SwiftKV：一种面向边缘的注意力算法与多头加速器，用于快速、高效的大语言模型解码

Junming Zhang，*IEEE* 研究生会员，Qinyan Zhang，Huajun Sun，*IEEE* 会员，Feiyang Gao，Sheng Hu，Rui Nie，Xiangshui Miao，*IEEE* 高级会员

arXiv:2601.10953v1 [cs.AR] 16 Jan 2026

摘要—**大语言模型的边缘加速对于其广泛应用至关重要；然而，在资源受限的边缘加速器上实现快速注意力推理和高效解码仍然具有挑战性。本文提出了 SwiftKV 注意力，这是一种逐令牌流水线、低延迟的单遍注意力推理算法。在该算法中，KV 缓存中的每个** (k_t, v_t) **仅在统一的逐令牌流水线中处理一次，无需分数物化、分块 Softmax 或第二遍处理，从而能够在仅使用一组硬件且无需资源密集型并行的情况下，实现边缘加速器上的快速执行。此外，为了解决现有加速器对多头大语言模型解码支持有限的问题，我们设计了 SwiftKV-MHA 加速器，使高精度注意力和低精度通用矩阵-向量乘法能够在同一处理器阵列上运行，从而实现快速、高效的多头并行解码。实验结果表明，在边缘加速器上，SwiftKV 注意力算法相较于原生注意力实现了** $7.16\times$ **的速度提升，并显著优于其他注意力算法。SwiftKV-MHA 进一步将注意力延迟降低了** $13.48\times$**；在相同设置下，与当前最先进的工作相比，其生成速度提升了** 17.4% **，令牌效率提高了** $1.98\times$ **。**

索引词—**Transformer 解码、注意力加速、多头大语言模型、边缘加速器、现场可编程门阵列。**

I. 引言

面向 THE 大语言模型（LLM）的边缘加速对于降低系统延迟、保护数据隐私和实现实时智能系统具有重要意义，近年来也取得了显著进展[1]–[3]。然而，在边缘加速器上加速大语言模型时，注意力计算延迟会变得十分突出，并且通常成为解码阶段的主要瓶颈。在代表性系统中，注意力占端到端延迟的 33.5% [4]和 43% [5]；随着上下文变长，这一问题会进一步加剧——当上下文长度达到 16K 时，注意力所占比例升至 60.2%，从而显著减慢生成速度 [6]。

现有边缘加速器加速策略主要聚焦于稀疏化和量化 [7]–[9]。传统的在线 Softmax 仅优化 Softmax，并非针对注意力设计（$QK^{\mathsf{T}}_{\text{cache}}$， PV_{cache}），且因反复访问注意力中间结果，仍会产生大量内存流量 [10], [19]。在此基础上，Flash Attention 及相关分块方法通过内核融合和分块进一步减少 I/O 内存访问，但其主要针对 GPU 训练而设计，且预填充 [10], [11]。然而，在解码过程中，当新生成的词元超出预定义的块边界时，无法始终保持分块结构，导致计算不得不等待块，从而显著降低了其在解码中的有效性。

本研究得到国家重点研发计划（2023YFB4402402）和国家自然科学基金（62474074）资助。（Junming Zhang 和 Qinyan Zhang 对本研究贡献相同。通讯作者：Huajun Sun。）

Junming Zhang、Qinyan Zhang、Huajun Sun、Feiyang Gao、Sheng Hu、Rui Nie 和 Xiangshui Miao 均来自华中科技大学集成电路学院及湖北省先进存储器重点实验室，地址为中国武汉 430074。

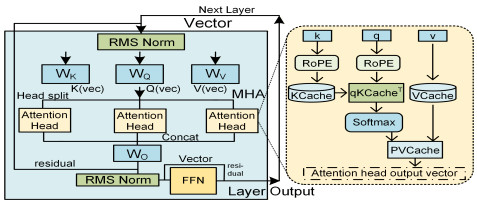

图 1. 解码过程中多头大语言模型层的结构。

具体而言，在资源受限的边缘加速器上，并行处理高维 $\mathbf{q}\mathbf{k}_i^{\mathsf{T}}$、Softmax 及后续运算颇具挑战，这会减慢注意力计算和词元生成 [5], [12]。因此，当仅有一组计算单元可用时，基于 GPU 的分块方法会失效，因为无法利用块级并行。此外，快速注意力推理通常需要密集的高精度乘法和逐步执行的*exp* [10], ，这与针对整数 MAC 阵列上的低比特 GEMV 优化的传统边缘加速器并不兼容。因此，亟需一种面向边缘的解码注意力算法，能够在不依赖并行的情况下实现快速、低延迟推理，并兼容低精度整数计算。

与此同时，在多头大语言模型的解码过程中，如图 1 所示，每个头都会独立执行完整注意力（包括旋转位置编码）。然而，传统的基于 GEMM 的加速方式会导致旋转位置编码、*exp* 和 Softmax 运算频繁发生跨模块数据移动，且对多头并行的支持不足 [12], [13]。目前，尚无处理器能够独立完成整个单头注意力工作流，也没有为快速解码优化的电路级旋转位置编码 [14]，因此亟需一种面向解码器的多头加速器。

在本文中，针对边缘加速器，我们提出了**SwiftKV 注意力**，这是一种用于解码阶段的快速注意力算法，以及与之对应的计算引擎 SwiftKVCore。我们进一步设计了**SwiftKV-MHA**，这是一种针对快速多头大语言模型解码优化的加速器。本文的贡献如下：

- 我们提出了**SwiftKV 注意力**，这是一种面向边缘、按令牌流水线执行的单遍注意力算法，无需额外的并行硬件计算资源，即可在边缘加速器上实现低延迟注意力。
- 我们设计了**SwiftKV-MHA**，这是一款多头加速器，采用

双模式 SKV 处理器阵列，其中每个处理器 (i) 可独立执行各个头的高精度注意力推理，(ii) 可执行高吞吐量低精度通用矩阵-向量乘法。它还集成了解码器专用 RoPEforfast，可实现高效的多头大语言模型解码。

- 我们对所提出的 SwiftKVAttention算法进行了评估和验证，并进一步分析和比较 SwiftKV-MHA 与当前最先进方法的性能。

II. 多头大语言模型解码结构

多头大语言模型在解码过程中单层的架构如图 1 所示。每个头都会对每头查询（$\mathbf{q}$）和键（$\mathbf{k}$）向量执行旋转位置编码（RoPE）。对于位置索引 $m \in \mathbb{N}$ 和基数 b （通常为$b = 10000$），定义角频率：

$$\omega_i = b^{-2(i-1)/d}, \quad i = 1, \ldots, d/2, \tag{1}$$

以及旋转角度：

$$\theta_i(m) = m\,\omega_i, \quad i = 1, \ldots, d/2. \tag{2}$$

RoPE 将每一对连续通道旋转 $\theta_i(m)$ 角度。用矩阵-向量形式表示：

$$\mathrm{RoPE}(\mathbf{q}, m) = \begin{pmatrix} \cos(m\theta_0) & -\sin(m\theta_0) & 0 & \cdots & 0 & \\ \sin(m\theta_0) & \cos(m\theta_0) & 0 & \cdots & 0 & \\ 0 & 0 & \cos(m\theta_1) & -\sin(m\theta_1) & \cdots & \\ 0 & 0 & \sin(m\theta_1) & \cos(m\theta_1) & \cdots & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & 0 & \cdots & \cos(m\theta_{d/2-1}) \\ 0 & 0 & 0 & 0 & \cdots & \sin(m\theta_{d/2-1}) \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix}. \tag{3}$$

对$\mathbf{k}$应用相同的旋转。随后计算注意力：

$$\mathrm{Attention}(\mathbf{q}, \mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}) = \mathrm{softmax}\left(\frac{\mathbf{q}\,\mathbf{K}_{\text{cache}}^{\top}}{\sqrt{d}}\right)\mathbf{V}_{\text{cache}}. \tag{4}$$

在本研究中，我们采用两个参数规模不同的大语言模型*LLaMA2-7B* 和*ChatGLM-6B*，用于硬件加速与验证。

III. 提出的SwiftKV-ATTENTION和SwiftKV-CORE

不同于传统的 Flash Attention 及其他针对 GPU 上预填充或训练的注意力加速算法， [10], [11] SwiftKV 可在边缘加速器上实现**逐令牌流水线、低延迟的单遍**注意力计算。

在解码过程中， $q \in \mathbb{R}^{1\times d}$ 每个注意力头中。对于 KV 缓存中的每个 k 和 v 向量（k_t， v_t），令牌 t处的注意力分数计算如下

$$s_t = \frac{qk_t^{\top}}{\sqrt{d}}. \tag{5}$$

SwiftKV 不会物化注意力分数，也不会执行分块 Softmax，而是对每个（k_t， v_t）仅处理一次。该分数 s_t 会与上一次注意力迭代中的 μ_{t-1} 进行比较。当 $s_t \le \mu_{t-1}$时，

$$\begin{cases} \beta = \exp(s_t - \mu_{t-1}), \quad Z_t = Z_{t-1} + \beta, \\ Y_t = Y_{t-1} + \beta v_t, \quad \mu_t = \mu_{t-1}. \end{cases} \tag{6}$$

当 $s_t > \mu_{t-1}$,

$$\begin{cases} \alpha = \exp(\mu_{t-1} - s_t), \quad Z_t = \alpha Z_{t-1} + 1, \\ Y_t = \alpha Y_{t-1} + v_t, \quad \mu_t = s_t. \end{cases} \tag{7}$$

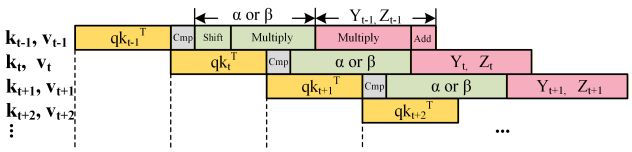

图 2. 面向边缘加速器的 SwiftKV 注意力流水线计算流程。

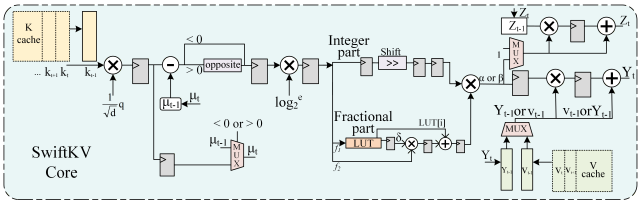

图 3. SwiftKV 注意力的硬件计算引擎：SwiftKV Core。

$Z_t \in \mathbb{R}$ 和 $Y_t \in \mathbb{R}^{1\times d}$。初始时， $\mu_1 = s_1$、 $Z_0 = 0$ 和 Y_0 被初始化为零向量。通过延后除法步骤，在处理完 KV 缓存中的所有 k 和 v 向量后，只需进行一次归一化即可得到最终的注意力输出：

$$\mathrm{Attention} = \frac{Y_T}{Z_T}. \tag{8}$$

在 SwiftKV 中，指数因子 α 和 β 始终位于 $(0, 1]$, ，因此适合通过硬件高效实现。指数的计算方式如下：

$$\exp(s_t - \mu_{t-1}) = 2^{(s_t-\mu_{t-1})\log_2 e} = 2^{n+f}, \tag{9}$$

其中， n表示整数部分，通过位移实现； f 表示小数部分。

在解码过程中，式 (5)–(9) 使边缘加速器能够以逐令牌流水线式的单遍过程计算注意力，只需完整扫描 KV 缓存一次。每个输入的 (k_t， v_t) 仅被处理一次，用于更新 (Z_t， Y_t)，从而避免分数物化、重新访问先前的令牌或执行第二遍计算。计算遵循统一的逐令牌操作序列，并且可自然地进行流水线处理：在后处理 qk_{t-1}^T的同时，为点积单元提取下一个 k_t 。由于高维 qk_i^T主导关键路径，其余所有更新都可以安排在其延迟时间内完成。因此，无需资源密集型并行，即可在单个硬件组上高效执行低延迟注意力推理。

此外，SwiftKV 在注意力计算中采用 32 位定点算术（FXP32、Q15.17），精度优于10^{-5}，足以满足准确推理的需求。这种定点设计可以复用用于低位宽整数通用矩阵-向量乘法的标准乘累加阵列，无需专用的注意力数据通路。

相应的硬件加速核心如图 3 所示。SwiftKV Core 由一个单独的 qk_i^T 点积部件、一个比较选择部件、一个*exp* 部件，以及一个用于（Z_t， Y_t）的更新部件组成。对于 KV 缓存中的每个键值对（k_t， v_t），它支持流水线 KV 缓存读取，并针对每个令牌执行单遍注意力计算。为使用 $f \in (-1,\ 0]$ 计算公式（9）中的分数项，我们使用带线性插值的 5 位查找表（LUT）对2^f 进行近似。我们将 f 分解为 $f = f_1 + f_2$，其中 f_1 包含前 5

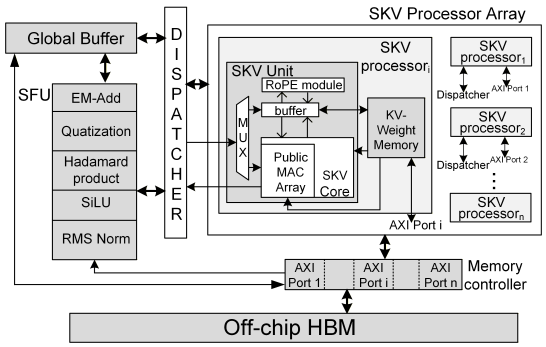

图 4. SwiftKV-MHA 加速器的整体架构。

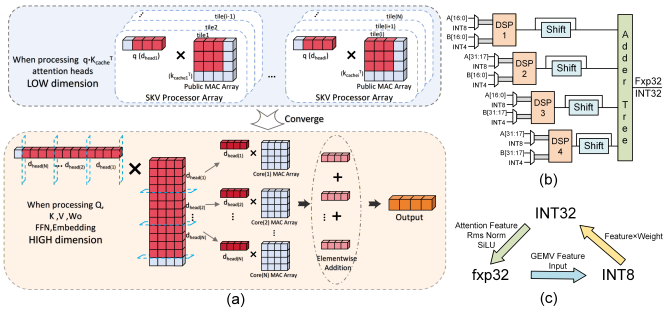

图 5. 快速通用矩阵-向量乘法与注意力计算示意图。(a) 使用多头计算逐注意力头注意力和高维通用矩阵-向量乘法。(b) 高精度 FXP32 注意力与低精度多路径通用矩阵-向量乘法共享相同的计算资源。(c) 计算过程中的精度转换。

个最高有效小数位，而 f_2 包含剩余的 12 位，从而得到索引 $i \in \{0, \ldots, 31\}$。查找表存储 $\mathrm{LUT}[i] = 2^{-i/32}$ 及相应的斜率 δ_i，并计算

$$2^f = \delta_i \cdot f_2 + \mathrm{LUT}[i]. \tag{10}$$

然后将该小数部分结果与移位项结合，得到整体指数。

IV. 多头加速器设计

A. 整体架构

SwiftKV-MHA 的整体架构如图 4 所示。它由 SKV 处理器阵列、分派器、特殊功能单元（SFU）、全局缓冲区和内存控制器组成。对于 6B–10B 范围内广泛使用的边缘导向模型，例如 *ChatGLM–6B*、*LLaMA 2–7B*、*LLaMA 3–8B*、*Qwen3–8B*，它们主要采用 32 头多头注意力。处理器阵列包含 32 个独立的 SKV 处理器，每个处理器负责一个头的注意力计算。每个处理器都配有一个 SKV 单元和专用的 KV 权重存储器。SKV 单元独立执行旋转位置编码和逐注意力头注意力推理，而 KV 权重存储器则提供分区的权重矩阵和 KV 缓存。

分派器在阵列、全局缓冲区和特殊功能单元之间传输数据。它将高维向量 $x \in \mathbb{R}^{1\times 4096}$ 分发给 32 个处理器，并将结果收集到全局缓冲区，或转发至特殊功能单元。特殊功能单元负责非乘累加运算：逐元素加法（EM-Add）、量化与类型转换（FXP32/INT32/INT8）、哈达玛积、SiLU 和均方根归一化。全局缓冲区暂存中间向量（**Q**、**K**、**V**、注意力输出和前馈网络激活值），并存储每一层的输入和输出。

在推理过程中，每个 Transformer 层都以 **W4A8** 精度运行。8 位输入向量 $x \in \mathbb{R}^{1\times 4096}$ 被分派到处理器阵列，以计算 **Q**、**K** 和 **V**。对于注意力，数据在特殊功能单元中转换并按头拆分；每个 SKV 处理器应用旋转位置编码并计算注意力。各头的输出经拼接和转换后，送回处理器阵列进行前馈网络计算。最后，前馈网络输出在特殊功能单元中经过均方根归一化和 SiLU 处理，并写回全局缓冲区，供下一层使用。

B. 快速通用矩阵-向量乘法与注意力计算

如图 5(a) 所示，SKV 处理器阵列同时支持高精度注意力和高吞吐量低精度整数通用矩阵-向量乘法。计算注意力时，对于向量$\mathbf{Q} \in \mathbb{R}^{1\times 4096}$，多头注意力会将其拆分为 32 个头，每个头由一个处理器处理。每个 SKV 单元中的 PublicMACArray 计算每个头的 qk_i^T，其中 q，$k_i \in \mathbb{R}^{1\times 128}$。执行 INT8 输入与 INT4 权重之间的通用矩阵-向量乘法时，输入也会被划分为 32 个、每个维度为 128 的数据块。每个处理器将分配给它的输入数据块与相应的权重矩阵相乘，生成一个部分结果；随后，特殊功能单元使用 EM-Add 将这 32 个部分结果相加，得到通用矩阵-向量乘法的输出。

借助 SwiftKV 注意力采用 FXP32 算术，同一套硬件可同时用于高精度注意力和权重向量通用矩阵-向量乘法。FXP32×FXP32→FXP32 和 INT4×INT8→INT32 均视为定点乘法（FXP(Q15.17)、INT4(Q4.0)、INT8(Q8.0)），因此可以共用乘累加单元，如图 5(b) 所示。一次典型的 FXP32×FXP32 乘法需要 4 个 DSP48E2 单元（27×18），而每次 INT4×INT8 运算只需一个数字信号处理器。对于通用矩阵-向量乘法，SKV 单元中的每个公共乘累加阵列包含 128 个数字信号处理器，因此每周期可执行一个 128 维点积。32 个处理器并行运行时，系统可通过单遍处理完成 4096 维点积，并借助流水线每周期生成一个输出元素。对于注意力，格式切换为 FXP32，其中点积中每个维度的乘法需要占用四个数字信号处理器。因此，同一个乘累加阵列每周期可计算一个 32 维点积，每个 qk_i^T需要 4 个周期。采用逐词元单遍流水线处理（见图 2），对上下文长度为 N 的序列执行注意力计算约需 $4N$ 周期数。

如图 5(c) 所示，每个 SKV 处理器对 xW_q、xW_k 和 xW_v 执行通用矩阵-向量乘法，生成 INT32 部分和，并在特殊功能单元中将其量化为 FXP32，以进行逐头注意力计算，随后再转换回 INT8 用于 xW_o。大多数数据类型转换都通过流水线与计算重叠，从而能够跨层高效执行通用矩阵-向量乘法和注意力计算。

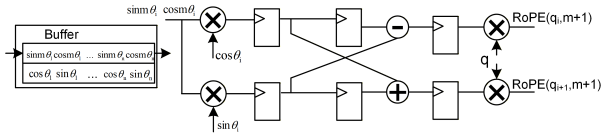

图 6. 解码器专用 RoPE 的框图。

表 I SwiftKV-MHA 的令牌推理准确率

	Top-1	Top-2	Top-3	Top-5
准确率	100%	100%	99%	98%

C. 单头注意力中的解码器专用 *RoPE*

在硬件中实现传统的旋转位置编码颇具挑战 [14]: CORDIC 仅支持 $[-\pi/2, \pi/2]$ 中的角度，而随着上下文长度增加，大语言模型中的旋转位置编码角度 $\gamma_{i,\ j} = i\theta_j$ 可能会变得很大。对于*LLaMA2-7B* 等模型，直接计算 $\cos(i\theta_j)$ 和 $\sin(i\theta_j)$ 的硬件开销很大。

解码期间，词元按顺序生成。对于位置为 $m{+}1$ 的新词元，m 等于当前上下文长度。我们将初始的 $\cos(\theta_i)$ 和 $\sin(\theta_i)$ 作为常量 a_i 和 b_i 存储在每个 SKV 单元 $i \in [1, d/2]$ 中，并缓存先前的 $\cos(m\theta_i)$ 和 $\sin(m\theta_i)$。因此，新生成词元的 RoPE 更新变为:

$$\begin{cases} \text{RoPE}(q_i, m+1) = q_i(a_i \cos m\theta_i - b_i \sin m\theta_i) - q_{i+1}(a_i \sin m\theta_i + b_i \cos m\theta_i), \\ \text{RoPE}(q_{i+1}, m+1) = q_i(a_i \sin m\theta_i + b_i \cos m\theta_i) + q_{i+1}(a_i \cos m\theta_i - b_i \sin m\theta_i). \end{cases} \tag{11}$$

如图 6 所示，这只需要四个乘法器；通过流水线，更新后的 q'_i 和 q'_{i+1} 可在三个周期数内生成。新的 RoPE(q, $m+1$) 保存在单元缓冲区中，而 RoPE(k, $m+1$) 则作为更新后的 KV 缓存写入高带宽内存。由于所有缓存的键都已进行位置编码，因此只有新词元的查询需要应用 RoPE，从而无需对整个 K 矩阵重新编码，并显著降低解码复杂度。

计算出**V** 向量后，从缓冲区读取 RoPE($q, m+1$) 并执行注意力计算。如图 4 所示，完整注意力头在全部 32 个 SKV 处理器上并行独立运行。因此，SKV 单元能够完成完整的单头注意力流程，并减少跨模块数据移动和等待延迟。

V. 实验结果

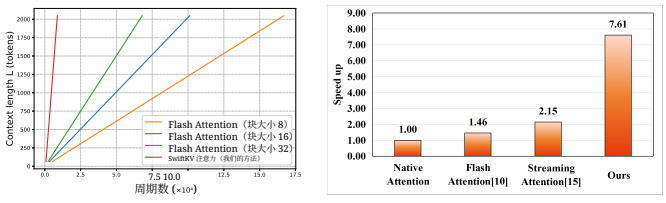

图 7. (a) SwiftKV 注意力与 Flash Attention 在不同块大小下的注意力计算时间随上下文长度的变化。(b) 在相同上下文长度下，SwiftKV 注意力与其他注意力基线的注意力计算加速比比较。

表 II SwiftKV-MHA 在 ALVEO U55C 上的硬件利用率

组件	LUT	FF	BRAM	DSP
SFU	14K	15K	46	38
分派器	148K	65K	0	0
处理器阵列	355K	328K	224	4480
全局缓冲区	0	0	136	0
总计	**517K** (39.6%)	**408K** (15.6%)	**406** (20.1%)	**4518** (50.1%)

我们评估并比较 SwiftKV 注意力加速算法的性能。为确保比较公平，所有设计均在相同的现场可编程门阵列平台上实现，采用相同的高带宽内存配置、相同的一组 *exp*单元，以及相同的流水线乘除单元来计算 qK^T、PV 和执行归一化。

我们首先使用分块方法，将 SwiftKV 与传统的 Flash Attention 进行比较。如图 7(a) 所示，在不同上下文长度和 Flash Attention 块大小（8/16/32）下，SwiftKV 的注意力延迟始终显著低于 Flash Attention。此外，在相同硬件条件下，我们比较了上下文长度固定为 512 时不同注意力加速方法的注意力计算速度。如图 7(b) 所示，以原生注意力为基准并将其归一化为 1，Flash Attention（块大小为 32） [10] 实现了 1.46× 倍加速，Streaming Attention [15] 实现了 2.15× 倍加速，而 SwiftKV 达到了 7.16× 倍。这显著提升了边缘加速器在解码期间的注意力计算速度，并降低了端到端延迟。

我们还根据公式 (9)–(10) 评估了基于查找表的 *exp* 实现的近似误差。在区间 $(-1, 0]$ 内，最大相对误差为 0.00586%，足以满足高精度前向推理的要求。为验证加速器的准确率，我们从 PG-19 数据集中抽取 100 个长度为 512 的序列，并运行*LLaMA2-7B*。我们对输出逻辑值进行排序，并将 Top-1 至 Top-5 词元与采用相同 W4A8 精度的桌面端结果进行比较。如表 I 所示，Top-1 和 Top-2 的准确率均为 100%，表明该方案适用于实际应用。SwiftKV-MHA 及相关实验使用 Vivado 2022.2 进行综合，并部署在 AMD Xilinx U55C 开发板上。资源利用率见表 II。

SwiftKV 注意力算法及 SKV 处理器阵列兼具高精度注意力计算和高吞吐量通用矩阵-向量乘法能力——如图 8(a) 所示，注意力延迟仅占大语言模型推理总延迟的 3.19%。与 [5], 中报告的原始 43.0% 相比，这一比例降低了 13.48×，证实了该加速器能够快速执行高精度注意力计算。对于通用矩阵-向量乘法运算（权重 × 激活值），处理器阵列可在单个周期内完成 4096 维向量的点积。在 225 MHz 下，其吞吐量达到 1836 GOPS，实现了高性能的通用矩阵 - 向量乘法计算。

表III展示了SwiftKV-MHA与其他运行*LLaMA2-7B*和*ChatGLM-6B*的加速器之间的对比。在相同的设置和条件下，每词元解码延迟仅为12.3毫秒，生成速度达到每秒81.5个词元，比现有

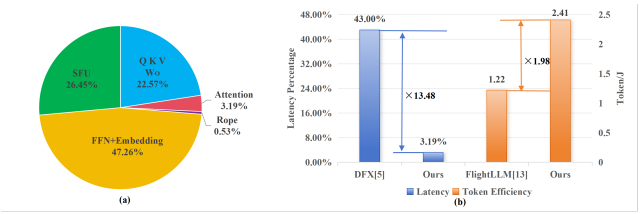

图 8. (a) 推理过程中*Llama2- 7B* 各模块的解码延迟分解。(b) SwiftKV-MHA 与其他加速器在注意力延迟（左）和令牌生成效率（右）方面的比较。

表 III 在相同实验设置下，面向 Llama-2 7B 和 ChatGLM-6B 推理的最先进基于现场可编程门阵列的变换器加速器性能比较

	FlightLLM [13] (U280)	EdgeLLM [9] (VCU128)		本工作 (U55C)	
模型	Llama-2-7B	Llama-2-7B	ChatGLM-6B	**Llama-2-7B**	**ChatGLM-6B**
量化 Type	~W4A8	W4A8		**W4A8**	
HBM 带宽	460GB/s	460GB/s		**460GB/s**	
频率 (MHz)	225	225		**225**	
使用了数字信号处理器	6345	4563		**4518**	
延迟 (ms)	18.2	14.4	11.7	**12.3**	**10.4**
速度 （令牌/秒）	55	69.4	85.8	**81.5**	**96.3**
系统功耗 (W)	45	56.8		**33.8**	
令牌/焦耳	1.22	1.22	1.51	**2.41**	**2.85**

表IV 与现有基于现场可编程门阵列的工作的比较

相关工作	MICRO'22 [5]	TCAS-I'23 [16]	ASP-DAC'24 [17]	TCAS-I'25 [18]	本文工作
平台	Alveo U280	ZCU102	Alveo U280	Alveo U50	**Alveo U55C**
模型	GPT2-1.5B	视觉 变换器	BERT-base	Swin 变换器	**Llama-2-7B**
频率 (MHz)	200	300	220	170	**225**
吞吐量 （每秒十亿次运算）	184.1	726.7	757.4	830.3	**1100.3**
能效 （每瓦每秒十亿次运算）	4.09	28.2	25.1	45.12	**60.12**

最先进的工作 [9]高出17.4%。SwiftKV-MHA现场可编程门阵列芯片的归一化功耗为18.3瓦，其中高带宽内存的功耗约为15.5瓦。能效如图8(b)所示。与以往最先进的工作 [9], [13]相比，令牌生成效率提升了1.98×。对于*LLaMA2-7B*，当上下文长度为512时，生成单个词元需要进行13.5十亿次运算。因此，SwiftKV-MHA的总体吞吐量达到1100.3每秒十亿次运算（13.5G × 81.5）。表IV将我们的工作与不同的基于注意力的变换器模型现场可编程门阵列加速器进行了比较。显而易见，SwiftKV-MHA在吞吐量和能效方面均优于以往的设计。

VI. 结论

我们提出了 SwiftKV 注意力，这是一种面向边缘、逐令牌、单遍注意力推理算法；同时提出 SwiftKV-MHA，这是一种快速、高能效的多头大语言模型解码加速器。SwiftKV 将注意力延迟降低了 13.48×，相较于原生注意力实现了 7.16× 的加速，同时速度提升 17.4%，能效也比最先进的研究成果 1.98× 更高。未来，该方案还可进一步与人工智能智能体集成，为实用的边缘侧大语言模型应用提供动力。

参考文献

[1] Dettmers T、Lewis M、Belkada Y 等。Gpt3. int8（）：面向大规模Transformer模型的 8位矩阵乘法[J]。载于《神经信息处理系统进展》，2022，35：30318-30332。[2] Xiao G、Lin J、Seznec M 等。SmoothQuant：面向大语言模型的精确且高效的训练后量化 [J]。载于《国际机器学习会议论文集》，2023：38087–38099。[3] Kim S 等。C-Transformer：一种具有大小核网络和隐式权重生成功能、面向大语言模型的 2.6–18.1 μJ/令牌同构DNN-Transformer模型/脉冲Transformer处理器 [J]。载于《IEEE国际固态电路会议论文集》（ISSCC），2024：368-369。[4] Z. Zhao、R. Cao、K.-F. Un、W.-H. Yu、P.-I. Mak 和 R. P. Martins，“一种采用输出块驻留数据流、用于目标识别应用的基于FPGA的Transformer加速器”，载于《IEEE电路与系统汇刊II：简报》，第70卷，第1期，第281-285页。[5] S. Hong 等，“DFX：一种用于加速基于Transformer的文本生成的低延迟多FPGA设备”，载于2022年第55届IEEE/ACM国际微体系结构研讨会（MICRO），2022，第616-630页。[6] A. K. Kamath、R. Prabhu、J. Mohan、S. Peter、R. Ramjee 和 A. Panwar，“POD-Attention：释放完整的预填充-解码重叠以实现更快的大语言模型推理”，载于《ACM程序语言与操作系统架构支持国际会议论文集》（ASPLOS），第2卷，2025，第897–912页。[7] M. Zhang、J. Cao、K. Shi 等，“FNM-Trans：具有完整 N:M稀疏性的高效基于FPGA的Transformer架构”，载于第61届ACM/IEEE设计自动化会议（DAC）论文集，2024，第1-6页。[8] P. Zhao 等，“HEAT：采用混合精度量化的高效视觉Transformer加速器”，载于《IEEE电路与系统汇刊II：简报》，第72卷，第5期，第758-762页，2025年5月。[9] M. Huang 等，“EdgeLLM：一种面向大语言模型的高能效CPU-FPGA异构边缘加速器”，载于《IEEE电路与系统汇刊I》，第72卷，第7期，第3352-3365页。[10] T. Dao、D. Y. Fu、S. Ermon、A. Rudra 和 C. R´e，“FlashAttention：具有IO感知的快速、内存高效精确注意力”，载于《神经信息处理系统进展》，第35卷，2022，第16344–16359页。[11] H. Liu、M. Zaharia 和 P. Abbeel，“采用分块Transformer模型实现近无限上下文的RingAttention”，载于《国际学习表征会议》（ICLR）论文集，2024。[12] J. Li 等，“在嵌入式FPGA上突破内存带宽和容量利用率极限，实现高效大语言模型解码”，载于《设计、自动化与测试欧洲会议》（DATE）论文集，2025，第1–7页。[13] S. Zeng 等，“FlightLLM：采用完整映射流程在FPGA上实现高效大语言模型推理”，载于《ACM/SIGDA现场可编程门阵列国际研讨会》（FPGA）论文集，2024年4月，第223–234页。[14] W. Li、G. Wang、D. Lyu、N. Xu 和 G. He，“大语言模型旋转位置嵌入的高效硬件架构设计”，载于《IEEE新兴与重点电路及系统专题期刊》，第15卷，第2期，第244-257页，2025年6月。[15] G. Islamoglu 等，“ITA：面向量化Transformer的能效型注意力和Softmax加速器”，载于2023年IEEE/ACM低功耗电子与设计国际研讨会（ISLPED），2023，第1-6页。[16] M. Huang、J. Luo、C. Ding、Z. Wei、S. Huang 和 H. Yu，“一种用于在边缘高效映射视觉Transformer的纯整数组向量脉动加速器”，载于《IEEE电路与系统汇刊I》，第70卷，第12期，第5289–5301页，2023年12月。[17] H. Wang、Y. Bai、J. Yu 和 K. Wang，“TransFRU：采用资源充分利用技术在FPGA上高效部署Transformer”，载于第29届亚太设计自动化会议（ASP-DAC）论文集，2024，第521-526页。[18] Q. Dong、S. Zhang 和 Z. Wang，“一种通过混合粒度稀疏性实现的高效基于窗口的视觉Transformer加速器”，载于《IEEE电路与系统汇刊I》，第72卷，第9期，第4751-4764页，2025年9月。[19] Milakov M、Gimelshein N。Softmax的在线归一化器计算[J]。arXiv预印本，arXiv:1805.02867，2018。