CommVQ：用于 KV 缓存压缩的可交换向量量化

李君彦 1 张扬 2 穆罕默德・优素福・哈桑 1 塔尔哈・查菲卡尔 1 蔡天乐 3 任志乐 4 郭鹏生 4 比纳齐尔・卡里姆扎德 4 科罗拉多・J・里德 4 王冲 4 甘创 1

arXiv:2506.18879v1 [cs.CL] 2025年6月23日

摘要

大型语言模型（LLMs）越来越多地被用于需要长上下文长度的应用中，但随着上下文的增长，键值（KV）缓存往往成为 GPU 上的内存瓶颈。为解决这一问题，我们提出了可交换向量量化（CommVQ）方法，以显著降低长上下文 LLM 推理时的内存使用量。我们首先引入带有轻量级编码器和码本的加法量化来压缩 KV 缓存，该缓存可通过简单的矩阵乘法进行解码。为进一步降低解码过程中的计算成本，我们将码本设计为与旋转位置嵌入（RoPE）可交换，并使用期望最大化（EM）算法对其进行训练。这使得解码能够高效地集成到自注意力机制中。我们的方法通过加法量化实现了高精度，并且借助与 RoPE 可交换的码本实现了低开销。在长上下文基准测试和 GSM8K 上的实验表明，我们的方法通过 2 位量化将 FP16 KV 缓存大小减少了 87.5%，同时性能优于最先进的 KV 缓存量化方法。值得注意的是，它支持 1 位 KV 缓存量化，且精度损失极小，使得 LLaMA-3.1 8B 模型能够在单个 RTX 4090 GPU 上以 128K 的上下文长度运行。源代码可在以下网址获取：https://github.com/UMass-Embodied-AGI/CommVQ。

1. 引言

我们正目睹大型语言模型（LLMs）的上下文长度不断增加的趋势日益明显。例如，最新的 LLaMA 3.1 模型（Dubey 等人，2024 年）支持长达 128K 的上下文长度，而近期的研究（Ding 等人，2024；Jin 等人，2024）更是将这一长度进一步延长，部分模型的上下文长度已超过 100 万。支持更长的上下文使大语言模型能够处理更丰富的输入并生成更多的令牌，从而提高它们执行更复杂任务和推理的能力（Wei 等人，2022）。

然而，这种增加的上下文长度对 GPU 内存使用提出了重大挑战。大语言模型（LLMs）中使用的因果注意力机制依赖键值（KV）缓存来加快推理速度。该缓存存储所有先前令牌的键和值，从而在生成下一个令牌时无需重新计算它们。随着上下文长度的增加，KV 缓存的大小成比例增长，最终成为内存使用的主要瓶颈——其占用的内存往往远超模型本身所需的内存。例如，一个 LLaMA 3.1 8B 模型在 FP16 精度下存储模型权重需要 16GB 内存。如果将上下文长度设置为最大的 128K，且批处理大小为 2，仅 KV 缓存就需要 88GB 内存。这使得在不进行 KV 缓存卸载的情况下，即使是 H100-80GB GPU 也无法运行推理。

减少 KV 缓存大小的努力仍在进行中（Shi 等人，2024；Yuan 等人，2024），其中 KV 缓存量化（Liu 等人，2024b；Hooper 等人，2024）是一种关键方法。量化通过减小 KV 缓存中表示每个 FP16 标量所用的位宽来降低内存占用。例如，与 FP16 相比，INT4 量化将内存使用量减少了四分之三。然而，这是有代价的：激进的量化（如 2 位甚至 1 位量化）会导致显著的信息丢失，严重降低模型性能。

为了应对这些挑战，我们提出了可交换向量量化（CommVQ），这是一种专为长上下文大语言模型设计的高效且准确的键值缓存量化新方法。与现有的将键值缓存中的每个标量独立处理的量化技术不同，CommVQ 在向量层面进行量化。具体而言，我们将每个令牌的键 / 值向量视为一个单一单元，而不是单独处理各个标量。为了实现这一点，

1 马萨诸塞大学阿默斯特分校 2 麻省理工学院 3 普林斯顿大学 4 苹果公司。通讯作者：李君岩 junyanli@umass.edu。

第 42 届国际机器学习会议论文集，加拿大温哥华。PMLR 267，2025 年。版权所有 2025 年，作者所有。

*所有实验均由马萨诸塞大学阿默斯特分校的李君彦（Junyan Li）完成。

我们利用加法量化（Babenko 和 Lempitsky，2014）——一种向量量化的变体，通过一个经过学习的码本将每个向量编码为低比特宽度的表示，在显著降低内存使用的同时最小化量化误差。

更重要的是，为了以计算高效的方式将加性量化整合到自注意力机制中，码本被创新性地设计为与旋转位置嵌入（RoPE）矩阵可交换（Su 等人，2024）。这使得键值（KV）解码的计算开销大幅降低，其中中间结果可以针对码本中的每个代码进行预计算，然后在计算冗长的键 - 查询乘积时被高效重用。

通过结合这些创新，CommVQ 在内存节省和准确性之间实现了更优的平衡。在两个长上下文基准测试（LongBench（Bai 等人，2023）和 InfiniteBench（Zhang 等人，2024b））以及一个专为复杂推理设计的基准测试 GSM8K（Cobbe 等人，2021）上进行的广泛评估表明，与最先进的 KV 缓存量化基线相比，我们通过 2 位量化实现了近乎无损的 KV 缓存压缩，性能优于其他方法。此外，我们实现了 1 位量化，其准确性显著高于现有基线，这证明了我们的方法在突破 KV 缓存压缩极限方面的有效性。我们的贡献总结如下：

- 不同于以往在 KV 缓存中对每个标量单独进行量化的工作，我们利用学习到的码本，将 KV 缓存中的每个向量作为一个整体量化为低位宽表示。
- 通过利用 RoPE 矩阵的交换性和自注意力的特性，我们将码本优化为具有 RoPE 交换性。这种优化使我们能够重新构建自注意力计算，从而更高效地整合解码过程。此外，我们提供了该方法的 Triton 实现，以展示实际的内存节省效果。
- 大量实验证明了我们方法的优越性，特别是在超低比特宽度 KV 缓存量化场景中（例如 1 位量化）。这为在有限的 GPU 内存约束下运行长上下文大语言模型提供了可能。

2. 相关工作

KV 缓存压缩。先前的几项研究已经探讨了 KV 缓存压缩，其大致可分为两类：令牌驱逐和量化。专注于令牌驱逐的方法（Xiao 等人，2024；Liu 等人，2024a；Zhang 等人（2023）旨在通过逐出不太重要的标记、仅存储最相关标记的键和值来减小 KV 缓存大小。这些方法与我们的方法互不冲突，并且有可能与我们的方法相结合，以实现更高的压缩率。

KV 缓存压缩的另一种重要方法是量化（Liu 等人，2024b；Hooper 等人，2024；Zhang 等人，2024a；Kumar，2024），这种方法通过降低 KV 缓存的位宽来减少其整体存储需求。虽然先前的研究已经成功实现了 KV 缓存的 4 位和 2 位量化，但很少有研究探索在长上下文大语言模型（LLMs）中实现 1 位量化的可行性。我们的方法通过引入一种新颖的基于向量的 KV 缓存量化技术来填补这一空白，该技术能够实现 1 位量化且精度损失极小。此外，我们还就如何重新构建自注意力机制以及如何使我们的量化方法实现更高效的集成提供了新的视角。

矢量量化。矢量量化（Gray，1984）是信号处理和机器学习领域中一种被广泛研究的技术，它使用一组更小的代表性向量（称为码本）来表示高维数据。矢量量化的变体，如乘积量化（PQ）（Jegou 等人，2010）和加法量化（AQ）（Babenko 与 Lempitsky，2014），已被提出以提高其效率和容量。

在机器学习中，矢量量化已成功应用于生成建模等领域（Van Den Oord 等人，2017；Esser 等人，2021）。然而，其在 KV 缓存压缩方面的潜在扩展仍在很大程度上未被探索。最近的一项研究 VQLLM（Kumar，2024）引入了残差矢量量化（RVQ）来压缩 KV 缓存。但他们实现的压缩率相对较低，且其基本的“先解码再自注意力”过程带来了显著的计算开销，限制了其在长上下文大语言模型服务中的实用性。关于如何优化矢量量化并将其与自注意力更好地结合，以实现更高效、更有效的 KV 缓存压缩，目前仍缺乏深入研究。

3. 预备知识

3.1. 自注意力与 KV 缓存

自注意力是大型语言模型（LLMs）的基本构建块。它接收查询（Q）、键（K）和值（V）矩阵作为输入，并生成与 Q 形状相同的输出（O）。自注意力中使用的因果注意力掩码使我们能够缓存键和值矩阵，从而显著加快令牌生成速度。在大型语言模型（LLMs）推理过程中，该过程分为两个阶段：

预填充阶段。在这个阶段，给定输入提示，会计算注意力输出，同时生成 KV 缓存。给定提示的隐藏状态.

$X \in \mathbb{R}^{N \times d}$，其中 N 是标记的数量，d 是

隐藏大小、Q、K 和 V 矩阵以及自注意力输出的计算如下：

$$Q = XW_Q, K = XW_K, V = XW_V$$
$$Self - Attn = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

这里，W_Q、W_K和$W_V \in \mathbb{R}^{d \times d}$是投影矩阵

分别用于查询、键和值。计算得到的 K 矩阵和 V 矩阵随后会被缓存，以供后续的解码阶段使用。

解码阶段。在此阶段，KV 缓存被重用于自注意力计算。给定当前输入隐藏状态$x \in \mathbb{R}^{1 \times d}$，KV 缓存更新如下：

$$K \leftarrow Concat(K, xW_K), V \leftarrow Concat(V, xW_V) \quad (2)$$

更新后的 KV 缓存随后用于计算自注意力输出，遵循与公式 1 相同的等式：

$$Q = xW_Q$$
$$Self - Attn = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

预填充阶段执行一次，用于处理输入令牌并生成第一个输出令牌。之后是解码阶段的多次迭代，该阶段会生成所有后续的输出令牌。

3.2. 旋转位置嵌入

位置编码被添加到查询（Q）和键（K）矩阵中，用于在自注意力计算过程中编码令牌位置信息。最近的开源大语言模型，如 LLaMA（Dubey 等人，2024）、Mistral（Jiang 等人，2023）和 Qwen（Yang 等人，2024），通常为此目的采用旋转位置嵌入（RoPE）（Su 等人，2024）：

$$q_m \leftarrow q_m R_m, k_m \leftarrow k_m R_m \quad (4)$$

其中$q_m, k_m \in \mathbb{R}^{1 \times d}$分别表示$m^{th}$令牌的查询和键向量，而$R_m \in \mathbb{R}^{d \times d}$表示应用于$m^{th}$令牌的 RoPE 矩阵。RoPE 矩阵是一种稀疏矩阵，仅在其 2×2 对角块中存在非零值。因此，RoPE 的应用可以通过首先将k_m向量（类似地，q_m：这里，我们以k_m为例）划分为多个二维子向量来重新表述：

$$k_m = [k1_x, k1_y, ..., k(d/2)_x, k(d/2)_y]$$
$$= [k_m^1, ..., k_m^{d/2}]$$

其中每个二维子向量$k_m^i = (k_{ix}, k_{iy})$由k_m向量中的两个标量组成。R_m的相应 2×2 对角子矩阵（记为$R_m^i \in \mathbb{R}^{2 \times 2}$）随后应用于每个子向量$k_m^i$

$$R_m^i = \begin{pmatrix} cosm\theta_i & -sinm\theta_i \\ sinm\theta_i & cosm\theta_i \end{pmatrix}$$

其中$\theta_i = 10000^{-2(i-1)/d}$。实际上，2×2 对角子矩阵$R_m^i$是一个旋转矩阵，它满足以下性质：

性质 1（交换律）：令$C \in \mathbb{R}^{2 \times 2}$定义为

$$C = \begin{pmatrix} x & y \\ -y & x \end{pmatrix}$$

满足以下交换性：

$$R_m^i C = C R_m^i \quad (8)$$

这一性质表明，在矩阵乘法下，C 与R^i 是可交换的。我们在 4.2 节中利用这一关键性质对我们的方法进行了优化。

4. 方法

在长上下文大语言模型推理场景中，KV 缓存是一个主要因素，因为存储和加载大型 KV 缓存会成为显著的内存和延迟瓶颈。因此，即使需要额外的编码和解码过程，对 KV 缓存进行压缩也是有利的。受向量量化（一种来自信号处理的经典量化技术）的启发，特别是其最新变体 —— 加法量化（Babenko & Lempitsky, 2014）的启发，我们采用了类似的方法，使用经过训练的编码器和码本将 KV 缓存中的每个向量量化为压缩表示，如 4.1 节所述。

为了在将向量量化高效整合到自注意力机制中，同时最大限度地减少额外解码过程带来的计算开销，我们通过设计一种 RoPE 可交换码本并重新排序自注意力中涉及的矩阵乘法，创新性地重新构建了自注意力计算。正如 4.2 节所详细阐述的，这种改进实现了大部分计算的复用，并显著降低了我们方法的计算成本。

4.1. 用于 KV 缓存的学习型加法量化

加法量化（Babenko 和 Lempitsky，2014）旨在将给定的 d 维向量表示为元素级的

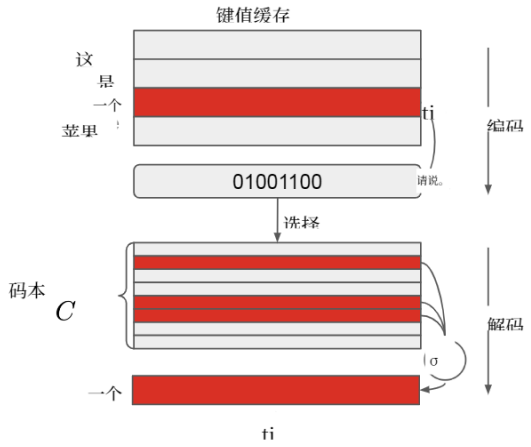


图 1. KV 缓存压缩的向量量化示意图。KV 缓存中的每个向量（对应一个标记）首先被编码为低位宽表示，显著减小了存储大小。此过程单独应用于 KV 缓存中的所有向量。需要时，使用码本加载压缩表示并将其解码回原始 KV 缓存。

从学习到的码本 C 中对多个向量进行求和。为简单起见，我们对键矩阵和值矩阵都采用逐令牌量化方案，这意味着每个令牌的 d 维键向量和值向量都会被单独量化。

受加法量化的启发，我们的方法融入了一个经过学习的编码器 E，用于将给定的 d 维向量编码为一个由 0 和 1 组成的二进制序列，其长度为N_c，还包含一个码本$C \in \mathbb{R}^{N_c \times d}$，通过简单的矩阵乘法即可将原始向量解码回来。如图 1 所示。

编码。给定i^{th}令牌、$t_i \in \mathbb{R}^d$的键或值向量，我们使用编码器 E 将该向量编码为量化向量$s_i \in 0,1^{N_c}$，即$s_i = E(t_i)$。在我们的实现中，编码器 E 由一个简单的线性层、一个激活函数以及后续用于输出的另一个线性层组成。使用 Gumbel-softmax（Jang 等人，2016）可使编码器实现端到端可微分。编码步骤在 KV 缓存生成之后进行，每个令牌的量化向量s_i被拼接并存储在一起，形成量化的 KV 缓存 S。

解码。加载 KV 缓存时，会检索量化的 KV 缓存 S。对于每个令牌的s_i，我们使用码本 C 执行简单的矩阵乘法，以重建解码张量$\hat{t}_i$：

$$\hat{t}_i = s_i C \quad (9)$$

然后，解码后的键和值可以参与子运算自注意力机制中的后续计算。

编码器 E 和码本 C 通过梯度下降进行优化，以最小化原始张量t_i与其解码后的对应张量$\hat{t}_i$之间的均方误差（MSE）损失。

KV 缓存缩减分析。每一层原始 FP16 KV 缓存的大小可以计算为$B \times N \times d \times 2 \times 16$位，其中 B 是批处理大小，N 是令牌数量，d 是隐藏大小。量化后，每一层量化后的 KV 缓存大小变为$B \times N \times N_c \times 2 \times 1$位。因此，缩减率$RR$可以表示为$1 - \frac{N_c}{16d}$。随后，用于表示 KV 缓存中每个标量的平均位宽为：

$$Avg.bit = 16 - 16 \times RR = \frac{N_c}{d} \quad (10)$$

例如，LLaMA-3.1-8B-Instruct 模型的键和值的隐藏大小 d 为 1024。如果我们将N_c设置为 1024，缩减率将为15，这相当于 1 位量化。我们将在实验部分使用平均比特作为衡量压缩率的指标。与 FP16 相比，平均比特越低，表示压缩率越高，GPU 内存使用量减少得越多。

计算复杂度。计算开销来自 KV 缓存的编码和解码两方面。由于 KV 缓存只需编码一次，因此计算开销的主要来源是在生成下一个令牌期间每次使用时对完整 KV 缓存的解码。

推理步骤 t 中使用 KV 缓存解码的自注意力计算可总结如下：

$$q \leftarrow qR_t \quad (11)$$

$$K \leftarrow \begin{bmatrix} s_0 C_K R_0 \\ s_1 C_K R_1 \\ \cdots \\ s_{N-1} C_K R_{N-1} \end{bmatrix}$$

$$V \leftarrow S_V C_V \quad (13)$$

$$Self-Attn = Softmax\left(\frac{qK^T}{\sqrt{d}}\right)V \quad (14)$$

其中，$q \in \mathbb{R}^{1 \times d}$是当前查询，N 是到目前为止生成的令牌数量，$S_V \in 0,1^{N \times N_c}$是量化值，$s_i \in 0,1^{N_c}$是令牌$i^{th}$的量化向量。

key、C_K、$C_V \in \mathbb{R}^{N_c \times d}$分别是 key 和 value 的码本，$R_i$表示$i^{th}$的 RoPE 矩阵。

token。计算复杂度可以表示为：

$$O((2d+1)N + 2dN_c)N) \quad (15)$$

其中 d 是隐藏维度，N 是 token 的数量，N_c是码本中的行数。第一个

公式 15 中的项对应公式 14 的计算，后者代表普通的自注意力。第二项$2dN_cN$说明了键和值的 KV 缓存解码过程，其中每一项都贡献了dN_cN。

比较式 15 中的第一项和第二项，我们可以发现，额外的矢量量化解码过程带来的计算开销是普通自注意力的N_c倍。这种开销的增加源于需要先对码本中的大量行进行求和以重构解码向量，之后解码向量才能参与自注意力计算。由于N_c通常很大（例如，在 LLaMA-3.1 8B 模型的 1 位量化中为 1024），因此该开销尤为显著。

在下一节中，我们将介绍我们新颖的重新设计，该设计利用可交换码本更高效地将向量量化与自注意力相结合。这种方法将显著降低整体计算开销。

4.2. 用于提升效率的可交换码本

为简化符号，我们专注于 √softmax 内的计算，并忽略$\sqrt{d}$常数：$o = qK^T$，其中输出 α 是一个 N 维向量。α 中的每个标量项o_i计算如下：

$$o_i = qR_t(s_iC_KR_i)^T = (qR_t)R^TC_Ks_i^T \quad (16)$$

其中，R_t表示查询的 RoPE 矩阵，R_i表示i^{th}键的 RoPE 矩阵，s_i是i^{th}键的量化向量。注意，当i变化时，qR_t和C_K保持不变。如果R_i也与 i 无关，那么$(qR_t)R^TC_K$的计算只需在不同的#\##上执行一次，从而显著降低计算成本。遗憾的是，由于R_i随 i 变化，这种简化是不可能的。

然而，如果我们能将码本C_K设计为与R_i可交换，那么式 16 的右侧就可以

被重写为$(qR_t)C_K^TR^TR_is_i^T$，这构成了大部分内容

计算，$(qR_t)C_K^T$，独立于i，因此可重复使用，从而节省大量计算资源。

接下来，我们将讨论如何定义我们新的交换码本C_K以及如何学习它。

设计一个可交换码本。如 3.2 节所述，由于 RoPE 矩阵是块对角的，我们可以将问题分解到二维子空间中，并在每个块内设计可交换码本。形式上，如式 5 和式 6 所定义，k_i^j和R_i^j分别表示k_i和R_i的第 j 个二维子向量 / 子矩阵。扩展性质 1，我们可以得到子空间码本的设计方案。

具体而言，令C_K^j为子空间 i 的码本集合。

关键向量，即

$$C_K^j = \left\{C_K^{j0}, C_K^{j1}, ..., C_K^{j(N_c'-1)}\right\}$$

其中N_c'是量化级别数，每个C_K^{jl}都是一个 2×2 矩阵，满足式 7 中定义的形式，因此具有交换性$R_i^jC_K^{jl} = C_K^{jl}R_i^j$。

k_i^j的量化向量，记为s_i^j，是一个 2

从$0, ..., N_c' - 1$取值的维度向量。解码后的密钥表示为

$$\hat{k}_i^j = \sum_{l=0}^{N_c'-1} \left[s_i^j = l\right] C_K^{jl}$$

，其中$[s_i^j = l]$是一个二维布尔指示向量，若s_i^j的对应维度等于 l，则每个维度为 1，否则为 0。

注意，式 18 中的解码方案比之前讨论的（式 9）更为复杂。这是因为之前我们对所有维度使用了相同的码本，而现在不同维度的码本是不同的。更多解释请参见附录 A.1。

采用新的解码方案后，交换性的优势仍然存在。要理解这一点，请注意式 16 现在应重写为

$$\begin{aligned}
\alpha_i &= \sum_j \left(q^jR_t^j\right)\left(\hat{k}_i^jR_i^j\right)^T \\
&= \sum_j \left(q^jR_t^j\right)\left(\sum_l \left[s_i^j = l\right] C_K^{jl}R_i^j\right)^T \\
&= \sum_{j,l} \left(q^jR_t^j\right) R_i^{jT}C_K^{jlT}\left[s_i^j = l\right]^T \\
&= \sum_{j,l} \left(q^jR_t^j\right) C_K^{jlT}R_i^{jT}\left[s_i^j = l\right]^T
\end{aligned}$$

。

第一个等式将内积分解为子向量的内积；最后一个等式应用了交换性性质，使得$(q^jR_t^j)C_K^{jlT}$可在i中重复使用。

学习码本。码本通过最小化重构误差来学习：

$$min_{C_K^j} \sum_i \left\|\hat{k}_i^j - k_i^j\right\|^2, s.t. Eqn.18 (20)$$

这是一个典型的聚类目标，可以通过类 EM 算法有效求解。如算法 1 所述，E 步在固定C_K^j的情况下对s_i^j进行最小化，M 步在固定s_i^j的情况下对C_K^j进行最小化。更多细节可参见附录 A.2，包括实际的更新公式以及用于稳定优化过程的技术。

算法 1 用于学习 RoPE 交换码本的 EM 算法。

1: 输入：校准集 K ∈RN×2
2: 参数：码本 Cj K
3: 目标：优化 Cj K 以最小化校准集 K 上的聚类误差。
4: 当 Cj K 收敛时
5: E 步骤：固定 Cj K，更新分配 S，使每个 k ∈K 被分配到其最近的聚类中心。
6: M 步骤：固定 S，更新 $\hat{i}_r$，使每个 k 与其分配的聚类中心之间的均方误差（MSE）损失最小化。
7: 结束循环

请注意，$\hat{i}$需要$2log2(N_c')$位，这相对于$\hat{i}$而言，限制了其实现高压缩率的能力。为了提高压缩效果，我们将连续的 g 个子向量分组，并在组内共享量化值，即$i.e.、\hat{i}^0 = \hat{i}^1 = \cdots = \hat{i}^{g-1}$。这使得整个向量能够用少得多的位来表示。

为了进一步提高量化精度，我们在量化误差张量上迭代应用聚类算法，每次使用新的码本，重复这一过程直到误差被充分最小化。具体来说，我们运行聚类算法 R 次，为子空间 j 生成 R 个码本$C\hat{i}_r$。R 是一个平衡量化精度和压缩率的超参数。这种迭代优化在概念上类似于残差矢量量化中的残差方法（Barnes 等人，1996），其中后续迭代专注于减少剩余误差。因此，量化2_a个 FP16 标量需要总共 R 个 S 实例，量化完整的 d 维向量所需的总比特数为$2Rlog2(N_c')\hat{d}$。相应地，平均量化比特可以计算为：

$$Avg.bit = \frac{Rlog2(N_c')}{} \quad (21)$$

例如，为了实现 1 位量化，我们设置了$N' - 64$、$R - 11$和$a - 64$。我们在附录 A.4 中提供了关于如何选择N'、R 和 g 的消融研究。

降低计算复杂度。对于值量化，我们保留原始方法，但重新排序矩阵乘法过程。具体来说，我们首先将经过 softmax 处理的注意力权重与s_{Tr}相乘，然后将结果与C_{Tr}相乘，如下所示：

$$Self - Attn = Softmax(A)S_{Tr}C_{Tr} \quad (22)$$

这种简单的矩阵乘法重新排序将计算复杂度从式 13、14 中的$O(dN_cN + dN)$降低到式 22 中的$O(N_cN + dN_c)$，假设 d 和N_c处于相似的量级，那么复杂度几乎降低了 d 倍。

通过将这些调整整合到原始的自注意力机制中，优化后的计算复杂度现在为：

$$O\left((Rd + N_c + 1)N + d(N_c + RN_c')\right) \quad (23)$$

与未优化的先解码后自注意力计算（公式 15）相比，这种优化有效地降低了计算成本。此前，其复杂度是原始自注意力的N_c倍；现在，它是 ap

大约高$\frac{Rd+1}{}$倍。由于 R 相对较小
超参数（例如，用于 1 位量化的$D - 11$），其开销仍然很小。

5. 实验

5.1. 设置

模型。我们使用最新的 LLaMA3.1-8B-Instruct 模型（Dubey 等人，2024）对 CommVQ 进行评估，该模型支持长达 128K token 的上下文长度。我们使用 FineWeb-Edu 数据集（Lozhkov 等人，2024）的一个子集来学习编码器和码本。我们展示了两种量化级别的评估结果：2 位和 1 位量化（详见附录 A.4 的码本配置）。为了证明我们方法的通用性，我们还在 LLaMA-2-8B（Touvron 等人，2023）和 Mistral-8B（Jiang 等人，2023）模型上进行了额外实验。

基线。我们将我们的方法与三种最新的键值缓存量化技术进行了比较：KIVI（Liu 等人，2024b）、KVQuant（Hooper 等人，2024）和 VQLLM（Kumar，2024）。KIVI 采用非对称量化，KVQuant 使用非均匀量化，VQLLM 则应用残差矢量量化。为保证公平性，我们在相同模型上使用它们的官方开源实现复现了其结果。我们将量化版本表示为 <方法>-<n>，其中<n>代表键值缓存中每个标量的位数。对于 VQLLM，在 2 位量化时，我们设置为$C - 256$、$K - 8$；在 1 位量化时，设置为$C - 256$、$K - 4$。

任务。为了评估我们的方法在长上下文大语言模型（LLMs）上的有效性，我们将其与基线方法一同在两个长上下文基准测试上进行测试：LongBench（Bai 等人，2023）和 InfiniteBench（Zhang 等人，2024b）。此外，为了评估模型执行复杂推理的能力，我们在 GSM8K（Cobbe 等人，2021）上对其进行了评估。除了任务得分外，我们还报告了每种方法的平均量化比特（记为 Avg. bit），以量化实际的 KV 缓存大小缩减。Avg. bit 越低，说明 KV 缓存所需的存储空间越少，从而节省的内存越多。对于基线方法，我们按照其各自论文中提供的计算方法来确定 Avg. bit。

Method	Avg. bit (↓)	Qasper	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	Average (↑)
FP16 Baseline	16	25.19	23.31	26.82	72.50	91.65	43.49	52.47	49.01	48.05
KIVI-2	3.00	22.71	24.33	27.29	72.50	92.06	43.26	51.32	47.53	47.62
KVQuant-2	2.33	41.86	22.37	25.76	69.00	89.00	42.09	36.22	36.51	45.35
VQLLM-2	2.00	32.39	25.20	26.22	69.95	92.01	41.03	40.58	36.19	45.45
CommVQ-2	**2.00**	24.67	24.36	26.48	72.50	91.92	43.98	53.02	46.92	**47.98**
KIVI-1	2.00	4.99	9.57	9.20	38.75	25.07	11.93	17.67	16.40	16.70
KVQuant-1	1.33	1.01	8.71	6.06	1.00	1.50	6.64	11.01	11.09	5.88
VQLLM-1	**1.00**	11.92	17.91	13.12	47.98	63.34	23.72	18.92	22.44	27.42
CommVQ-1	1.03	18.86	23.02	24.34	69.00	91.61	41.83	48.78	42.08	**44.94**

表 1. LLaMA-3.1-8B-Instruct 模型的 LongBench 评估。

Method	Avg. bit (↓)	R.PK	R.Num	R.KV	En.Sum	En.QA	En.MC	En.Dia	Code.D	Math.F	Average (↑)
FP16 Baseline	16	100.00	99.49	55.20	26.74	14.28	66.81	20.00	22.08	33.43	48.67
KIVI-2	3.00	100.00	97.80	0.60	25.41	13.90	66.81	22.50	23.35	33.71	42.68
KVQuant-2	2.33	98.81	88.81	0.00	25.02	7.77	35.81	8.00	25.63	10.29	33.34
VQLLM-2	2.00	100.00	97.96	0.00	18.27	8.09	44.54	9.50	21.83	29.71	36.66
CommVQ-2	**2.00**	100.00	93.39	12.20	24.14	14.57	67.25	18.00	22.08	33.14	**42.75**
KIVI-1	2.00	1.86	0.00	0.00	12.44	3.24	55.46	4.50	23.86	34.00	15.04
KVQuant-1	1.33	0.00	0.00	0.00	17.83	1.36	0.44	2.00	0.25	1.14	2.56
VQLLM-1	**1.00**	82.37	14.75	0.00	10.46	2.69	25.33	1.50	21.83	7.43	18.48
CommVQ-1	1.03	99.15	62.37	0.00	19.34	11.30	65.50	18.00	22.08	33.14	**36.76**

表 2. LLaMA-3.1-8B-Instruct 模型的 InfiniteBench 评估。

5.2. 长上下文基准测试评估

LongBench 评估。LongBench（Bai 等人，2023）是一个用于评估模型在长上下文任务（如多文档问答、摘要和代码补全）上表现的基准。遵循 KIVI（Liu 等人，2024b），我们在四个子组的相同八项任务上评估性能，并报告单项得分和平均得分。最大序列长度设置为 128K。

实验结果如表 1 所示。我们的 2 位量化模型在大多数任务上实现了无损精度，平均得分与 FP16 模型几乎相同，同时优于其他基线模型，并且节省了更多内存。与 KIVI 相比，我们的方法多节省了 33% 的内存，同时获得了更高的平均得分。在平均量化位数相近的基线模型中，我们的方法表现显著更优，得分比 KVQuant 高 2.63%，比 VQLLM 高 2.53%。对于 1 位量化，我们的方法大幅优于其他基线模型，平均得分比 VQLLM 高 17.52%，同时保持了相当的内存节省。这最大限度地减少了与 FP16 模型相比的精度下降。

InfiniteBench 评估。InfiniteBench（Zhang 等人，2024b）在超长语境下对模型进行评估，模拟具有接近无限输入长度的现实世界场景。其任务包括多种类型的检索、问答、摘要和代码调试。在我们的实验中，最大的序列长度设置为 128K。

实验结果如表 2 所示。我们的方法在超长上下文场景中继续表现出色，在 1 位量化方面的优势更为明显。对于需要准确信息的挑战性任务，例如检索（R.PK、R.Num 和 R.KV），其他方法无法产生准确结果，而我们的方法即使在低量化水平下仍保留了一定的能力。这得益于我们的码本设计，它有效地保留了 KV 缓存中的信息，附录 A.5 中的表 11 所示的我们方法较低的量化误差（以均方误差衡量）也证实了这一点。

大海捞针评估。我们进一步使用 LLaMA-3.1 8B 模型在 “大海捞针（NIAH）” 基准测试上评估我们的方法，该基准测试专门针对长上下文环境中的检索能力，要求模型识别嵌入在大量无关文本中的一小段信息。这项测试是衡量量化情况下 KV 缓存和注意力机制保存程度的重要指标。

图 2 中的实验结果表明，我们的 2 位量化模型成功保留了完整的检索能力，性能与 FP16 基线相当。这证实了即使在高强度压缩下，我们的方法所带来的性能下降也微乎其微。更值得注意的是，我们的 1 位 CommVQ 变体比 KIVI 的 1 位变体具有更高的检索精度，这凸显了我们的码本设计在保留关键注意力方面的有效性。

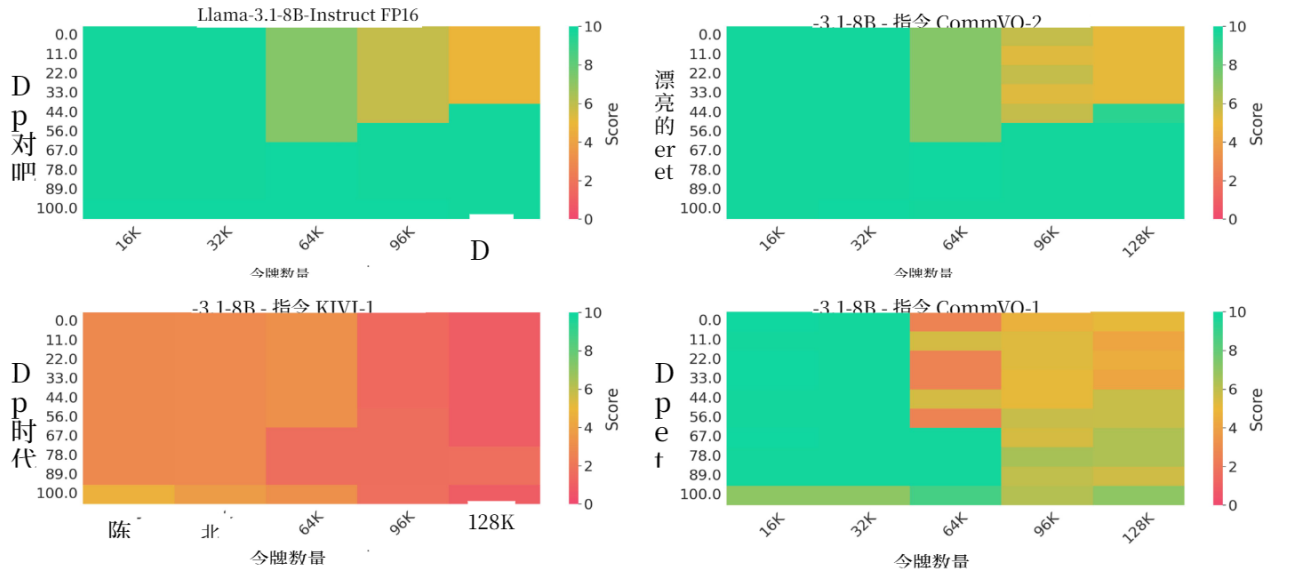


图 2. 使用 LLaMA-3.1 8B 模型进行大海捞针测试。我们展示了 FP16 基线（左上）、KIVI-1（左下）、CommVQ-2（右上）和 CommVQ-1（右下）的测试结果。我们方法的 2 位版本与 FP16 性能相当，而 1 位 CommVQ 变体优于 KIVI，这表明在极端压缩情况下具有很强的检索保真度。

Method	Avg. bit (↓)	GSM8K (↑)
FP16 Baseline	16	76.27
KIVI-2	3.00	73.69
VQLLM-2	2.00	52.69
CommVQ-2	**2.00**	**76.04**
KIVI-1	2.00	2.20
VQLLM-1	**1.00**	1.67
CommVQ-1	1.03	**66.57**

表 3. LLaMA-3.1-8BInstruct 模型的 GSM8K 基准测试评估。采用精确匹配准确率作为衡量指标。

信号。这些发现进一步表明，即使在极高的压缩比下，CommVQ 在信息密集型检索任务中仍能保持较高的保真度。

5.3. GSM8K 评估

为了证明我们的方法在具有挑战性和复杂性的任务上的有效性，我们在 GSM8K（Cobbe 等人，2021）上进行了实验。GSM8K 是一个严格的基准，由人类专家精心设计的高质量、语言多样的数学问题组成。这些问题需要复杂的多步骤推理，并涉及复杂的算术运算，这使得 GSM8K 成为评估复杂推理能力的理想基准。如表 3 所示，我们的 2 位量化模型优于其他基线模型，保持了显著更高的准确率，比 KIVI 高出 2.35%，比 VQ-LLM 高出 23.35%，同时表现出

Model		Avg. bit (↓)	LongBench (↑)
Llama-2-7B	FP16	16	48.43
	KIVI	3.00	47.14
	CommVQ	**2.00**	**47.27**
Mistral-7B	FP16	16	53.40
	KIVI	3.00	52.78
	CommVQ	**2.00**	**53.04**

表 4. 全精度（FP16）、KIVI 和 CommVQ 在另外两个大型语言模型上进行模型消融实验的性能对比。

与 FP16 相比，精度仅下降 0.23% 这一极小幅度。此外，我们的基线模型在 1 位量化下难以生成准确结果，而我们的方法即使在这种极端压缩情况下，仍能展现出强大的推理能力。

5.4. 模型消融实验

我们还将我们的方法应用于另外两个长上下文大语言模型：LLaMA-2-7B（来自 Together.ai 的 32K 上下文长度版本）和 Mistral-7B-v0.3（原生支持 32K 上下文长度）。我们在 LongBench 上对这两个模型进行了评估，将它们的平均分数与 FP16 基线和 KIVI 进行了比较。实验结果汇总于表 4 中。对于 LLaMA-2 和 Mistral 这两个模型，我们的方法在保持 FP16 基线平均分数的同时，比 KIVI 实现了更好的压缩率 - 准确率权衡。这些结果凸显了我们方法的广泛适用性和有效性。

Latency (ms)	8K	32K	128K
Naive Impl.	2.4	9.2	36.6
Optimized Impl.	0.4	1.1	3.8
Speedup	**6.0**	**8.4**	**9.6**

表 5. 朴素实现与利用交换码本的优化实现之间的延迟比较。针对 8K、32K 和 128K 的上下文长度，测量了每层每令牌的延迟，单位为毫秒（ms）。

Method	FineWeb-Edu	GSM-8K	Repobench-p	KV_Retrieval
FP16	10.17	5.67	2.20	31.93
CommVQ-2	11.54	6.14	2.78	32.72
PPL Diff	+1.37	+0.47	+0.58	+0.79

表 6. 不同领域下 FP16 基准与 CommVQ-2 的困惑度（PPL）比较。

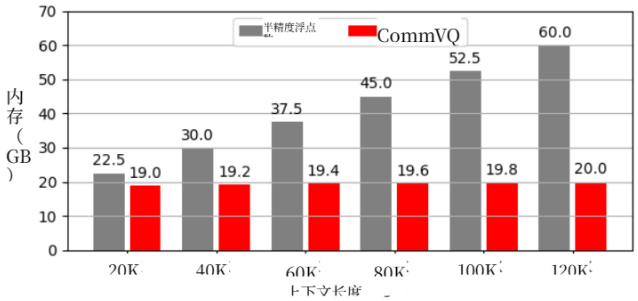
5.5. 域偏移下的鲁棒性分析

我们的码本和编码器在 FineWebEdu（Lozhkov 等人，2024）预训练数据集上进行训练。为了评估在测试领域与训练领域不同时它们的稳健性，我们使用 LLaMA-3.1 8B 模型进行了分析。具体而言，我们在四个不同的数据集上比较了 CommVQ-2 与 FP16 基线的困惑度：FineWeb-Edu（一个通用文本数据集，也是训练集）、GSM-8K（Cobbe 等人，2021，一个数学数据集）、Repobench-p（Bai 等人，2023，一个代码检索和补全数据集）以及 InfiniteBench（Zhang 等人，2024b）中的$KV\text{Retrieval}$（一个合成 UUID 键值检索数据集）。第一个数据集代表域内评估，而后三个代表域偏移评估，即码本和编码器在通用文本上训练，在数学、代码和合成 UUID 数据上测试。结果汇总于表 6 中。

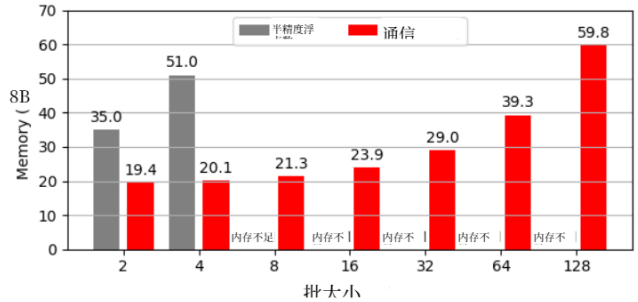
与域内评估相比，我们发现由于域偏移导致的困惑度（PPL）没有显著增加。这表明，我们的方法在与校准数据不同的各个域中都表现稳定良好，包括不太可能出现在校准集中的合成 UUID 数据。总体而言，我们得出结论：我们的方法在域偏移情况下具有鲁棒性和可泛化性。

5.6. 效率结果

在表 5 中，我们展示了延迟对比，以说明我们的方法（记为优化实现）通过减少计算量所带来的影响，而朴素实现并未利用可交换码本来减少计算量。在上下文长度为 8K、32K 和 128K 时，优化实现始终比朴素实现有速度提升。



(a) 内存使用量与上下文长度的关系（批次大小 = 1）。



(b) 内存使用量与批处理大小的关系（上下文长度 = 32K）。

与 FP16 模型相比。实验在 LLaMA 图 3 上进行。CommVQ（1 位）3.1-8B-Instruct 模型的每令牌解码内存使用情况。

验证了交换码本在减少计算开销方面的有效性。

我们还实现了 Triton 内核，以实现实际的内存节省。图 3 突出显示了使用 LLaMA-3.1-8B-Instruct 模型时，在 H100-80GB GPU 上测量的实际每令牌解码内存节省情况。120K 的上下文长度在 FP16 格式下需要 60GB 内存，而我们的方法将其减少到 20GB，使得在单个消费级 GPU（如 RTX 4090）上进行推理成为可能。对于 32K 的上下文长度，FP16 格式在批量大小为 8 时会出现内存不足（OOM），但我们的方法可以扩展到批量大小 128。这改进了长上下文和大批量的服务能力，使许多应用（如长文档问答）受益匪浅。我们还在附录 A.3 中提供了关于码本大小的额外分析，表明其大小可以忽略不计，特别是与长上下文的大型 KV 缓存相比时。

6. 结论

我们提出了 CommVQ，这是一种适用于长上下文大语言模型（LLMs）的新型 KV 缓存量化方法。通过利用向量量化和 RoPE 交换码本，CommVQ 在显著减小 KV 缓存大小的同时，保持了较高的计算效率。在长上下文基准测试中的评估表明，CommVQ 优于现有的 KV 缓存量化方法，能够实现更具内存效率和可扩展性的长上下文大语言模型推理。

影响说明

本文介绍了旨在推动机器学习领域发展的研究工作。我们的研究可能会带来诸多社会影响，但我们认为在此无需特别强调其中任何一项。

参考文献

Babenko, A. 和 Lempitsky, V. 用于极端向量压缩的加法量化。见《IEEE 计算机视觉与模式识别会议论文集》，第 931-938 页，2014 年。

白宇、吕鑫、张静、吕华、唐杰、黄章、杜哲、刘星、曾安、侯亮等人。Longbench：一个用于长上下文理解的双语多任务基准。arXiv 预印本 arXiv:2308.14508，2023。

巴恩斯、C. F.、里兹维、S. A. 和纳斯拉巴迪、N. M. 残差矢量量化的进展：综述。《IEEE 图像处理汇刊》，5 (2)：226-262，1996。

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R. 等人。训练验证器解决数学文字题。arXiv 预印本 arXiv:2110.14168，2021。

丁怡、张丽莉、张聪、徐阳、尚楠、徐静、杨峰和杨梅。Longrope：将大语言模型的上下文窗口扩展到 200 万个 token 以上。arXiv 预印本 arXiv:2402.13753，2024 年。

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A. 等人。llama 3 模型集群。arXiv 预印本 arXiv:2407.21783，2024。

Esser, P., Rombach, R., 以及 Ommer, B. 用于高分辨率图像合成的驯服 Transformer。见《IEEE/CVF 计算机视觉与模式识别会议论文集》，第 12873-12883 页，2021 年。

Gray, R. 矢量量化。《IEEE Assp 杂志》，1 (2)：4–29，1984。

Hooper, C.、Kim, S.、Mohammadzadeh, H.、Mahoney, M. W.、Shao, Y. S.、Keutzer, K. 和 Gholami, A.。Kvquant：通过 kv 缓存量化实现千万级上下文长度大模型推理。arXiv 预印本 arXiv:2401.18079，2024。

张（E.）、顾（S.）和普尔（B.）。使用 Gumbel-softmax 的分类重参数化。arXiv 预印本 arXiv:1611.01144，2016。

Jegou, H.、Douze, M. 和 Schmid, C. 用于最近邻搜索的乘积量化。《IEEE 模式分析与机器智能汇刊》，33 (1)：117-128，2010。

江，A. Q.、萨布莱罗勒，A.、门施，A.、班福德，C.、查普洛特，D. S.、卡萨斯，D. d. l.、布雷桑，F.、伦杰尔，G.、兰普尔，G.、索尔尼耶，L. 等。Mistral 7b。arXiv 预印本 arXiv:2310.06825，2023。

金，H.，韩，X.，杨，J.，江，Z.，刘，Z.，常，C.Y.，陈，H.，和胡，X.。大语言模型或许是长语言模型：无需调优即可自扩展大语言模型的上下文窗口。arXiv 预印本 arXiv:2401.01325，2024。

库马尔，A. 用于大型语言模型中 KV 缓存压缩的残差矢量量化。arXiv 预印本 arXiv:2410.15704，2024。

兰利，P. 撰写机器学习论文。见兰利，P.（编），《第 17 届国际机器学习会议论文集》（ICML 2000），第 1207-1216 页，加利福尼亚州斯坦福，2000 年。摩根・考夫曼出版社。

刘 Z.、德赛 A.、廖 F.、王 W.、谢 V.、徐 Z.、基里利迪斯 A. 和施里瓦斯塔瓦 A.。《剪刀手：在测试时利用重要性假设的持久性实现大语言模型键值缓存压缩》。《神经信息处理系统进展》，第 36 卷，2024a。

刘，Z.，袁，J.，金，H.，钟，S.，徐，Z.，布雷弗曼，V.，陈，B.，和胡，X. Kivi：一种无需调优的非对称 2 位键值缓存量化方法。arXiv 预印本 arXiv:2402.02750，2024b。

Lozhkov, A.、Ben Allal, L.、von Werra, L. 和 Wolf, T.《Fineweb-edu：精选教育内容集》，2024 年。网址：https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu。

施磊、张海、姚远、李子、赵华。降低成本：优化大语言模型 KV 缓存消耗的方法综述。arXiv 预印本 arXiv:2407.18003，2024。

苏杰、艾哈迈德・M.、卢宇、潘森、博伟和刘毅。Roformer：具有旋转位置嵌入的增强型 Transformer。《神经计算》，568：127063，2024。

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S. 等人。Llama 2：开放基础模型与微调聊天模型。arXiv 预印本 arXiv:2307.09288，2023。

Van Den Oord, A., Vinyals, O., 等。神经离散表示学习。神经信息处理系统进展，30, 2017.

魏捷、王旭、舒尔曼斯・D.、博斯马・M.、夏峰、池E.、勒・Q.V.、周 D. 等人。思维链提示引发大型语言模型的推理。《神经信息处理系统进展》，35:24824–24837，2022。

肖刚、唐杰、左军、郭军、杨松、唐华、付宇、韩松。Duoattention：通过检索和流头实现高效的长上下文大模型推理。arXiv 预印本 arXiv:2410.10819，2024。

杨 A、杨 B、张 B、许 B、郑 B、于 B、李 C、刘 D、黄 F、魏 H 等。Qwen2.5 技术报告。arXiv 预印本 arXiv:2412.15115，2024。

袁杰、刘浩、钟思、庄耀南、李思、王广、乐迪、金浩、乔杜里·V.、徐志等。KV 缓存压缩，但我们必须付出什么代价？长上下文能力方法的综合基准测试。arXiv 预印本 arXiv:2407.01527，2024。

张涛、易佳、徐志和阿尼尔·施里瓦斯塔瓦。《Kv 缓存为每通道 1 位：通过耦合量化实现高效的大语言模型推理》。arXiv 预印本 arXiv:2405.03917，2024a。

张希、陈阳、胡松、徐志、陈静、郝敏、韩旭、泰泽、王硕、刘震等。∞Bench：将长上下文评估扩展到 10 万 token 以上。《第 62 届计算语言学协会年会论文集（第一卷：长论文）》，第 15262-15277 页，2024b。

张泽、盛宇、周涛、陈涛、郑亮、蔡然、宋哲、田宇、C. R´e、C. Barrett 等。H2o：用于大型语言模型高效生成推理的重击者预言机。《神经信息处理系统进展》，36：34661–34710，2023。

A. 附录

A.1. 使用交换码本进行编码和解码的说明

在本节中，我们将额外解释如式 18 所定义的二维子向量k_i^j是如何使用量化向量s_i^j和码本C_K^j来表示的。

我们首先将量化过程表述为一个聚类问题，其中聚类中心定义如下：

$$c_{a,b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} C_K^j[a] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} C_K^j[b] (24)$$

此处，$C_K^j[a]$和$C_K^j[b]$代表C_K^j中的a^{th}和b^{th} 2×2子码本。因此，码本C_K^j共形成$N_c'^2$个聚类中心。然后，我们可以将k_i^j量化为其最近的聚类中心，并使用$s = a, b$作为量化表示来代表k_i^j。解码时，我们使用分配的聚类中心来近似k_i^j，因此

$$\begin{aligned} k_i^j &= c_{a,b} \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} C_K^j[a] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} C_K^j[b] \end{aligned}$$

而公式 25 的作用与公式 18 完全相同。

A.2. EM 算法实现细节

如 4.2 节所述，我们应用一种类 EM 算法来学习码本。在本节中，我们提供该算法的细节。我们使用 FineWeb-Edu（Lozhkov 等人，2024）的一个子集作为校准集 K，并在该校准集上优化码本。E 步很简单，只需将 K 中的每个向量 k 分配给其最近的聚类中心。对于 M 步，我们推导了一个闭式公式，用于在给定当前分配 S 的情况下更新c_i。回顾c_i的定义：

$$c_i = \{c_i^0, c_i^1, \ldots, c_i^{(N_c'-1)}\} (26)$$

$$C_K^{jl} = \begin{pmatrix} x_l & y_l - y_l & x_l \end{pmatrix} (27)$$

我们首先定义一些有用的术语：

$$\begin{aligned} \phi &= [x_i, y_i]_{i=0}^{N_c'-1} \\ m &= [m_{ij}]_{i,j=0}^{N_c'-1, N_c'-1} \\ S &= diag\left(N_{ij}, N_{ij}\right)_{i,j=0}^{N_c'-1, N_c'-1} \end{aligned} (28)$$

其中 ϕ 是一个$2N_c'$维向量，表示码本c_i，我们将对其进行优化。m 是一个$N_c'^2$维向量，表示分配给每个聚类中心的数据点的均值。S 是一个$2N_c' \times 2N_c'$矩阵，其对角线上是分配给每个聚类中心的数据点总数。

接下来，我们定义一个辅助常数矩阵$T \in -1, 0, 1^{(2N_c'^2) \times (2N_c')}$，其元素由以下各式给出：

$$T_{2(xN_c'+y),2x} = 1, T_{2(xN_c'+y),2y+1} = -1 \eqno(29)$$

$$T_{2(xN_c'+y),2x} = 1, T_{2(xN_c'+y),2x+1} = 1 (30)$$

(31)

对于所有 x , $y \in 0, 1, \ldots, N_c' - 1$，且 T 的所有其他元素均为零。

然后我们可以将式 18 重写为矩阵形式：

$$min_\phi (T\phi - m)^T S (T\phi - m) (32)$$

闭式解由

$$c^* = (c^T c)^{-1} c^T c_m \quad (33)$$

给出

我们迭代更新 ϕ 直到其收敛。在实验过程中，我们发现学习过程不稳定，且经常难以优化，这是因为在我们的案例中聚类中心的数量较多（例如，对于$N_c' = 64$，存在 4096 个聚类中心），并且聚类中心需要满足公式 24。为了稳定优化过程，我们采用了两种技术。第一种技术是软聚类中心分配，即不将数据点硬分配给其最近的聚类中心，而是将其以不同的权重分配到所有聚类中心。分配给每个中心的权重取决于该中心与该数据点的距离，距离最近的中心获得最高权重。具体来说，设 N 为校准集 K 中的数据点数量，N_{cc}为聚类中心的数量，权重矩阵$W \in \mathbb{R}^{N \times N_{cc}}$的计算方式如下：

$$W_{ij} = \frac{e^{-D_{ij}}}{\sum_{} e^{-D_{ik}}} \quad (34)$$

其中，$D \in \mathbb{R}^{N \times N_{cc}}$是 L2 距离矩阵，用于衡量每个数据点与每个聚类中心之间的 L2 距离。然后，我们使用 W 而非硬分配来计算公式 28 中的 m 和 S。

此外，我们通过实验观察到，在早期迭代中，W 的分布需要更平滑，以防止聚类中心失效。相比之下，在后期迭代中，W 的更尖锐分布有助于实现更好的收敛。因此，我们在公式 35 中引入温度退火来调节分布：

$$W_{ij} = \frac{e^{-\frac{D_{ij}}{T}}}{\sum_{} e^{-\frac{D_{ik}}{T}}} \quad (35)$$

其中，T 是温度参数，它会随着迭代次数呈指数衰减。

A.3. 码本大小分析

我们的方法使用带有码本的矢量量化来压缩 KV 缓存，这需要额外的 GPU 内存来存储码本。码本配置如表 8 所示，并以 FP16 格式存储。总码本大小计算如下：值码本大小（MB）= Nc × d × 2（36）

$$KeyCodebookSize(MB) = 2 \times 2 \times N_c' \times R \times \frac{d}{g} \times 2 \quad (37)$$

其中 d 是大语言模型（LLM）的隐藏层大小。对于 LLaMA-3.1-8B-Instruct 模型，$d = 1024$。N_c是值码本中的行数。对于键码本，N_c'表示量化等级的数量，R 表示残余量化的数量。

我们分析了 LLaMA-3.1-8B-Instruct 模型在 2 位和 1 位量化时的码本大小，如表 7 所示。作为对比，上下文长度为 128K 的 KV 缓存，其值和键各需要 256 MB。值得注意的是，码本大小不随令牌数量变化而改变，这使得其在长上下文大语言模型推理中的内存开销可以忽略不计。

A.4. 交换码本配置的消融研究

我们使用专门设计的可交换码本对关键缓存进行量化，如 4.2 节所述。有三个超参数控制压缩级别：N_c'（量化级别数）、R（残差量化数）和 g（一组中共享相同量化向量 s 的子向量数）。回顾计算平均量化比特的公式：

$$Avg.bit = \frac{Rlog2(N_c')}{g} \quad (38)$$

本节中的所有实验均使用 LLaMA-3.1-8B-Instruct 模型进行，量化误差（以均方误差 MSE 衡量）是通过 LLaMA 模型第一层的键缓存，在 FineWeb-Edu 数据集（Lozhkov 等人，2024）的一个子集上计算得出的。

	Avg. Bit	
	1 bit	2 bit
Value Codebook	2.00 MB	4.00 MB
Key Codebook	2.75 MB	5.25 MB

表 7. 码本大小分析。我们基于 LLaMA-3.1-8B-Instruct 模型计算码本大小。

	1 bit	2 bit
N_c	1024	2048
R	11	21
$N_{c'}$	64	64

表 8. 码本配置。N_c用于值码本，而 R、$N_{c'}$用于键码本。

在我们的第一项消融研究中，我们在保持平均量化比特数相同的情况下，研究了 g 对$R-1$的影响，如表 9 所示。我们还针对不同的 g 改变 R，以将平均量化比特数保持为 1，如表 10 所示。

g	$N_{c'}$	R	MSE
8	2	1	0.2699
16	4	1	0.2011
32	16	1	0.1265
64	64	1	**0.0906**

表 9. 在保持平均比特数一致的情况下，使用$R=1$对 g 进行的消融研究。

g	$N_{c'}$	R	Avg. bit	MSE
8	2	8	1 bit	0.0798
16	4	8	1 bit	0.0790
32	16	8	1 bit	0.0254
64	64	11	1 bit	**0.0095**

表 10. 在保持平均比特数为 1 的情况下，对 g 和 R 进行的消融研究。

从表 9 和表 10 中我们可以得出结论，当保持平均量化位数相同时，更大的 g 会导致更低的量化误差，不过这是以$N_{c'}$数量的增加为代价的，正如式 23 所述，这会导致计算复杂度的增加。对比表 9 和表 10 中具有相同 g 和$N_{c'}$的同一行，我们可以进一步得出结论，当保持 g 和$N_{c'}$不变时，更大的 R 会带来更低的量化误差。

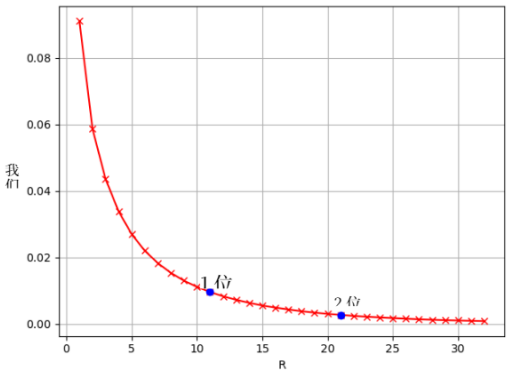


图 4. 保持$g=64$和$N_{c'}=64$不变时对 R 的消融研究

接下来，我们保留$g=64$和$N_{c'}=64$，通过改变 R 来获取不同压缩率下的量化误差。结果如图 4 所示。1 位量化和 2 位量化的 R 值在图中用蓝色圆点标注。可以看出，增大 R 会持续降低量化误差，但代价是更高的平均量化位数。

总之，更大的 g 和更大的 R 会带来更好的量化精度，但也会导致更高的计算量和更低的压缩率。因此，要在量化精度、计算成本和

压缩率方面，我们为所有主要实验设置了$g = 64$、$N_c' = 64$，1 位量化设置为$R = 11$，2 位量化设置为$R = 21$。

A.5. 量化误差比较

我们进行了实验，以比较我们的方法与基准方法的量化误差（量化误差通过原始 KV 缓存和解码后的 KV 缓存之间的均方误差（MSE）来衡量）。具体而言，我们使用 LLaMA-3.1-8B-Instruct 模型第一层的数值缓存进行均方误差计算。非对称量化作为比较的基准。从表 11 可以看出，我们基于加法量化的方法优于非对称量化，尤其是在量化位数较低的情况下，例如 1 位量化时。

Method	**Avg. bit**	**MSE**
Asymmetric quant.	2 bit	0.00030
CommVQ		**0.00014**
Asymmetric quant.	1 bit	0.00380
CommVQ		**0.00027**

表 11. CommVQ 与 KIVI 中使用的非对称量化（Liu 等人，2024b）之间的均方误差（MSE）比较，该比较使用 LLaMA-3.1-8B-Instruct 第一层的缓存值矩阵计算得出。均方误差是在 FineWeb-Edu 数据集（Lozhkov 等人，2024）的一个小子集上评估的。