FlexLLM：用于灵活混合大语言模型加速器设计的可组合高层次综合库

张家豪*、何子凡*、Nicholas Fraser†、Michaela Blott†、孙逸舟*、丛京生*
*计算机科学，加利福尼亚大学洛杉矶分校，加利福尼亚州
†AMD，都柏林，爱尔兰

arXiv:2601.15710v1 [cs.AR] 22 Jan 2026

摘要——**我们提出了 FlexLLM，这是一种可组合的高层次综合（HLS）库，用于快速开发领域专用大语言模型加速器。FlexLLM 提供了面向阶段定制推理的关键架构设计空间，使混合设计能够针对预填充和解码分别定制时间复用与空间数据流，并提供完整的量化套件以支持准确的低比特部署。借助 FlexLLM，我们仅用不到两个月、仅编写 1K 行代码，就为 Llama-3.2 1B 模型构建了完整的推理系统。该系统包括：（1）采用硬件高效量化的阶段定制加速器（12.68 WikiText-2 困惑度），性能超越 SpinQuant 基线；（2）用于高效长上下文处理的分层记忆 Transformer（HMT）插件。在采用 16 纳米工艺的 AMD U280 FPGA 上，与运行 BF16 推理的英伟达 A100 GPU（7 纳米）相比，该加速器实现了 1.29× 倍端到端加速、1.64× 倍更高的解码吞吐量以及 3.14× 倍更高的能效；在采用 7 纳米工艺的 V80 FPGA 上，预计分别可达到 4.71×倍、6.55×倍和 4.13×倍。在长上下文场景中，集成 HMT 插件可将预填充延迟降低 23.23× ，并将上下文窗口扩展 64×，相较于 A100 基线，在 U280/V80 上分别实现 1.10×/4.86× 倍更低的端到端延迟和 5.21×/6.27× 倍更高的能效。因此，FlexLLM 以极少的人工工作弥合了大语言模型推理中的算法创新与高性能加速器之间的鸿沟。**

I. 引言

大语言模型（LLM）已成为现代最具变革性的技术之一，在自然语言处理 [1], [2], 代码生成 [3], [4], 和多模态理解 [5], [6]等领域取得了突破。尽管云服务提供商仍在大力投资基于 GPU 的基础设施，以支持大规模推理 [7], ，但市场对**领域专用加速器**的需求日益增长。这类加速器可在不同部署环境中提供定制化、高效且可扩展的替代方案，覆盖从数据中心 [8], [9]到边缘平台 [10], [11]的各种场景。

与通用中央处理器和 GPU 相比，领域专用加速器在以往的深度神经网络工作负载（例如卷积神经网络 [12], [13], 残差网络 [14], [15], [16]）中展现出显著的性能和能效优势。然而，现代大语言模型与传统深度神经网络有很大不同：它们包含数十亿个参数，且计算和内存访问模式 [17], [18], [19], 更加复杂，因此对计算吞吐量、内存带宽和架构灵活性的需求高得多。这些特性给高效大语言模型加速器的设计带来了三项关键挑战：

挑战 1：预填充与解码阶段的计算与内存行为差异。基于 Transformer 解码器的大语言模型推理包含两个瓶颈截然不同的阶段。在预填充阶段，模型并行处理提示，提供较高的算术强度和丰富的并行性。在解码阶段，词元以自回归方式生成；计算主要受数据依赖和频繁的内存访问影响，因此严重受内存带宽限制。因此，同一模型在预填充阶段通常受计算限制，而在解码阶段则受内存带宽和依赖关系限制，导致单一服务流水线中存在相互冲突的优化目标。

现有 FPGA 加速器主要采用**时序架构** [8], [20] 或**空间架构** [21], [22], [23], [24]。时序设计在各层之间复用共享计算引擎，但由于缓冲容量有限，在预填充阶段会产生频繁的片外流量，并且难以在单个引擎内支持异构内核和不同精度（图 1(b)(c)）。空间设计将内核映射到专用模块，并通过片上 FIFO 流式传输中间数据，但当内核延迟不均衡或固有依赖占主导时，会出现流水线停顿（图 1(d)(e)）。尽管两种范式存在差异，但它们通常共享同一个隐含选择：使用单一统一架构服务于两个阶段。

然而，由于预填充和解码受不同资源的瓶颈制约，统一的（时序、空间或甚至混合）设计不可避免地会对一个阶段过度优化，却无法充分满足另一个阶段的需求。换言之，任何**“一刀切”**的架构从根本上都不适配大语言模型服务：预填充和解码必须针对各自阶段进行定制，让每个阶段采用不同组合的空间并行与时间复用，以匹配其各自不同的计算与内存约束。

挑战 2：平衡模型准确率与低比特计算和内存效率。大语言模型的参数数量极其庞大，因此压缩对于在加速器上部署至关重要 [9], [20]。量化是降低计算和内存开销的有效方法；然而，先前的 FPGA 大语言模型加速器往往依赖简单的整数型量化，当量化位宽降至接近 GPU 级吞吐量所需的激进低比特范围时，模型准确率可能严重下降，导致这类设计不适用于实际部署。例如，对 Llama 3.2-1B 应用 INT4 SmoothQuant [25] 或 GPTQ [26] ，会使其在 WikiText-2 [27]上的困惑度升至 $1e2$ 以上。鉴于 FPGA 相较于 GPU 的计算能力有限，实用的大语言模型加速必须依赖先进的量化方法与面向硬件的协同设计，而非简单的量化流水线。

挑战 3：特定模型加速器的人工工作量大、迭代速度慢。随着大语言模型架构快速演进，开发特定模型加速器仍然需要投入大量人力。基于 RTL 的实现可能超过 10 万行代码，并带来陡峭的学习曲线，使其

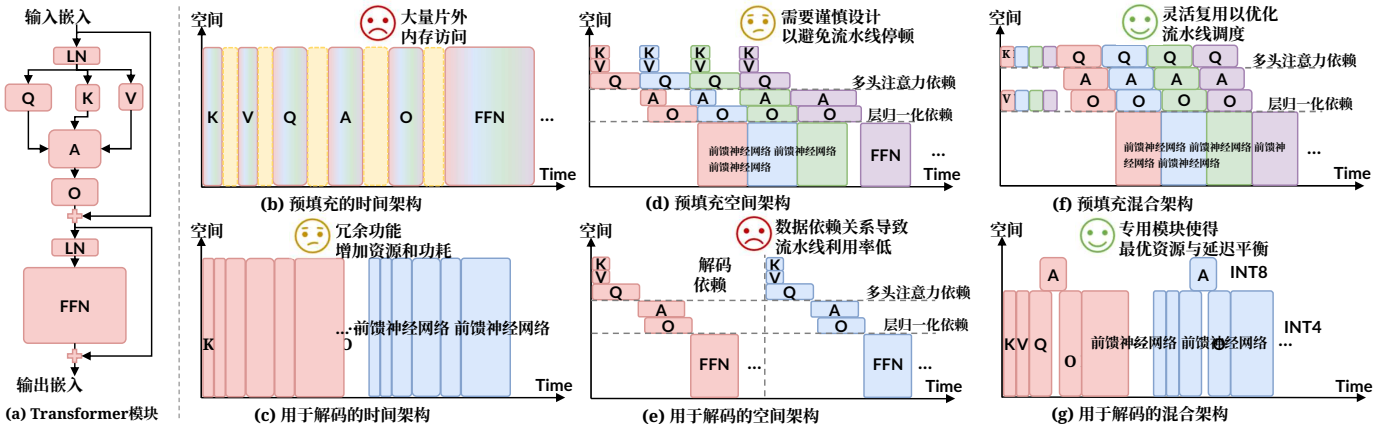


图 1. 大语言模型加速器在预填充和解码阶段采用的三种架构风格对比。不同颜色表示不同的词元，块表示硬件模块。为清晰起见，图中仅展示线性层，其中 A 表示多头注意力（此处为分组查询注意力），O 表示输出投影。(a) 一个 Transformer 模块。(b–c) 时序架构通过模块复用实现较高的利用率，但会受到频繁片外访存和灵活性有限的影响。(d–e) 空间架构为每个内核配置专用模块，实现完全片上流处理，但容易受到流水线停顿的影响。(f–g) 混合架构结合时间复用和空间并行性，在利用率、延迟和灵活性之间取得平衡。

难以跟上大语言模型创新的步伐进行迭代。近期的高层次综合框架（如 Allo [23] 和 StreamTensor [24]）通过基于 MLIR 的映射和自动化编译，减轻了 RTL 的负担，但它们主要面向空间架构，对于专门优化预填充和解码的混合设计支持有限。

为缩小这一生产力差距，亟需一种支持灵活架构设计的高级编程框架，以应对面向大语言模型的加速器的快速发展。如果缺少可组合的框架，加速器开发周期（通常为 1–2 年）就无法跟上大语言模型的快速演进；新模型往往每隔几个月就会发布。我们需要一种框架，能够开放关键的架构设计自由度——尤其是面向不同阶段的定制化混合设计和先进量化技术——同时便于机器学习和加速器开发人员使用。

受这些挑战的启发，我们开发了**FlexLLM**，这是一个用于领域专用大语言模型加速器设计的可组合高层次综合库。FlexLLM 基于 TAPA [28], 构建，提供高度参数化的模板化模块，可快速构建混合加速器。值得注意的是，据我们所知，***FlexLLM*** 首次明确突破了统一设计范式，实现了按阶段定制的混合架构。这显著提升了设计灵活性和加速器性能，同时为未来的大语言模型加速器开发提供了可复用的方法。借助 FlexLLM，我们构建了一个高性能加速器系统，仅用不到 1K 行 C++ 代码，便集成了先进的大语言模型技术。这证明 FlexLLM 能够快速融入新的算法创新，同时大幅减少开发工作量。论文发表后，我们将开源 FlexLLM。

总而言之，我们的主要贡献包括：

- 我们提出 FlexLLM，这是一个可组合高层次综合库，支持为大语言模型加速器灵活构建混合式、各阶段定制的架构。其高度模板化的模块可快速进行针对特定模型的定制，同时显著降低设计复杂度和代码量。
- 我们集成了一套全面的量化技术栈，涵盖动态和静态变体、多种对称性与粒度选项，以及异常值处理模块，从而能够在定制加速器平台上准确、高效地部署大语言模型。据我们所知，这为现有大语言模型加速器框架提供了最先进的量化支持。
- 借助 FlexLLM，我们为 Llama-3.2 1B 构建了按阶段定制的混合加速器，并采用硬件感知的 W4A4KV8 SpinQuant [27] 方案，将 WikiText-2 困惑度从 13.30（原始 SpinQuant）降至 12.68。与 NVIDIA A100 GPU 上的 BF16 基线相比，我们在 AMD Alveo U280 上的 FPGA 实现实现了 1.29×端到端加速、1.64× 更高的解码吞吐量和 3.14× 更优的能效；在 Versal V80 上，预计分别可提升 4.71×、6.55×和 4.13×。
- 我们进一步在 FlexLLM 上实现了分层记忆 Transformer（HMT） [29] 插件，用于长上下文处理，将预填充延迟最多降低 23.23×，并将有效上下文窗口扩展超过64× ，同时资源开销低于 7.5%、延迟开销仅为 0.6%，展现了 FlexLLM 在长上下文加速方面的可扩展性。

II. 背景

A. 大语言模型推理阶段的特征分析

Transformer 架构 [30] 是大多数现代大语言模型的基础。每个 Transformer 层主要由多头自注意力（MHA）模块和前馈神经网络（FFN）组成，后接归一化和残差连接，如图 1(a) 所示。这种高度规整的结构使基于内核的硬件库成为加速器设计中自然且高效的选择。

推理过程中，仅解码器大语言模型通常分为两个阶段执行：预填充和解码。在预填充阶段，模型并行处理整个提示，并计算所有词元的激活值，呈现出较高的算术强度和充足的跨

表 I NVIDIA A100 GPU、AMD ALVEO U280 FPGA 和 VERSAL V80 FPGA 的关键硬件指标比较。

指标	U280	V80	A100 80GB PCIe
工艺节点	台积电 16纳米	台积电 7纳米	台积电 7纳米
峰值计算性能	8 FP32 万亿次浮点运算每秒	58 FP32 万亿次浮点运算每秒	312 FP32 万亿次浮点运算每秒
峰值 HBM 带宽	460 GB/s	820 GB/s	1,935 GB/s
HBM 容量	8 GB	32 GB	80 GB
峰值功耗	75 W	190 W	300 W

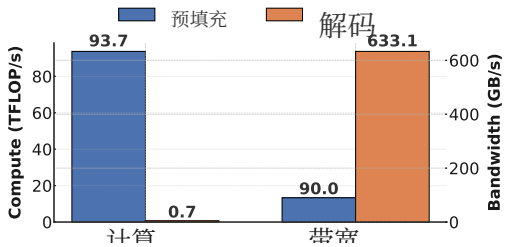


图2。NVIDIA A100 GPU 上 BF16 Llama-3.2 1B 模型在预填充和解码阶段的计算吞吐量与内存带宽利用率分析。

词元并行性，因此整体上受计算能力限制。在解码阶段，模型在复用键值（KV）缓存的同时自回归地生成词元；此时计算主要受顺序依赖和 KV 访问支配，瓶颈也从计算吞吐量转向内存带宽。

为量化这一阶段差异，我们在 NVIDIA A100 上运行 BF16 Llama-3.2 1B [31]，使用 vLLM [7]，分别测量预填充和解码过程中处理和生成 1K 个 token 时的计算与内存带宽利用率（图2）。得益于大规模并行计算和高度优化的内核，GPU 在预填充阶段实现了较高的利用率和强劲的性能。相比之下，在解码阶段，计算利用率急剧下降，内存带宽成为主要瓶颈，这证实了两个阶段对资源的需求存在根本差异。

表 I 将具有代表性的 FPGA 平台 [32], [33]与 A100 GPU [34]进行了比较。值得注意的是，GPU 的解码阶段通常无法充分利用其峰值带宽和计算性能，而现代FPGA 提供了充足的峰值计算性能和高带宽内存带宽；当量化减少内存流量时，这些资源足以满足解码需求。因此，借助有效的低比特量化，以及针对内存复用和流水线效率协同设计的架构，领域专用加速器可以具备很强的竞争力——尤其适用于以解码为主、序列较长的生成工作负载，在此类负载中，令牌延迟和能效至关重要 [9], [35], [36]。

B. 高效硬件大语言模型量化

量化对于在领域专用加速器上实现高效的大语言模型部署至关重要。一般的 N 位整数型量化可表示为

$$X_q = s \cdot \text{round}\left(\frac{X-b}{s}\right) + b,$$

其中 X 是浮点（FP）张量，X_q 是量化整数张量，s 和 b 是缩放因子和零点偏移。对称量化使用 $s = \frac{\max(|X|)}{2^{N-1}-1}$ 和 $b = 0$，而非对称量化使用 $s = \frac{\max(X)-\min(X)}{2^N-1}$ 和 $b = \min(X)$。当 s，b 在离线预先计算时，量化是静态 的；当它们在运行时测量时，则是动态的。

静态量化天然兼容预训练权重，并且由于无需运行时校准，也十分适用于激活值，因此已被近期的加速器广泛采用 [20]，[9]。然而，即使采用 SmoothQuant 等先进方案 [25]，其精度在 INT4 下通常仍会大幅下降。相比之下，细粒度动态量化会引入额外开销和逐令牌依赖屏障，但结合异常值处理技术（例如 SpinQuant 中引入的激活旋转和快速哈达玛变换（FHT））后，便成为实现高精度激进低比特部署的一条实用途径 [27]。

C. 相关加速器与框架

表 II 总结了具有代表性的面向 Transformer 的加速器与框架。

FlightLLM [20]采用一种**时序** 设计，在共享引擎上按顺序执行各层，以最大限度地提高复用率。虽然利用率可能较高，但有限的片上缓冲会导致频繁的片外访问，而单体式引擎限制了对异构内核/精度的支持以及快速模型定制，从而增加了延迟、能耗和重新设计的工作量。

相比之下，FINN [21], hls4ml [22],Allo [23]和 StreamTen- sor [24] 均采用完全的**空间数据流**范式，将模块映射到专用硬件上，并在片上进行流式数据传输。它们为定制化提供了有用的抽象或编译支持，但在固有依赖和不均衡的内核延迟下，流水线停顿仍然限制了解码性能，从而使吞吐量难以维持。

混合架构旨在通过灵活的调度和模块级定制，结合时序设计与空间设计的优势。例如，SSR [37]和 InTAR [38]实例化了一组固定的线性引擎和非线性引擎，从而支持模块间流式传输以及引擎内核切换。然而，现有混合架构在很大程度上仍在各个推理阶段采用统一架构；固定的引擎配置限制了针对不同大语言模型以及预填充和解码阶段显著不同瓶颈的深入优化。

要充分发挥混合大语言模型加速的潜力，就必须采用**面向阶段定制的**设计，使预填充和解码采用不同的模块配置与调度策略，以匹配各自不同的硬件特性（图 1(f)(g)）。下一节将介绍 FlexLLM 如何通过面向阶段的模块优化和快速的混合架构构建实现这一目标，同时集成广泛的量化支持，构建实用的高性能大语言模型加速器，并将从模型到芯片的周期缩短至不到两个月。

III. FlexLLM框架

A. 阶段定制模块模板

如 II-A 节所述，预填充和解码对资源的需求存在根本差异，因此单一的统一架构不可避免地会牺牲其中一个阶段的性能。FlexLLM 通过提供**阶段定制模块模板**来解决这一问题，从而开放合适的

表 II 相关加速器与框架比较。

加速器框架	贡献类型	架构类型	可扩展性	阶段定制内核优化	支持的精度	支持的量化类型
FlightLLM [20]	加速器	时序	不可扩展	✗	FP32，INT1∼8	静态
FINN [21], hls4ml [22]	编译器框架	空间	自动映射	✗	FP32，INT1∼8	静态
Allo [23]	领域特定语言	空间	需要手动设计	✗	FP32, INT4/8	静态
StreamTensor [24]	编译器框架	空间	自动优化	✗	FP32，INT8	静态
SSR [37], InTAR [38]	加速器	混合	需要手动设计	✗	FP32、INT8	静态
FlexLLM（我们的方法）	HLS 内核库	混合	模板化内核组合	✓	FP16/32，INT1∼8	静态/动态

并行性调节选项，便于快速构建具有均衡流水线的混合加速器。

预填充阶段模块： 预填充具有丰富的令牌间并行性。FlexLLM 将多个 token 的激活值打包，并对线性层和非线性层同时进行处理（图 3(a)）。我们将这种令牌间并行性定义为令牌并行（***TP***）。由于线性层占据了主要计算量，我们进一步引入权重并行（***WP***），以从片外存储器并行获取并传输多个权重通道。每个预填充线性模块都实现一个二维脉动阵列，包含$TP \times WP$ 个处理单元（PE），并针对支持的精度进行优化，使启动间隔（II）为一个周期。

给定长度为 l_p的提示，输入维度为 d_{in} 、输出维度为 d_{out} 的线性层的理论预填充延迟可表示为

$$T^{p}_{\text{linear}} = \frac{l_p \, d_{in} \, d_{out}}{TP \cdot WP}, \tag{1}$$

相应的片外带宽需求为

$$BW^{p}_{\text{linear}} = B_W \cdot WP \cdot F, \tag{2}$$

其中， B_W 为每个权重元素的字节数， F 为工作频率。这些参数使用户能够在满足内存带宽限制的同时，平衡层吞吐量。

解码阶段模块： 解码受自回归依赖关系的限制，因此并行性主要局限于单个 token 内。为此，FlexLLM 通过将每个输出隐藏向量划分为多个块并行计算，并在片上进行归约，从而利用 token 内并行性。我们将其称为块并行（***BP***）。每个解码线性层模块还支持权重并行（***WP***），其实现方式是使用BP 组一维脉动阵列，每组包含WP/BP 个处理单元（图3(b)）。对于解码长度 l_d，具有相同 d_{in} 和 d_{out}的线性层的理想延迟

is

$$T^{d}_{\text{linear}} = \frac{l_d \, d_{in} \, d_{out}}{WP}. \tag{3}$$

片外带宽遵循公式2。与预填充相比，解码处理单元不会在多个 token 间共享权重，因此相同的资源预算通常可以支持更高的权重并行，从而增加带宽需求，并促使我们谨慎地进行阶段定制调优。

量化模块： 量化模块使用与非线性模块相同的可配置并行性（预填充时采用 张量并行，解码时采用 块并行 ）。如图 3(c)所示，量化器使用缩放因子和零偏移量将浮点输入转换为低位整数；这些缩放因子和零偏移量可以预先加载（静态），也可以在线计算（动态）。内核计算完成后，反量化器使用相同的缩放因子和偏移量，以及缓存在片上的辅助数据（例如逐通道权重缩放因子和求和结果），重建浮点输出。总体而言，我们的框架支持

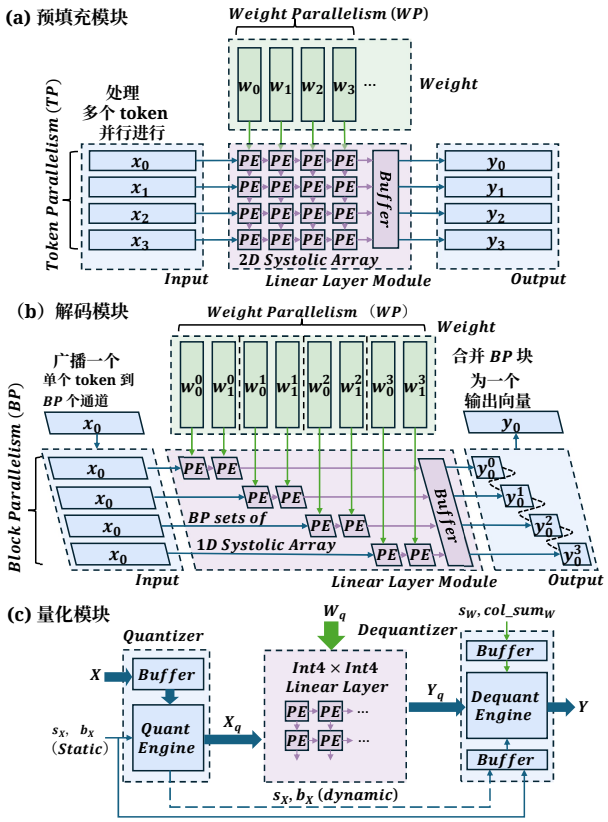


图 3. 阶段定制模块设计：(a) 预填充，(b) 解码，(c) 量化模块集成。

静态/动态和对称/非对称量化，并支持逐张量、逐词元和逐通道粒度，还包含旋转和快速哈达玛变换等离群值处理模块。

库组件：遵循这些原则，FlexLLM 基于 TAPA 提供了全面的模块库，包含超过 1 万行高度参数化的代码。该库包括核心大语言模型内核、量化/反量化模块，以及用于片上流处理、缓冲和内存访问管理的辅助组件。表 III 总结了关键模块及其可配置参数。这些模板支持快速、可组合的混合架构探索，同时便于非专业开发者使用，将模型到芯片的开发周期从数月缩短至数周。

B. 混合加速器构建与优化

基于 FlexLLM 和 TAPA高层次综合流程，用户可以高效探索大语言模型加速器的混合架构，并快速从设计探索过渡到板上实现。图4展示了一个简化示例，说明如何以混合方式组合多个模块。

表 III FlexLLM 提供的主要模块模板、可配置参数及模块接口概览。

	模块 模板	模块可配置参数	模块接口
内核库	线性层	数据类型，token_并行性（预填充），块_并行（解码）， 头_并行（多头注意力），权重_并行性，头_数量（多头注意力）， 最大输入_维度、最大输出_维度、最大序列_长度	in_流，w_流，out_流， 输入_维度、输出_维度、序列_长度
	非线性层 (旋转位置编码，Softmax，层归一化，...)	数据类型，token_并行性（预填充），块_并行（解码）， 头部_并行（多头注意力），头部_数量（多头注意力），最大_输入输出_维度，最大_序列_长度	输入_流，输出_流，io_维度， 序列_长度
量化库	静态/动态 量化层	输入_数据类型，输入_量化_位，token_并行性（预填充）， 块_并行（解码），头部_并行（多头注意力）， 输入_量化_类型（对称/非对称），输入_量化_粒度（逐张量/token）， 头部_数量（多头注意力），最大_IO_维度，最大_序列_长度	in_流，in_缩放因子_流， 输入_零点_流，量化_输入_流， 输入输出_维度，序列_长度
	静态/动态 反量化层	输入_量化_位，权重_量化_位，输出_数据类型，token_并行性（预填充）， 块_并行（解码），头部_并行（多头注意力）， 输入_量化_类型（对称/非对称），输入_量化_粒度（逐张量/token）， 权重_量化_类型（对称/非对称），权重_量化_粒度（逐张量/通道）， 头_数（多头注意力），最大_输入输出_维度，最大_序列_长度	量化_输出_流， 输入_缩放因子_流，输入_零点_流， 权重_缩放因子_流，权重_列_和_流， 输出_流、IO_维度、序列_长度

```
// Temporal-reuse: inst identical template sequentially
void Linear_Layer_KQ_reused(
    istream<vector<float, TP>>& in_stream,
    istream<vector<float, WP>>& w_stream,
    ostream<vector<float, TP>>& out_stream,
    int seq_len
){
    pref_Linear_Layer<float, TP, WP, HID_DIM, HID_DIM>(
        in_stream, w_stream, out_stream, seq_len
    );
    pref_Linear_Layer<float, TP, WP, HID_DIM, HID_DIM>(
        in_stream, w_stream, out_stream, seq_len
    );
}
void RoPE_layer_KQ_reused(...);
// Spatial-dataflow: connect modules in top via stream
void Top_Module(...){
    stream<vector<float, TP>> LL_in_stream;
    stream<vector<float, TP>> RoPE_in_stream;
    ...
    tapa::task()
    .invoke(Linear_Layer_KQ_reused, LL_in_stream, ...)
    .invoke(RoPE_layer_KQ_reused, RoPE_in_stream, ...)
    .invoke(...);
}
```

图4。展示使用 FlexLLM 构建混合架构的示例代码，结合了时间复用和空间数据流。

在时序复用部分，同一个模板化模块会在单个函数中实例化并调用多次，以顺序处理相似操作。在此示例中，线性层和旋转位置编码的预填充模块会同时用于键 和查询计算。此方法通过在相似任务间复用计算资源，最大限度地提高硬件利用率，同时尽可能减少冗余模块实例化。

在空间数据流部分，每个实例化的模块都会在顶层被显式调用，并通过片上FIFO连接，以实现并行执行和流式通信。这种基于模块的组合利用了模块间流水线化和片上数据流，从而实现高吞吐量以及计算阶段之间的重叠。

时序与空间风格相结合，能够在混合加速器设计中有效平衡性能与资源效率。我们通过第四节中的一个案例研究，详细阐述分阶段定制混合设计的方法。使用 FlexLLM 组合出加速器后，用户可以利用其与 AutoBridge [39]的无缝兼容性来优化布局布线（P&R），并行探索设计方案，从而实现更高的频率和性能。这一端到端工具链能够加快迭代，并支持高效的板上部署。

四、案例研究1：LLAMA 3.2-1B ＋ SPINQUANT

本节将展示如何利用 FlexLLM 为 Llama-3.2 1B 模型构建采用阶段定制架构的高性能加速器。具体而言，我们首先改进 SpinQuant框架，以获得更适合硬件的表示，同时尽可能降低精度下降。随后，我们利用 FlexLLM 中可组合的模块，为预填充和解码构建定制架构。表IV总结了主要模块组件及其对应的代码行数和预估开发工作量。

A. 硬件高效模型量化

SpinQuant 通过利用学习得到的旋转来缓解激活值离群点，从而提高低比特精度 [27]。虽然许多旋转可以折叠到权重中，或通过 FHT 高效实现，但剩余的边界旋转仍需要与完整线性层相当的 FP 计算，这在 FPGA 上代价高昂。此外，基线 SpinQuant 设置将 MHA查询路径和最终的词汇投影（lm_head）保留为 FP，导致大量硬件效率未得到发挥。

为获得一种精度损失最小的实用模型，我们进行了以下面向硬件的改进：

移除边界旋转。我们在微调过程中将每个解码器层中的首个和末个旋转吸收到相邻权重中，从而将其移除。这样既保留了异常值抑制效果，又彻底消除了浮点边界操作。

以更高精度量化注意力。由于多头注意力对精度更敏感，我们将注意力量化为 INT8，同时将其他线性层保持为 INT4。为简化硬件设计，多头注意力采用静态对称逐张量量化，其余线性层则采用动态非对称逐词元量化，以保留精度。

整数词汇投影。我们进一步将 lm_head 量化为 INT4，与其他线性层保持一致，从而降低资源成本并提升解码吞吐量。

表 V 报告了在 WikiText-2 上进行的消融实验。以 BF16（困惑度 = 8.94）为起点，原始 SpinQuant 配置（INT4 线性层搭配 BF16-INT4 注意力）得到的困惑度为 = 13.30。将注意力精度提高到 INT8 可恢复大部分损失的精度；使用静态 INT8 会使困惑度略微升高，但能简化硬件。

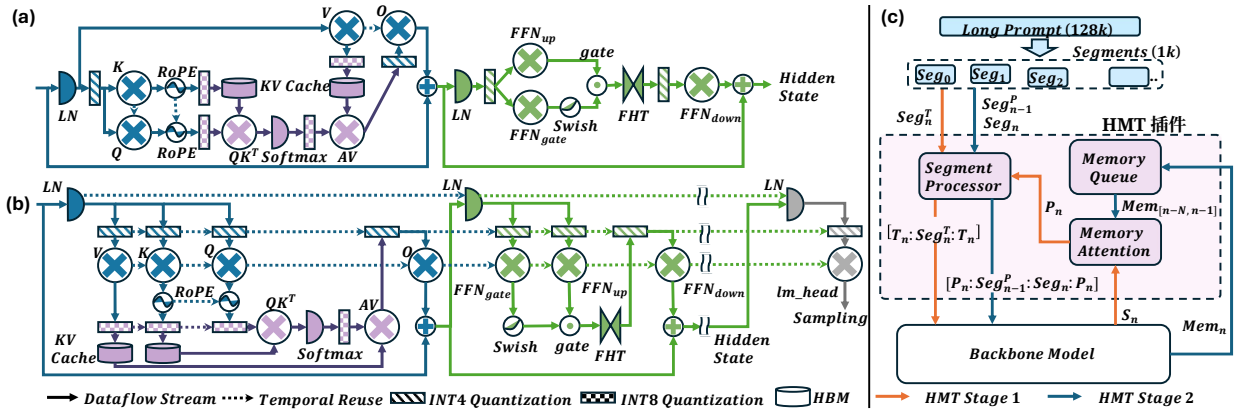


图 5.（a）预填充与（b）解码的混合架构设计，以及（c）HMT 插件的集成示意图。

最后，量化 lm_head 仅会略微增加困惑度，同时使所有线性层构成的流水线实现全整数运算。

B. 架构设计

我们采用 FlexLLM 的阶段定制模块，分别为预填充和解码实现独立架构，以适应各自不同的瓶颈，从而提升性能和效率。

预填充阶段：预填充受益于令牌间并行和流水化流处理。与为每个内核实例化专用模块的纯空间设计不同，我们的预填充架构将流式数据流与选择性时间复用相结合。如图5(a)所示，Q_proj 和 K_proj 共用相同的线性层和旋转位置编码模块，V_proj 和 O_proj 也采用同样的方式。我们并行计算键和值，并将其存储在片外高带宽内存中，之后其余内核以完全流水化的数据流方式执行。流式连接最大限度地提升了跨 token 的重叠，而时间复用则缓解了数据依赖导致的空闲，并有助于在多头注意力中形成均衡流水线。

根据公式 1，预填充延迟的下界为

$$T_p = \frac{Nl_p}{TP}\left(\frac{d_h d_{kv}}{WP_{kqvo}} + \max\left(\frac{d_h^2}{WP_{kqvo}}, \frac{d_h l_p}{WP_{mha}}, \frac{d_h d_{ffn}}{WP_{ffn}}\right)\right) \quad (4)$$

其中，N 为解码器层数，d_h、d_{kv}和 d_{ffn}分别为隐藏维度、KV 维度和前馈神经网络中间维度。权重并行$_{kqvo}$、权重并行$_{mha}$和权重并行$_{ffn}$ 表示相应模块的权重并行设置。峰值带宽需求为

$$BW_p = F\left(B_W^{int4}(2WP_{kqvo} + 3WP_{ffn}) + B_W^{int8} \cdot 2WP_{mha}\right) \quad (5)$$

其中 B_W 是每个权重元素的字节数，F 表示频率。

在预填充阶段，非线性开销主要随张量并行缩放，而线性模块则随张量并行 和权重并行共同缩放。我们通过整数线性规划（整数线性规划），在硬件约束（资源和内存带宽）下调整这些参数，以最小化 T_p 并获得最优、流水线化良好的设计。

解码阶段：自回归依赖使解码过程无法进行令牌间并行，从而使跨词元的空间重叠不可行。为最大化运行时硬件利用率，我们对所有INT4量化、线性层和非线性层进行模块的时序复用，同时保持多头注意力内部以及复用的量化、

表IV 各设计组件的模块使用情况、代码规模和开发工作量汇总。

设计	主要模块用途	LoC	从开发到流片的工作量
Llama-3.2 1B	线性层、旋转位置编码、多头注意力、键值缓存，层归一化、残差连接、Softmax，Swish、门控、采样	~ 600	~4 周 × 人
SpinQuant	动态非对称 INT4 / 静态对称 INT8 量化 / 反量化，快速哈达玛变换	~ 200	~1 周 × 人
HMT 插件	线性层、多头注意力、键值缓存	~ 200	~1 周 × 人

表 V LLAMA-3.2-1B 的消融研究（困惑度越低越好）。

量化配置	W	A	Attn	词汇量 Wiki-2 困惑度
不量化	BF16	BF16	BF16	BF16	8.94
Q0（SpinQuant）	INT4	INT4	BF16-INT4	BF16	13.30
Q1	INT4	INT4	动态 INT8	BF16	12.07
Q2	INT4	INT4	静态 INT8	BF16	12.28
第三季度（最终）	INT4	INT4	静态 INT8	INT4	12.68

权重/激活值/注意力/词汇表：权重/激活值/注意力/语言模型输出头精度。

线性模块、非线性模块之间的数据流连通性，如图5(b)所示。该设计实现了令牌内并行与头间重叠，从而有效利用可用的并发性。

解码阶段的理论延迟上界和内存带宽如公式6和公式7所示，其中权重并行$_{int4}$表示投影、前馈神经网络和语言模型输出头层所共用的权重并行 。我们同样通过整数线性规划，在硬件约束下调整块并行数和权重并行数，以最小化目标设备上的 Td 。

$$T_d = l_d\left(\frac{N(2d_h d_{kv} + d_h^2 + 3d_h d_{ffn}) + dd_{lm_head}}{WP_{int4}} + \max\left(\frac{Nd_h^2}{WP_{int4}}, \frac{Nd_h(l_p + 0.5l_d)}{WP_{mha}}\right)\right) \quad (6)$$

$$BW_d = F(B_W^{int4} WP_{int4} + 2B_W^{int8} WP_{mha}) \quad (7)$$

实际上，解码通常采用更大的权重并行设置，从而产生扇出较高、规模过大的线性模块。我们通过将其划分为多个相同的子模块来缓解这一问题，从而提高布局规划的灵活性，并提升布局布线期间可实现的频率。

五、案例研究2：HMT 集成

虽然量化降低了计算量和内存流量，但长上下文提示仍会在计算和 KV 访问方面产生二次方注意力开销，导致延迟和能耗较高。为此，我们将分层记忆

表 VI Llama 3.2-1B 模型、加速器和 HMT 插件的参数配置，以及在 U280 FPGA 上的板载频率、硬件资源利用率和延迟，以及在 V80 FPGA 上的综合结果（带 * 标记的为估算值）。

模型/设备设计		参数	频率	配置逻辑块	数字信号处理器	查找表	FF	块 RAM / UltraRAM	延迟
Llama 3.2-1B	–	L=16, d=2048, d_{kv}=512, d_{ffn}=8192, d_{lm_head}=128256	–	–	–	–	–	– / –	–
U280	预填充架构	张量并行=8, 权重并行$_{kqvo}$=24, 权重并行$_{mha}$=16, 权重并行$_{ffn}$=96	304 兆赫	66%	29%	39%	24%	35% / 11%	1.65 秒/1k 词元
	解码架构	块并行=16，权重并行$_{mlp}$=1024，权重并行$_{mha}$=256	292 兆赫	76%	18%	44%	28%	41% / 15%	6.94 秒/1k 词元
	HMT 插件	N=64,块并行=4,权重并行$_{mem\ 注意力}$=4	290 兆赫	7.5%	1.5%	5.3%	1.9%	4.3% / 3.8%	8.44 毫秒/片段
V80	预填充架构	张量并行=16，权重并行$_{kqvo}$=32，权重并行$_{mha}$=32，权重并行$_{ffn}$=128	300兆赫*	58%	26%	37%	20%	22% / 9%	0.61秒/1k 词元*
	解码架构	块并行=64，权重并行$_{intd}$=4096，权重并行$_{mha}$=1024	300兆赫*	75%	25%	42%	22%	36% / 20%	1.68秒/1k 词元*
	HMT 插件	N=64，块并行=4，权重并行$_{mem\ 注意力}$=8	300 兆赫*	3.8%	0.7%	3.3%	0.9%	2.4% / 1.9%	6.50 毫秒/片段*

Transformer（HMT）插件集成到我们的加速器系统中。HMT 会递归地将上下文片段压缩为紧凑的记忆嵌入，并通过记忆注意力通路检索相关信息，从而将长上下文处理的复杂度从序列长度的二次方降低到线性。此外，这项研究也展示了 FlexLLM 在实现新型大语言模型机制方面的灵活性和效率。

FlexLLM 的可组合设计使此集成十分轻量：HMT 插件主要通过复用现有的线性层和注意力模块来实现其片段编码器、记忆生成和历史检索流程，所需额外硬件和设计工作极少。

图 5(c) 展示了端到端工作流程。长提示被拆分为多个短片段。对于每个片段，片段处理器首先将当前片段的前半部分（Seg_n^T）与主题令牌嵌入 T_n 拼接，形成摘要提示，并将其发送至大语言模型加速器，以生成主题摘要向量 S_n。随后，HMT 插件对 S_n 与记忆队列中最近的 N 个记忆嵌入 $\{Mem_{n-N}, \ldots, Mem_{n-1}\}$ 执行交叉注意力，生成检索提示嵌入 P_n。

接下来，HMT 插件通过拼接完整片段、 P_n 和来自前一片段的短期上下文片段（Seg_{n-1}^P），构建增强提示。大语言模型加速器处理此增强输入，以生成新的长期记忆嵌入 Mem_n，该嵌入会追加到记忆队列中，以供未来检索。

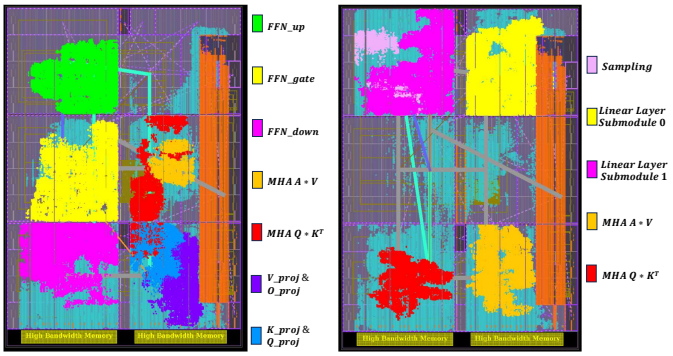
VI-B2 节中的实验结果表明，基于 FlexLLM 构建的 HMT 插件能够以极低的硬件开销和几乎可忽略的额外延迟，在骨干模型实现之上支持可扩展的上下文窗口，从而显著拓展大语言模型加速器在长上下文工作负载中的适用范围。

VI. 评估

A. 实验设置

硬件平台：我们选用采用台积电 16 纳米工艺的 AMD Alveo U280 [32]和采用 7 纳米工艺的 Versal V80 [33]作为目标加速器平台，并选用采用 7 纳米工艺的 NVIDIA A100 作为 GPU 代表。表 I 汇总了关键硬件规格。

加速器实现：使用 FlexLLM，我们为经过 SpinQuant 量化的 Llama-3.2 1B 模型，分别在 U280 和 V80 上实现了针对各阶段定制的加速器。图 6 展示了 U280 布局。我们还集成了 HMT 插件，以支持长上下文处理。表 VI 汇总了两个平台上的模型参数、架构配置、实现频率和资源利用率。在 U280 上，该设计可在板上运行，并支持快速重新配置（~0.3 秒），其性能



(a) (b)

图 6. U280 上量化 Llama-3.2 1B 的 (a) 预填充架构和 (b) 解码架构的实现布局。

直接在硬件上测得。对于 V80 实现，我们使用 Vitis/Vivado 2024.2 和 AVED 工具链进行 RTL 综合和布局布线，并根据 U280 实现的性能缩放，估算板上性能。我们使用采用 W4A8KV8 SmoothQuant 的 Allo [23] 作为最先进的加速器基线。

GPU 基线：我们使用 HuggingFace BF16 Llama-3.2 1B 模型作为 GPU 基线。由于 vLLM [7]不支持 SpinQuant，因此我们改用 GPTQ [26]和 Marlin [40]将模型量化为 INT4，并在 A100 GPU 上使用 vLLM 对其进行评估。这一设置为我们的 FPGA 实现提供了公平的 GPU 参照。

评估指标：我们使用三个关键指标评估基于 FPGA 的加速器和 GPU 基线：(1) 端到端延迟，(2) 解码吞吐量，以及 (3) 能效。端到端延迟衡量总推理时间（预填充 +解码），而解码吞吐量表示解码过程中每秒生成的词元数。我们记录 A100 和 U280 的板载采样平均功耗，并采用 V80 的综合报告数据，计算每焦耳生成令牌数（token/J），以评估能效。

B. 实验结果

1) 案例研究 *1*：典型序列长度：我们在一系列预填充和解码序列长度下评估了所提出的加速器和 GPU 基线（图7）。正如预期，在预填充较长、解码长度较短的场景（例如， [1024, 256]）中，GPU 展现出明显优势，其强大的并行计算能力可实现超快的提示词处理。然而，随着解码长度增加，GPU 的性能瓶颈转向内存带宽；与此同时，我们的 FPGA 加速器通过针对不同阶段定制的设计和灵活的并行性，保持了更高的效率。

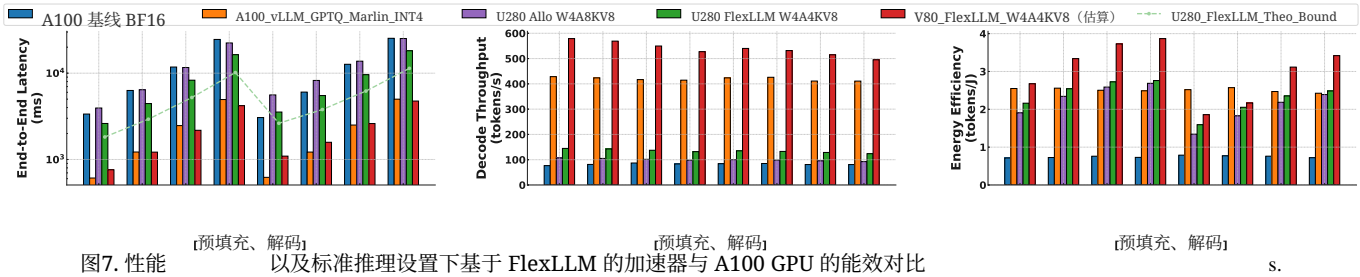


图7. 性能 以及标准推理设置下基于 FlexLLM 的加速器与 A100 GPU 的能效对比 s.

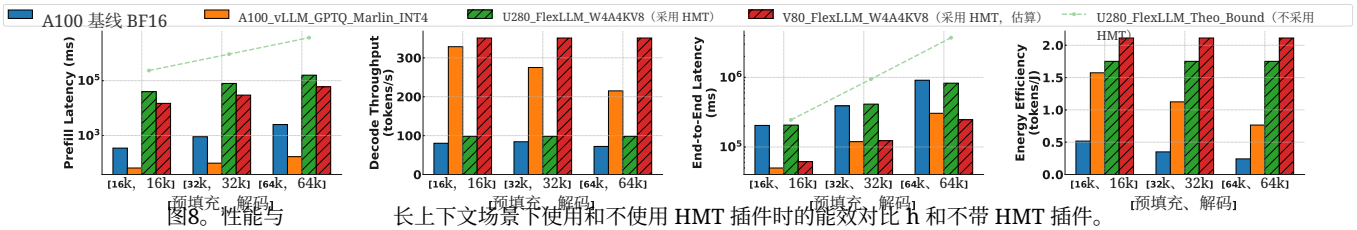


图8。性能与 长上下文场景下使用和不使用 HMT 插件时的能效对比 h 和不带 HMT 插件。

平均而言，基于 FlexLLM 的 U280 加速器相较于 A100 BF16 基线，实现了**1.29**× 端到端加速比和**1.64**× 更高的解码吞吐量；相较于 Allo，也分别实现了**1.46**×和**1.35**×的提升。V80 设计的端到端加速比最高可达**4.71**× ，解码吞吐量则最高提升**6.55**×。对于更长的解码工作负载（例如，[512, 2048] 和 [1024, 2048]），V80 的性能甚至超过了在 vLLM 下运行 INT4 GPTQ-Marlin 的 A100。尽管 A100 的峰值带宽更高，但解码性能受有效带宽利用率限制：A100+vLLM 的平均带宽利用率仅为 13.06%，而我们的 V80 设计平均达到**38.61%** ，峰值高达**74.93%** 。这些结果凸显了 FlexLLM 支持的阶段定制架构在内存受限解码中的优势。

在能效方面，基于 FlexLLM 的加速器优于 A100 BF16 基线，平均每焦耳生成令牌数分别提升 **3.14**× （U280）和 **4.13**×（V80）。在较长的解码长度（1024/2048）下，即使是 16 纳米 U280，其性能也超过了使用 vLLM 运行 INT4 GPTQ-Marlin 的 7 纳米 A100，展现出出色的利用率和能效。与 Allo 相比，我们的加速器进一步实现了 **1.10**× 更高的能效。这些结果证实，FlexLLM 能够实现领域专用加速器，在不同工艺节点下，其性能和能效均优于通用 GPU。

2）案例研究 *2*：使用 *HMT* 插件的长上下文：在长上下文推理中，由于 FPGA 高带宽内存容量有限，不使用 HMT 的加速器容易耗尽内存。即使片外存储器足以支持 KV 缓存，在 64K 个 token 下，U280 上的理论预填充延迟也可能超过一小时，使得部署变得不切实际。这促使人们针对大语言模型加速器设计序列压缩方案，例如 HMT。

如表 VI 所示，U280 上的 HMT 插件消耗的资源不到总资源的 **7.5%** ，每个片段仅增加 8.44ms 的延迟（占端到端延迟的 **<0.6%** ）。FlexLLM 的模板化模块性进一步支持调整插件的资源—延迟权衡，以适配不同的大语言模型加速器。

图 8 展示了采用 HMT 时的性能和能效。由于峰值吞吐量更高，不采用 HMT 的 A100 在预填充阶段仍快于基于 FPGA 的加速器——但与不采用 HMT 时的理论延迟上界相比，HMT 可将预填充延迟最多降低**23.23**×，使长提示词的首令牌生成时间进入实用范围。在端到端性能方面，得益于更高的解码吞吐量，以及 HMT 的压缩将解码随序列长度的扩展从二次转变为线性，采用 HMT 增强的加速器重新超越了 A100。与 A100 BF16 基线相比，U280 的解码吞吐量最高提升 **1.36**× ，端到端延迟最高降低**1.10**×；V80 则分别最高提升 **4.86**×和**3.70**×。

随着上下文增长，HMT 也能保持能效稳定：长提示词下，A100 的 tokens/J 急剧下降，而加入 HMT 的 U280 和 V80 则能维持较高能效，其 tokens/J 最高分别比 A100 BF16 基线高 **5.21**× 和 **6.27**× ，比 GPTQ- Marlin+vLLM 高 **1.65**× 和 **1.99**× 。这些结果不仅凸显了 HMT 插件对于实现可扩展长上下文推理的关键作用，也体现了我们的 FlexLLM 框架在大语言模型加速器设计方面的灵活性和可扩展性。

VII. 结论与展望

我们提出了**FlexLLM**，这是一种可组合的高层次综合框架，可用于快速构建和优化领域专用大语言模型加速器。FlexLLM支持针对预填充和解码阶段定制的混合架构，并提供全面的量化技术栈，以实现准确的低比特推理，从而连接模型层面的创新与硬件层面的实现。在Llama-3.2 1B模型上的实验结果表明，FlexLLM能够构建高性能、低能耗的大语言模型推理设计，其性能可媲美甚至超越运行vLLM的NVIDIA A100 GPU。这些结果彰显了FlexLLM能够以极少的人工投入，将算法创新转化为高效的硬件实现。展望未来，FlexLLM为在未来平台上实现领域专用大语言模型加速，以及将可组合的高层次综合方法扩展至下一代Transformer模型奠定了基础。

参考文献

[1] J. Devlin、M.-W. Chang、K. Lee和K. Toutanova，“BERT：面向语言理解的深度双向变换器预训练”，载于《2019年计算语言学协会北美分会会议论文集：人类语言技术》，第1卷（长短论文），2019年，第4171–4186页。[2] C. Raffel、N. Shazeer、A. Roberts、K. Lee、S. Narang、M. Matena、Y. Zhou、W. Li和P. J. Liu，“探索采用统一文本到文本变换器进行迁移学习的极限”，《机器学习研究期刊》，第21卷，第140期，第1–67页，2020年。[3] M. Chen、J. Tworek、H. Jun、Q. Yuan、H. P. D. O. Pinto、J. Kaplan、H. Edwards、Y. Burda、N. Joseph、G. Brockman等，“评估基于代码训练的大语言模型”，arXiv预印本 arXiv:2107 03374，2021年。[4] Y. Li、D. Choi、J. Chung、N. Kushman、J. Schrittwieser、R. Leblond、T. Eccles、J. Keeling、F. Gimeno、A. Dal Lago等，“利用AlphaCode实现竞赛级代码生成”，《科学》，第378卷，第6624期，第1092–1097页，2022年。[5] J. Achiam、S. Adler、S. Agarwal、L. Ahmad、I. Akkaya、F. L. Aleman、D. Almeida、J. Altenschmidt、S. Altman、S. Anadkat等，“GPT-4技术报告”，arXiv预印本 arXiv:2303.08774，2023年。[6] J.-B. Alayrac、J. Donahue、P. Luc、A. Miech、I. Barr、Y. Hasson、K. Lenc、A. Mensch、K. Millican、M. Reynolds等，“Flamingo：用于少样本学习的视觉语言模型”，《神经信息处理系统进展》，第35卷，第23 716–23 736页，2022年。[7] W. Kwon、Z. Li、S. Zhuang、Y. Sheng、L. Zheng、C. H. Yu、J. Gonzalez、H. Zhang和I. Stoica，“利用PagedAttention实现大语言模型服务的高效内存管理”，第611–626页，2023年。[8] S. Hong、S. Moon、J. Kim、S. Lee、M. Kim、D. Lee和J.-Y. Kim，“DFX：用于加速基于变换器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）》。IEEE，2022年，第616–630页。[9] H. Chen、J. Zhang、Y. Du、S. Xiang、Z. Yue、N. Zhang、Y. Cai和Z. Zhang，“理解基于FPGA的空间加速在大语言模型推理中的潜力”，《ACM可重构技术与系统汇刊》，第18卷，29，2024年1，1第。页–。[10] Y.-H. Chen、T.-J. Yang、J. Emer和V. Sze，“Eyeriss v2：面向移动设备上新兴深度神经网络的灵活加速器”，《IEEE新兴与重点电路和系统期刊》，第9卷，第2期，第292–308页，2019年。[11] M. Huang、A. Shen、K. Li、H. Peng、B. Li、Y. Su和H. Yu，“EdgeLLM：面向大语言模型的高效CPU-FPGA异构边缘加速器”，《IEEE电路与系统汇刊I：常规论文》，2025年。[12] C. Zhang、Z. Fang、P. Zhou、P. Pan和J. Cong，“Caffeine：迈向深度卷积神经网络的统一表示与加速”，载于《第35届计算机辅助设计国际会议论文集》，2016年，第1–8页。[13] S. Basalama、A. Sohrabizadeh、J. Wang、L. Guo和J. Cong，“FlexCNN：用于在FPGA上组合CNN加速器的端到端框架”，《ACM可重构技术与系统汇刊》，第16卷，第2期，第1–32页，2023年。[14] X. Li、L. Ding、L. Wang和F. Cao，“FPGA加速深度残差学习以实现图像识别”，载于《2017年IEEE第2届信息技术、网络、电子与自动化控制会议（ITNEC）》。IEEE，2017年，第837–840页。[15] Y. Ma、M. Kim、Y. Cao、S. Vrudhula和J.-s. Seo，“用于深度残差网络的端到端可扩展FPGA加速器”，载于《2017年IEEE国际电路与系统研讨会（ISCAS）》。IEEE，2017年，第1–4页。[16] Y. Guan、H. Liang、N. Xu、W. Wang、S. Shi、X. Chen、G. Sun、W. Zhang和J. Cong，“FP-DNN：一种利用寄存器传输级-高层次综合混合模板将深度神经网络映射到FPGA的自动化框架”，载于《2017年IEEE第25届现场可编程定制计算机年度国际研讨会（FCCM）》。IEEE，2017年，第152–159页。[17] T. Brown、B. Mann、N. Ryder、M. Subbiah、J. D. Kaplan、P. Dhariwal、A. Neelakantan、P. Shyam、G. Sastry、A. Askell等，“大语言模型是少样本学习器”，第33卷，2020年，第1877–1901页。[18] A. Dubey、A. Jauhri、A. Pandey、A. Kadian、A. Al-Dahle、A. Letman、A. Mathur、A. Schelten、A. Yang、A. Fan等，“Llama 3模型家族”，arXiv电子预印本，第arXiv–2407页，2024年。

[19] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan 等，“DeepSeek-V3技术报告”，arXiv预印本 arXiv:2412.19437，2024年。[20] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang 等，“FlightLLM：基于FPGA完整映射流程的高效大语言模型推理”，载于《2024年ACM/SIGDA国际现场可编程门阵列研讨会论文集》，2024年，第223–234页。[21] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O’brien, Y. Umuroglu, M. Leeser 和 K. Vissers，“FINN-R：用于快速探索量化神经网络的端到端深度学习框架”，《ACM可重构技术与系统汇刊》（TRETS），第11卷，第3期，第1–23页，2018年。[22] FastML团队，“fastmachinelearning/hls4ml”，2025年。 [在线]。网址：https://github.com/fastmachinelearning/hls4ml [23] H. Chen, N. Zhang, S. Xiang, Z. Zeng, M. Dai 和 Z. Zhang，“Allo：一种用于可组合加速器设计的编程模型”，《ACM编程语言会议论文集》，第8卷，620，2024年，593期，PLDI，第–页。[24] H. Ye 和 D. Chen，“StreamTensor：让张量在大语言模型数据流加速器中流动”，载于《第58届IEEE/ACM国际微体系结构研讨会®论文集》，2025年，第201–216页。[25] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth 和 S. Han，“SmoothQuant：面向大语言模型的准确、高效训练后量化”，载于《国际机器学习会议》。PMLR，2023年，第38 087–38 099页。[26] E. Frantar, S. Ashkboos, T. Hoefler 和 D. Alistarh，“GPTQ：面向生成式预训练变换器的准确训练后量化”，arXiv预印本 arXiv:2210.17323，2022年。[27] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian 和 T. Blankevoort，“SpinQuant：通过学习得到的旋转实现大语言模型量化”，arXiv预印本 arXiv:2405.16406，2024年。[28] L. Guo, Y. Chi, J. Lau, L. Song, X. Tian, M. Khatti, W. Qiao, J. Wang, E. Ustun, Z. Fang 等，“TAPA：一种面向现代FPGA的可扩展任务并行数据流编程框架，协同优化高层次综合与物理设计”，《ACM可重构技术与系统汇刊》，第16卷，第4期，第1–31页，2023年。[29] Z. He, Y. Cao, Z. Qin, N. Prakriya, Y. Sun 和 J. Cong，“HMT：用于高效长上下文语言处理的分层记忆 Transformer”，arXiv预印本 arXiv:2405.06067，2024年。[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser 和 I. Polosukhin，“《Attention Is All You Need》”，神经信息处理系统进展，第30卷，2017年。[31] Meta。（2025年）访问 meta-llama/Llama-3.2-1B。 [在线]。网址：https://huggingface.co/meta-llama/Llama-3.2-1B/ [32] AMD。（2023年）Alveo U280数据中心加速卡数据手册（ds 963 [ ] 963 280 ）。[33] 2024 80 1013 在线。网址：https://docs.amd.com/r/en-US/ds -u ——。（）Alveo v数据中心加速卡数据手册（ds）。[在线]。网址：https://docs.amd.com/r/en-US/ds1013-v80/Adaptive- SoC-Resources[34] 英伟达。（2022年）NVIDIA A100 Tensor Core GPU数据手册（80 GB，PCIe）。 [在线]。网址：https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf [35] S. Joel, J. Wu 和 F. Fard，“面向低资源和领域专用编程语言的基于大语言模型的代码生成综述”，《ACM软件工程与方法论汇刊》，2024年。[36] C. G´omez-Rodr´ıguez 和 P. Williams，“《模型联盟》：对大语言模型创意写作能力的全面评估”，arXiv预印本 arXiv:2310.08433，2023年。[37] J. Zhuang, Z. Yang, S. Ji, H. Huang, A. K. Jones, J. Hu, Y. Shi 和 P. Zhou，“SSR：用于变换器加速中延迟与吞吐量权衡的空间序列混合架构”，载于《2024年ACM/SIGDA国际现场可编程门阵列研讨会论文集》，2024年，第55–66页。[38] Z. He, A. Truong, Y. Cao 和 J. Cong，“InTAR：面向深度神经网络中高数据量变化的任务间自动可重配置加速器设计”，载于《2025年IEEE第33届年度国际现场可编程定制计算机研讨会》（FCCM）。IEEE，2025年，第123–132页。[39] L. Guo, Y. Chi, J. Wang, J. Lau, W. Qiao, E. Ustun, Z. Zhang 和 J. Cong，“AutoBridge：结合粗粒度布局规划与流水线化，实现多芯粒FPGA上的高频高层次综合设计”，载于《2021年ACM/SIGDA

国际现场可编程门阵列研讨会，2021年，第 81–92 页。[40] E. Frantar、R. L. Castro、J. Chen、T. Hoefler 和 D. Alistarh，“Marlin：大语言模型的混合精度自回归并行推理”，载于第 *30* 届 *ACM SIGPLAN* 原理与并行编程年度研讨会论文集，2025年，第 239–251 页。