 使用的块大小均设置为 64 个词元。所有实验均使用固定的随机种子 42。对于所有基线方法，我们将可用的图形处理器内存设置为 10GB，将 CPU 内存设置为 24GB。

5.2 端到端性能

模型性能。 图9 展示了 ContiguousKV 与现有基于卸载的推理系统在四个数据集和三个大语言模型上的模型性能。由于 AS+LRU 会加载完整的 KV 缓存，因此其模型性能不会随着预算比例的增加而变化。总体而言，在几乎所有配置下，与采用稀疏 KV 缓存算法的这些系统相比，ContiguousKV 的性能最佳。具体来说，在不同的 KV 预算比例下，与 IMPRESS 相比，ContiguousKV 的准确率平均分别提升了 7.69%、4.81%、3.58% 和 1.63%

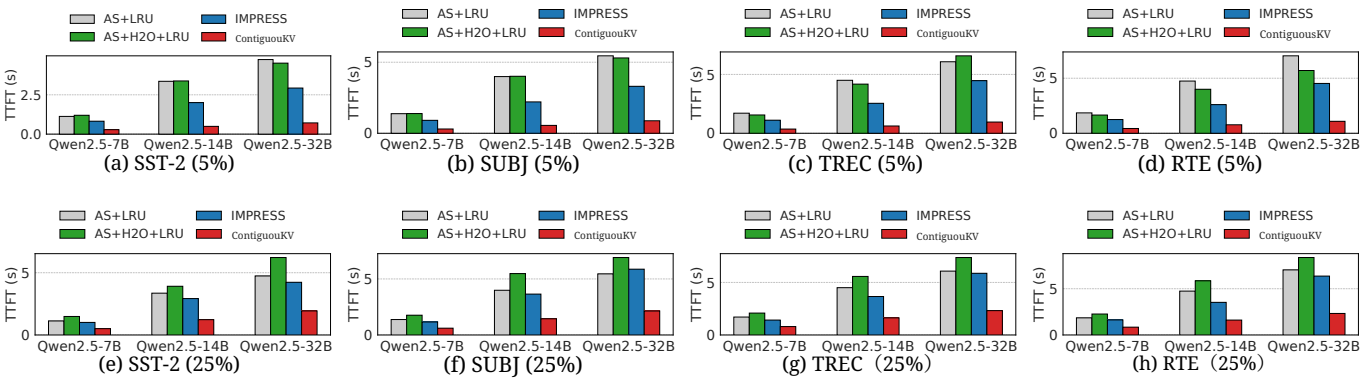


图 10：四个数据集和三种大语言模型上不同推理系统的平均首令牌生成时间。我们报告了两种键值缓存预算比例（5% 和 25% ）下的结果。

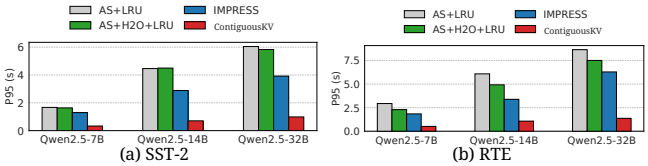


图11：键值缓存预算比例为 5% 时，不同推理系统在代表性数据集上的 P95 尾延迟。

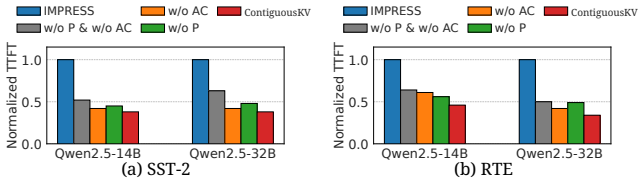


图12：各项优化措施影响的消融研究。

与 IMPRESS 相比，准确率平均提升了 7.69%、4.81%、3.58% 和 1.63%；与 AS+H2O+LFU 相比，平均提升了 10.23%、7.08%、3.56% 和 2.04%。在不同预算比例下，与 AS+LRU 相比，ContiguousKV 的准确率平均分别下降了 5.62%、3.56%、1.21 % 和 0.04%。随着模型规模扩大，ContiguousKV 在所有模型规模下均优于现有推理系统。尤其是在较大的模型（如 Qwen2.5-32B）上，ContiguousKV 的提升更为显著：与 IMPRESS 和 AS+H2O+LFU 相比，分别提升了 7.71% 和 6.99%。与 AS+LRU 相比，ContiguousKV 的性能降幅最高为 2.94%。这些结果表明，ContiguousKV 的 SemChunk 能为大语言模型保留足够的语义，从而维持可接受的性能。

平均首令牌生成时间和 P95 尾延迟。在评估之前，我们会预热所有基线方法的图形处理器和中央处理器缓存。我们在两个具有代表性的预算比例（5% 和 25%）下，评估 ContiguousKV 与其他系统的平均首令牌生成时间。如图 10 所示，在 5% 键值预算比例下，ContiguousKV 相较于 AS 16×、AS83×和85×分别实现了时间归约+LRU、AS+H2O+ LFU 和 IMPRESS。图 10中还可以观察到，在 25% 预算比例下速度也保持一致提升，但这些加速比低于 5% 键值预算比例下的结果。就模型规模而言，ContiguousKV 在不同模型规模下实现了相近的加速比。这些

结果表明，我们的 ContiguousKV 能够在较低预算比例下实现最大加速。这是因为 ContiguousKV 不会引入读放大，从而节省了输入/输出带宽。

我们还在图 11 中，使用两个具有代表性的数据集评估了 ContiguousKV 及其他系统的 P95 尾延迟。为更清晰地展示结果，我们将 KV 缓存预算比例设置为 5%。与现有系统相比，我们的 ContiguousKV 显著降低了不同模型的 P95 尾延迟，平均分别为 0.42 秒、0.89 秒和 1.17 秒。这些结果表明，我们的 ContiguousKV 具有更好的稳定性和鲁棒性。

5.3 消融研究

单项优化的影响。 为评估 ContiguousKV 中所提出的各项技术的影响，我们使用两个模型（Qwen2.5-14B 和 Qwen2.5-32B）在两个数据集（RTE 和 SST-2）上进行了消融研究。预算比例为 25%。图 12展示了消融研究结果。我们测试了 ContiguousKV 的三种变体：

1. **不使用 P 且不使用 AC**：ContiguousKV 不进行预取，并默认使用 LFU 作为缓存策略。2. **不使用 AC**：ContiguousKV 使用 LFU 作为缓存策略。3. **不使用 P**：ContiguousKV 不进行预取。

图 12 中的实验结果表明，与 IMPRESS 相比，主要的性能提升源于连续分块，

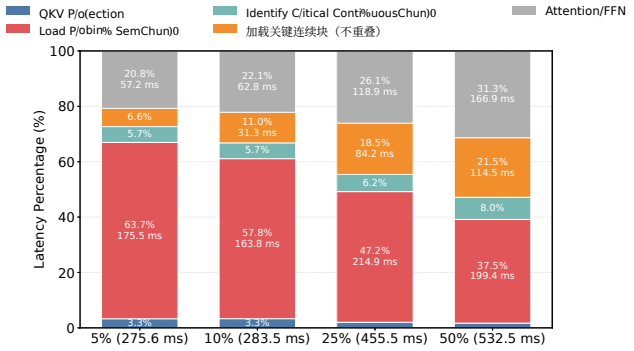


图 13：ContiguousKV 的延迟分解。

这种方式使系统粒度与算法粒度保持一致。这是因为这项优化能够从根本上减轻读放大。复用感知的异步预取和注意力感知缓存管理可以进一步加速推理。不过，它们在不同模型和数据集上的贡献各不相同。例如，服务更大的模型（如 Qwen2.5-32B）时，预取带来的收益更大。这可能是因为更大的模型需要更长的计算时间，因此可以与更多 I/O 操作重叠。与 SST-2 数据集相比，新缓存策略对 RTE 数据集的贡献更大。这可能与各数据集访问模式的差异有关。

延迟分解。 与图 5类似，我们还展示了在不同 KV 缓存预算比例下，使用 Qwen2.5-7B 在 RTE 数据集上运行 ContiguousKV 时的延迟分解。我们对每个阶段的延迟进行归一化，并报告各阶段占总延迟的比例。如图 13 所示，加载关键 SemChunks 阶段所占的比例显著降低，从而减少了总体延迟。这是因为大多数 I/O 操作都与计算并行执行，隐藏了顺序 I/O 的时间成本。由于我们没有优化加载探测 SemChunks 阶段，该阶段的延迟与图 5 中的 IMPRESS 相近，因此其所占比例显著增加。值得注意的是，计算时间成本也显著降低。这是因为对齐消除了从多个数据块中重新组装稀疏 token 的过程。

输入/输出归约。 我们还进行了实验，以展示表 2中的输入/输出归约情况。我们通过统计从固态硬盘加载的词元数量来量化归约，并报告归一化结果。结果表明，与 IMPRESS 相比，我们的 ContiguousKV 平均可实现约 16.33 倍的归约。这也证明了我们对齐策略的有效性。

表2：ContiguousKV 和 IMPRESS 从固态硬盘加载的词元数量的归一化结果。所有数据均基于 IMPRESS 的结果进行归一化。

数据集	模型	IMPRESS	ContiguousKV
SST-2	7B	100%	6.21%
	14B	100%	6.64%
RTE	7B	100%	6.00%
	14B	100%	5.71%

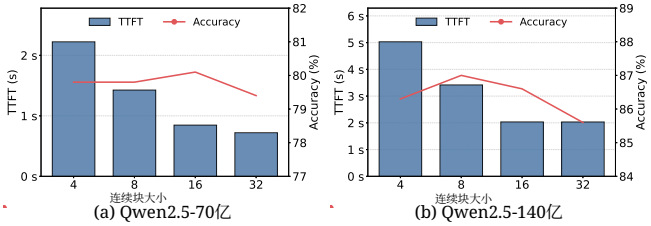


图 14：不同 SemChunk 块大小下准确率与首令牌生成时间的权衡。

5.4 敏感性分析

SemChunk 大小。 由于我们对齐了系统粒度与算法粒度，SemChunk 大小不仅会影响模型性能，也会影响模型效率。我们评估了不同 SemChunk 大小（范围为 4 到 32）下 ContiguousKV 的准确率和平均首令牌生成时间。如图 14所示，随着 SemChunk 大小增加，模型性能先上升后下降。这是因为丰富的语义有助于提升大语言模型的理解能力，但信息过多可能会引入噪声。在效率方面，由于我们已将系统粒度与算法粒度对齐，较大的 SemChunk 大小能够利用更多输入/输出带宽，从而加快推理速度。本文选择 16 作为 SemChunk 大小，因为它在准确率和效率之间实现了良好的权衡。

周期大小。 周期大小决定了共享关键语义块索引的层数。借助我们的预取机制，较大的周期大小可以让更多输入/输出操作与计算重叠。我们还评估了 ContiguousKV 在不同周期大小（从 4 到 16）下的准确率和平均首令牌生成时间。如图 15所示，随着周期大小增加，模型性能相应下降，而模型效率则有所提升。这是因为复用策略是一种近似过程，随着复用层数增加，误差会不断累积。因此，本文选择周期大小为 8，以实现更好的权衡。

前缀大小扩展性。我们进一步通过改变共享前缀

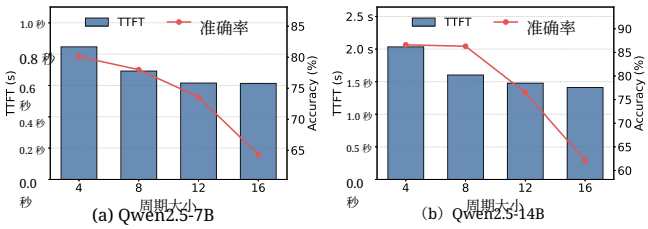


图 15：不同周期大小下准确率与首令牌生成时间之间的权衡。

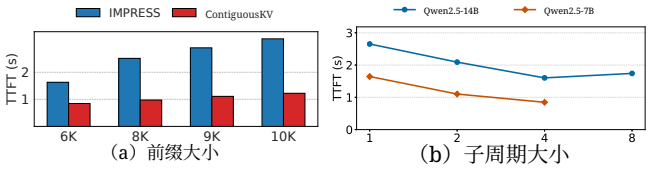


图16：不同子周期大小和不同前缀大小下的平均首令牌生成时间。

长度，在 Qwen2.5-7B 上以 25% 的键值缓存预算测试 ContiguousKV 和 IMPRESS 的扩展性。图中的结果16(a) 表明，ContiguousKV 始终优于 IMPRESS，且前缀越长，性能优势越明显。例如，使用 10K 词元的前缀时，ContiguousKV 可实现 2。6× 倍加速。这表明，我们的系统能够高效处理不断增长的上下文长度，而不会像以往系统那样出现严重的延迟膨胀。

子周期大小。 我们使用两个模型评估了连续 KV 在不同子周期大小下的平均首令牌生成时间。如图 16（b）所示，较大的子周期大小通常能够通过更好地重叠输入/输出与计算来降低首令牌生成时间。本文选择大小为 4，这是因为它能够在预取积极性与可用于重叠的计算时间之间实现最佳平衡。当子周期大小等于完整周期大小（例如 8）时，由于消除了周期内流水线处理，效率会降低。

6 相关工作

基于卸载的推理系统。 为满足长上下文大语言模型推理期间庞大的键值缓存需求，一系列研究探索了卸载策略。其中一类工作 [13, 47] 将键值缓存从图形处理器扩展到 CPU 主存，并将其视为更大的二级缓存层。这些工作还将驻留于 CPU 的缓存组织成检索数据库，以支持稀疏注意力。虽然这缓解了图形处理器内存压力，但其容量仍受 DRAM 大小限制，而且检索索引通常需要针对新的输入模式重新构建，因此会带来相当大的开销。另一类与我们的工作更为接近的系统 [6, 11, 19, 29], 采用三级存储层级（图形处理器内存、CPU 内存和固态硬盘），以提供近乎无限的容量。然而，这些系统为了摊销输入/输出开销，以粗粒度的固定大小块管理数据；当其与细粒度的语义感知剪枝算法结合时，会导致严重的读放大。ContiguousKV 通过引入连续块作为剪枝、存储、传输和缓存的统一单元，直接解决了这一根本性的粒度不匹配问题，从而消除读放大及其导致的输入/输出浪费。

KV 缓存剪枝与量化。 KV 缓存剪枝方法会识别并仅保留一部分重要的 KV 对。H2O [50],StreamingLLM [41],和 SnapKV [20], 等令牌级剪枝技术会为单个词元评分并进行选择。尽管这类方法能有效缩减规模，但以令牌粒度进行操作可能会破坏局部语义连贯性，从而显著降低生成质量。近期的 ChunkAttention [45]和 ChunkKV [25] 等分块级方法会将连续词元分组，并在块级别进行重要性选择，从而更好地保留上下文语义；ContiguousKV 采用了这一思路，并将其扩展到系统设计中。另一方面，量化技术 [14, 23, 27, 35] 通过降低每个 KV 元素的位宽（例如，从 FP16 降至 8 位或 4 位）来减少内存占用。然而，随着上下文长度增长至数万个词元，节省的绝对空间会逐渐减少；而且，如果仍需从慢速存储中加载整个量化缓存，仅靠量化无法解决根本的 I/O 瓶颈。ContiguousKV 在摄取时剪枝并选择性加载，从根本上减少了需要传输的数据量，这一优势与量化相辅相成。

其他大语言模型服务系统。 大量研究致力于优化大语言模型服务的其他方面。vLLM [18] 和 SGLang [51] 等系统专注于在解码期间高效管理动态键值缓存的图形处理器内存，主要采用分页和高级调度。DistServe [52] 和 Mooncake [30],等其他项目则提出在集群中对预填充阶段和解码阶段进行解耦，以提高整体有效吞吐量。这些优化与 ContiguousKV 的关注点基本正交，后者旨在降低使用共享的卸载前缀 KV 缓存的工作负载在重新预填充阶段的输入/输出延迟。ContiguousKV 的技术可以与这些系统集成，进一步提升端到端性能。

7 结论

本文识别并解决了处理共享上下文工作负载的大语言模型服务系统中的关键重新预填充瓶颈。通过详细分析，我们发现了两项根本性低效：语义剪枝与物理 I/O 操作之间存在粒度不匹配，导致读放大；重新预填充过程中的顺序依赖则造成资源利用不足。我们的系统 ContiguousKV 引入了 ContiguousChunk

抽象，使 I/O 粒度与算法粒度相匹配，确保每次 I/O 操作都能准确获取所需的语义连贯单元，从而消除无效的数据传输。我们设计了两级预取机制，利用 Transformer 层间注意力模式在周期内和跨周期的相似性，在当前层计算的同时异步推测性地加载 KV 缓存数据。这种流水线处理打破了此前造成空闲气泡的顺序依赖。此外，我们的注意力感知缓存管理策略根据模型自身的注意力分数动态确定数据块的优先级，确保语义价值最高的前缀数据保留在更快的内存层级中。使用 Qwen2.5 模型进行的评估表明，与最先进的解决方案相比，ContiguousKV 将首令牌生成时间降低了 3.85×。

参考文献

[1] Joshua Ainslie、James Lee-Thorp、Michiel de Jong、Yury Zemlyanskiy、Federico Lebrón 和 Sumit Sang- hai。GQA：基于多头检查点训练广义多查询 Transformer 模型。载于*2023* 年自然语言处理实证方法会议（*EMNLP*）论文集，第 4895–4901 页，2023 年。

[2] Yushi Bai、Shangqing Tu、Jiajie Zhang、Hao Peng、Xi-aozhi Wang、Xin Lv、Shulin Cao、Jiazheng Xu、Lei Hou、Yuxiao Dong、Jie Tang 和 Juanzi Li。Longbench v2：深入理解和推理真实长上下文多任务。载于第*63*届计算语言学协会年会（*ACL*）论文集，第 3639–3664 页，2025 年。

[3]Luisa Bentivogli、Bernardo Magnini、Ido Dagan、Hoa Trang Dang 和 Danilo Gia mpiccolo。第五届 PASCAL 文本蕴含识别挑战赛。载于第二届文本分析会议（*TAC*）论文集，2009 年。

[4] Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Pondé de Oliveira Pinto、Jared Kaplan、Harri Edwards、Yuri Burda、Nicholas Joseph、Greg Brockman、Alex Ray、Raul Puri、Gretchen Krueger、Michael Petrov、Heidy Khlaaf、Girish Sastry、Pamela Mishkin、Brooke Chan、Scott Gray、Nick Ryder、Mikhail Pavlov、Alethea Power、Lukasz Kaiser、Mohammad Bavarian、Clemens Winter、Philippe Tillet、Felipe Petroski Such、Dave Cummings、Matthias Plappert、Fotios Chantzis、Eliza- beth Barnes、Ariel Herbert-Voss、William Hebgen Guss、Alex Nichol、Alex Paino、Nikolas Tezak、Jie Tang、Igor Babuschkin、Suchir Balaji、Shantanu Jain、William Saunders、Christopher Hesse、Andrew N. Carr、Jan Leike、Joshua Achiam、Vedant Misra、Evan Morikawa、Alec Radford、Matthew Knight、Miles Brundage、Mira Murati、Katie Mayer、Peter Welinder、Bob McGrew、Dario Amodei、Sam McCandlish、Ilya Sutskever 和 Wojciech Zaremba。评估基于代码训练的大语言模型。*CoRR*，abs/2107.03374，2021年。

[5] Maximillian Chen、Ruoxi Sun、Tomas Pfister 和 Ser- can Ö. Arik。《学习如何澄清：通过基于行动的对比式自训练进行多轮对话》。载于第十三届国际学习表征会议（*ICLR*）论文集，2025 年。

[6] 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感知多层前缀键值存储系统》。载于第 *23* 届 *USENIX* 文件与存储技术会议（*FAST*）论文集，第 187–201 页。

[7] Kaustubh Deshpande、Ved Sirdeshmukh、Johannes Baptist Mols、Lifeng Jin、Ed-Yeremai Hernandez-Cardona、Dean Lee、Jeremy Kritz、Willow E. Primack、Summer Yue 和 Chen Xing。《Multichallenge：一项真实的多轮对话评测基准，对前沿大语言模型提出挑战》。载于计算语言学协会（*ACL*）论文集，第 18632–18702 页，2025 年。

[8] Amit Elhelo 和 Mor Geva。从参数推断注意力头的功能。载于第 *63* 届计算语言学协会年会论文集（*ACL*），第 17701–17733 页，2025。

[9] Zhiting Fan、Ruizhe Chen、Tianxiang Hu 和 Zuozhu Liu。Fairmt-bench：用于评测对话式大语言模型多轮对话公平性的基准。在第十三届国际学习表征会议（*ICLR*）论文集，2025。

[10] Harsh Gajjar。HIA（健康洞察智能体），2025。

[11] Bin Gao、Zhuomin He、Puru Sharma、Qingxuan Kang、Djordje Jevdjic、Junbo Deng、Xingkun Yang、Zhou Yu 和 Pengfei Zuo。借助缓存注意力，为多轮对话提供高性价比大语言模型服务。载于 Saurabh Bagchi 和 Yiying Zhang 编，*2024* 年 *USENIX* 年度技术会议论文集（*USENIX ATC*），第 111–126 页，2024。

[12] Tianyu Gao、Adam Fisch 和 Danqi Chen。《让预训练语言模型成为更出色的少样本学习器》。载于第*59*届计算语言学协会年会和第*11*届国际自然语言处理联合会议（*ACL/IJCNLP*）论文集，第3816–3830页，2021年。

[13] Junhui He、Junna Xing、Nan Wang、Rui Xu、Shangyu Wu、Peng Zhou、Qiang Liu、Chun Jason Xue 和 Qin- gan Li。《A^2ats：通过窗口化旋转位置嵌入和查询感知向量量化进行基于检索的 KV 缓存压缩》。载于计算语言学协会（*ACL*）论文集，第12451–12463页，2025。

[14] Coleman Hooper、Sehoon Kim、Hiva Mohammadzadeh、Michael W. Mahoney、Yakun Sophia Shao、Kurt Keutzer 和 Amir Gholami。Kvquant：借助 KV缓存量化实现 1000 万上下文长度的大语言模型推理。载于《*2024* 年神经信息处理系统进展》第 *38* 卷（*NeurIPS*），2024 年。

[15] Yulong Hui、Yao Lu 和 Huanchen Zhang。UDA：面向真实世界文档分析中检索增强生成的基准测试套件。载于*2024* 年第 *38* 届神经信息处理系统进展（*NeurIPS*）论文集，2024 年。[16] Jinwoo Jeong 和 Jeongseob Ahn。通过高效资源管理加速多轮对话的大语言模型服务。载于第 *30* 届 *ACM* 国际编程语言与操作系统架构支持会议（*ASPLOS*）论文集，第 1–15 页，2025 年。[17] Yuri Kuratov、Aydar Bulatov、Petr Anokhin、Ivan Rodkin、Dmitry Sorokin、Artyom Y. Sorokin 和 Mikhail Burtsev。Babilong：通过长上下文推理测试大语言模型的极限。载于*2024* 年第 *38* 届神经信息处理系统进展（*NeurIPS*）论文集，2024 年。[18] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、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。载于第 *29* 届操作系统原理研讨会（*SOSP*）论文集，第 611–626 页，2023 年。[19] Wonbeom Lee、Jungi Lee、Junghwan Seo 和 Jaewoong Sim。Infinigen：通过动态KV缓存管理实现大语言模型的高效生成式推理。载于第 *18* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI*）论文集，第 155–172 页，2024 年。[20] 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想要什么。载于*2024* 年第 *38* 届神经信息处理系统进展（*NeurIPS*）论文集，2024 年。

[21] Jianxun Lian、Yuxuan Lei、Xu Huang、Jing Yao、Wei Xu 和 Xing Xie。Recai：利用大语言模型打造新一代推荐系统。载于*ACM Web* 会议（*WWW*）伴随论文集，第 1031–1034 页，2024 年。

[22] Jiaheng Liu、Ken Deng、Congnan Liu、Jian Yang、Shukai Liu、He Zhu、Peng Zhao、Linzheng Chai、Yanan Wu、Ke Jin、Ge Zhang、Zekun Moore Wang、Guoan Zhang、Yingshui Tan、Bangyu Xiang、Zhaoxiang Zhang、Wenbo Su 和 Bo Zheng。M2RC-EVAL：大规模多语言仓库级代码补全评测。载于《第 *63* 届计算语言学协会年会论文集》（*ACL*），第 15661–15684 页，2025 年。

[23] Peiyu Liu、Ze-Feng Gao、Xin Zhao、Yipeng Ma、Tao Wang 和 Ji-Rong Wen。通过矩阵分解解锁用于 KV 缓存压缩的无数据低比特量化。载于计算语言学协会第*62*届年会论文集（*ACL*），第2430–2440页，2024年。[24] Tianyang Liu、Canwen Xu 和 Julian J. McAuley。Re-pobench：代码仓库级代码自动补全系统基准测试。载于第*12*届国际学习表征会议论文集（*ICLR*），2024年。[25] Xiang Liu、Zhenheng Tang、Peijie Dong、Zeyu Li、Bo Li、Xuming Hu 和 Xiaowen Chu。Chunkkv：在保留语义的同时压缩 KV 缓存，以实现高效的长上下文大语言模型推理。载于*2025*年神经信息处理系统进展第*39*卷论文集（*NeurIPS*），2025年。[26] Zichang Liu、Aditya Desai、Fangshuo Liao、Weitao Wang、Victor Xie、Zhaozhuo Xu、Anastasios Kyrillidis 和 Anshumali Shrivastava。Scissorhands：利用重要性持久性假设，在测试时压缩大语言模型 KV 缓存。载于*2023*年神经信息处理系统进展第*36*卷论文集（*NeurIPS*），2023年。[27] Zirui Liu、Jiayi Yuan、Hongye Jin、Shaochen (Henry) Zhong、Zhaozhuo Xu、Vladimir Braverman、Beidi Chen 和 Xia Hu。KIVI：无需调优的 KV 缓存非对称 2 位量化。载于第*41*届国际机器学习会议论文集（*ICML*），2024年。[28] Bo Pang 和 Lillian Lee。情感教育：基于最小割的主观性摘要进行情感分析。载于第*42*届计算语言学协会年会论文集（*ACL*），第271–278页，2004年。

[29] Ramya Prabhu、Ajay Nayak、Jayashree Mohan、Ramachandran Ramjee 和 Ashish Panwar。vattention：为不使用 PagedAttention 的大语言模型服务提供动态内存管理。载于第*30*届 *ACM* 编程语言与操作系统架构支持国际会议论文集（*ASPLOS*），第1133–1150页，2025年。

[30] Ruoyu Qin、Zheming Li、Weiran He、Jialei Cui、Feng Ren、Mingxing Zhang、Yongwei Wu、Weimin Zheng 和 Xinran Xu。Mooncake：以更多存储换取更少计算——一种以键值缓存为核心的大语言模型聊天机器人服务架构。载于第*23*届 *USENIX*文件与存储技术会议（*FAST*）论文集，第 155–170 页，2025 年。

[31] Soo Ryu。Transformer 语言模型中的合理性处理：聚焦于注意力头在 GPT 中的作用。载于 Houda Bouamor、Juan Pino 和 Kalika Bali 编，计算语言学协会（*EMNLP*）论文集，第 356–369 页，2023 年。[32] Ying Sheng、Lianmin Zheng、Binhang Yuan、Zhuohan Li、Max Ryabinin、Beidi Chen、Percy Liang、Christopher Ré、Ion Stoica 和 Ce Zhang。Flexgen：使用单个图形处理器实现大语言模型的高吞吐量生成式推理。载于第 *40* 届国际机器学习会议（*ICML*）论文集，第 202 卷，机器学习研究论文集，第 31094–31116 页。PMLR，2023 年。[33] Je onghoon Shim、Gyuhyeon Seo、Cheongsu Lim 和 Yohan Jo。Tooldial：面向工具增强型语言模型的多轮对话生成方法。载于第 *13* 届国际学习表征会议（*ICLR*）论文集，2025 年。[34] Richard Socher、Alex Perelygin、Jean Wu、Jason Chuang、Christopher D. Manning、Andrew Y. Ng 和 Christopher Potts。情感树库上的语义组合性递归深度模型。载于*2013* 年自然语言处理经验方法会议（*EMNLP*）论文集，第 1631–1642 页，2013 年。[35] Yi Su、Yuechi Zhou、Quantong Qiu、Juntao Li、Qingrong Xia、Ping Li、Xinyu Duan、Zhefeng Wang 和 Min Zhang。通过追踪离群词元实现准确的 KV 缓存量化。载于计算语言学协会第 *63* 届年会（*ACL*）论文集，第 12895–12915 页，2025 年。[36] Llama Team。Llama 3 模型家族。*CoRR*，abs/2407.21783，2024 年。[37] Ellen M. Voorhees 和 Dawn M. Tice。构建问答测试集。载于第 *23* 届 *ACM* 信息检索研究与发展国际会议（*SIGIR*），第 200–207 页，2000 年。

[38] Di Wu、Wasi Uddin Ahmad、Dejiao Zhang、Murali Kr- ishna Ramanathan 和 Xiaofei Ma。Repoformer：面向仓库级代码补全的选择性检索。载于第*41*届国际机器学习会议（*ICML*）论文集，2024 年。

[39] Shangyu Wu、Ying Xiong、Yufei Cui、Xue Liu、Buzhou Tang、Tei-Wei Kuo 和 Chun Jason Xue。Refu- sion：通过计算高效的检索表示融合提升自然语言理解。载于第*12*届国际学习表征会议（*ICLR*）论文集，2024 年。

[40] Shangyu Wu、Ying Xiong、Yufei Cui、Haolun Wu、Can Chen、Ye Yuan、Lianming Huang、Xue Liu、Tei-Wei Kuo、Nan Guan 和 Chun Jason Xue。《自然语言处理中的检索增强生成：综述》。*CoRR*，abs/2407.13193，2024。[41] Guangxuan Xiao、Yuandong Tian、Beidi Chen、Song Han 和 Mike Lewis。《利用注意力汇聚点实现高效的流式语言模型》。发表于第十二届国际学习表征会议（*ICLR*）论文集，2024。[42] An Yang、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chengyuan Li、Dayiheng Liu、Fei Huang、Haoran Wei、Huan Lin、Jian Yang、Jianhong Tu、Jianwei Zhang、Jianxin Yang、Jiaxi Yang、Jingren Zhou、Junyang Lin、Kai Dang、Keming Lu、Keqin Bao、Kexin Yang、Le Yu、Mei Li、Mingfeng Xue、Pei Zhang、Qin Zhu、Rui Men、Runji Lin、Tianhao Li、Tingyu Xia、Xingzhang Ren、Xuancheng Ren、Yang Fan、Yang Su、Yichang Zhang、Yu Wan、Yuqiong Liu、Zeyu Cui、Zhenru Zhang 和 Zihan Qiu。《Qwen2.5 技术报告》。*CoRR*，abs/2412.15115，2024。[43] Jiayi Yao、Hanchen Li、Yuhan Liu、Siddhant Ray、Yi- hua Cheng、Qizheng Zhang、Kuntai Du、Shan Lu 和 Junchen Jiang。《Cacheblend：利用缓存知识融合实现面向检索增强生成的快速大语言模型服务》。发表于第*20*届欧洲计算机系统会议（*EuroSys*）论文集，第 94–109 页，2025。[44] Zijun Yao、Weijian Qi、Liangming Pan、Shulin Cao、Lin- mei Hu、Weichuan Liu、Lei Hou 和 Juanzi Li。《Seakr：面向自适应检索增强生成的自感知知识检索》。发表于第*63*届计算语言学协会年会（*ACL*）论文集，第 27022–27043 页，2025。[45] Lu Ye、Ze Tao、Yong Huang 和 Yang Li。《Chunkat- tention：利用前缀感知键值实现高效自注意力

缓存与两阶段分区。发表于第*62*届计算语言学协会年会（*ACL*）论文集，第 11608–11620 页，2024。

[46] Chong Zhang、Xinyi Liu、Mingyu Jin、Zhongmou Zhang、Lingyao Li、Zhenting Wang、Wenyue Hua、Dong Shu、Suiyuan Zhu、Xiaobo Jin、Sujian Li、Mengnan Du 和 Yongfeng Zhang。人工智能与金融相遇（StockAgent）：基于大语言模型的股票交易，应用于模拟现实环境。*CoRR*，abs/2407.18957，2024。

[47] Hailin Zhang、Xiaodong Ji、Yilin Chen、Fangcheng Fu、Xupeng Miao、Xiaonan Nie、Weipeng Chen 和 Bin Cui。PQCache：面向长上下文大语言模型推理的基于乘积量化的键值缓存。*2025* 年 *ACM* 数据管理大会（*SIGMOD*）论文集，3(3):201:1–201:30，2025。

[48] Minwei Zhang、Haifeng Sun、Jingyu Wang、Shaolong Li、Wanyi Ning、Qi Qi、Zirui Zhuang 和 Jianxin Liao。ClusterAttn：在内在注意力集群下压缩键值缓存。载于第 *63* 届计算语言学协会年会（*ACL*）论文集，第 14451–14473 页，2025。

[49] Susan Zhang、Stephen Roller、Naman Goyal、Mikel Artetxe、Moya Chen、Shuohui Chen、Christopher De- wan、Mona T. Diab、Xian Li、Xi Victoria Lin、Todor Mihaylov、Myle Ott、Sam Shleifer、Kurt Shuster、Daniel Simig、Punit Singh Koura、Anjali Sridhar、Tianlu Wang 和 Luke Zettlemoyer。OPT：开放式预训练 Transformer 语言模型。*CoRR*，abs/2205.01068，2022。

[50] Zhenyu Zhang、Ying Sheng、Tianyi Zhou、Tianlong Chen、Lianmin Zheng、Ruisi Cai、Zhao S ong、Yuan- dong Tian、Christopher Ré、Clark W. Barrett、Zhangyang Wang 和 Beidi Chen。H2O：面向大型语言模型高效生成式推理的重击者预言机。载于*2023*年神经信息处理系统进展第*36*卷（*NeurIPS*），2023。

[51] Lianmin Zheng、Liangsheng Yin、Zhiqiang Xie、Chuyue Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica、Joseph E. Gonzalez、Clark W. Barrett 和 Ying Sheng。《Sglang：结构化语言模型程序的高效执行》。载于 *2024* 年神经信息处理系统进展第 *38* 卷会议论文集（*NeurIPS*），2024 年。

[52] Yinmin Zhong、Shengyu Liu、Junda Chen、Jianbo Hu、Yibo Zhu、Xuanzhe Liu、Xin Jin 和 Hao Zhang。Dist-serve：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*）论文集，第193–210页，2024。

[53] Zhenhong Zhou、Haiyang Yu、Xinghua Zhang、Rongwu Xu、Fei Huang、Kun Wang、Yang Liu、Junfeng Fang 和 Yongbin Li。论注意力头在大语言模型安全中的作用。载于第十三届国际学习表征会议（*ICLR*）论文集，2025。