arXiv:2410.04466v3 [cs.AR] 23 Jan 2025

Large Language Model Inference Acceleration: A Comprehensive Hardware Perspective

JINHAO LI, Shanghai Jiao Tong University, China
JIAMING XU, Shanghai Jiao Tong University & Infinigence-AI, China
SHAN HUANG, Shanghai Jiao Tong University, China
YONGHUA CHEN and WEN LI, Infinigence-AI, China
JUN LIU, YAOXIU LIAN, JIAYI PAN, LI DING, and HAO ZHOU, Shanghai Jiao Tong University, China
YU WANG, Tsinghua University, China
GUOHAO DAI*, Shanghai Jiao Tong University & Infinigence-AI, China

Large Language Models (LLMs) have demonstrated remarkable capabilities across various fields, from natural language understanding to text generation. Compared to non-generative LLMs like BERT and DeBERTa, generative LLMs like GPT series and Llama series are currently the main focus due to their superior algorithmic performance. The advancements in generative LLMs are closely intertwined with the development of hardware capabilities. Various hardware platforms exhibit distinct hardware characteristics, which can help improve LLM inference performance. Therefore, this paper comprehensively surveys efficient generative LLM inference on different hardware platforms. First, we provide an overview of the algorithm architecture of mainstream generative LLMs and delve into the inference process. Then, we summarize different optimization methods for different platforms such as CPU, GPU, FPGA, ASIC, and PIM/NDP, and provide inference results for generative LLMs. Furthermore, we perform a qualitative and quantitative comparison of inference performance with batch sizes 1 and 8 on different hardware platforms by considering hardware power consumption, absolute inference speed (tokens/s), and energy efficiency (tokens/J). We compare the performance of the same optimization methods across different hardware platforms, the performance across different hardware platforms, and the performance of different methods on the same hardware platform. This provides a systematic and comprehensive summary of existing inference acceleration work by integrating software optimization methods and hardware platforms. We point out that the development of edge intelligence has gained significant momentum, driven by the increasing capability of LLMs and the increasing demands of edge applications. And three trends (multimodality, inference-time compute, and higher inference energy efficiency) are promising to redefine the capabilities of edge artificial intelligence systems. Our project is available at https://dai.sjtu.edu.cn/project.html.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

*Corresponding author.

Authors' Contact Information: Jinhao Li, kimholee@sjtu.edu.cn, Shanghai Jiao Tong University, Shanghai, China; Jiaming Xu, Shanghai Jiao Tong University & Infinigence-AI, Shanghai, China; Shan Huang, Shanghai Jiao Tong University, Shanghai, China; Yonghua Chen; Wen Li, Infinigence-AI, Shanghai, China; Jun Liu; Yaoxiu Lian; Jiayi Pan; Li Ding; Hao Zhou, Shanghai Jiao Tong University, Shanghai, China; Yu Wang, Tsinghua University, Beijing, China; Guohao Dai, daiguohao@sjtu.edu.cn, Shanghai Jiao Tong University & Infinigence-AI, Shanghai, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

Additional Key Words and Phrases: Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, Yu Wang, and Guohao Dai. 2018. Large Language Model Inference Acceleration: A Comprehensive Hardware Perspective. *J. ACM* 37, 4, Article 111 (August 2018), 51 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Large Language Models (LLMs) have become cornerstones of modern artificial intelligence, demonstrating remarkable capabilities across a spectrum of fields, from natural language understanding to text generation [1–5]. LLMs can be categorized into two primary types: generative LLMs and non-generative LLMs. Non-generative LLMs, such as BERT [6], RoBERTa [7], ELECTRA [8], and DeBERTa [9], are designed to classify and make predictions based on input text. These models typically range in size from millions of parameters, allowing them to excel in tasks that require discernment and nuanced understanding. BERT, introduced in 2018, only has 340 million parameters. RoBERTa, introduced in 2019, slightly increases to 355 million parameters. And DeBERTa, released in 2021, increases to 1.5 billion parameters. Generative LLMs, like GPT series [10–13], T5 [14], OPT [15], BLOOM [16], and Llama series [17–19], have taken language generation to new heights. The model size increase of generative LLMs [10–57] are particularly notable in the past 6 years, as shown in Figure 1. In 2018, GPT1 has only 110 million parameters, which grows to 1.5 billion in GPT2 in 2019. GPT3, launched in 2020, grows to 175B parameters dramatically, and GPT3.5 maintains the same size. After 2022, the model size maintains to several hundreds and thousands of billions like GPT4, Llama3, and Grok1 [58]. The evolution of LLMs is characterized by an exponential growth in model parameters, which enhances their performance and versatility. Compared to non-generative LLMs, generative LLMs are currently the primary focus of research and development in the field of LLMs for their superior algorithmic performance. In recent years, after reaching the trillion-parameter scale in 2022, the parameter size of generative LLMs has stopped growing at an exponential rate. Two main reasons can explain this phenomenon: (1) As the amount of computation increases, the demand for computing power also rises significantly. The slow growth of hardware capabilities, particularly the slowing down of Moore's Law [59], limits the improvement of single-chip computing power. (2) Researchers have found that model performance is not solely dependent on the number of parameters, but also on the quantity and quality of training data [60]. By providing more qualified training tokens, the algorithmic performance can be further improved [18, 19]. At the same time, the model size of generative LLMs have shifted from "increasing" to "remaining stable" or even "shrinking". More models with fewer parameters are released, which are more suitable for deployment on edge devices.

The advancements in generative LLMs are closely intertwined with the development of hardware capabilities. Due to the continuation of Moore's Law, from 2018 to 2022, GPU manufacturing processes have progressed from 12nm to 3nm, and the floating-point performance of single GPU die has increased from 130 TFLOPS to 989 TFLOPS. During model training, GPUs are used predominantly due to the user-friendliness of the CUDA programming stack [61] and the high scalability of GPU chips (*e.g.* NVLink [62]). During inference, various hardware options like CPU, GPU, FPGA, and ASIC exhibit distinct hardware characteristics, which can help improving LLM inference performance. CPUs offer high programmability with a computing power of approximately 4 to 70 TOPS and with power consumption around from 4W to >200W. Modern CPUs (including some System-on-Chips, SoCs) enhance AI performance by integrating domain-specific architecture (DSA) units. These include Apple's Neural Engine in the M2 Ultra [63], Qualcomm's NPU in the Snapdragon 8 Gen3 [64], and Intel's AVX/AMX ISA extensions [65]. GPUs excel in parallelism and computing power, delivering between ~70 to >1000 TOPS and featuring an impressive memory bandwidth of

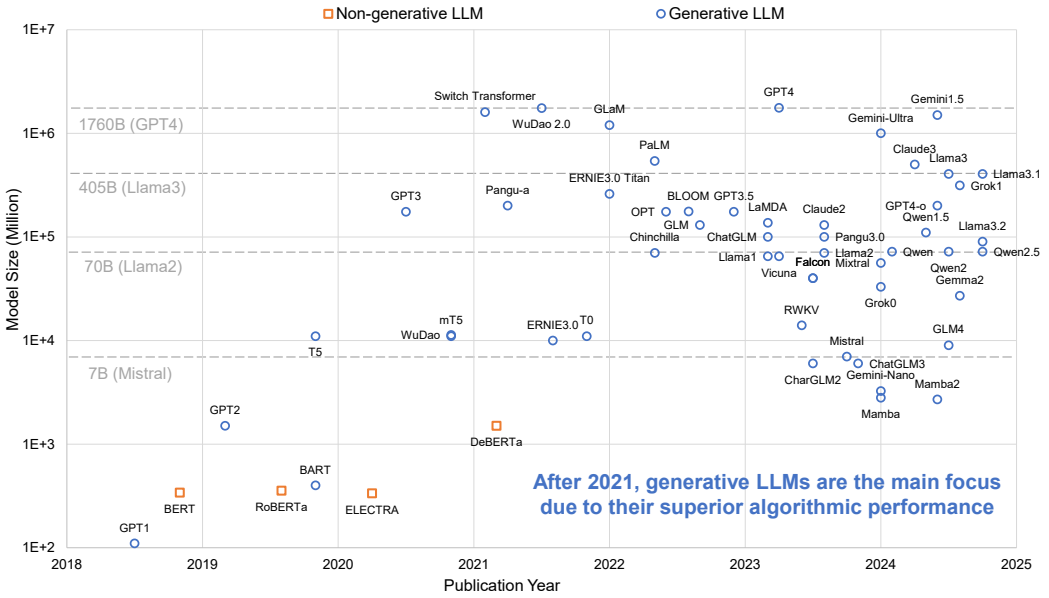

Fig. 1. Typical LLM model size in the past six years.

up to 1555 GB/s. On one hand, GPUs integrate a large number of SIMD cores and Tensor Cores in NVIDIA V100/A100/H100 [66–68] or Matrix Cores in AMD Instinct MI100/MI200/MI300 series [69–71] to enhance computing powers. On the other hand, GPUs support lower precision computations, such as INT8, FP8 and INT4 [67, 68], which allows for more multiplication units to be packed into a given chip area. Nevertheless, their power consumption is significantly higher, ranging from ~20W to >700W. FPGAs offer substantial parallelism and optimization capabilities, with computing performance between 50 to 100 TOPS. They are also more power-efficient, consuming about 75 to 100W. AMD's Xilinx offers Zynq [72], Virtex [73], and Versal [74] FPGA series for edge-side and cloud-side computing. The Zynq series, like ZCU102, combine ARM processors with FPGA programmable logic, has a power consumption of 10W-30W. The Virtex series, such as the VU13P, are designed for high-performance computing and data centers, providing up to 6.2 TFLOPS of computing power, with 3,840 DSP units and power consumption ranging from 30W to 70W. The Versal series, such as the VCK5000, integrate AI Engines [75] and efficient parallel computation units, offering >50 TOPS of computing power and 4,000 DSP units, with power consumption of 225W. ASICs like Groq LPU [76] and Cerebras WSE-3 [77] are often designed for specific applications with the customized architecture, offering higher computational efficiency and better energy efficiency.

Here, we list and compare the existing LLM inference surveys in Table 1. Previous surveys [78–82] primarily summarize various software optimization methods like quantization, sparsity, fast decoding for generative LLMs from an algorithm perspective. However, they do not consider that different optimization methods exhibit different inference performance across different hardware platforms, and similarly, they also lack a fair and quantitative comparison. Surveys [83, 84] focus on accelerating transformer-based LLMs, including non-generative LLMs like BERT and a small number of generative LLMs like GPT, but merely list the work done on different hardware platforms. And they lack a summary and abstraction of the optimization methods used by different accelerators. Additionally, it only provides a relative comparison of speedup and energy efficiency with different baselines, which is unfair. Like [83, 84], surveys [85, 86] mainly focus on non-generative LLMs on one or two specific hardware platforms. Our survey focuses solely on generative LLMs, summarizing

various software optimization methods in conjunction with multiple hardware platforms, including CPUs, GPUs, FPGAs, ASICs, and PIM/NDPs. What matters most for generative LLMs is the absolute inference speed and the inference energy efficiency. Therefore, for the first time, we use the number of tokens generated per second (tokens per second, tokens/s) and the number of tokens generated per joule (tokens per joule, tokens/J) to evaluate LLM acceleration. Besides these two metrics, we also conduct the comparison on **(1) the performance of the same optimization methods across different hardware platforms**, **(2) the performance across different hardware platforms**, and **(3) the performance of different methods on the same hardware platform**. This can provide the systematic and comprehensive summary of existing inference accelerations with software optimization methods and hardware platforms.

Table 1. Comparison of existing LLM surveys

Survey	Generative LLM	Software Optimization	Hardware Platforms					Quantitative Comparison
			CPU	GPU	FPGA	ASIC	PIM/NDP	
[78–82]	✓	✓	✗	✗	✗	✗	✗	✗
[83, 84]	✗	✗	✓	✓	✓	✓	✓	✓
[85]	✗	✗	✗	✗	✗	✗	✓	✓
[86]	✗	✓	✗	✗	✓	✓	✗	✗
Ours	✓	✓	✓	✓	✓	✓	✓	✓

This survey aims to systematically summarize the optimization methods for generative LLMs across different hardware platforms. Section 2 delves into the inference process of LLMs, providing an overview of the architecture and functioning of mainstream generative LLMs. Section 3 first summarizes the different optimization methods on various platforms such as CPU, GPU, FPGA, ASIC, and PIM/NDP in tabular form, and then provides a detailed description of each method and related works. Additionally, for each method, we also perform a qualified and quantitative comparison to show the difference among the hardware platforms. Section 4 performs a qualitative and quantitative comparison of inference performance with batch sizes 1 and 8 on different hardware platforms. Furthermore, in section 5, we point out that three trends (multimodality, inference-time compute, and higher inference energy efficiency) are promising to redefine the capabilities of edge artificial intelligence systems. Section 6 summarizes the work of this survey.

2 GENERATIVE LLM ARCHITECTURE

2.1 Overview

The most common generative LLM is based on the transformer structure due to its abilities for capturing long-term dependencies [87]. The inference of generative LLM consists of two stages, the prefill stage and the decode stage, as shown in Figure 2. In the prefill stage, the input text is converted into embeddings and input into the LLM all at once. Each transformer layer performs operations, and the intermediate results (Key and Value, KV cache) are saved for reuse during the decode stage. After completing calculating the last layer and LM head, the first token (‘My’) is generated. Then, in the decode stage, the LLM generates each token output autoregressively and updates the KV cache each time until generating the stop string (‘</s>’). For daily application scenarios where the number of input tokens is from 128 to 256 and output tokens larger than 32, we obtain that the time proportion of the decode stage exceeds 80% by profiling Llama2-7B on single NVIDIA A100 GPU, as shown in Figure 2(c).

In the prefill stage, the attention computation has quadratical computational and storage complexity related with input text length. To address this issue, many hardware-efficient LLMs are designed for more efficient processing of longer length. Therefore, we summarize these LLMs

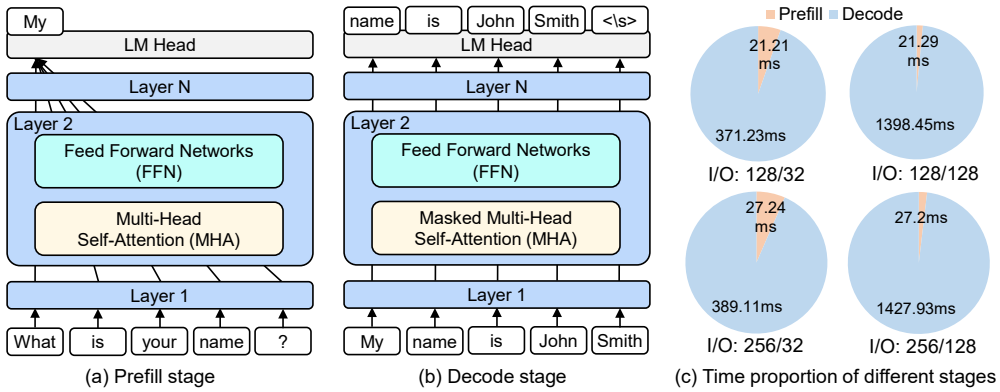

Fig. 2. LLM inference includes prefill and decode stages. During inference in daily scenarios (input tokens: 128≤I≤256, output tokens: O≥32), the time of decode stage is dominant.

in sub-section 2.2. And for the decode stage, we summarize hardware optimization methods for different platforms such as CPU, GPU, FPGA, ASIC, and PIM/NDP, and provide inference results for generative LLMs in latter sub-sections.

2.2 Efficient Generative LLM

2.2.1 Attention-based LLM. In LLMs, the most common type of attention is self-attention, where each token in the input sequence attends to every other token, capturing both local and global dependencies. This is achieved by computing three vectors for each token: Query (Q), Key (K), and Value (V). The attention scores are calculated as the scaled dot product of Q and K, followed by a softmax operation to determine the weight of each token in the sequence. These weights are then used to compute a weighted sum of the V vectors, producing the attention output.

$$O = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Many methods focus on simplifying the attention mechanism. Transformer-XL [88] adopts a segment-level recurrence mechanism and a positional encoding scheme to learn dependencies beyond a fixed length without disrupting temporal coherence. Linear Transformer [89] represents self-attention as a linear dot product of kernel feature maps and alters the computation order by leveraging the associativity of matrix multiplication. This modification reduces the complexity from $O(L^2)$ to $O(L)$, where L is the context length, significantly accelerating the computation of autoregressive Transformers. Another efficient structure is the Attention-Free Transformer (AFT) [90]. Unlike vanilla transformers, which first compute the query-key product, AFT combines the key and value with a set of learned positional biases before performing element-wise multiplication with the query. As a result, the memory complexity of AFT is linear with respect to both the context size and feature dimensions, enabling support for larger input lengths and model sizes. Based on AFT, the Receptance Weighted Key Value (RWKV) [39] combines the efficient parallel training capabilities of Transformers with the efficient inference of RNNs. It leverages linear attention mechanisms and allows the model to be expressed as either a transformer or an RNN. It also enables parallel computation during training while maintaining constant computational and memory complexity during inference. DiJiang [91] introduces a novel frequency-domain kernelization method based on the Discrete Cosine Transform (DCT). It points out that improving attention mechanisms often requires extensive retraining, which is impractical for large language models with vast numbers of parameters. This approach enables the conversion of a pre-trained standard Transformer into

a model with linear complexity and low training costs, utilizing a weighted quasi-Monte Carlo method for sampling. Extensive experiments demonstrate that this method achieves performance comparable to the vanilla transformer while significantly reducing training costs and substantially increasing inference speed.

2.2.2 SSM-based LLM. State Space Model (SSM) defines a linear mapping from an input x to output y through a hidden state h, where A is the state matrix, B is the input matrix, and C is the output matrix. In generative LLMs, SSM can understand and compress the input text into hidden states, and then generate output text based on these states.

$$\begin{aligned} h &= A \times h + B \times x \\ y &= C \times h \end{aligned} \tag{2}$$

The Structured State Space Sequence Model (S4) [92] involves conditioning the matrix A with low-rank corrections, enabling it to be stably diagonalized, and simplifying the SSM to computations that involve an in-depth exploration of the Cauchy kernel. It offers significantly higher computational efficiency compared to previous methods while retaining its theoretical advantages. The Gated State Space Model (GSS) [93] is built on the effectiveness of gated activation functions. GSS demonstrates significantly faster training speeds on TPUs compared to S4, and it competes effectively with several Transformer-based LLMs. Hyena [94] addresses that existing sub-quadratic methods based on low-rank and sparse approximations need to be combined with dense attention layers to match the performance of transformers. Therefore, it introduces a sub-quadratic direct replacement for attention, constructed using interleaved implicit parameterized long convolutions and data-controlled gating. Hyena can improve accuracy by over 50 points compared to operators relying on state space models and other implicit and explicit methods. Due to the inability to perform content-based reasoning for linear attention, gated convolution, recurrent models, and S4, Mamba [43] makes the SSM parameters a function of the input and enables the model to selectively propagate or forget information along the sequence length dimension based on the current token. Additionally, a hardware-aware parallel algorithm was designed for the recurrent mode, enhancing computational efficiency. DenseSSM [95] enhance Mamba by selectively integrating shallow layer hidden states into deeper layers. Despite the dense connections, DenseSSM maintains both training parallelism and inference efficiency. Mamba2 [44] demonstrates that transformer and SSM model families are closely related through various decompositions of a well-studied class of structured quasi-separable matrices. Mamba2 also introduces the State Space Duality (SSD) framework, with its core layer being an improved version of the selective SSM used in Mamba and offering a 2-8× speedup.

2.2.3 Hybrid LLM. Some other LLMs integrate attention-based and SSM-based LLMs, leveraging the complete information extraction ability of attention and the information compression capability of SSM to enhance the performance for long inputs. The Block-State-Transformer (BST) [96] integrates an SSM sublayer for long-range contextualization with a block-transformer sublayer for short-term sequence representation. This architecture combines the strengths of SSMs and block attention, and explores three distinct, fully parallelizable variants. Griffin [97] combines gated linear recurrence with local attention, featuring the Hawk layer (a type of RNN with gated linear recurrence). Jamba [98] interleaves blocks of transformer and Mamba layers, harnessing the strengths of both model families. In some of these layers, mixture of expert (MoE) is added to increase model capacity while keeping the number of active parameters manageable. Unlike BST, Infini-Transformer [99] combines masked local attention and long-term linear attention within a single Transformer block. This Infini-Attention mechanism incorporates compressed memory into the original attention mechanism within the constraints of limited memory and computational

resources. MEGALODON [100] is a neural architecture designed for efficient sequence modeling with infinite context length. It builds on the MEGA architecture (exponential moving average with gated attention) and introduces complex exponential moving average (CEMA), timestep normalization layer, normalized attention mechanism, and a pre-norm configuration with two-hop residuals to enhance its capability and stability.

3 OPTIMIZATIONS ON HARDWARE PLATFORMS

In this section, we provide an overview of the hardware platforms and various optimization techniques used in LLM inference. As shown in Table 2, the hardware platforms include CPU, GPU, FPGA, ASIC, and PIM/NDP, while the optimization methods include quantization, sparsity, fast decoding, operator optimization, heterogeneous cooperation, and homogeneous cooperation. In the following sections, we will provide a detailed explanation of the principles of each optimization method and related works, followed by a qualified and quantitative comparison.

3.1 Quantization

3.1.1 Overview. Quantization converts the model's weights and activations from high-precision formats (32-bit floating-point numbers) to low-precision formats (such as 4-bit integers). This process aims to reduce the model's storage requirements and computational costs while maintaining its accuracy. From the perspective of data format, quantization includes uniform and non-uniform quantization. Uniform quantization is a method where the value range is divided into several equal intervals. In uniform quantization, the entire range of values is partitioned into equally sized intervals, with each interval mapped to a discrete representation value. These discrete values are typically represented using fewer bits (e.g., 8 bits). The advantages of uniform quantization include its simplicity and high computational efficiency. However, it may not effectively capture the data distribution characteristics, especially when the data distribution is uneven, potentially leading to significant information loss. Non-uniform quantization, on the other hand, uses intervals of varying sizes based on the actual data distribution. It divides the data range into different-sized intervals, for example, using smaller intervals in regions where the data distribution is dense and larger intervals where the distribution is sparse. This approach can better preserve the details and features of the data, thus improving the model's accuracy. Non-uniform quantization typically requires additional computation and storage to manage the quantization intervals, but it provides higher precision and effectiveness in quantization.

Granularity in quantization is crucial for determining model performance and efficiency. The granularity are group-wise, channel-wise, and tensor-wise. Group-wise granularity is a coarser approach where multiple channels or layers are quantized with the same parameters. This means that within a group, all channels or layers use identical quantization settings. The advantage of group-level granularity is its simplicity and relatively low computational and storage overhead. However, it may not capture the individual characteristics of each channel or layer as effectively, potentially resulting in some compromise in model performance. Channel-wise granularity involves quantizing each channel individually within the model. Each channel can have its own quantization parameters, allowing for more precise adjustments according to the weight distribution and activation characteristics of each channel. This granularity offers a balance between precision and flexibility, though it increases the complexity of implementation and computation. Tensor-wise granularity is the most detailed approach, where each tensor (such as weight tensors or activation tensors) is quantized separately. This means that each tensor has its own quantization parameters, enabling the highest degree of adaptation to the specific characteristics of each tensor and providing the best precision. However, this level of granularity comes with the highest computational and

Table 2. Existing generative LLM inference optimizations on different hardware platforms

Methods	CPU	GPU	FPGA	ASIC	PIM/NDP
Quantization	Shen et al. [101], T-MAC [102], Snapdragon 8 Gen3 [64], llama.cpp [103], NoMAD-Attention [104]	GPTQ [105], AWQ [106], SpQR [107], SqueezeLLM [108], LLM-MQ [109], APTQ [110], Li et al. [111], LUT-GEMM [112], FLUTE [113], FP6-LLM [114], LLM.int8 [115], SmoothQuant [116], QUIK [117], Atom [118], LLM-FP4 [119]	FlexRun [120], HLSTransform [121], SECDA-LLM [122], Chen et al. [123], FlightLLM [124], EdgeLLM [125]	FIGNA [126], MECLA [127], OliVe [128], Li et al. [129], Tender [130]	Guo et al. [131], TransPIM [132], Sharda et al. [133]
Sparsity	Turbo Sparse [134], ProSparse [135]	LLM-pruner [136], SparseGPT [137], Wanda [138], E-Sparse [139], Flash-LLM [140], Agarwalla et al. [141], DejaVu [142], Sparse Transformer [143], Bigbird [144], StreamingLLM [145], Longformer [146], Adaptively Sparse Attention [147], Reformer [148], Sparse Flash Attention [149], Sparse Sinkhorn Attention [150], H_2O [151]	FlightLLM [124], EdgeLLM [125]	Spatten [152], TF-MVP [153], SOFA [154]	LauWS [155], HARDSEA [156], Sharda et al. [133]
Fast Decoding		LLMA [157], Speculative decoding [158], Lookahead [159], Medusa [160], EAGLE [161, 162], Ouroboros [163], Sequoia [164], Draft&Verify [165], Kangaroo [166], LayerSkip [167], Adainfer [168], RAEE [169], MOD [170]		C-Transformer [171]	SpecPIM [172]
Operator Optimization		FlashAttention [173, 174], FlashDecoding [175], FlashDecoding++ [176], DeepSpeed [177], vLLM [178], OpenPPL [179], cuBLAS [180], TensorRT-LLM [181], CUTLASS [182], ByteTransformer [183]		LPU [184], Groq LPU [76], Cerebras WSE-3 [77], Smax [185], MARCA [186], TCP [187], Habana Gaudi [188], Gaudi2 [189], Gaudi3 [190]	PIMnast [191], AttentionLego [192], PIM-GPT [193], SAL-PIM [194], PipePIM [195]
Heterogeneous Cooperation	Kim et al. [196], PowerInfer [197], PowerInfer-2 [198]				NeuPIMs [199], IANUS [200], MoNDE [201], Sharda et al. [133], AttAcc [202, 203], Kang et al. [204], Kim et al. [205], H3D-Transformer [206], CXL-PNM [207], 3D-HI [208], SK Hynix AiMX/AiMX-xPU [209, 210], Cambricon-LLM [211]
Homogeneous Cooperation	He et al. [212, 213]		DFX [214]		

storage costs and is the most complex to implement. There are two main quantization methods: weight-only quantization and weight-activation quantization.

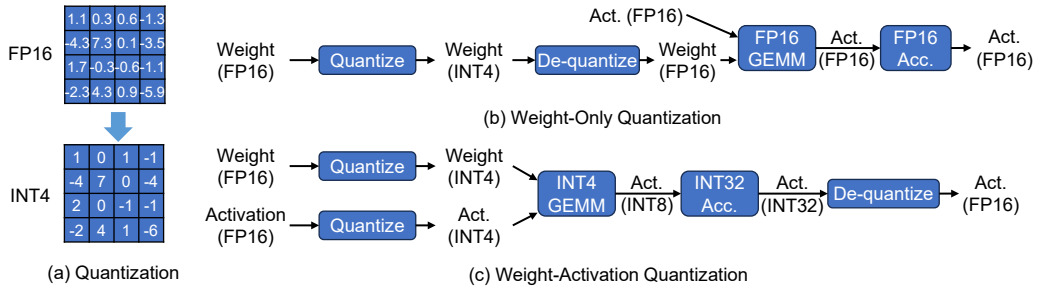

Fig. 3. Two main quantization methods: weight-only quantization and weight-activation quantization.

Weight-Only Quantization. Weight-only quantization involves converting the model's weight parameters from high-precision formats (like 32-bit floating-point numbers) to low-precision formats (such as 8-bit integers). This process typically includes discretizing the weights by mapping them to a finite set of discrete values and then representing these values with fewer bits (e.g., 8 bits). This approach significantly reduces storage requirements and accelerates computation. Weight-only quantization can be implemented using methods such as uniform quantization, which divides the weight range into equal intervals, or non-uniform quantization, which adjusts the intervals based on the distribution of weights to better preserve model accuracy. Matrix decomposition quantization is a specialized method where a large matrix is approximated by the product of several smaller matrices. This technique reduces the computational and storage requirements by representing a large matrix with multiple smaller matrices, which can be stored and processed in lower precision formats. This method is particularly beneficial for managing extremely large models, as it helps lower computational complexity and storage overhead.

Weight-Activation Quantization. Weight-activation quantization extends the concept of weight-only quantization to include the activations generated during model inference. In this method, both the weights and the activations at each layer are quantized to lower precision formats. This reduces memory bandwidth requirements and enhances inference speed. The challenge with weight-activation quantization is to manage the trade-off between quantization errors and model accuracy. Techniques such as dynamic range quantization or specific quantization schemes are used to balance precision and computational efficiency. Table 3 shows the usage of two quantization methods across different hardware platforms.

Table 3. Quantization on CPU, GPU, FPGA, ASIC, and PIM/NDP

Hardware	Weight-Only Quantization	Weight-Activation Quantization
CPU	✓	✗
GPU	✓	✓
FPGA	✗	✓
ASIC	✗	✓
PIM/NDP	✗	✓

3.1.2 CPU. **Weight-Only Quantization.** The optimization methods of quantization on CPUs mainly focus on weight-only quantization. Shen et al. [101] leverage Intel Neural Compressor to automate the INT4 quantization process with negligible accuracy loss, supporting various

quantization recipes such as GPTQ [105], AWQ [106] and TEQ [215]. They further develop a tensor library tailored for CPUs, which supports mainstream instruction sets like AVX2, AVX512, AVX512_VNNI, and AMX. By providing INT4 dequantization kernels on x86 CPUs, the experimental results on mainstream LLMs including Llama2, Llama and GPT-NeoX shows the latency of token generation is ranging from 12.5 tokens/s to 50 tokens/s for models with parameters ranging from 6B to 20B, by using a single socket of 4th Generation Intel Xeon Scalable Processors [65]. Qualcomm Snapdragon 8 Gen3 SoC [64] utilizes its proprietary Hexagon 700 AI processor and quantization techniques to support the efficient LLM execution. For Llama2-7B with 4-bit quantization, it achieves about 15 tokens/s. Some open-source repositories like llama.cpp [103] are designed for efficient LLM inference across diverse hardware platforms including CPUs, GPUs and ASICs. For Llama2-7B with 4-bit quantization, llama.cpp achieves 6 tokens/s with a single core and 32 tokens/s with eight cores on Apple M2-Ultra processors. For Llama2-7B with 2-bit quantization, llama.cpp achieves 4 tokens/s with a single core and 21 tokens/s with eight cores on M2-Ultra.

Due to the overheads of weight dequantization from integer to floating, T-MAC [102] leverages lookup tables (LUTs) for efficient low-bit LLM inference on edge CPUs, circumventing the need for dequantization and mixed precision matrix multiplication. For Llama2-7B with 4-bit quantization, T-MAC achieves 10 tokens/s with a single core and 38 tokens/s with eight cores on Apple M2-Ultra processors [63], and 3 tokens/s on Raspberry Pi 5 [216] integrated with ARM Cortex-A76. For Llama2-7B with 2-bit quantization, T-MAC achieves 17 tokens/s with a single core and 50 tokens/s with eight cores on M2-Ultra, and 6 tokens/s on Raspberry Pi 5. Furthermore, due to the vast quantities of expensive Multiply-Add (MAD) matrix operations in the attention computations, NoMAD-Attention [104] design an efficient attention algorithm that replaces MAD operations with in-register lookups. Through hardware-aware algorithmic designs, NoMAD-Attention achieves the computation of attention scores using repeated fast accesses to SIMD registers despite their highly limited sizes. Empirical evaluations demonstrate that for CodeLlama-7B with 4-bit quantization, NoMAD-Attention achieves 9 tokens/s with short context (e.g. 128) and 4 tokens/s with long context (e.g. 16k) on 2 Intel Xeon E5-2695 V3 14-core CPUs.

3.1.3 GPU. **Weight-Only Quantization.** GPTQ [105] is an one-shot weight quantization method based on approximate second-order information and error compensation, that is both highly-accurate and highly-efficient. It can quantize GPT models with 175 billion parameters in approximately four GPU hours, reducing the bitwidth down to 3-bit or 4-bit per weight, with negligible accuracy degradation relative to the uncompressed baseline. Experimental results show that the average time of per token of 3-bit OPT-175B model obtained via GPTQ running on a single A100 (80GB) is 14.1 tokens/s, which is about 3.25× faster than the FP16 version (running on 5 GPUs). On more accessible GPUs, such as the NVIDIA A6000 (48GB), the average time of per token is 7.7 tokens/s (running on 2 GPUs), which is about 4.53× faster than the FP16 version (running on 8 GPUs). AWQ [106] is based on the observation that protecting 1% of salient weights whose activations are extremely large can greatly reduce quantization error. It first searches for the optimal per-channel scaling and then multiplies the salient weights with the per-channel scalings. It also reduces the bitwidth down to 3 or 4 bits per weight. Experimental results with INT4 implementation show that for Llama-2-7B, it improves the inference speed from 52 tokens/s to 194 tokens/s on RTX 4090 desktop GPU (3.73× speedup). For Llama-2-13B, the inference speed is 110 tokens/s on RTX 4090 desktop GPU. On the laptop RTX 4070 GPU (8GB), it is able to run Llama-2-13B models at 33 tokens/s, while the FP16 implementation cannot fit 7B models.

To further reduce the accuracy loss for smaller models in the 1-10B parameter range, SpQR [107] works by identifying and isolating outlier weights, which cause particularly-large quantization errors, and storing them in higher precision like half data type (16-bit), while compressing all

other weights to 3-4 bits, and achieves relative accuracy losses of less than 1% in perplexity for highly-accurate LLaMA and Falcon LLMs. Experimental results show that SpQR with 3-bit and 16-bit quantization achieves 57 tokens/s, 44 tokens/s, 22 tokens/s and 12 tokens/s on A100 GPU, respectively. Unlike SpQR, SqueezeLLM [108] proposes a sensitivity-based non-uniform quantization method, which searches for the optimal bit precision assignment based on second-order information. It also applies dense and sparse decomposition that stores outliers and sensitive weight values in an efficient sparse format. Experimental results show that SqueezeLLM with 3bit and 16-bit quantization achieves 63.5 tokens/s, 49.2 tokens/s, 29.1 tokens/s and 14.5 tokens/s on A6000 GPU, respectively. LLM-MQ [109] proposes sensitivity-based precision allocation to assign the proper bitwidth for each layer within the given budget for weight memory based on their first-order information and quantization error. It also develops an efficient CUDA core kernels to accelerate LLMs by fusing the dequantization and general matrix-vector multiplication (GEMV). LLM-MQ deploys INT4 quantized Llama2-7B model on NVIDIA T4 GPU achieves up to 1.6× end-to-end speedup compared to the pytorch FP16 baseline. APTQ [110] proposes an attention-aware 2/4-bit mixed-precision quantization for LLMs, which considers not only the second-order information of each layer's weights, but also, for the first time, the nonlinear effect of attention outputs on the entire model. Li et al. [111] are the first to propose an intra-weight mixed-precision quantization for LLMs to further reduce accuracy loss under 3-bit. By applying 2/4-bit mixed-precision quantization with memory alignment and exclusive 2-bit sparse outlier reservation with minimum speed degradation, it achieves 2.91-bit for each weight considering all scales/zeros for different models with negligible loss. Additionally, they design an asynchronous dequantization and fuse the dequantization and GEMV kernels during inference. For Llama2-7B, it achieves 45.2 tokens/s on RTX 3090 GPU and 34.0 tokens/s on RTX 2080 GPU.

LUT-GEMM [112] proposes an efficient LUT-based GPU kernel for quantized matrix multiplication, which not only eliminates the resource-intensive dequantization process but also reduces computational costs compared to previous kernels for weight-only quantization. The impact of LUT-GEMM is facilitated by implementing high compression ratios through low-bit quantization and efficient LUT-based operations. For Llama-7B with 4-bit quantization, it achieves 163.9 tokens/s on A100 GPU, achieving a remarkable 1.64× token generation latency improvement compared to the pytorch FP16 baseline. FLUTE [113] is a flexible lookup table engine for LUT-quantized LLMs, which uses offline restructuring of the quantized weight matrix to minimize bit manipulations associated with unpacking, and vectorization and duplication of the lookup table to mitigate shared memory bandwidth constraints. For Llama3-8B with 4-bit quantization, it achieves 91.3-99.8 tokens/s and 113.7-121.7 tokens/s on NVIDIA A6000 and A100 GPUs, respectively. For Llama3-8B with 3-bit quantization, it achieves 91.9-110.0 tokens/s and 117.7-135.5 tokens/s on NVIDIA A6000 and A100 GPUs, respectively.

To effectively reduce the size of LLMs and preserve the model accuracy, FP6-LLM [114] proposes FP6 quantization on GPUs with TC-FPx, the first full-stack GPU kernel design scheme with unified Tensor Core support of float-point weights for various quantization bit-width. It solves the unfriendly memory access of model weights with irregular bit-width and high runtime overhead of weight de-quantization. Experimental results shows that for Llama2-13B with FP6 quantization, it achieves about 55 tokens/s on NVIDIA A100 GPU.

Weight-Activation Quantization. In addition to hardware units that support FP16 computations, NVIDIA GPUs also provide hardware units that support INT4, INT8, and FP8 computations. The number of these computation units can be 2× and 4× greater than FP16 on each chip. Compared to weight-only quantization, weight-activation quantization can utilize INT4, INT8, and FP8 computations, thereby maximizing the peak computational performance of the GPU. Since the prefill phase in LLM inference is compute-bound, weight-activation quantization can significantly

enhance performance during this stage. LLM.int8 [115] uses vector-wise quantization with separate normalization constants for each inner product in the matrix multiplication, to quantize most of the features. For the outliers, it isolates the outlier feature dimensions into a 16-bit matrix multiplication while still more than 99.9% of values are multiplied in 8-bit. For BLOOM-176B model, LLM.int8 achieves 4.05 tokens/s, 30.3 tokens/s and 109.77 tokens/s for batch size 1, 8 and 32, respectively, on 3 A100 GPUs in decode phase. The inference speed is slightly slower but close to 16-bit inference with less GPU consumption. SmoothQuant [116] enables 8-bit weight and 8-bit activation (W8A8) quantization for LLMs. Based on the fact that weights are easy to quantize while activations are not, SmoothQuant smooths the activation outliers by offline migrating the quantization difficulty from activations to weights with a mathematically equivalent transformation. SmoothQuant enables an INT8 quantization of both weights and activations for all the matrix multiplications in LLMs. SmoothQuant achieves up to 1.56× speedup and 2× memory reduction for LLMs with negligible loss in accuracy. QUIK [117] is for the first time, that the majority of inference computations for LLMs can be performed with both weights and activations being cast to 4 bits. QUIK compresses most of the weights and activations to 4-bit, while keeping some outlier weights and activations in higher-precision. It also provides GPU kernels matching the QUIK format with highly-efficient layer-wise runtimes, which lead to practical end-to-end throughput improvements of up to 3.4× relative to FP16 execution in prefill phase.

Prevalent quantization schemes (e.g., W8A8) cannot fully leverage the capabilities of modern GPUs, such as 4-bit integer operators, resulting in sub-optimal performance. To maximize the throughput, Atom [118] significantly boosts serving throughput by using low-bit operators and considerably reduces memory consumption via low-bit quantization. It attains high accuracy by applying a novel mixed-precision and fine-grained quantization process. For single batch inference, Atom can achieve about 30 tokens/s for on a NVIDIA RTX 4090 GPU. Atom improves end-to-end throughput by up to 7.73× compared to the FP16 and by 2.53× compared to INT8 quantization, while maintaining the same latency target.

Compared to integer quantization, floating-point (FP) quantization can better handle long-tail or bell-shaped distributions, and it has emerged as a default choice in many hardware platforms. LLM-FP4 [119] quantizes both weights and activations in LLMs down to 4-bit floating-point values (W4A4) with negligible accuracy loss. Due to the lack of PF4 computing unit in GPUs, its decoding speed maybe slower than FP16 baseline.

3.1.4 FPGA. **Weight-Activation Quantization.** FlexRun [120] uses 8-bit quantization (W8A8), conducts an in-depth design space exploration to find the best accelerator architecture for a target LLM model, and automatically reconfigures the accelerator based on the exploration results. With the implementation on Intel Stratix 10 GX and MX FPGAs, FlexRun outperforms the current state-of-the-art FPGA-based accelerator by 1.15×–1.50× for GPT2, respectively. Compared to Nvidia's V100 GPU, FlexRun achieves 2.69× higher performance on average for various GPT2 models. HLSTransform [121] uses HLS to design a FPGA accelerator and synthesis combined with pipelining, memory unrolling, and memory partitioning and transfer optimizations, with the addition of 8-bit integer quantization (W8A8). On a tiny model with 110 million parameters, HLSTransform achieves 57.11 tokens/s on Xilinx Virtex UltraScale+ VU9P FPGA. SECDA-LLM [122] utilizes quantization (W3A8) and designs an efficient FPGA-based LLM accelerators for the llama.cpp inference framework. By deploying on the PYNQ-Z1 board, it achieves 0.588 tokens/s for the TinyLlama model (1.1B). Chen et al. [123] investigate the feasibility and potential of model-speciic spatial acceleration for LLM inference on FPGAs. They introduce a comprehensive analytical model to estimate the LLM inference performance of FPGA accelerator with W4A8 quantization, and provide a library of high-level synthesis (HLS) kernels that are composable and reusable. For

Llama2-7B, during prefilling phase, they can achieves about 213 tokens/s, 43 tokens/s, and 320 tokens/s on Xilinx Alveo U280, VCK5000, and VHK158 FPGAs, respectively. During decode stage, they can achieves about 200 tokens/s, 40 tokens/s, and 333 tokens/s on Xilinx Alveo U280, VCK5000, and VHK158 FPGAs, respectively.

3.1.5 ASIC. **Weight-Only Quantization.** Despite the memory footprint reduction achieved by weight-only quantization, the actual computing performance is not really improved due to dequantization from integer to float. FIGNA [126] proposes dedicated FP-INT arithmetic units designed specifically for FP-INT MAC operations and integrates them on the accelerator. FIGNA with FP16-INT4 provides 3.2768 TOPS computing power and 26.58W power consumption by considering all memory access in 28nm at 100MHz. Estimated result shows that for OPT-6.7B it can achieve 21.332 tokens/s in decode stage. Different from normal quantization methods, MECLA [127] proposes a parameter-efficient scaling sub-matrix partition method (SSMP) to decompose large weight matrices into several tiny-scale source sub-matrices (SS) and derived sub-matrices (DS). For memory issues, SSMP avoids accessing the full weight matrix but only requires small SS and DS scaling scalars. For computation issues, the proposed accelerator fully exploits the intermediate data reuse of matrix multiplication via on-chip matrix regrouping, inner-product multiplication re-association, and outer-product partial sum reuse. Totally, it can reduce 83.6% memory access and 72.2% computation. MECLA provides 14.008 TOPS computing power and ~96W (1.9763W+94W) power consumption by considering all memory access under 28nm. For Llama2-7B and BLOOM-7B, compared to NVIDIA V100 GPU, MECLA achieves 6.74× and 5.91× inference speedup (~161 tokens/s and 141 tokens/s, respectively).

Weight-Activation Quantization. Based on the key insight that outliers are important while the normal values next to them are not, OliVe [128] adopts an outlier-victim pair (OVP) quantization and handles outlier values locally with low hardware overheads. This enables a memory-aligned W4A4/W8A8 quantization, which can be efficiently integrated to the existing hardware accelerators like systolic array and tensor core. OliVe provides 0.71 TOPS computing power and ~8W (0.2806W+7.9872W) power consumption by considering all memory access under 22nm. Estimated results shows that for OPT-6.7B it can achieve 9.173 tokens/s in decode stage. Li et al. [129] uniformly group weights and activations to ensure workload balance for hardware, and propose two approaches called channel sorting and channel selection to enhance the performance of quantization. It provides 1.43 TOPS computing power and ~8.5W (0.472W+7.9872W) power consumption by considering all memory access under 65nm. Estimated results shows that for OPT-6.7B it can achieve 19.733 tokens/s in decode stage. Tender [130] decomposes weight and activation matrices by groups with different size to smooth the impact of outliers. And the format of scale factors are powers of two apart, which avoids explicit dequantization and extension to the commodity tensor compute hardware. It is 7.174W (1.60W+5.574W) power consumption by considering HBM2 memory access under 28nm. Result shows that for OPT-6.7B, Tender achieves 1.33× speedup (53.33 tokens/s) than NVIDIA A100 GPU.

3.1.6 PIM/NDP. ReRAM-based analog PIM architectures perform integer MVMs using voltage, current, and conductance in the analog domain, limiting their application to the more accurate floating point (FP) data format. Guo et al. [131] propose an ReRAM and 3D-SRAM-based hybrid PIM architecture with non-uniform data format, achieving FP-based algorithm accuracy, high device utilization, and high energy efficiency. At the software level, they first analyze the impact of quantization errors on the accuracy of attention-free LLMs. For the quantization error-insensitive MVM operations, they propose the PIM-oriented exponent-free non-uniform (PN) data format. The proposed PN format can be flexibly adjusted to fit the data distribution and approach the accuracy of the FP format using bit-slicing-based full INT operations. For the quantization error-sensitive EWM

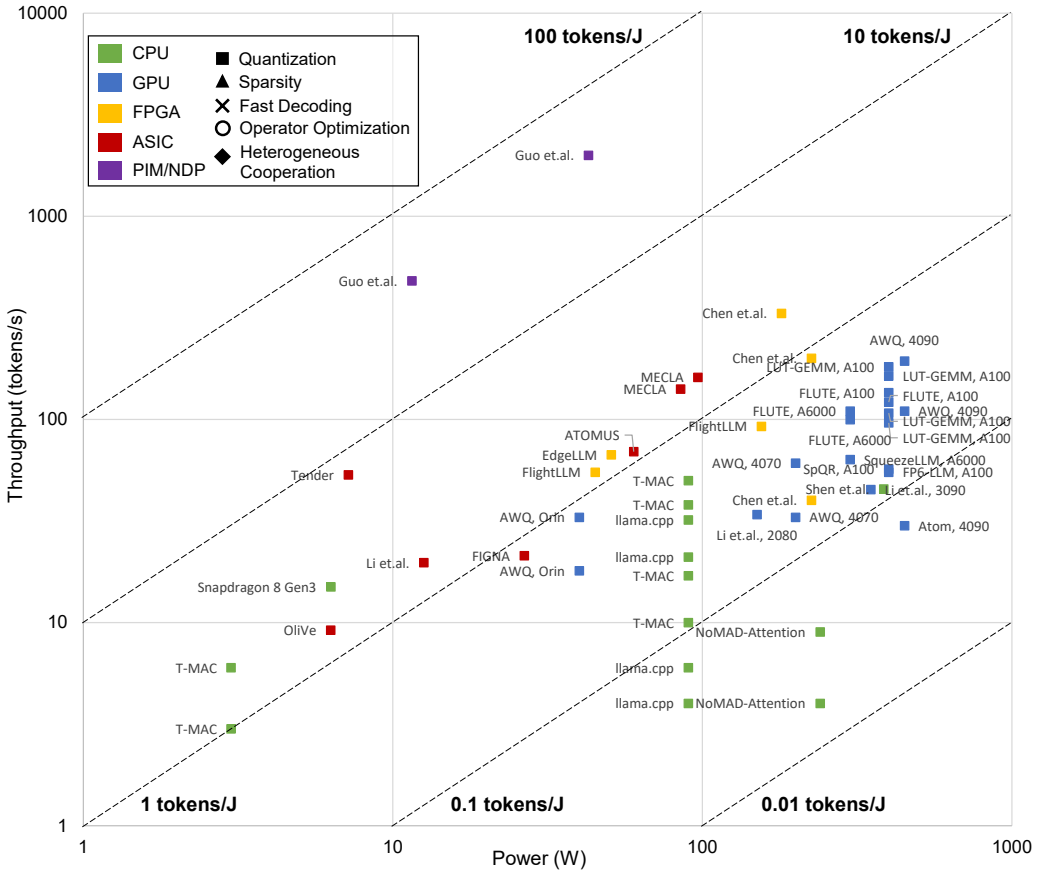

Fig. 4. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with quantization.

operations, they introduce the multiplication free approximated FP multiplications to reduce the additional hardware overhead for PIM. At the hardware level, they propose a hybrid PIM architecture, including an ReRAM analog PIM using shift-and-add for PN-based MVMs, and a 3D-SRAM digital PIM with high utilization for multiplication-free FP-based element-wise operations. Extensive experiments show that the proposed PIM architecture achieves up to 89× and 16× speedup with 2537× and 12× energy efficiency improvement compared with GPU and PIM-baseline, respectively. TransPIM [132] is a memory-based acceleration for Transformer using software and hardware co-design. In the software-level, TransPIM adopts a token-based dataflow to avoid the expensive inter-layer data movements introduced by previous layer-based dataflow. In the hardware-level, TransPIM introduces lightweight modifications in the conventional HBM architecture to support PIM-NMC hybrid processing and efficient data communication for accelerating Transformer-based models. TransPIM system uses the 8GB HBM as the memory with $2.15mm^2$ area overhead and about 40.01W power consumption. Experimental results show that for GPT2 models, TransPIM achieves at least 22.1× speedup than NVIDIA RTX 2080Ti GPU. Other PIM/NDP accelerators like TransPIM [132] and Sharda's method [133] also involve the quantization to further improve LLM inference.

3.1.7 Quantitative Comparison. We first compare the power consumption, inference speed and energy efficiency for different hardware platforms, in Figure 4. For quantization, power consumption ranges from 3W to 450W, with inference speeds between 3 tokens/s and 1998 tokens/s. The energy efficiency ranges from 0.0167 tokens/J to 46.66 tokens/J. T-MAC [102] (CPU) achieves the lowest power consumption with 3W and Guo et al. [131] (PIM/NDP) achieves the highest throughput with batch size 1 (pre-silicon simulation result). And Guo et al. [131] also achieves the highest energy efficiency with 46.66 tokens/J.

- For CPUs, power consumption ranges from 3W to 385W, with inference speeds between 3 tokens/s and 50 tokens/s, located in the bottom part of the figure. The energy efficiency ranges from 0.0167 tokens/J to 2.38 token/J. Additionally, we observe edge CPUs (including CPU SoCs) with 3W to 6W power consumption exhibit higher energy efficiency (0.544 tokens/J to 2.38 tokens/J).
- For GPUs, power consumption ranges from 40W to 450W, with inference speeds between 18 tokens/s and 194 tokens/s, situated in the upper right part of the figure. The energy efficiency ranges from 0.0667 tokens/J to 0.825 token/J. Compared to other hardware, GPUs can achieve higher absolute inference speeds due to their high computing power and high bandwidth. When quantization methods are used, the memory access bottlenecks in LLM inference are alleviated, further unlocking computing power.
- For FPGAs, power consumption ranges from 45W to 225W, with inference speeds between 40 tokens/s and 333 tokens/s, also in the upper right part of the figure. The energy efficiency ranges from over 0.178 tokens/J to 1.85 tokens/J, which is higher than GPUs and server CPUs.
- For ASICs, power consumption ranges from 6.3W to 96.66W, with inference speeds between 9.173 tokens/s and 161.086 tokens/s, found in the upper left section of the figure. The energy efficiency ranges from 0.803 tokens/J to over 7.434 tokens/J, outperforming CPU, GPU and FPGA hardware platforms.
- For PIM/NDPs, power consumption ranges from 11.516W to 42.819W, with inference speeds between 481 tokens/s and 1998 tokens/s, found in the upper left section of the graph. The energy efficiency outperforms other hardware platforms.

Overall, both weight-only quantization and weight-activation quantization methods can enhance absolute inference speed and improve energy efficiency. Weight-only quantization reduces bandwidth requirements but introduces additional dequantization operations, which can increase hardware power consumption while improving absolute speed. On the other hand, weight-activation quantization reduces the hardware compute unit area and power consumption by using smaller-width computation units, leading to improved absolute speed while lowering overall hardware power consumption.

3.2 Sparsity

3.2.1 Overview. Sparsity reduces the number of non-zero elements and skip the multiplication and addition with zero to improve efficiency of computation and storage. Due to the presence of attention computations in standard transformer-based large models, sparsification methods include not only weight sparsity and activation sparsity but also attention sparsity. **Weight sparsity** is primarily achieved through pruning methods, including global pruning, layer-wise pruning, and structured pruning, which reduce the size of weight matrices and leverage sparse matrix libraries for optimization. **Activation sparsity** focuses on reducing the computation of activation values by employing techniques such as activation pruning (e.g., threshold pruning) and dynamic sparsity, with hardware optimizations utilizing sparse data structures to enhance efficiency. **Attention sparsity** addresses the optimization of computations in self-attention mechanisms, employing

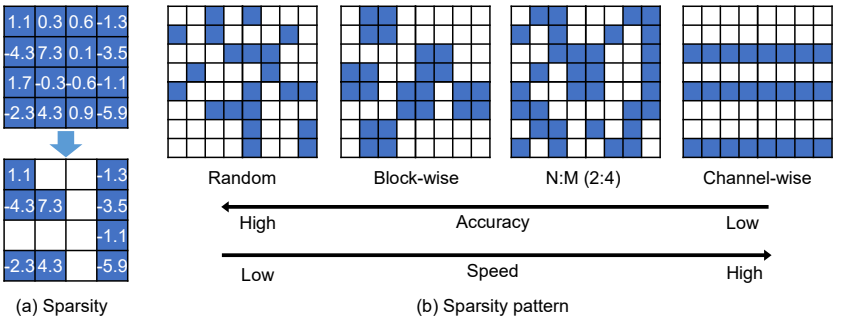

Fig. 5. Sparsity and sparsity patterns.

methods like local attention, block-wise attention, and sparse attention matrices, which reduce computational load by limiting the calculation scope or using sparse matrix storage. These sparsity strategies help improve model inference efficiency, particularly when dealing with large-scale data and complex tasks.

Sparsity patterns can be categorized into random and structured sparsity as shown in Figure 5. **Random pattern** involves a random distribution of zero elements within the matrix, achieving higher accuracy but potentially lower speed for computation. **Structured pattern** applies a specific pattern to the sparsity, improving computational efficiency by aligning with hardware optimizations. Within structured sparsity, common patterns include block-wise sparsity, N:M sparsity, channel-wise sparsity and some combinations of structured pattern sparsity. These structured patterns offer predictable and optimized computational benefits. Block-wise sparsity involves dividing the weight matrix into smaller blocks and applying sparsity within each block. N:M sparsity retains M non-zero elements out of every N elements, improving efficiency through hardware acceleration. NVIDIA's 2:4 sparse Tensor Core is a representative hardware unit for N:M sparsity, capable of achieving up to 2× computational acceleration. Channel-wise sparsity aims to prune entire channels in a matrix, significantly reducing computation and storage needs. Table 4 shows the usage of three sparsity methods across different hardware platforms.

Table 4. Sparsity on CPU, GPU, FPGA, ASIC, and PIM/NDP

Hardware	Weight Sparsity	Activation Sparsity	Attention Sparsity
CPU	✗	✓	✗
GPU	✓	✗	✓
FPGA	✗	✗	✗
ASIC	✗	✗	✓
PIM/NDP	✗	✗	✗

3.2.2 CPU. **Activation Sparsity.** Activation sparsity is determined by activation functions. Commonly using SwiGLU [217] and GeGLU [218] exhibits limited sparsity for LLMs, but simply replacing these functions with ReLU fails to achieve sufficient sparsity. Turbo Sparse [134] proposes the dReLU activation function to improve LLM activation sparsity, along with a high-quality training data mixture ratio to facilitate effective sparsity. By applying their sparsity method to the Mistral and Mixtral models, only 2.5 billion (35.7%) and 4.3 billion (9.2%) parameters are activated per inference iteration, respectively. For Mistral-7B, Turbo Sparse achieves 8.71 tokens/s and 9.94 tokens/s on Intel i9-14900HX processor and Intel i7-12700K processor, respectively. For Mixtral-47B with 4-bit quantization, Turbo Sparse achieves 16.1 tokens/s, 11.98 tokens/s and 11.1 tokens/s

on Intel i9-14900HX, Intel i7-12700K and SnapDragon 8 Gen3, respectively. ProSparse [135] also introduces activation function substitution, progressive sparsity regularization, and activation threshold shifting to help non-ReLU LLMs obtain high activation sparsity without performance degradation. For Llama2-7B and Llama2-13B, ProSparse achieves high sparsity of 89.32% and 88.80%, and 16.3 tokens/s and 8.67 tokens/s, respectively, based on PowerInfer [197] framework.

3.2.3 GPU. **Weight Sparsity.** LLM-pruner [136], adopts structural pruning that selectively removes non-critical coupled structures based on gradient information, maximally preserving the majority of the LLM's functionality. To this end, the performance of pruned models can be efficiently recovered through tuning techniques, LoRA, in merely 3 hours, requiring only 50K data. We validate the LLM-Pruner on three LLMs, including Llama, Vicuna, and ChatGLM, and demonstrate that the compressed models still exhibit satisfactory capabilities in zero-shot classification and generation.

LLMs can be pruned to at least 50% sparsity in one-shot, without any retraining, at minimal loss of accuracy. SparseGPT [137] requires a sophisticated weight update procedure in an iterative pruning process. Wanda [138] prunes weights with the smallest magnitudes multiplied by the corresponding input activations, on a per-output basis. Notably, Wanda requires no retraining or weight update, where pruning process is faster. Besides unstructured pattern, these two methods generalizes to semi-structured N:M (2:4 and 4:8) patterns. E-Sparse [139] introduces entropy to quantify the information richness within each channel (intra-channel) of the input features, and adopts it to enhance the feature norms (crosschannel) as a metric to evaluate parameter importance. Furthermore, it proposes Channel Shuffle to reorder the information distribution in LLMs to obtain N:M Sparsity with less information loss. 2:4 sparsity as supported by NVIDIA GPUs of generation Ampere and newer theoretically offers 2× acceleration of matrix multiplications. In practical, 2:4 sparsity can achieve 1.54×-1.79× speedup for MatMul, and end-to-end speedups are about 1.21×-1.25× (due to some extra overheads from e.g. attention).

Based on the key observation that the bottleneck of LLM inference is the skinny matrix multiplications, Flash-LLM [140] proposes a general Load-as-Sparse and Compute-as-Dense methodology for unstructured sparse matrix multiplication. Flash-LLM proposes a new sparse format called Tiled-CSL to relieve the memory bandwidth bottleneck and support the tile-by-tile SpMM execution with tensor cores. For OPT-30B, Flash-LLM achieves 80% sparsity with 1.44% accuracy decrease and about 290 tokens/s, 500 tokens/s, 800 tokens/s, and 1187 tokens/s on single A100 GPU with batch sizes 8, 16, 32, and 64, respectively.

Agarwalla et al. [141] combine the SparseGPT one-shot pruning method and sparse pretraining to pretrain a high sparsity LLM. They deploy model on GPU and CPU by utilizing Neural Magic's DeepSparse engine and Neural Magic's nm-vllm engine, respectively. For Llama-7B, on NVIDIA A10 GPU, they achieve 44.4 tokens/s and 47.9 tokens/s with 50% sparsity and 70% sparsity, respectively. On AMD EPYC 9R14 Processor, they achieve 4.4 tokens/s and 6.9 tokens/s with 50% sparsity and 70% sparsity, respectively.

Existing methods require costly retraining, forgo LLM's in-context learning ability, or do not yield wall-clock time speedup on modern hardware. DejaVu [142] predicts contextual sparsity on the fly given inputs to each layer, along with an asynchronous and hardware-aware implementation that speeds up LLM inference. For OPT-175B model, DejaVu achieves up to 75% sparsity and 50 tokens/s on 8 A100-80GB GPUs with batch size 1, which is over 2× and 6× faster than FasterTransformer and Hugging Face implementation, respectively.

Attention Sparsity. During the prefilling phase of LLM inference, attention computation complexity scales quadratically with input sequence length. Given limited GPU computing and memory resources, attention sparsification can reduce the number of attention values to accelerate prefilling phase. For static sparsity, Sparse Transformer [143], StreamingLLM [145], Bigbird [144],

and Longformer [146] use the manual combination of global and local patterns to replace the full attention patterns. The local pattern captures the local context of each token within a fixed size or stride while the global pattern captures the relationship between the specific tokens to all other tokens. For Llama2-7B and Llama2-13B models, StreamingLLM achieves 15.38-32.26 tokens/s and 9.43-20.83 tokens/s on single NVIDIA A6000 GPU, respectively. For dynaimic sparsity, Adaptively Sparse Attention [147] replaces softmax with α-entmax, a differentiable generalization of softmax that allows low-scoring words to receive precisely zero weight and drops parts of the context that are no longer required for future generation. Reformer [148] replaces dot-product attention by using locality-sensitive hashing, changing the complexity from $O(L^2)$ to $O(LlogL)$, where L is the sequence length. Sparse Flash Attention [149] extends FlashAttention [219] GPU kernel and encompasses key/query dropping and hashing-based attention. Sparse Sinkhorn Attention [150] adopts a learned sorting network to align keys with their relevant query buckets, ensuring that attention is computed only between the corresponding query-key pairs. H_2O [151] observes that a small portion of tokens (called Heavy Hitters, H_2) contributes most of the value when computing attention scores. H_2O introduces a dynamic attention sparsification method to adopt KV cache eviction policy that dynamically retains a balance of recent and H_2 tokens. For OPT-6.7B model, H_2O with 20% H_2 achieves 30.4 tokens/s on single NVIDIA T4 GPU.

3.2.4 FPGA. **Weight Sparsity.** FlightLLM [124] is the first real FPGA-based LLM accelerator which proposes a configurable sparse DSP chain to support different sparsity patterns with high computation efficiency. Then, it proposes an always-on-chip decode scheme to boost memory bandwidth with mixed-precision support. Finally, it proposes a length adaptive compilation method to reduce the compilation overhead. For Llama2-7B model, FlightLLM achieves 55 tokens/s and 92.5 tokens/s with batch size 1 on the Xilinx Alveo U280 FPGA and Versal VHK158 FPGA, respectively. EdgeLLM [125] integrates 4-bit weight-only quantization and utilizes log-scale structural sparsity for weight parameters in the matrix multiplication operator. For ChatGLM2-6B model, it achieves average 67 tokens/s with 55.07W power consumption on AMD Xilinx VCU128 FPGA.

3.2.5 ASIC. **Attention Sparsity.** Spatten [152] leverages token sparsity, head sparsity, and quantization opportunities to reduce the attention computation and memory access. It assesses the cumulative importance of each word by aggregating the attention matrix columns, subsequently pruning tokens with minimal cumulative significance from the input in subsequent layers. It provides 2.88 TOPS computing power and 8.3W power consumption by considering all memory access under 40nm. Experimental results shows that for GPT2-Medium, it can achieve about 35.86 tokens/s in decode stage. TF-MVP [153] quantitatively analyzes sparsity patterns of pruned-transformer models with the cutting-edge fine-grained pruning scheme for the first time and presents the mixed-length vector pruning (MVP) procedure by utilizing this direction strength. From hardware perspective, it introduces the TF-MVP architecture, a sparsity-aware cost-efficient accelerator design dedicated to the proposed pruned-transformer models. Implemented in a 28nm CMOS technology at 400MHz, TF-MVP provides 0.835 TOPS with 1.721W on-chip power consumption for accelerating GPT-2 small model. SOFA [154] predicts attention sparsity by using log-based add-only operations to avoid the significant overhead of prediction. Then, a distributed sorting and a sorted updating FlashAttention mechanism are proposed with a cross-stage coordinated tiling principle, which enables fine-grained and lightweight coordination among stages, helping optimize memory access and latency. SOFA provides 24.423 TOPS computing power and 3.4W power consumption under 28nm. For Llama2-7B, compared to NVIDIA A100 GPU, it achieves 9.5× inference speedup in prefill stage.

3.2.6 PIM/NDP. **Weight Sparsity.** LauWS [155] proposes an unstructured sparsity method for NDP systems and is evaluated on a practical GDDR6-based bank NDP. Not only the overall features of the total matrix but the features of the local region are preserved by ignoring non-feature values as much as possible, which is beneficial to the trade-off between high sparsity and least accuracy loss. Compared to dense models, it achieves 1.23× and 1.24× speedup for GPT-2 small model and OPT-125M at 50% sparsity, respectively. Sharda et al. [133] propose to use the capacitorless 3D stackable DRAM to store much larger LLMs compared to conventional DRAM at higher density. To reduce the intermediate data size, they propose to use a layer-wise sparsity-quantization hybrid (LSQH) algorithm, which induces sparsity based on calculations performed using low-bit quantization to reduce both the energy consumption and the data storage requirements. Finally, a 3D heterogeneously integrated accelerator is designed by stacking a 3D DRAM with logic dies designed in the 3nm technology node at 1GHz. The evaluation shows that for Llama2-13B, it achieves 163k tokens/s in prefill stage with 193W power consumption.

Attention Sparsity. HARDSEA [156] proposes an attention sparsity method by predicting lightweight token relevance and design a hybrid analog-ReRAM and digital-SRAM in-memory computing accelerator. It employs ReRAM-CIM, whose precision is sensitive to circuit non-idealities, to take charge of token relevance prediction where only computing monotonicity is demanded. The SRAM-CIM, utilized for exact sparse attention computing, is reorganized as an on-memory-boundary computing scheme, thus adapting to irregular sparsity patterns. Experimental results show that HARDSEA prunes BERT and GPT-2 small model to 20% sparsity without accuracy loss, achieving 5.8×–6.7× speedup over NVIDIA RTX 3090 GPU.

3.2.7 Comparison. In Figure 6, for sparsity, power consumption ranges from 30W to 400W, with inference speeds between 8.67 tokens/s and 92.5 tokens/s. The energy efficiency ranges from 0.069 tokens/J to 1.421 tokens/J. TF-MVP [153] (ASIC) achieves the lowest power consumption with 30.0167W and FlightLLM [124] (FPGA) achieves the highest throughput with batch size 1 (though it also applies quantization). And TF-MVP (ASIC) achieves the highest energy efficiency with 1.421 tokens/J.

- For CPUs, power consumption ranges from 55W to 125W, with inference speeds between 8.67 tokens/s and 16.3 tokens/s, located in the bottom part of the figure. The energy efficiency ranges from 0.069 tokens/J to 0.158 token/J, which is much lower than FPGAs and ASICs. Currently, no edge-side CPUs have adopted sparsity methods to accelerate LLM inference.
- For GPUs, power consumption ranges from 70W to 400W, with inference speeds between 20.83 tokens/s and 50 tokens/s, situated in the middle right part of the figure. The energy efficiency ranges from 0.069 tokens/J to 0.434 token/J. Compared to CPUs, GPUs can achieve higher absolute inference speeds due to their high computing power and high bandwidth. However, the energy efficiency difference between CPUs is not significant.
- For FPGAs, power consumption ranges from 45W to 155W, with inference speeds between 55 tokens/s and 92.5 tokens/s, also in the upper right part of the figure. The energy efficiency ranges from over 0.597 tokens/J to 1.32 tokens/J, which is much higher than GPUs and CPUs.

3.3 Fast Decoding

3.3.1 Overview. Traditional autoregressive decoding typically generates text token by token, choosing only the highest probability token at each step, known as greedy sampling. While this approach is simple and easy to implement, it may lead to a lack of diversity and creativity in the generated results. Another autoregressive method called nucleus sampling (or top-p sampling) [220] considers multiple candidates during generation by setting a cumulative probability threshold p, allowing for sampling within a certain range. Although this method offers more diversity than

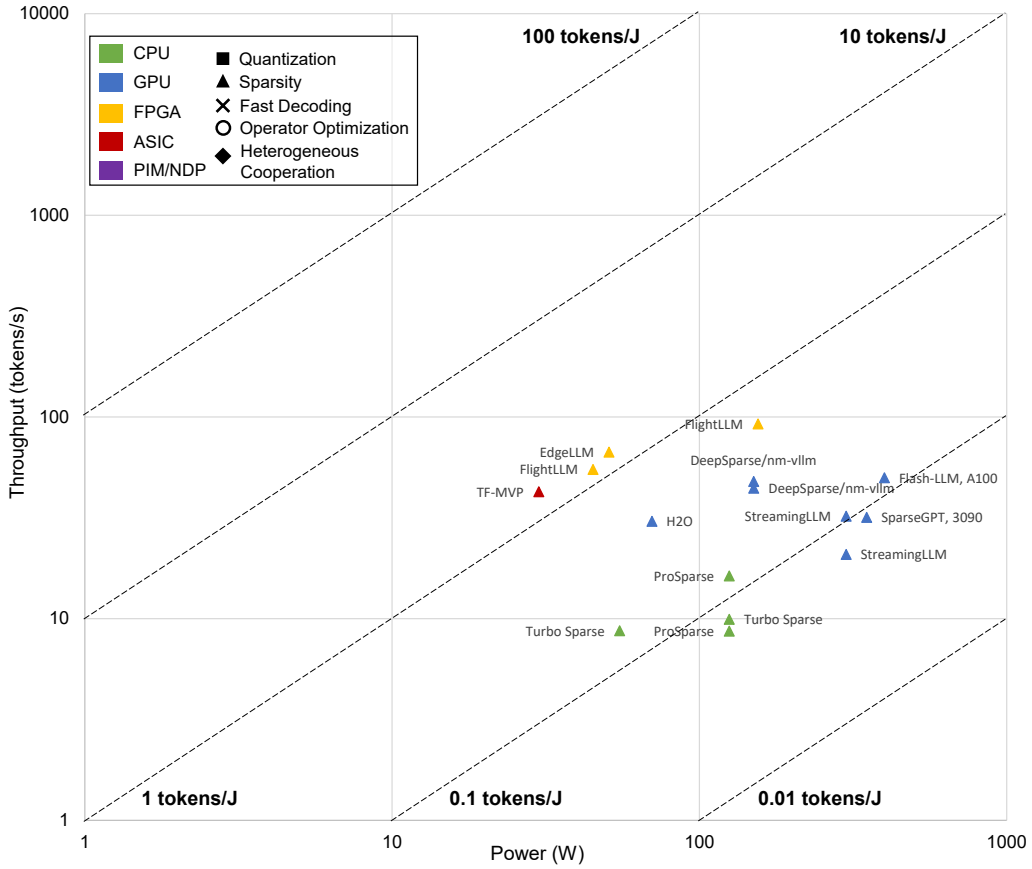

Fig. 6. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with sparsity.

greedy sampling, it still operates in a step-by-step generation manner. Currently, fast decoding techniques can be mainly divided into two categories: speculative decoding and skip layer.

Speculative Decoding. Speculative decoding is a technique for enhancing the generation efficiency of large language models (LLMs). Its core principle lies in using a draft model to quickly generate candidate outputs, which are then evaluated in depth by a main model, thereby accelerating the text generation process. In the implementation of speculative decoding, a smaller draft model is first used to quickly generate multiple candidate words. This model can evaluate the context in a short time and propose various possible output options. Subsequently, the main model performs parallel evaluations of these candidates, calculating their probabilities or scores, and ultimately selects the candidate with the highest score for actual generation. Through this approach, speculative decoding combines speed and accuracy, significantly reducing the computation time while maintaining the quality of the generated text. Common choices for draft models include: one option is to directly use a specific layer from the Transformer model, leveraging the existing architecture to maintain a certain level of feature extraction capability while accelerating inference speed; another option is to train a separate small model, which typically has fewer parameters and a simpler structure, focusing on rapidly generating candidate words and optimizing performance for specific tasks. Both methods have their advantages, and the choice can be made based on specific application needs, quality requirements, and computational resource constraints.

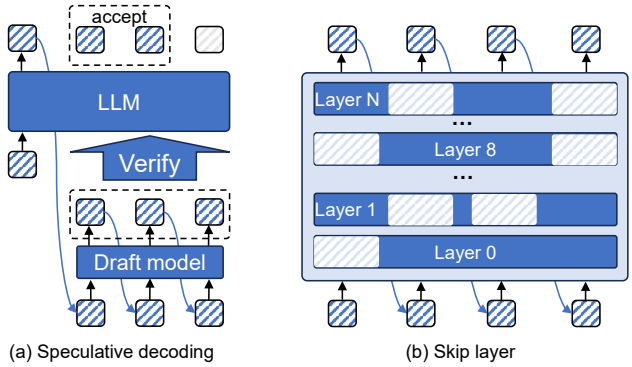

Fig. 7. Fast decoding.

Skip Layer. The working principle of skip layer technology is to dynamically and selectively skip certain layers during the model inference process, thereby reducing computational load and increasing generation speed. In practical implementation, the model evaluates the importance of each layer for the current task while processing input, and decides whether to execute the computation of specific layers based on preset heuristic rules or learned strategies. In this method, the model typically consists of multiple layers, such as self-attention and feedforward layers in a Transformer. During inference, when the input features are relatively simple or the context complexity is low, the model can choose to skip the computation of certain intermediate layers. This choice may be based on real-time assessments, allowing the model to dynamically adjust its computation path according to different inputs. To effectively implement skip layer, the model often requires optimization during the training phase, learning when to skip which layers. This can be achieved through methods such as policy gradients or reinforcement learning, enabling the model to adapt flexibly across various tasks. By skipping unnecessary layers, skip layer technology can significantly accelerate inference speed while reducing computational resource consumption, making it particularly suitable for real-time systems and resource-constrained environments. Overall, this method enhances the efficiency and applicability of large language models by intelligently selecting the computation process.

Table 5. Fast decoding on CPU, GPU, FPGA, ASIC, and PIM/NDP

Hardware	Speculative Decoding	Skip Layer
CPU	✗	✗
GPU	✓	✓
FPGA	✗	✗
ASIC	✓	✗
PIM/NDP	✓	✗

3.3.2 GPU. **Speculative Decoding.** Speculative decoding [158] is proposed to overcome the inherently sequential process in the autoregressive decoding of LLM. The essential decoding mechanism is to make predictions (*i.e.*, draft tokens) parallelly for multiple time steps and then select the longest prefix verified by a scoring model as the final output. Lookahead decoding [159] adopts the *Guess-and-Verify* paradigm as the whole decoding mechanism which generates the draft tokens by *n*-gram method and verifies the draft tokens during the forward at the same time. For Llama-7B and Llama-13B models with different datasets, it achieves 65.12-94.51 tokens/s and 56.01-90.05

tokens/s on a single NVIDIA A100-80GB GPU. To improve the acceptance rate while maintaining generation quality, many works focus on the scheme of generating draft tokens. Medusa [160] adds extra decoding heads to generate multiple subsequent tokens in parallel and uses a tree-based attention mechanism to construct multiple candidate continuations and verify them simultaneously in each decoding step. For Vicuna-7B and Vicuna-13B, it achieves 129.86 tokens/s and 98.54 tokens/s on a single NVIDIA A100-80GB GPU. EAGLE [161, 162] selects a single transformer layer with the same configuration as LLM as the draft model to make predictions autoregressively and combines the feature and token embedding as the input of the draft model. During the verification phase on the target model, EAGLE chooses the tree-based attention mechanism similar to Lookahead and Medusa to ensure the correct relationship between the draft tokens. For Vicuna-7B, Vicuna-13B, Llama2-Chat-7B, and Llama-2-Chat-13B models, it achieves 139.95 tokens/s, 132.31 tokens/s, 133.98 tokens/s, and 156.80 tokens/s on a single NVIDIA A100-80GB GPU. Based on the basic paradigm of speculative decoding for prediction by draft models and verification by target models, some studies explore the optimizations on the modules in it. Ouroboros [163] constructs a phrase candidate pool from the verification process of LLMs to provide candidates for the draft token generation of the draft model. Different from Medusa and EAGLE, Ouroboros uses the smaller LLMs (*e.g.*, DeepSeek-7B) as the draft models for the target models (*e.g.*, DeepSeek-34B). For Yi-34B, DeepSeek-34B and CodeLlama-34B models, it achieves 61.20 tokens/s, 41.00 tokens/s and 39.2 tokens/s on a single NVIDIA A100-80GB GPU. During the prediction phase, Sequoia [164] introduces a dynamic programming algorithm and a hardware-aware tree optimizer to find the optimal tree structure based on the runtime features and the given hardware platform. During the verification phase, Sequoia uses a novel sampling and verification method that outperforms prior work across different decoding temperatures. For Llama2-7B and Llama2-13B models, it achieves 169.68 tokens/s and 149.20 tokens/s on a single NVIDIA A100-80GB GPU.

The studies mentioned above all require the help of an auxiliary model (*e.g.*, a single transformer layer in EAGLE, several Medusa heads in Medusa, or a smaller LLM like Llama2-7B in Ouroboros) or the statistical methods (*e.g.*, *n*-gram in Lookahead) to generate the predicted tokens and then the target model is utilized to verify the predicted tokens. Some other studies [165–167] also explore the LLM inference acceleration without the need of auxiliary models, called self-speculative decoding. Draft&Verify [165] generates draft tokens by selectively skipping certain intermediate layers of LLMs. Subsequently, the draft tokens will be verified in one forward pass. For the Llama2-13B model, it achieves 62.23 tokens/s on a single NVIDIA A100-40GB GPU. Kangaroo [166] adopts a fixed shallow sub-network of the LLM as the draft model, with the remaining layers serving as the target model. To enhance the representation ability of the draft model (*i.e.*, the shallow sub-network), it trains an adapter module to follow the sub-network. For Vicuna-7B and Vicuna-13B models, it achieves 138.14 tokens/s and 105.89 tokens/s on a single NVIDIA A100-80GB GPU. LayerSkip [167] proposes to exit at early layers and verify and correct with remaining layers of the model. During training, it applies layer dropout with low dropout rates for earlier layers and higher dropout rates for later layers, and adds an early exit loss to increase the accuracy of early exit at earlier layers. For the Llama2-13B model, it achieves 66.37 tokens/s on a single NVIDIA H100 GPU. LLMA [157] is motivated by the observation that there are abundant identical text spans between the decoding result by an LLM and the reference that is available in many real-world scenarios (e.g., retrieved documents). LLMA first selects a text span from the reference and copies its tokens to the decoder and then efficiently checks the tokens' appropriateness as the decoding result in parallel within one decoding step. For Llama-7B and Llama-13B models, it achieves 59.2 tokens/s and 41.1 tokens/s on a single NVIDIA V100 GPU, respectively.

Skip Layer. The skip layer method [168–170] is proposed based on the idea that not all layers of LLMs are necessary during inference. AdaInfer [168] statistically analyzes the activated layers

across tasks and proposes a simple algorithm to determine the inference termination moment based on the input instance adaptively. Due to the introduction of the overhead in each layer, which is not friendly to the decoding phase, AdaInfer only takes some Benchmarks for the Q&A tasks and achieves 25.2 tokens/s for Llama2-7B on a single NVIDIA V100 GPU. Based on the basic dataflow of Adainfer, RAEE [169] proposes to build the retrieval database to store the token information offline and leverages the information of the retrieved similar token by searching the pre-built retrieval database to guide the backbone model to exit at the layer. For the LM-BFF which is finetuned based on RoBERT-large-350M, RAEE achieves 26.23 tokens/s. The studies mentioned above both use continuous shallow layers for inference, called early exiting. MOD [170] decides whether to skip the current layer or not by pretraining the model to add a router in each layer like Mixture-of-Experts. This achieves a dynamic selection of partial layers for computation instead of forcing continuous layers. The model of MOD is not open source and no corresponding results on throughput are given in the paper.

3.3.3 ASIC. **Speculative Decoding.** C-Transformer [171] adopts a big-little network, which is composed of the original GPT-2 big model and a 1/10× smaller model, and a reconfigurable homogeneous architecture to increase hardware utilization and energy efficiency. During inference, only the little model computation is performed, and if the prediction probability of a specific token is over a predefined threshold, the big model computation is skipped leading to memory access reduction. Then, workloads are divided into two domains: adder-based spike domain which is efficient for small input values, and multiplier-based non-spike domain which is efficient for large input values. It has 1.431W power consumption and is fabricated in 28nm CMOS technology. For GPT-2 large model, C-Transformer achieves 2146.75 tokens/s in prefill stage.

3.3.4 PIM/NDP. **Speculative Decoding.** SpecPIM [172] aims to accelerate speculative inference on the PIM-enabled system by extensively exploring the heterogeneity brought by both the algorithm and the architecture. It constructs the architecture design space to satisfy each model's disparate resource demands and dedicates the dataflow design space to fully utilize the system's hardware resources. Based on the co-design space, it also proposes a design space exploration framework to provide the optimal design under different target scenarios. Compared with speculative inference on GPUs and existing PIM-based LLM accelerators, SpecPIM achieves 1.52×/2.02× geomean speedup and 6.67×/2.68× geomean higher energy efficiency.

3.3.5 Comparison. In Figure 8, for fast decoding, power consumption ranges from 17.499W to 700W, with inference speeds between 24.7 tokens/s and 174.018 tokens/s. The energy efficiency ranges from 0.082 tokens/J to 9.944 tokens/J. SpecPIM [172] (PIM/NDP) achieves the lowest power consumption with 17.499W, the highest throughput with batch size 1, and the highest energy efficiency.

- For GPUs, power consumption ranges from 267W to 700W, with inference speeds between 24.7 tokens/s and 169.68 tokens/s. The energy efficiency ranges from 0.082 tokens/J to 0.587 tokens/J.
- For PIM/NDPs, SpecPIM is the only one using speculative decoding, which outperforms GPU platforms.

3.4 Operator Optimization

3.4.1 Overview. Improving the execution efficiency of operators is crucial for LLM eras, which not only involves enhancing computational speed but also maximizing resource utilization. As the scale and complexity of models continue to increase, traditional operator execution methods become increasingly inefficient, prompting the need to explore various optimization strategies.

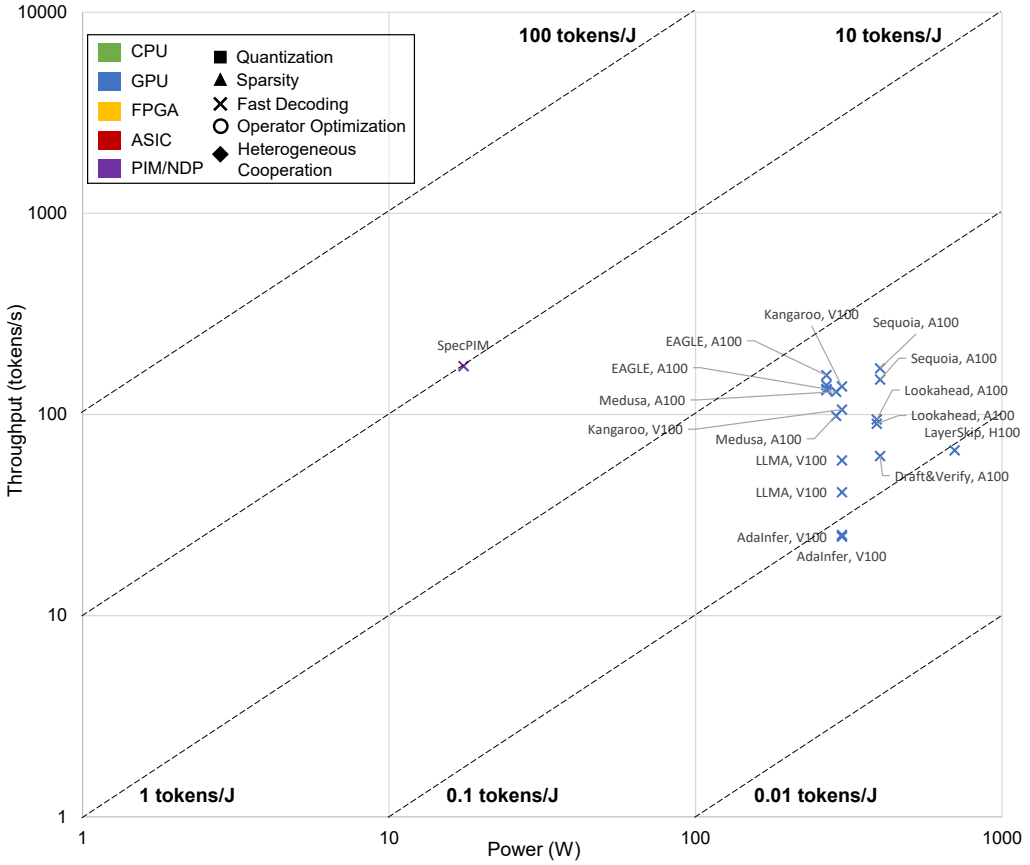

Fig. 8. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with fast decoding.

The following four methods provide effective solutions for operator optimization, significantly enhancing the performance and responsiveness of models across different hardware platforms. **Fusion.** Operator fusion reduces the storage and transmission needs of intermediate data by merging multiple independent operators into a single entity. This approach not only lowers I/O overhead but also reduces redundant computations during the execution process, thereby improving overall efficiency. Operator fusion enables hardware to utilize cache and bandwidth more effectively, significantly boosting computational performance. **Nonlinear Function Approximation.** Common nonlinear activation functions in deep learning models often require specialized hardware support, which can occupy substantial chip resources. By employing linear approximations, we can achieve computations using less expensive hardware while maintaining algorithmic accuracy. This optimization strategy makes complex nonlinear calculations more efficient, making it suitable for resource-constrained environments. **Coarse-grained Processing.** When handling large-scale matrix operations, fine-grained computational units may lead to frequent resource scheduling and contention, introducing additional overhead. Coarse-grained processing simplifies the scheduling process by merging multiple small computational units into larger ones, reducing resource contention. This method effectively enhances computational coherence and efficiency, particularly in parallel computing environments. **Storage Optimization.** Storage optimization strategies focus on the arrangement of data in memory, aiming to minimize latency caused by memory access

during computation. By strategically organizing data storage locations and access patterns, we can significantly improve data access efficiency and enhance overall computational performance. This optimization is closely related not only to hardware performance but also to algorithm design.

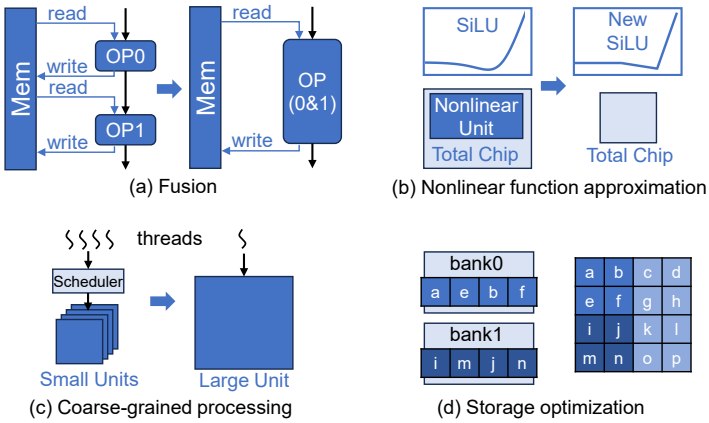

Fig. 9. Operator optimizations.

Table 6 shows the usage of these four operator optimization methods across different hardware platforms. By comprehensively applying these four optimization methods, the efficiency of operator execution can be significantly improved, allowing deep learning models to handle complex tasks more effectively. Selecting and combining these optimization strategies appropriately for specific application scenarios and hardware platforms will be key to achieving high-performance computing.

Table 6. Operator optimization on CPU, GPU, FPGA, ASIC, and PIM/NDP

Hardware	Fusion	Nonlinear Function Approximation	Coarse-grained Processing	Storage Optimization
CPU	✗	✗	✗	✗
GPU	✓	✗	✗	✗
FPGA	✗	✗	✗	✗
ASIC	✓	✓	✓	✗
PIM/NDP	✗	✓	✓	✓

3.4.2 GPU. **Fusion.** To address the quadratic memory requirements in the attention computation, FlashAttention [173, 174] fuses the attention operation into a single operator by tiling input matrices (Q, K, V) and the attention matrix into blocks. Based on FlashAttention, FlashDecoding [175] proposes additionally the parallel computation along the feature dimension, improving performance for small batch size during the decoding phase. FlashDecoding achieves 95.07 tokens/s and 54.19 tokens/s for Llama2-7B on a single NVIDIA A100 and RTX 3090 GPU respectively. Subsequently, FlashDecoding++ [176] optimizes the synchronization overhead in softmax computation by pre-determining a unified maximum based on statistical analysis in advance, enabling the parallel execution of subsequent operations and improving efficiency in typical LLMs like Llama2 and ChatGLM [33]. FlashDecoding++ achieves 115.57 tokens/s and 61.66 tokens/s for Llama2-7B on a single NVIDIA A100 and RTX 3090 GPU respectively. The linear operator is widely used in deep learning. In the common framework for deep learning (*e.g.*, Pytorch), the backend of the linear operator is usually the General Matrix-Matrix Multiplication (GEMM) operation supported by NVIDIA. The naive implementation by HuggingFace achieves 44.60 tokens/s and 36.29 tokens/s

for Llama2-7B on a single NVIDIA A100 and RTX 3090 GPU respectively. In the LLM framework (*e.g.*, DeepSpeed [177], vLLM [178], and OpenPPL [179]), the GEMM implementation provided by cuBLAS [180] APIs is generally optimized. For the Llama2-7B model on a single NVIDIA A100 GPU, they achieve 89.24 tokens/s, 88.45 tokens/s and 93.63 tokens/s. For the Llama2-7B model on a single NVIDIA RTX 3090 GPU, they achieve 51.28 tokens/s, 53.31 tokens/s and 55.99 tokens/s. To address the inefficiency of GEMM during decoding due to the reduced dimensions, TensorRT-LLM [181] introduces a dedicated General Matrix-Vector Multiplication (GEMV) implementation to support the decoding phase of LLM when batch size equals 1. TensorRT-LLM achieves 98.19 tokens/s for Llama2-7B on a single NVIDIA A100 GPU. For the smaller batch size during the decoding phase, FlashDecoding++ [176] introduces FlatGEMM to address the inefficiencies in cuBLAS [180] and CUTLASS [182] libraries and employs fine-grained tiling and double buffering techniques to improve parallelism and reduce the latency of memory access. Moreover, it adopts a heuristic selection mechanism to dynamically select the most efficient operator based on the input. The performance of FlashDecoding++ is shown above. Operator fusion [173, 176, 177, 183] is a common and effective optimization to reduce the runtime memory access, eliminate kernel launching overhead and enhance parallelism. ByteTransformer [183] and DeepSpeed [177] fuse the main operator (*e.g.*, GEMM) and the lightweight operators(*e.g.*, residual adding, layernorm and activation functions) into a single kernel to reduce the kernel launching overhead. FlashAttention [173] mentioned above fuse the attention operator into one single kernel, reducing significantly the overhead of memory access and the memory requirements. FlashDecoding++ [176] also achieves the integration of seven fused kernels in a transformer block.

3.4.3 ASIC. **Fusion.** In 2020, Groq company introduces the Tensor Streaming Processor (TSP) architecture [221], a functionally-sliced microarchitecture with memory units interleaved with vector and matrix compute units in order to take advantage of dataflow locality. The first TSP implementation yields a computational density of more than 1 TOPS/mm^2 for its 25×29 mm 14nm chip at 900 MHz. In 2022, Groq company introduces a novel software-defined communication approach for large-scale interconnection networks of TSP elements [222]. This scalable communication fabric is based on a software-defined dragonfly topology, ultimately yielding a parallel machine learning system with elasticity to support a variety of workloads. Each TSP contributes 220 MB to the global memory capacity, with the maximum capacity limited only by the network's scale. The large-scale parallel system achieves up to 10,440 TSPs and more than 2 TB of global memory accessible in less than 3ms of end-to-end system latency. Based on two previous works, Groq Language Processing Unit (LPU) [76] is fabricated in 14nm with 185W power consumption. According to [223], LPU achieves 814 tokens/s for Gemma-7B model. Another commercial company, HyperAccel, also proposes a LLM inference chip with dataflow architecture. Latency Processing Unit (LPU) [184] introduces streamlined hardware that maximizes the effective memory bandwidth usage during end-to-end inference regardless of the model size to achieve up to 90% bandwidth utilization for high-speed text generation. It consists of expandable synchronization link (ESL) that hides bulk of the data synchronization latency in a multi-device system to achieve nearperfect scalability. Its on-chip power is 0.284W sythesised by Samsung 4nm and the total system power is 86W with 96GB HBM3. For OPT-1.3B/6.7B/13B/30B, LPU achieves 769.23 tokens/s, 217.39 tokens/s, 112.40 tokens/s and 49.26 tokens/s, respectively. The Wafer-Scale Engine (WSE-3) [77], which powers the Cerebras CS-3 system, is the largest chip ever built. The WSE-3 is 57× larger than the largest GPU, has 52× more compute cores, and 880× more high performance on-chip memory. The only wafer scale processor ever produced, it contains 4 trillion transistors, 900,000 AI-optimized cores, and 44 gigabytes of high performance on-wafer memory. Each wafer consists of 84 dies with

40GB on-chip memory and 97W power consumption. For Llama3.1-8B, it can achieve about 1800 tokens/s, which is 20× faster than hyperscale clouds.

Nonlinear Function Approximation. Constant Softmax (ConSmax) [185] is a software-hardware co-design as an efficient Softmax alternative. ConSmax employs differentiable normalization parameters to remove the maximum searching and denominator summation in softmax. It allows for massive parallelization while performing the critical tasks of softmax. In addition, a scalable ConSmax hardware utilizing a bitwidth-split LUT can produce lossless non-linear operation and support mix-precision computing. It further facilitates efficient LLM inference. Experimental results show that ConSmax achieves a minuscule power consumption of 0.43mW and area of $0.001mm^2$ at 1GHz working frequency and 22nm CMOS technology. MARCA [186] is the first accelerator with reconfigurable architecture tailored for Mamba model. It proposes a reduction alternative PE array architecture for both linear and element-wise operations. For linear operations, the reduction tree connected to PE arrays is enabled and executes the reduction operation. For element-wise operations, the reduction tree is disabled and the output bypasses. And it proposes a reusable nonlinear function unit based on the reconfigurable PE and decomposes the exponential function and activation function (SiLU) into element-wise operations to reuse the reconfigurable PEs. Extensive experiments show that MARCA can achieve 23.78 tokens/s with batch size 1 for Mamba-2.8B model with 10.33W power consumption (11.89 tokens/s for ∼ 7B model).

Coarse-grained Processing. Tensor Contraction Processor (TCP) [187], is composed of coarse-grained PEs, which are designed to be flexible enough to be configured as a large-scale single unit or a set of small independent compute units. TCP chip is designed and fabricated in 5nm technology with 256MB SRAM and 1.5 TB/s 48GB HBM3 under 150W. For Llama2-7B model, TCP achieves about 125 tokens/s with batch size 1 and 1176 tokens/s with batch size 8. Habana Gaudi [188] and Gaudi2 [189] consists of two main compute engines, Matrix Multiplication Engine (MME) and Tensor Processor Core (TPC) cluster. The TPC unit is a SIMD processor tailor-made for general deep learning operations. The biggest difference between GPU and Gaudi architecture is the size of MME. Gaudi architecture can handle 256×256 matrix multiplication, which requires 512 input elements per cycle while GPU architecture with 16×16 Tensor Core requires 8K input elements. Therefore, Gaudi architecture can save 16× less read bandwidth requirements. Besides compute engine, Gaudi2 includes 96 GB of HBM2E memories delivering 2.45 TB/sec bandwidth, in addition to a 48 MB of local SRAM. Gaudi is fabricated in 16nm and Gaudi2 is fabricated in 7nm technique node with 600W power consumption. For Llama2-7B and Llama2-13B models, Gaudi2 achieves from 81.97 to 111.11 tokens/s and from 49.02 to 64.52 tokens/s with batch size 1, respectively.

3.4.4 PIM/NDP. **Fusion.** Modern DRAMs have multiple banks to serve multiple memory requests in parallel. However, when two requests go to the same bank, they have to be served serially, exacerbating the high latency of off-chip memory. Therefore, Kim et al. [224] propose Subarray-Level Parallelism (SALP) to overlap the latencies of different requests that go to the same bank. Based on SALP, SAL-PIM [194] proposes a subarray-level processing-in-memory architecture includes three architectural features. Two types of ALUs (SALU and C-ALU) are integrated into the subarray-level and the channel-level, respectively. S-ALU utilizes higher bandwidth than bank-level PIM to compute memory bound operation, and C-ALU supports accumulation and reduce-sum operation for multiple banks. The SAL-PIM architecture is implemented by HBM2 8GB in 28nm CMOS technology with 68.973W power consumption. As a result, when the input size is from 32 to 128 and the output size is from 1 to 256, SAL-PIM achieves average 1.83× inference speedup for the text generation based on the GPT-2 medium model (345M) compared to the NVIDIA Titan RTX GPU. PipePIM [195] introduces pipelining and dual buffering to maximize CU utilization in a digital PIM. PipePIM consists of two primary schemes: subarray-level pipelining (SAPI) and a dual

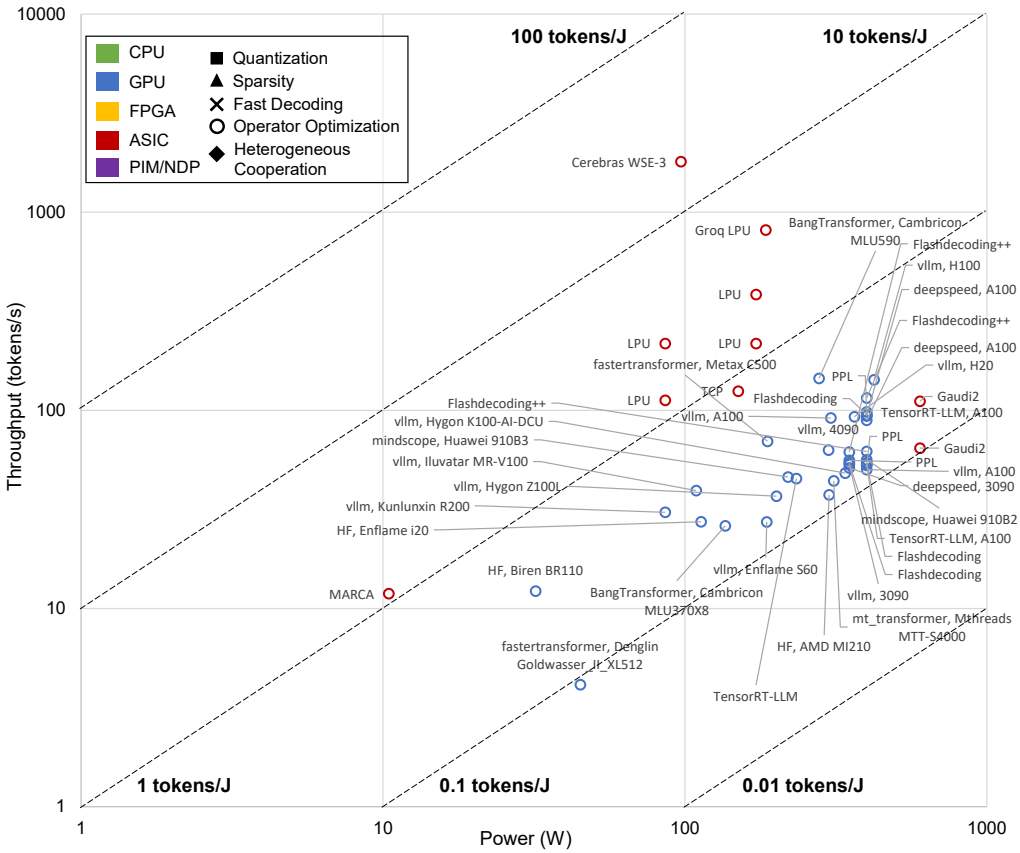

Fig. 10. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with operator optimization.

vector buffer. The key ideas are to process activation, computation, and precharging on different subarrays in a pipelined manner by SAPI and concurrently perform buffer writes and computation by a dual vector buffer. It is simulated in 22nm CMOS technology with $88.82mm^2$ and $86.36mm^2$ area. For LLMs like LLaMA and Mistral, the results show 1.2× and 1.14× speedup in Newton while 1.21× and 1.15× speedup in HBM-PIM [225].

Nonlinear Function Approximation. AttentionLego [192] proposes a PIM-based matrix-vector multiplication and look-up table-based Softmax design. PIM-GPT [193], which achieves end-to-end acceleration of GPT inference with high performance and high energy efficiency. At the hardware level, PIM-GPT is a hybrid system that includes DRAM-based PIM chips to accelerate VMM near data and an application-specific integrated circuit (ASIC) to support other functions that are too expensive for PIM including necessary nonlinear functions, data communication, and instructions for the PIM chips. At the software level, the mapping scheme is optimized to efficiently support the GPT dataflow. Overall, PIM-GPT achieves 89× speedup and 221× energy efficiency over NVIDIA T4 GPU on 8 GPT models with up to 1.4 billion parameters. SAL-PIM [194] also adopts LUT-based linear interpolation to perform complex nonlinear functions in PIM.

Storage Optimization. By observing that a key impediment to truly harness PIM acceleration is deducing optimal data-placement to place the matrix in memory banks, PIMnast [191] proposes matrix tiling/ordering algorithms to tackle these factors and identify orchestration knobs that

impact PIM acceleration. For OPT-6.7B model, compared to the SoC baseline (AMD Ryzen PRO 7040 Series processors comprising eight CPU cores, 12 compute units of GPU cores, 16 AIE tiles, and eight channels of LPDDR5 memory for a peak memory bandwidth of 120 GB/s), PIMnast achieves 4.5× speedup for per-token latency.

3.4.5 Comparison. In Figure 10, for operator optimization, power consumption ranges from 10.44W to 600W, with inference speeds between 4.13 tokens/s and 1800 tokens/s. The energy efficiency ranges from 0.092 tokens/J to 18.556 tokens/J. MARCA [186] (ASIC) achieves the lowest power consumption with 10.44W. Cerebras WSE-3 [77] (ASIC) achieves the highest throughput with batch size 1 (post-silicon results) and the highest energy efficiency.

- For GPUs, power consumption ranges from 32W to 400W, with inference speeds between 4.13 tokens/s and 145.04 tokens/s. The energy efficiency ranges from 0.092 tokens/J to 0.522 tokens/J.
- For ASICs, power consumption ranges from 10.44W to 600W, with inference speeds between 11.89 tokens/s and 1800 tokens/s, found in the upper left section of the figure. The energy efficiency ranges from 0.108 tokens/J to over 18.556 tokens/J, outperforming the GPU platform.

3.5 Heterogeneous Cooperation

3.5.1 Overview. Heterogeneous cooperation combines different types of computing platforms, such as CPUs, GPUs, FPGAs, and PIM/NDPs, to optimize system performance, energy efficiency, and flexibility. Each computing unit has unique strengths for specific tasks; for example, GPUs excel at parallel processing, FPGAs offer customizable hardware acceleration, and PIM/NDPs are specialized for memory-bound operations. By distributing tasks to the most suitable hardware, heterogeneous cooperation enhances computing efficiency, reduces power consumption, and lowers latency. It can be broadly categorized into two types, computing enhancement and memory enhancement.

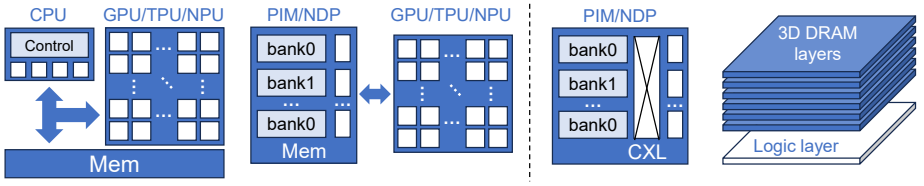

Fig. 11. Heterogeneous cooperation.

Computing Enhancement. Commonly, CPUs and PIMs may struggle when handling large-scale computation tasks. For instance, while CPUs offer versatility and flexibility, they are not well-suited for highly parallel processing. PIM reduces data transfer by performing computations within memory, but its computational power is limited. In such scenarios, powerful parallel computing hardware such as GPUs, NPUs, or TPUs are introduced to assist. In LLM inference, computation-intensive computations such as attention calculations are placed on GPUs, NPUs, or TPUs, while other computations can be handled by CPUs or PIMs.

Memory Enhancement. focuses on two key aspects: storage capacity enhancement and bandwidth enhancement. As we move into the LLM eras, computational tasks are becoming increasingly complex, leading to greater demands on memory capacity and bandwidth. From the perspective of storage capacity, memory enhancement is achieved by integrating more on-chip memory by using 3D storage stacks on the same chip area. This allows more data and models to be totally stored in a limited on-chip space, reducing the need to access external storage and improving overall efficiency. Another memory enhancement is to accelerate memory-bound computations by

increasing data transfer speeds. For heterogeneous cooperation, bandwidth often restricts computational performance. To address this issue, much methods are proposed to support high-speed on-chip [226–228] and chip-to-chip interconnect, such as Compute Express Link (CXL) [229] and NVLink [62]. These techniques enable low-latency, high-bandwidth data transmission between different chips, significantly speeding up data exchange processes.

Table 7. Heterogeneous cooperation on CPU, GPU, FPGA, ASIC, and PIM/NDP

Hardware	Computing Enhancement	Memory Enhancement
CPU	✓	✗
GPU	✗	✗
FPGA	✗	✗
ASIC	✗	✗
PIM/NDP	✓	✓

3.5.2 CPU. **Computing Enhancement. (1) CPU-GPU.** Only using the CPU may result in slower performance, so many methods employ a combination of CPU and GPU to enhance LLM inference speed. For personal computers, PowerInfer [197] proposes that the hot-activated neurons should be preloaded onto the GPU for fast access, while cold-activated neurons are computed on the CPU, thus significantly reducing GPU memory demands and CPU-GPU data transfers. PowerInfer further integrates adaptive predictors and neuron-aware sparse operators. For various models (OPT-30B/66B, Falcon-40B, and Llama-70B) on a Intel i9-13900K processor equipped with an NVIDIA RTX 4090, it achieves 8.32 tokens/s on average. On a Intel i7-12700K processor equipped with an NVIDIA RTX 2080Ti, it achieves 5.77 tokens/s on average. For smartphones, PowerInfer-2 [198] further leverages the highly heterogeneous XPUs present in smartphone SoCs, such as asymmetric big.LITTLE CPU cores, GPU, and NPU. On OnePlus 12 equipped with SnapDragon 8 Gen 3, PowerInfer-2 achieves 11.7 tokens/s and 10.5 tokens/s on Llama-7B and Mistral-7B, respectively. On OnePlus Ace 2 equipped with SnapDragon 8+ Gen 1, PowerInfer-2 achieves 6.5 tokens/s and 6.3 tokens/s on Llama-7B and Mistral-7B, respectively. For servers, Kim et al. [196] propose an adaptive model to determine the LLM layers to be run on CPU and GPU, respectively, based on the memory capacity requirement and arithmetic intensity. They then propose CPU-GPU cooperative computing that exploits the AMX extensions of the latest Intel CPU. The evaluation demonstrates that for OPT-30B model, CPU-GPU cooperative computing achieves 336 tokens/s with batch size 1 and input tokens 2016 in prefill stage. In decode stage, it achieves about 25 tokens/s with batch size 90 and input tokens 2016, and about 300 tokens/s with batch size 1400 and input tokens 64.

3.5.3 PIM/NDP. **Computing Enhancement. (1) PIM-NPU.** NeuPIMs [199] and IANUS [200] are heterogeneous PIM acceleration systems that jointly exploits a conventional GEMM-focused NPU and GEMV-optimized PIM devices. NeuPIMs [199] first proposes dual row buffers in each bank, facilitating the simultaneous management of memory read/write operations and PIM commands, to enable concurrent operations on both NPU and PIM platforms. Further, it employs a runtime sub-batch interleaving technique to maximize concurrent execution for the inherent dependencies between GEMM and GEMV in LLMs. Our evaluation demonstrates that NeuPIMs with ~76W (0.6348W+75W) power consumption (32GB HBM) achieves ~3k tokens/s with batch size 8 for GPT3-7B. IANUS [200] proposes novel PIM access scheduling that manages not only the scheduling of normal memory accesses and PIM computations but also workload mapping across the PIM and the NPU. The evaluations show that for GPT-6.7B model, IANUS achieves 127.1 tokens/s with about 240W power consumption. Cambricon-LLM [211] proposes a chiplet-based hybrid architecture

with NPU and a dedicated NAND flash chip to enable efficient LLM inference. It utilizes both the high computing capability of NPU and the data capacity of the NAND flash chip. The NAND flash chip is enhanced by in-flash computing to perform precise lightweight on-die processing, and the NPU performs matrix operations and handles special function computations. Experimental results show that Cambricon-LLM with ~37W power consumption achieves 3.44 tokens/s (~) and 36.34 tokens/s for 70B and 7B LLMs, which is over 22× to 45× faster than existing flash-offloading technologies, respectively. **(2) PIM-GPU.** MoNDE [201] is a near-data computing solution that efficiently enables Mixture-of-Experts (MoE) LLM inference on heterogeneous hardwares. MoNDE reduces the volume of MoE parameter movement by transferring only the hot experts to the GPU, while computing the remaining cold experts inside the host memory device. MoNDE can achieve inference latency comparable to an ideal GPU system with infinite memory. AttAcc [202, 203] is also an heterogeneous system equipped with GPUs to accelerate the attention layers. Compared to the monolithic state-of-the-art GPU system, AttAcc achieves significantly higher throughput (up to 2.81×) and energy efficiency (up to 2.67×) for GPT3-175B model. In 2024, SK Hynix proposes LPDDR6-based heterogeneous LLM system called AiMX-xPU[210], which consists 1 NVIDIA H100 GPU and 3 AiMX, achieving 167 tokens/s, 220 tokens/s, and 900 tokens/s with batch size 1, 8, and 32 for OPT-30B. **(3) PIM-FPGA.** Kang et al. [204] put the memory-bound GEMV calculations including projections, feed-forward-network layer, and the last fully-connected layer with pre-trained weight on HBM2-PIM and evaluate system with PIM-powered Xilinx Alveo U280 board. For GPT-1.3B model, after scaling the bandwidth by 8× for PIM technology, it can achieve about 347.83 tokens/s, which is 1.6× faster than the NVIDIA A100 GPU. From 2022 to 2023, SK Hynix proposes GDDR6-based accelerator-in-memory named AiM [230, 231] and integrates it with 2 XLINX XCVU9P FPGA chips for control and communication as a prototype called AiMX [209] to achieve 330 tokens/s for OPT-6.7B with batch size 1. **(4) PIM-TPU.** H3D-Transformer [206] proposes a heterogeneous 3D-based accelerator design for transformer models, which adopts an interposer substrate with multiple 3D memory/logic hybrid cubes optimized for accelerating different MatMul workloads. An approximate computing scheme is proposed to take advantage of heterogeneous computing paradigms of mixed-signal compute-in-memory (CIM) and digital tensor processing units (TPU). From the system-level evaluation results, 10 TOPS/W energy efficiency is achieved for the GPT2 model, which is about 2.6×-3.1× higher than the baseline with 7nm TPU and stacked FeFET memory. 3D-HI [208] leverage chiplet-based heterogeneous integration to design a Network-on-Interposer (NoI) architecture to accelerate LLM inference. 3D-HI uses the streaming multiprocessors along with the associated memory controllers (SM-MCs) for multi-head attention and ReRAM cores for the feed-forward network, which optimize both achievable energy efficiency and throughput. Further, vertical integration on top of a 2.5D interposer helps to enhance overall system performance and alleviates the issue of memory bottlenecks. Experimental results demonstrate that 3D-HI lowers the latency and energy consumption by up to 22.8× and 5.36× with respect to an equivalent state-of-the-art chiplet-based platform.

Memory Enhancement. (1) PIM-CXL. As the frequent transfers of these model parameters and intermediate values are performed over relatively slow device-to-device interconnects such as PCIe or NVLink, they become the key bottleneck for efficient acceleration of LLMs. Kim et al. [205] exploit PIM, which uses bank-level parallelization to provide higher internal memory bandwidth compared to traditional DRAM, resulting in a significant increase in on-DRAM compute bandwidth. In addition to achieving high capacity through Compute eXpress Link (CXL) memory expansion, CXL-PNM demonstrates performance improvements by integrating computational logic into memory products, consequently increasing internal memory bandwidth. Evaluation results show that the performance of memory-bounded LLMs was significantly improved with PIM and PNM by up to 4.5× and 4.4×, respectively. CXL-PNM [207] first devises an LPDDR5X-based CXL memory architecture with

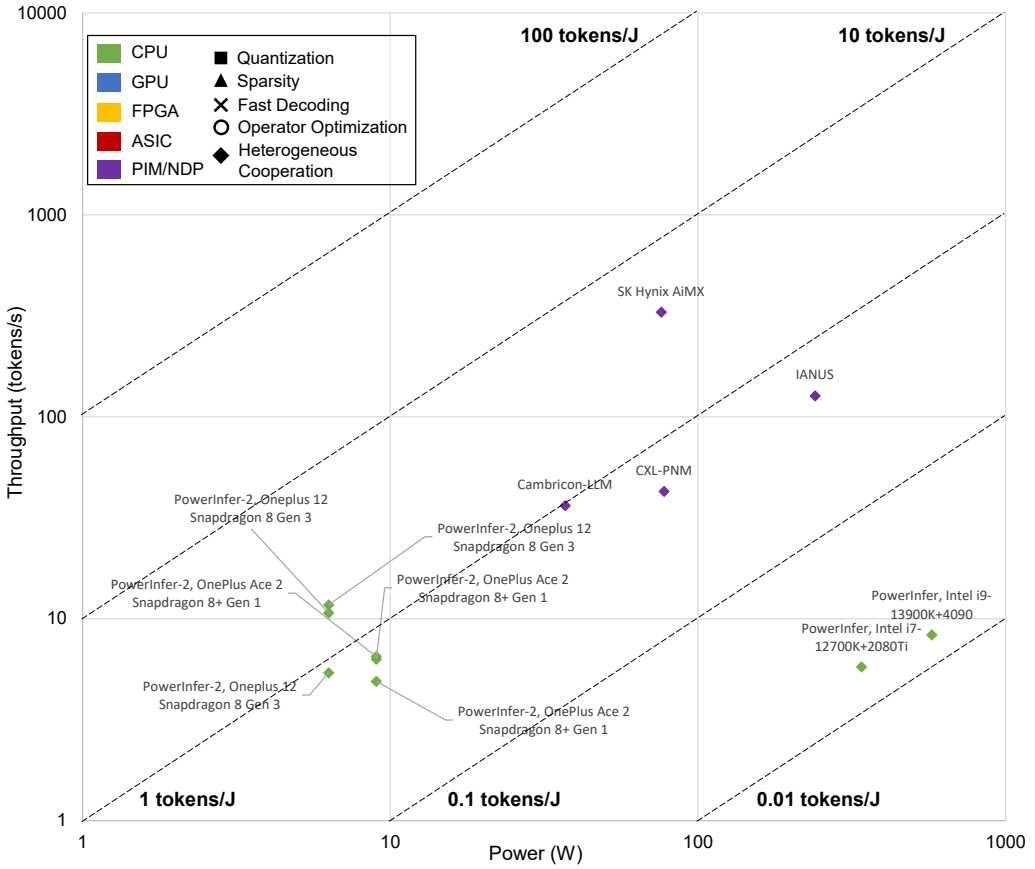

Fig. 12. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with heterogeneous cooperation.

512GB of capacity and 1.1TB/s of bandwidth, which boasts 16× larger capacity and 10× higher bandwidth than GDDR6 and DDR5-based CXL memory architectures, respectively, under a module form-factor constraint. Second, it designs a CXL-PNM controller architecture integrated with an LLM inference accelerator, exploiting the unique capabilities of such CXL memory to overcome the disadvantages of competing technologies such as HBM-PIM and AxDIMM. The evaluation shows that for OPT-13B model, CXL-PNM achieves 42.68 tokens/s with 77.6W power consumption. **(2) PIM-3D Stack.** Sharda et al. [133] propose to use the capacitorless 3D stackable DRAM to store much larger LLMs compared to conventional DRAM at higher density. To reduce the intermediate data size, they propose to use a layer-wise sparsity-quantization hybrid (LSQH) algorithm, which induces sparsity based on calculations performed using low-bit quantization to reduce both the energy consumption and the data storage requirements. Finally, a 3D heterogeneously integrated accelerator is designed by stacking a 3D DRAM with logic dies designed in the 3nm technology node at 1GHz. The evaluation shows that for Llama2-13B, it achieves 163k tokens/s in prefill stage with 193W power consumption.

3.5.4 Comparison. In Figure 12, for heterogeneous cooperation, power consumption ranges from 6.3W to 575W, with inference speeds between 4.9 tokens/s and 330 tokens/s. The energy efficiency ranges from 0.0145 tokens/J to 4.342 tokens/J. PowerInfer-2 [198] with Snapdragon 8 Gen 3 (CPU)

achieves the lowest power consumption with 6.3W. SK Hynix AiMX [209] (PIM/NDP) achieves the highest throughput with batch size 1 and the highest energy efficiency.

- For CPUs, power consumption ranges from 6.3W to 575W, with inference speeds between 4.9 tokens/s and 11.7 tokens/s, situated in the bottom part of the figure. The energy efficiency ranges from 0.0145 tokens/J to 1.857 tokens/J. In the end-side scenario, the CPU can achieve higher energy efficiency.
- For PIM/NDPs, power consumption ranges from 37W to 240W, with inference speeds between 36.34 tokens/s and 330 tokens/s, found in the upper part of the graph. The energy efficiency ranges from 0.55 tokens/J to 4.342 tokens/J, similar with edge-side CPU platforms.

3.6 Homogeneous Cooperation

3.6.1 Overview. Due to the large size and high computational demands of LLMs, homogeneous cooperation can also enhance LLM inference. Distributed computing like model parallelism is aimed at addressing memory limitations associated with LLMs. As model sizes continue to grow, a single hardware unit may not be able to accommodate the entire model. Model parallelism splits the model into multiple parts, with different hardware units processing different segments of the model concurrently.

3.6.2 CPU. Distributed computing emerges as a prevalent strategy to mitigate single-node memory constraints and expedite LLM inference performance. He et al. [212] propose an efficient distributed inference optimization solution for LLMs on CPUs. On four 5th Gen Intel Xeon Scalable Processors the result shows the generation speed on Qwen-72B is 7.14 tokens/s. He et al. [213] also propose new attention flow, SlimAttention, to reduce the KV cache size and ensure precision for efficient LLM inference on CPUs. The experimental results on Llama2-70b shows the latency of token generation is 4 tokens/s with 2 sockets and 11.4 tokens/s with 8 sockets on Intel Xeon 8563C CPUs.

3.6.3 FPGA. DFX [214] is a multi-FPGA acceleration which uses model parallelism and optimized dataflow to improve LLM inference speed in both prefilling and decoding phases. With the implementation on four Xilinx Alveo U280 FPGAs, DFX achieves about 120 tokens/s for GPT2-1.5B model.

4 FURTHER COMPARISON

4.1 Setup

In this section, we compare the performance of state-of-the-art optimization methods on hardware accelerators including CPUs, GPUs, FPGAs, ASICs and PIM/NDPs for generative LLM. In traditional hardware capability comparisons, the focus is usually on peak computational performance and power consumption. However, in the context of generative large models, the emphasis shifts to inference speed (**tokens per second, tokens/s**) and hardware power consumption (**Watt, W**). The slope of the curve representing inference speed on the Y-axis versus hardware power consumption on the X-axis indicates the hardware efficiency in terms of tokens per joule (**tokens/J**). Therefore, we have collected data on each hardware platform along with all optimization methods, highlighting their inference speed (mentioned explicitly or estimated) and actual power consumption while running generative large models ($\sim$ 7 billion parameters) with batch size 1 and 8, as shown in Figure 13 and Figure 14, respectively.

In collecting the data, the sources for absolute inference speed are categorized into five types:

- For most CPU and GPU hardware platforms, the absolute inference speed is typically reported in the literature alongside the specific model being run.

- For GPU platforms with open-source and reproducible code, we directly measure the absolute inference speed through actual execution.
- There is limited work on FPGA hardware platforms. Some papers provide verified absolute inference data through physical testing, while others offer estimates without hardware validation. Consequently, we also include estimated absolute inference speeds for these cases.
- For most ASIC platforms, we first define a post-silicon work [130] with an absolute inference speed as the baseline and scale the results by the peak computational performance (TOPS or GOPS).
- Some works provide acceleration ratios relative to GPUs or other hardware platforms. In these cases, we scale existing data according to the reported acceleration ratios to estimate the absolute inference speed.

And the sources of power consumption are categorized into three types:

- For most CPU, and some GPU and FPGA hardware platforms, the papers typically specify the hardware model and quantity used. We use these specifications to determine the actual power consumption.
- For GPU platforms with open-source and reproducible code, we directly measure the actual power consumption through real-world execution.
- For most ASIC and PIM/NDP platforms, we consider both on-chip and off-chip memory access power consumption. On-chip power consumption is often reported in the literature. Off-chip power consumption is calculated by multiplying the energy per bit of the off-chip memory type by the model parameter count and then by the absolute inference speed.

4.2 Hardware Comparison

4.2.1 Small batch size (bs=1). In Figure 13, for **CPU** platforms, power consumption ranges from 3W to 575W, with inference speeds between 3 tokens/s and 50 tokens/s, located in the bottom part of the figure. The energy efficiency ranges from 0.014 tokens/J to 2.38 token/J. Additionally, we observe edge-side CPUs (including CPU SoCs) with 3W to 6W power consumption exhibit higher energy efficiency (0.544 tokens/J to 2.38 tokens/J) compared to GPUs. For **GPU** platforms, power consumption ranges from 40W to 700W, with inference speeds between 18 tokens/s and 194 tokens/s, situated in the middle right part of the figure. The energy efficiency ranges from 0.067 tokens/J to 0.825 token/J. For **FPGA** platforms, power consumption ranges from 45W to 225W, with inference speeds between 40 tokens/s and 333 tokens/s, also in the middle part of the figure. The energy efficiency ranges from over 0.178 tokens/J to 1.85 tokens/J, which is higher than GPU platforms and similar with edge-side CPUs. For **ASIC** platforms, power consumption ranges from 6.3W to 600W, with inference speeds between 9.173 tokens/s and 1800 tokens/s, found in the upper left part of the figure. The energy efficiency ranges from 0.108 tokens/J to over 18.557 tokens/J, outperforming CPU, GPU and FPGA platforms. For **PIM/NDP** platforms, power consumption ranges from 11.516W to 240W, with inference speeds between 42.68 tokens/s and 1998 tokens/s, located in the upper left part of the figure. The energy efficiency ranges from 0.53 tokens/J to 46.66 tokens/J, outperforming other hardware platforms.

4.2.2 Large batch size (bs=8). Compared to small batch size, the results of inference throughput on large batch size are limited. We can only collect seldom results in Figure 14. For **GPU** platforms, power consumption ranges from 45W to 464W, with inference speeds between 46 tokens/s and 1033 tokens/s. The energy efficiency ranges from 0.527 tokens/J to 3.155 token/J. For **FPGA** platforms, power consumption ranges from 180W to 255W, with inference speeds between 220 tokens/s and 370 tokens/s, also in the middle right part of the figure. The energy efficiency ranges from over 0.978 tokens/J to 2.056 tokens/J. For **ASIC** platforms, power consumption ranges from 150W to 600W,

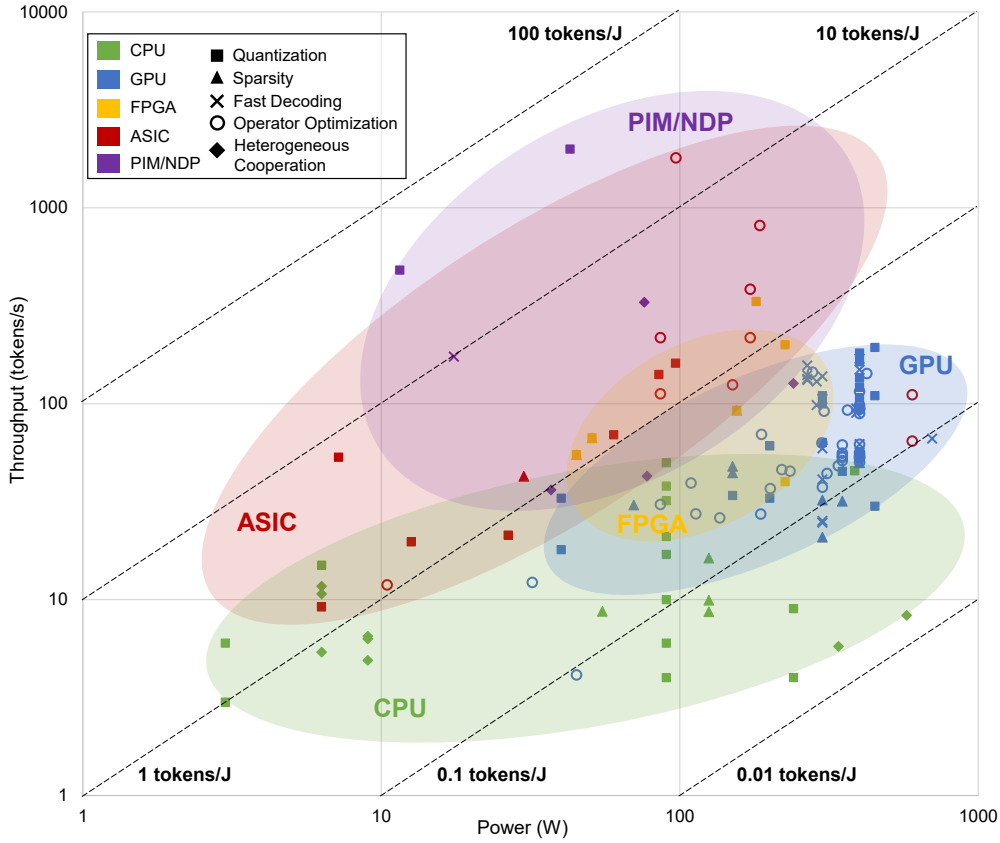

Fig. 13. LLM (~ 7 billion parameters) decode stage throughput (batch size 1) vs power on different platforms with different optimization methods.

with inference speeds between 516 tokens/s and 1176 tokens/s. The energy efficiency ranges from 0.86 tokens/J to over 7.84 tokens/J. For **PIM/NDP** platforms, power consumption is about 75W, with inference speeds about 3000 tokens/s. The energy efficiency is up to 40 tokens/J. Compared to small batch size 1, larger batch size 8 can achieve significantly higher throughput. For example, on a GPU, throughput increases from 18-194 tokens/s to 46-1033 tokens/s, representing an improvement of 2.56×-5.32×. Similarly, energy efficiency improves from 0.067-0.825 tokens/J to 0.527-3.155 tokens/J, an increase of 3.82×-7.87×.

4.3 Optimization Method Comparison

4.3.1 Small batch size (bs=1). We then compare different optimization methods on the same platforms. In Figure 13, the methods employed on **CPU** platforms include quantization, sparsity, and heterogeneous cooperation. Quantization can achieve higher absolute inference speeds, reaching approximately 50 tokens/s, while sparsity and heterogeneous cooperation achieve speeds of 16.3 tokens/s and 11.7 tokens/s, respectively. Quantization and heterogeneous cooperation can achieve higher energy efficiency with 2.38 tokens/J and 1.86 tokens/J, respectively, compared to 0.16 tokens/J for sparsity. On **GPU** platforms, the methods used include quantization, sparsity, fast decoding, and operator optimization. Quantization, fast decoding, and operator optimization can achieve higher absolute inference speeds, reaching approximately 194 tokens/s, 169.68 tokens/s, and 145.04 tokens/s, respectively, while sparsity achieves only about 50 tokens/s. Regarding efficiency,

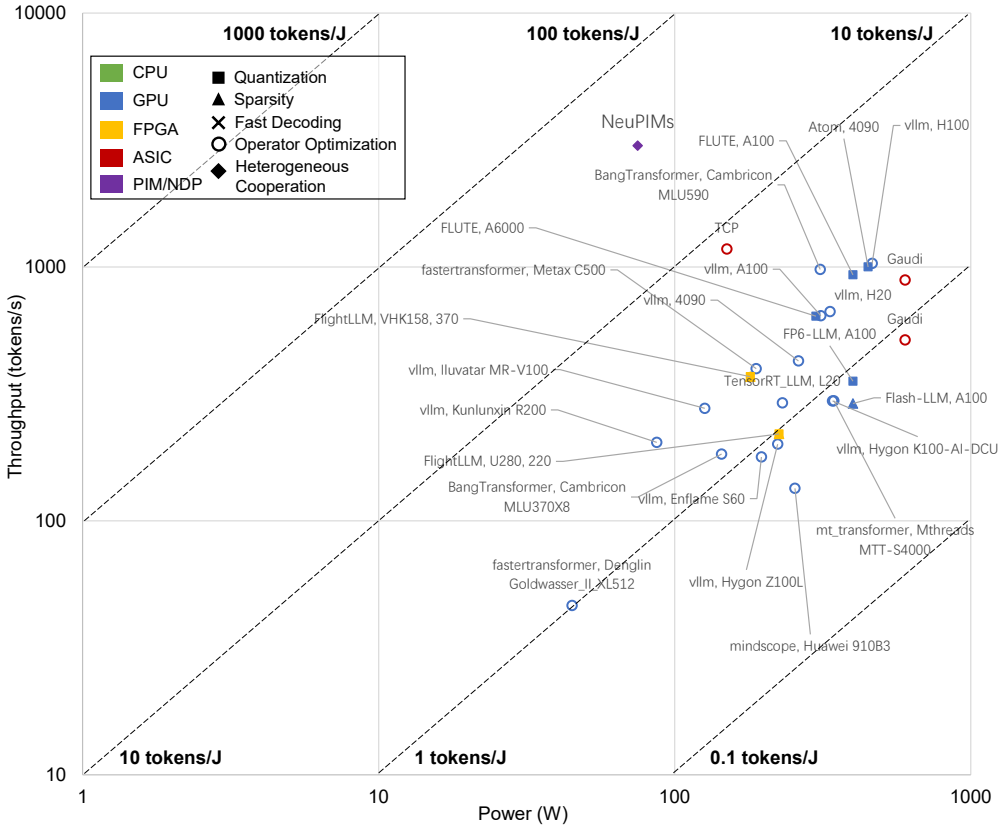

Fig. 14. LLM (~ 7 billion parameters) decode stage throughput (batch size 8) vs power on different platforms with different optimization methods.

these three methods show relatively small differences in performance compared to sparsity (0.16 tokens/J), with efficiencies of 0.825 tokens/J, 0.587 tokens/J, and 0.46 tokens/J, respectively. On **FPGA** platforms, the methods employed include quantization and sparsity, which are often used together. The highest on-board deployed speed reaches 92.5 tokens/s, with an efficiency of up to 1.32 tokens/J. Some works estimate that FPGA can achieve a maximum inference speed of 333 tokens/s and an efficiency of 1.85 tokens/J. On **ASIC** platforms, the methods employed include quantization, sparsity, and operator optimization. Operator optimization can achieve the highest absolute inference speeds, reaching approximately 1800 tokens/s. In comparison, sparsity and quantization achieve speeds of 42.664 tokens/s and 161.086 tokens/s, respectively. In terms of efficiency, quantization and operator optimization offer higher energy efficiency with 7.434 tokens/J and 18.557 tokens/J, respectively. Sparsity shows lower energy efficiency with only 1.421 tokens/J. On **PIM/NDP** platforms, the main methods used are quantization, fast decoding, and heterogeneous cooperation. Quantization can achieve higher throughput with 1998 tokens/s while fast decoding and heterogeneous cooperation only achieve 174.018 tokens/s and 330 tokens/s, respectively. In terms of efficiency, quantization can achieve higher energy efficiency with 46.66 tokens/J while fast decoding and heterogeneous cooperation only achieve 9.94 tokens/J and 4.34 tokens/s, respectively.

4.3.2 Large batch size (bs=8). In Figure 14, on **GPU** platforms, the methods used include quantization, sparsity, and operator optimization. Quantization and operator optimization achieve higher absolute inference speeds, reaching approximately 1000 tokens/s and 1033 tokens/s, respectively,

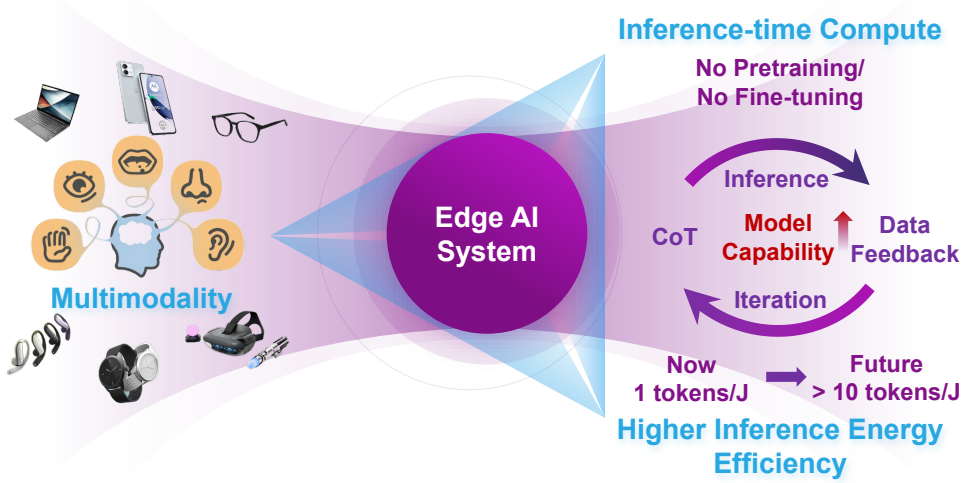

Fig. 15. The trends of edge AI system: multimodality, inference-time computation, and higher inference energy efficiency.

while sparsity achieves about 290 tokens/s. Regarding efficiency, quantization, and operator optimization also achieve higher energy efficiency with 3.155 tokens/J and 2.328 tokens/s compared to sparsity (0.725 tokens/J). On **FPGA** platforms, the methods employed include quantization and sparsity, which are often used together. The highest inference speed reaches 370 tokens/s, with an efficiency of up to 2.056 tokens/J. On **ASIC** platforms, the methods employed include operator optimization. It can achieve higher absolute inference speeds, reaching approximately 1176.47 tokens/s with 7.843 tokens/J energy efficiency. On **PIM/NDP** platforms, the main methods used are heterogeneous cooperation. Heterogeneous cooperation can achieve an inference speed of about 3000 tokens/s and 40 tokens/J.

5 FURTHER DISCUSSION

The development of edge intelligence has gained significant momentum, driven by the increasing capability of LLMs and the increasing demands of edge applications. As we look toward the future, three trends emerge: **multimodality**, **inference-time compute**, and **higher inference energy efficiency**, which is promising to redefine the capabilities of edge AI systems, as shown in Fig. 15. Multimodality focuses on integrating diverse data types, such as visual, auditory, textual, and sensory inputs, to enhance the system's ability equipped with LLMs to understand and adapt to complex scenarios, supporting a wide range of edge devices like PCs, smartphones, AR glasses, and smartwatches. Inference-time compute emphasizes the deep thinking during LLM inference leveraging iterative processes such as Chain-of-Thought (CoT) reasoning to continuously improve model capabilities. It is achieved by performing a post-training process. Higher energy efficiency highlights the demand for increasing energy efficiency from 1 tokens/J to >10 tokens/J for high real-time scenarios, such as embodied intelligence represented by robotics control and autonomous driving.

5.1 Multimodality

As shown in Fig. 16, we summarize the release dates and the number of training text tokens for typical LLMs over the past five years. The number of training tokens has been growing at a rate of 2.5× per year. GPT-3 is trained using only 0.4 trillion text tokens, while Qwen2.5 and Llama3.1 use 18 trillion and 15 trillion tokens, respectively. According to the result of Epoch AI, the total number

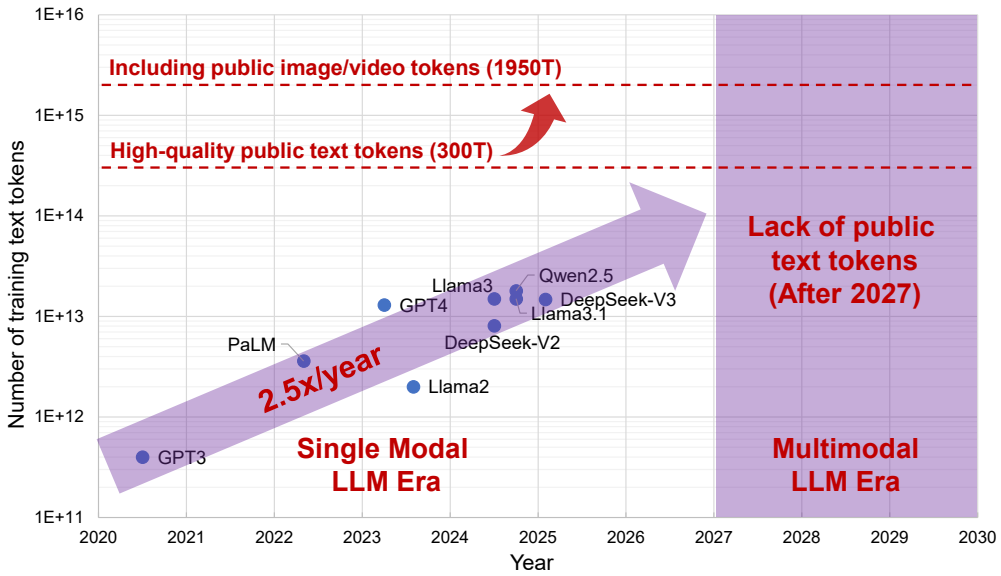

Fig. 16. The number of training text tokens for typical single modal LLMs in the past 5 years. We will lack the public text tokens after 2027. By using multimodal tokens for training, we can continue the LLMs' scaling law.

of human-generated public text tokens is ~300 trillion [232]. At this growth rate, it is expected that by 2027, we will face a shortage of available public text tokens for training LLMs. In addition to text tokens, there is a vast amount of other modalities of tokens on the Internet, such as image and video tokens (total ~1650 trillion [232]). Therefore, from the perspective of token availability, by leveraging multiple modalities of tokens for training LLMs, we can continue the LLMs' scaling law [60]. From the perspective of human interaction, humans rely on various senses as gateways to interact with the physical world, which is inherently multimodal. Recent research has also proven that multimodal learning provably performs better than single modal due to the more accurate estimate of the latent space representation [233]. These factors indicate that future LLMs should have the capability of multimodal interactions to perceive, process, and generate information across multiple modalities.

Currently, more than 80% multimodal LLMs focus on both vision and language tasks [234]. As shown in Fig. 17, their development can be roughly divided into three stages. First, from May to September 2023, the model size is small and the accuracy ranges from 20% to 40%. Then, from December 2023 to April 2024, with the release of GPT4v, the model accuracy significantly improves, reaching nearly 60%. After May 2024 with the release of GPT4o, more multimodal models are released and the accuracy reaches almost 80%. It should be noted that the development of multimodal LLMs (MLLMs) heavily relies on the construction of multimodal datasets. And another significant trend in MLLMs is the concept of native multimodality. By aligning diverse modalities during the training stage, native MLLMs enable end-to-end inputs and outputs.

5.2 Inference-time Compute

In the LLM era, data is the fossil fuel, while the age of pre-training will end because the data is not growing [236]. Inference-time compute, as represented by OpenAI's o1 paradigm [237], is now seen as the critical pathway to enhance model capabilities in the next LLM era. The principles of o1 can be illustrated through its distinction from GPT4, as shown in Fig. 18(a). During the training stage, GPT4 follows the conventional workflow: pre-training a base model followed by fine-tuning

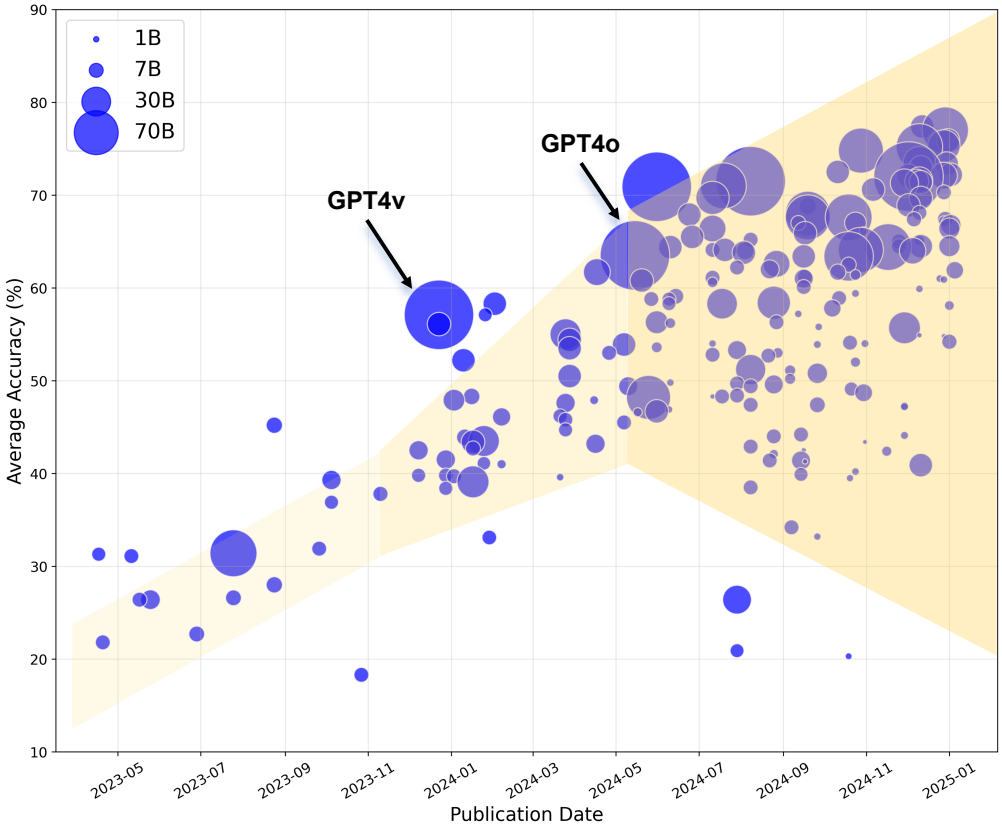

Fig. 17. The timeline of multimodal (Vision-Language) LLMs: average accuracy, publication date, and model size. The results are collected from OpenCompass [235].

through reinforcement learning from human feedback (RLHF). In contrast, o1 introduces several innovations in the post-training stage. Beyond incorporating reinforcement learning (RL), o1 places particular emphasis on safety optimization to ensure that o1 is equipped for more nuanced tasks and safe decision-making during deployment. The difference between GPT4 and o1 becomes more pronounced in the inference stage. GPT4 generates answers directly without explicitly involving iterative reasoning. By comparison, o1 introduces a "Think&Summary" phase before producing answers. This step leverages advanced techniques such as Chain-of-Thought (CoT) reasoning and tree search to perform more intricate logical reasoning and step-by-step problem solving. Compared to GPT4o, the o1 paradigm achieves a 50% improvement in accuracy on the complex logical reasoning tasks like AI Math Evaluation (AIME) test dataset. However, the inference cost is increased by 10-100× [238].

Inference-time compute can enhance the model capabilities but also substantially increase inference time, particularly for complex reasoning tasks. As shown in Fig. 18(b), in the GSM8K benchmark, we employ Llama2-7B as the backbone LLM, combined with the open-source LLM-Reasoners [239]. The inference time is increased by 678× compared to using Llama2-7B directly. This substantial time overhead primarily stems from the iterative multi-step process inherent in inference-time compute, which is necessary to satisfy the requirements of more complex reasoning task. Therefore, choosing the appropriate scenarios and conditions for employing inference-time compute remains a topic worthy of further research. Additionally, inference-time compute also

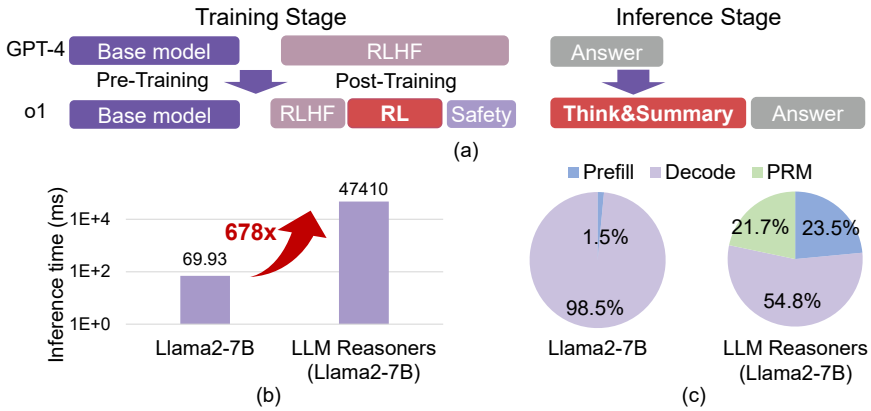

Fig. 18. Inference-time Compute: (a) the difference of training and inference stages, (b) the increment of inference time, and (c) runtime breakdown of inference time.

changes the runtime breakdown of the LLM inference. As shown in Fig. 18(c), the proportion of the prefill stage significantly increases from 1.5% to 23.5%, while the decode stage decreases from 98.5% to 54.8%. The increase in the prefill stage is partly due to the need to merge the human input tokens with the template input tokens as the whole LLM input, which effectively extends the input sequence length. Moreover, in the iterative process, intermediate output tokens are cumulatively fed back as inputs, leading to a gradual increase in the sequence length for prefill. Meanwhile, the introduction of the Process Reward Model (PRM) accounts for 21.7% of the time. This shift of runtime breakdown highlights that future optimization methods for inference-time compute should consider all prefill stage, decode stage, and PRM simultaneously.

Inference-time compute is poised to become the foundational LLM inference paradigm of next-generation AI systems. For edge applications like autonomous driving and robotics, the advanced model capabilities are essential while the exponential growth of inference cost presents a significant challenge for both hardware and software. Real-time responsiveness, higher throughput, and efficient resource utilization are becoming essential attributes for supporting these edge applications.

5.3 Higher Inference Energy Efficiency

In future edge applications, particularly in high real-time scenarios such as embodied intelligence represented by robotics control and autonomous driving, the system's control frequency needs to reach 100-1000 Hz [240–243]. This implies that systems using LLMs as the decision-making center for actions or instructions should achieve an inference speed of at least 100-1000 tokens/s. Since edge devices need to consider energy supply, the power of edge chips should be <20W (the power consumption of human brain [244, 245]) and then the inference efficiency should exceed 10 tokens/J. However, as shown in the Fig. 19, current commercial edge chips such as Tesla RPU [246] and NVIDIA Jetson Orin [247], even with software optimization methods, only achieve an actual inference speed of 10-30 tokens/s and an efficiency of <1 tokens/J. This presents a gap of 1-2 orders of magnitude in both absolute inference speed and efficiency compared to future edge application demands.

To achieve this goal, we should continue the algorithm-hardware co-design approach. On the algorithm side, more aggressive model compression methods can be employed with high-quality datasets for retraining to minimize accuracy loss. On the hardware side, in addition to designing specific hardware units to support algorithm optimizations, it is necessary to design more innovative chip architectures, such as 3D DRAM-stacked integrated architectures that significantly reduce data

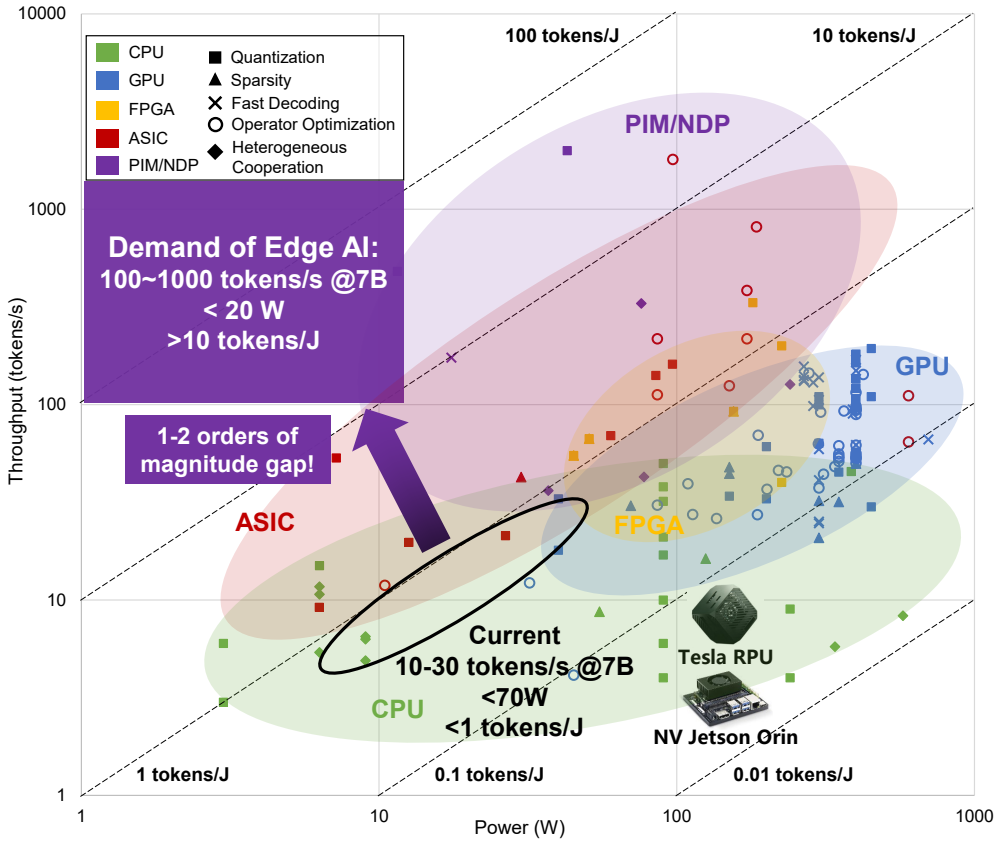

Fig. 19. The performance gap between the future and the current edge AI systems.

transfer latency and energy consumption, dataflow architectures that optimize data transfer paths to improve computational efficiency, and wafer-scale architectures that provide higher computational and storage capabilities through ultra-large-scale integration. These co-design approaches can effectively bridge the current gaps in inference speed and energy efficiency of edge-side chips, meeting the demands of future applications.

6 CONCLUSION

Generative LLMs like GPT series and Llama series are currently the main focus due to their high algorithm performance. The advancements in generative LLMs are closely intertwined with the development of hardware capabilities. This paper presents a comprehensive survey of efficient generative LLM inference on different hardware platforms. We provide an overview of the algorithm architecture of mainstream generative LLMs and summarize different optimization methods for different platforms such as CPU, GPU, FPGA, ASIC, and PIM/NDP. Furthermore, we perform a qualitative and quantitative comparison of inference performance with batch sizes 1 and 8 on different hardware platforms by considering hardware power consumption, absolute inference speed (tokens/s), and energy efficiency (tokens/J). We point out that the development of edge intelligence has gained significant momentum, and three trends (multimodality, inference-time compute, and higher inference energy efficiency) are promising to redefine the capabilities of edge AI systems. Based on the software and hardware optimization, future edge chips are expected to

achieve high throughput and low energy consumption, eliminating the 1-2 orders of magnitude gap in edge AI inference.

REFERENCES

[1] Yuhong Mo, Hao Qin, Yushan Dong, Ziyi Zhu, and Zhenglin Li. Large language model (llm) ai text generation detection based on transformer deep learning algorithm. *arXiv preprint arXiv:2405.06652*, 2024.

[2] Yonghui Wu. Large language model and text generation. In *Natural Language Processing in Biomedicine: A Practical Guide*, pages 265–297. Springer, 2024.

[3] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Pre-trained language models for text generation: A survey. *ACM Computing Surveys*, 56(9):1–39, 2024.

[4] Lei Shu, Liangchen Luo, Jayakumar Hoskere, Yun Zhu, Yinxiao Liu, Simon Tong, Jindong Chen, and Lei Meng. Rewritelm: An instruction-tuned large language model for text rewriting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18970–18980, 2024.

[5] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[6] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[8] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[9] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[11] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[12] Tom B Brown. Language models are few-shot learners. *arXiv preprint ArXiv:2005.14165*, 2020.

[13] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[14] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[15] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[16] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. 2023.

[17] AI Meta. Introducing llama: A foundational, 65-billion-parameter large language model. *Meta AI*, 2023.

[18] Hugo Touvron, Louis Martin, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[19] Meta AI. Introducing meta llama 3: The most capable openly available llm to date, 2024. Accessed: 2024-09-30.

[20] M Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[21] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.

[22] L Xue. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

[23] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[24] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. *arXiv preprint arXiv:2104.12369*, 2021.

[25] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*, 2021.

[26] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

[27] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[28] Shuohuan Wang, Yu Sun, Yang Xiang, Zhihua Wu, Siyu Ding, Weibao Gong, Shikun Feng, Junyuan Shang, Yanbin Zhao, Chao Pang, et al. Ernie 3.0 titan: Exploring larger-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2112.12731*, 2021.

[29] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[30] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[31] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[32] Anthropic. About claude models, 2024. Accessed: 2024-09-30.

[33] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

[34] Hugging Face. Chatglm2-6b model documentation, 2024. Accessed: 2024-09-30.

[35] Hugging Face. Chatglm3-6b model documentation, 2024. Accessed: 2024-09-30.

[36] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[37] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[38] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[39] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[40] x.ai. Announcing grok, 2024. Accessed: 2024-09-30.

[41] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[42] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[43] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[44] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[45] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[46] OpenAI. Introducing GPT-4O, 2024. Accessed: 2024-09-30.

[47] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[48] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

[49] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
[50] Qwen Team. Introducing qwen1.5, February 2024.
[51] Meta AI. Introducing llama 3.1: Our most capable models to date, 2024. Accessed: 2024-09-30.
[52] Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024. Accessed: 2024-09-30.
[53] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.
[54] DeepMind. Gemini ultra, 2024. Accessed: 2024-09-30.
[55] Anthropic. Claude 3 haiku: our fastest model yet, 2024. Accessed: 2024-09-30.
[56] OpenAI. Gpt-3.5 turbo model documentation, 2024. Accessed: 2024-09-30.
[57] Grok. Introducing grok-0, 2024. Accessed: 2024-09-30.
[58] xAI. Open release of grok-1, 2024. https://x.ai/blog/grok-os.
[59] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.
[60] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
[61] Shane Cook. *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.
[62] Denis Foley and John Danskin. Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*, 37(2):7–17, 2017.
[63] Apple Inc. Apple introduces m2 ultra, 2023. https://www.apple.com/newsroom/2023/06/apple-introduces-m2-ultra/.
[64] Inc. Qualcomm Technologies. Snapdragon 8 gen 3 mobile platform, 2024. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3-mobile-platform.
[65] Intel Inc. 4th gen intel xeon scalable processors with built-in accelerators, 2024. https://www.intel.com/content/www/us/en/products/docs/processors/xeon-accelerated/4th-gen-xeon-scalable-processors.html.
[66] NVIDIA Inc. Nvidia tesla v100 gpu architecture, 2017. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.
[67] NVIDIA Inc. Nvidia a100 tensor core gpu architecture, 2020. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.
[68] NVIDIA Inc. Nvidia h100 tensor core gpu architecture, 2022. https://resources.nvidia.com/en-us-data-center-overview/gtc22-whitepaper-hopper.
[69] AMD Inc. Amd cdna architecture, 2020. https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna-white-paper.pdf.
[70] AMD Inc. Amd cdna 2 architecture, 2021. https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf.
[71] AMD Inc. Amd cdna 3 architecture, 2023. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf.
[72] AMD. Amd zynq™ 7000 socs, 2024. https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html.
[73] AMD. Amd virtex™ 7 fpgas, 2024. https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-7.html.
[74] AMD. Amd versal™ adaptive socs, 2024. https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal.html.
[75] AMD. Versal ai engine, 2024. https://www.xilinx.com/products/intellectual-property/versal-ai-engine.html.
[76] Groq Inc. What is a language processing unit?, 2024. https://wow.groq.com/wp-content/uploads/2024/07/GroqThoughts_WhatIsALPU-vF.pdf.
[77] Sean Lie. Wafer-scale ai: Gpu impossible performance. In *2024 IEEE Hot Chips 36 Symposium (HCS)*, pages 1–71. IEEE Computer Society, 2024.
[78] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*, 2023.
[79] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633*, 2023.
[80] Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. Mobile edge intelligence for large language models: A contemporary survey. *arXiv preprint arXiv:2407.18921*, 2024.
[81] Seungcheol Park, Jaehyeon Choi, Sojin Lee, and U Kang. A comprehensive survey of compression algorithms for language models. *arXiv preprint arXiv:2401.15347*, 2024.
[82] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.
[83] Christoforos Kachris. A survey on hardware accelerators for large language models. *arXiv preprint arXiv:2401.09890*, 2024.

[84] Nikoletta Koilia and Christoforos Kachris. Hardware acceleration of llms: A comprehensive survey and comparison. *arXiv preprint arXiv:2409.03384*, 2024.
[85] Christopher Wolters, Xiaoxuan Yang, Ulf Schlichtmann, and Toyotaro Suzumura. Memory is all you need: An overview of compute-in-memory architectures for accelerating large language model inference. *arXiv preprint arXiv:2406.08413*, 2024.
[86] Beom Jin Kang, Hae In Lee, Seok Kyu Yoon, Young Chan Kim, Sang Beom Jeong, Hyun Kim, et al. A survey of fpga and asic designs for transformer inference acceleration and optimization. *Journal of Systems Architecture*, page 103247, 2024.
[87] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
[88] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
[89] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.
[90] Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.
[91] Hanting Chen, Zhicheng Liu, Xutao Wang, Yuchuan Tian, and Yunhe Wang. Dijiang: Efficient large language models through compact kernelization. *arXiv preprint arXiv:2403.19928*, 2024.
[92] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.
[93] Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.
[94] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.
[95] Wei He, Kai Han, Yehui Tang, Chengcheng Wang, Yujie Yang, Tianyu Guo, and Yunhe Wang. Densemamba: State space models with dense hidden connection for efficient large language models. *arXiv preprint arXiv:2403.00818*, 2024.
[96] Jonathan Pilault, Mahan Fathi, Orhan Firat, Chris Pal, Pierre-Luc Bacon, and Ross Goroshin. Block-state transformers. *Advances in Neural Information Processing Systems*, 36, 2024.
[97] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.
[98] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.
[99] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.
[100] Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *arXiv preprint arXiv:2404.08801*, 2024.
[101] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus. *arXiv preprint arXiv:2311.00502*, 2023.
[102] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. *arXiv preprint arXiv:2407.00088*, 2024.
[103] ggerganov. llama.cpp, 2024. https://github.com/ggerganov/llama.cpp.
[104] Tianyi Zhang, Jonah Wonkyu Yi, Bowen Yao, Zhaozhuo Xu, and Anshumali Shrivastava. Nomad-attention: Efficient llm inference on cpus through multiply-add-free attention. *arXiv preprint arXiv:2403.01273*, 2024.
[105] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
[106] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.
[107] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.
[108] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

[109] Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. Llm-mq: Mixed-precision quantization for efficient llm deployment. In *The Efficient Natural Language and Speech Processing Workshop with NeurIPS*, volume 9, 2023.

[110] Ziyi Guan, Hantao Huang, Yupeng Su, Hong Huang, Ngai Wong, and Hao Yu. Aptq: Attention-aware post-training mixed-precision quantization for large language models. *arXiv preprint arXiv:2402.14866*, 2024.

[111] Jinhao Li, Shiyao Li, Jiaming Xu, Shan Huang, Yaoxiu Lian, Jun Liu, Yu Wang, and Guohao Dai. Enabling fast 2-bit llm on gpus: Memory alignment, sparse outlier, and asynchronous dequantization. *arXiv preprint arXiv:2311.16442*, 2023.

[112] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

[113] Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized llms. *arXiv preprint arXiv:2407.10960*, 2024.

[114] Haojun Xia, Zhen Zheng, Xiaoxia Wu, Shiyang Chen, Zhewei Yao, Stephen Youn, Arash Bakhtiari, Michael Wyatt, Donglin Zhuang, Zhongzhu Zhou, et al. Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design. *arXiv preprint arXiv:2401.14112*, 2024.

[115] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

[116] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[117] Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. Towards end-to-end 4-bit inference on generative large language models. *arXiv preprint arXiv:2310.09259*, 2023.

[118] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

[119] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. Llm-fp4: 4-bit floating-point quantized transformers. *arXiv preprint arXiv:2310.16836*, 2023.

[120] Suyeon Hur, Seongmin Na, Dongup Kwon, Joonsung Kim, Andrew Boutros, Eriko Nurvitadhi, and Jangwoo Kim. A fast and flexible fpga-based accelerator for natural language processing neural networks. *ACM Transactions on Architecture and Code Optimization*, 20(1):1–24, 2023.

[121] Andy He, Darren Key, Mason Bulling, Andrew Chang, Skyler Shapiro, and Everett Lee. Hlstransform: Energy-efficient llama 2 inference on fpgas via high level synthesis. *arXiv preprint arXiv:2405.00738*, 2024.

[122] Jude Haris, Rappy Saha, Wenhao Hu, and José Cano. Designing efficient llm accelerators for edge devices. *arXiv preprint arXiv:2408.00462*, 2024.

[123] Hongzheng Chen, Jiahao Zhang, Yixiao Du, Shaojie Xiang, Zichao Yue, Niansong Zhang, Yaohui Cai, and Zhiru Zhang. Understanding the potential of fpga-based spatial acceleration for large language model inference. *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[124] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, et al. Flightllm: Efficient large language model inference with a complete mapping flow on fpgas. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 223–234, 2024.

[125] Mingqiang Huang, Ao Shen, Kai Li, Haoxiang Peng, Boyu Li, and Hao Yu. Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models. *arXiv preprint arXiv:2407.21325*, 2024.

[126] Jaeyong Jang, Yulhwa Kim, Juheun Lee, and Jae-Joon Kim. Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 760–773. IEEE, 2024.

[127] Yubin Qin, Yang Wang, Zhiren Zhao, Xiaolong Yang, Yang Zhou, Shaojun Wei, Yang Hu, and Shouyi Yin. Mecla: Memory-compute-efficient llm accelerator with scaling sub-matrix partition. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1032–1047. IEEE, 2024.

[128] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–15, 2023.

[129] Wenjie Li, Aokun Hu, Ningyi Xu, and Guanghui He. Quantization and hardware architecture co-design for matrix-vector multiplications of large language models. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[130] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. Tender: Accelerating large language models via tensor decomposition and runtime requantization. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1048–1062, 2024.

[131] Lidong Guo, Zhenhua Zhu, Tengxuan Liu, Xuefei Ning, Shiyao Li, Guohao Dai, Huazhong Yang, Wangyang Fu, and Yu Wang. Towards floating point-based attention-free llm: Hybrid pim with non-uniform data format and reduced multiplications. 2024.

[132] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1071–1085. IEEE, 2022.

[133] Janak Sharda, Po-Kai Hsu, and Shimeng Yu. Accelerator design using 3d stacked capacitorless dram for large language models. In *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pages 487–491. IEEE, 2024.

[134] Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. Turbo sparse: Achieving llm sota performance with minimal activated parameters. *arXiv preprint arXiv:2406.05955*, 2024.

[135] Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, et al. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *arXiv preprint arXiv:2402.13516*, 2024.

[136] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

[137] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[138] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[139] Yun Li, Lin Niu, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. E-sparse: Boosting the large language model inference through entropy-based n: M sparsity. *arXiv preprint arXiv:2310.15929*, 2023.

[140] Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Proceedings of the VLDB Endowment*, 17(2):211–224, 2023.

[141] Abhinav Agarwalla, Abhay Gupta, Alexandre Marques, Shubhra Pandit, Michael Goin, Eldar Kurtic, Kevin Leong, Tuan Nguyen, Mahmoud Salem, Dan Alistarh, et al. Enabling high-sparsity foundational llama models with efficient pretraining and deployment. *arXiv preprint arXiv:2405.03594*, 2024.

[142] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.

[143] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[144] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

[145] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *ICLR*, 2024.

[146] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[147] Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.

[148] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[149] Matteo Pagliardini, Daniele Paliotta, Martin Jaggi, and François Fleuret. Faster causal attention over large sequences through sparse flash attention. *arXiv preprint arXiv:2306.01160*, 2023.

[150] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR, 2020.

[151] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[152] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

[153] Eunji Yoo, Gunho Park, Jung Gyu Min, Se Jung Kwon, Baeseong Park, Dongsoo Lee, and Youngjoo Lee. Tf-mvp: Novel sparsity-aware transformer accelerator with mixed-length vector pruning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[154] Huizheng Wang, Jiahao Fang, Xinru Tang, Zhiheng Yue, Jinxi Li, Yubin Qin, Sihan Guan, Qize Yang, Yang Wang, Chao Li, et al. Sofa: A compute-memory optimized sparsity accelerator via cross-stage coordinated tiling. *arXiv preprint arXiv:2407.10416*, 2024.

[155] Zixu Li, Wang Wang, Xin Zhong, Manni Li, Jiayu Yang, Yinyin Lin, Guhyun Kim, Yosub Song, Chengchen Wang, and Xiankui Xiong. Lauws: Local adaptive unstructured weight sparsity of load balance for dnn in near-data processing. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2024.

[156] Shiwei Liu, Chen Mu, Hao Jiang, Yunzhengmao Wang, Jinshan Zhang, Feng Lin, Keji Zhou, Qi Liu, and Chixiao Chen. Hardsea: Hybrid analog-reram clustering and digital-sram in-memory computing accelerator for dynamic sparse self-attention in transformer. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[157] Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023.

[158] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

[159] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

[160] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[161] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[162] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024.

[163] Weilin Zhao, Yuxiang Huang, Xu Han, Chaojun Xiao, Zhiyuan Liu, and Maosong Sun. Ouroboros: Speculative decoding with large model enhanced drafting. *arXiv preprint arXiv:2402.13720*, 2024.

[164] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024.

[165] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023.

[166] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024.

[167] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[168] Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*, 2024.

[169] Lianming Huang, Shangyu Wu, Yufei Cui, Ying Xiong, Xue Liu, Tei-Wei Kuo, Nan Guan, and Chun Jason Xue. Raee: A training-free retrieval-augmented early exiting framework for efficient inference. *arXiv preprint arXiv:2405.15198*, 2024.

[170] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[171] Sangyeob Kim, Sangjin Kim, Wooyoung Jo, Soyeon Kim, Seongyon Hong, and Hoi-Jun Yoo. 20.5 c-transformer: A 2.6-18.1 μj/token homogeneous dnn-transformer/spiking-transformer processor with big-little network and implicit weight generation for large language models. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 67, pages 368–370. IEEE, 2024.

[172] Cong Li, Zhe Zhou, Size Zheng, Jiaxi Zhang, Yun Liang, and Guangyu Sun. Specpim: Accelerating speculative inference on pim-enabled system via architecture-dataflow co-exploration. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 950–965, 2024.

[173] Tri Dao et al. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[174] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[175] Tri Dao et al. Flash-decoding for long-context inference. [Online], 2023. https://crfm.stanford.edu/2023/10/12/flashdecoding.html.

[176] Ke Hong et al. Flashdecoding++: Faster large language model inference on gpus, 2024.

[177] Reza Yazdani Aminabadi et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*,

pages 1–15. IEEE, 2022.
[178] Woosuk Kwon et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.
[179] Sensetime. Openppl: A high-performance deep learning inference platform. [Online], 2023. https://openppl.ai/home.
[180] NVIDIA. cublas: Basic linear algebra on nvidia gpus. [Online], 2017. https://developer.nvidia.com/cublas.
[181] Neal Vaidya et al. Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available. [Online], 2023. https://github.com/NVIDIA/TensorRT-LLM.
[182] NVIDIA. Cutlass: Cuda templates for linear algebra subroutines. [Online], 2017. https://github.com/NVIDIA/cutlass.
[183] Yujia Zhai et al. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 344–355. IEEE, 2023.
[184] Seungjae Moon, Jung-Hoon Kim, Junsoo Kim, Seongmin Hong, Junseo Cha, Minsu Kim, Sukbin Lim, Gyubin Choi, Dongjin Seo, Jongho Kim, et al. Lpu: A latency-optimized and highly scalable processor for large language model inference. *IEEE Micro*, 2024.
[185] Shiwei Liu, Guanchen Tao, Yifei Zou, Derek Chow, Zichen Fan, Kauna Lei, Bangfei Pan, Dennis Sylvester, Gregory Kielian, and Mehdi Saligane. Consmax: Hardware-friendly alternative softmax with learnable parameters. *arXiv preprint arXiv:2402.10930*, 2024.
[186] Jinhao Li, Shan Huang, Jiaming Xu, Jun Liu, Li Ding, Ningyi Xu, and Guohao Dai. Marca: Mamba accelerator with reconfigurable architecture. *arXiv preprint arXiv:2409.11440*, 2024.
[187] Hanjoon Kim, Younggeun Choi, Junyoung Park, Byeongwook Bae, Hyunmin Jeong, Sang Min Lee, Jeseung Yeon, Minho Kim, Changjae Park, Boncheol Gu, et al. Tcp: A tensor contraction processor for ai workloads industrial product. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 890–902. IEEE, 2024.
[188] Intel Inc. Intel®gaudi® ai accelerator first generation deep learning training and inference processor, 2020. https://habana.ai/products/gaudi/.
[189] Intel Inc. Intel®gaudi® 2 ai accelerator high performance acceleration for genai and llms, 2023. https://habana.ai/products/gaudi2/.
[190] Roman Kaplan. Intel gaudi 3 ai accelerator: Architected for gen ai training and inference. In *2024 IEEE Hot Chips 36 Symposium (HCS)*, pages 1–16. IEEE, 2024.
[191] Mohamed Assem Ibrahim, Mahzabeen Islam, and Shaizeen Aga. Balanced data placement for gemv acceleration with processing-in-memory. *arXiv preprint arXiv:2403.20297*, 2024.
[192] Rongqing Cong, Wenyang He, Mingxuan Li, Bangning Luo, Zebin Yang, Yuchao Yang, Ru Huang, and Bonan Yan. Attentionlego: An open-source building block for spatially-scalable large language model accelerator with processing-in-memory technology. *arXiv preprint arXiv:2401.11459*, 2024.
[193] Yuting Wu, Ziyu Wang, and Wei D Lu. Pim gpt a hybrid process in memory accelerator for autoregressive transformers. *npj Unconventional Computing*, 1(1):4, 2024.
[194] Wontak Han, Hyunjun Cho, Donghyuk Kim, and Joo-Young Kim. Sal-pim: A subarray-level processing-in-memory architecture with lut-based linear interpolation for transformer-based text generation. *arXiv preprint arXiv:2401.17005*, 2024.
[195] Taeyang Jeong and Eui-Young Chung. Pipepim: Maximizing computing unit utilization in ml-oriented digital pim by pipelining and dual buffering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.
[196] Hyungyo Kim, Gaohan Ye, Nachuan Wang, Amir Yazdanbakhsh, and Nam Sung Kim. Exploiting intel® advanced matrix extensions (amx) for large language model inference. *IEEE Computer Architecture Letters*, 2024.
[197] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv preprint arXiv:2312.12456*, 2023.
[198] Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*, 2024.
[199] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 722–737, 2024.
[200] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, et al. Ianus: Integrated accelerator based on npu-pim unified memory system. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 545–560, 2024.
[201] Taehyun Kim, Kwanseok Choi, Youngmock Cho, Jaehoon Cho, Hyuk-Jae Lee, and Jaewoong Sim. Monde: Mixture of near-data experts for large-scale sparse models. *arXiv preprint arXiv:2405.18832*, 2024.

[202] Jaewan Choi, Jaehyun Park, Kwanhee Kyung, Nam Sung Kim, and Jung Ho Ahn. Unleashing the potential of pim: Accelerating large batched inference of transformer-based generative models. *IEEE Computer Architecture Letters*, 2023.

[203] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. Attacc! unleashing the power of pim for batched transformer-based generative model inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 103–119, 2024.

[204] Shin-haeng Kang, Sukhan Lee, and Kyomin Sohn. The era of generative artificial intelligence: In-memory computing perspective. In *2023 International Electron Devices Meeting (IEDM)*, pages 1–4. IEEE, 2023.

[205] Byeongho Kim, Sanghoon Cha, Sangsoo Park, Jieun Lee, Sukhan Lee, Shin-haeng Kang, Jinin So, Kyungsoo Kim, Jin Jung, Jong-Geon Lee, et al. The breakthrough memory solutions for improved performance on llm inference. *IEEE Micro*, 2024.

[206] Yandong Luo and Shimeng Yu. H3d-transformer: A heterogeneous 3d (h3d) computing platform for transformer model acceleration on edge devices. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–19, 2024.

[207] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, et al. An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 970–982. IEEE, 2024.

[208] Harsh Sharma, Pratyush Dhingra, Janardhan Rao Doppa, Umit Ogras, and Partha Pratim Pande. A heterogeneous chiplet architecture for accelerating end-to-end transformer models. *arXiv preprint arXiv:2312.11750*, 2023.

[209] SK Hynix. Cost-effective llm inference solution using sk hynix's aim (accelerator-in-memory), 2023. https://sc23.supercomputing.org/proceedings/exhibitor_forum/exhibitor_forum_pages/exforum133.html.

[210] Guhyun Kim, Jinkwon Kim, Nahsung Kim, Woojae Shin, Jongsoon Won, Hyunha Joo, Haerang Choi, Byeongju An, Gyeongcheol Shin, Dayeon Yun, et al. Sk hynix ai-specific computing memory solution: From aim device to heterogeneous aimx-xpu system for comprehensive llm inference. In *2024 IEEE Hot Chips 36 Symposium (HCS)*, pages 1–26. IEEE Computer Society, 2024.

[211] Zhongkai Yu, Shengwen Liang, Tianyun Ma, Yunke Cai, Ziyuan Nan, Di Huang, Xinkai Song, Yifan Hao, Jie Zhang, Tian Zhi, et al. Cambricon-llm: A chiplet-based hybrid architecture for on-device inference of 70b llm. *arXiv preprint arXiv:2409.15654*, 2024.

[212] Pujiang He, Shan Zhou, Changqing Li, Wenhuan Huang, Weifei Yu, Duyi Wang, Chen Meng, and Sheng Gui. Distributed inference performance optimization for llms on cpus. *arXiv preprint arXiv:2407.00029*, 2024.

[213] Pujiang He, Shan Zhou, Wenhuan Huang, Changqing Li, Duyi Wang, Bin Guo, Chen Meng, Sheng Gui, Weifei Yu, and Yi Xie. Inference performance optimization for large language models on cpus. *arXiv preprint arXiv:2407.07304*, 2024.

[214] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 616–630. IEEE, 2022.

[215] Wenhua Cheng, Yiyang Cai, Kaokao Lv, and Haihao Shen. Teq: Trainable equivalent transformation for quantization of llms. *arXiv preprint arXiv:2310.10944*, 2023.

[216] Raspberry Pi. Raspberry pi 5, 2024. https://www.raspberrypi.com/products/raspberry-pi-5/.

[217] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[218] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[219] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[220] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[221] Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, et al. Think fast: A tensor streaming processor (tsp) for accelerating deep learning workloads. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 145–158. IEEE, 2020.

[222] Dennis Abts, Garrin Kimmell, Andrew Ling, John Kim, Matt Boyd, Andrew Bitar, Sahil Parmar, Ibrahim Ahmed, Roberto DiCecco, David Han, et al. A software-defined tensor streaming multiprocessor for large-scale machine learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 567–580, 2022.

[223] Groq Inc. Groundbreaking gemma 7b performance running on the groq lpu™ inference engine, 2024. https://wow.groq.com/groundbreaking-gemma-7b-performance-running-on-the-groq-lpu-inference-engine/.

[224] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (salp) in dram. *ACM SIGARCH Computer Architecture News*, 40(3):368–379, 2012.

[225] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, et al. Hardware architecture and software stack for pim based on commercial dram technology: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 43–56. IEEE, 2021.

[226] Jinhao Li, Chong Qu, Fan Wu, and Jianfei Jiang. A 4gbps dppm on-chip serial link based on pipelined vernier-tdc. In *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pages 1–3. IEEE, 2020.

[227] Ang Li, Jianfei Jiang, Qin Wang, Naifeng Jing, Zizheng Dong, Shuya Ji, Xiulan Cheng, and Yuhang Zhao. A method to improve 3d interconnections resource utilization and reliability in hybrid bonding process considering the effects on signal integrity. In *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*, pages 2131–2136. IEEE, 2023.

[228] Jinhao Li, Jianfei Jiang, Qin Wang, Naifeng Jing, Weiguang Sheng, and Guanghui He. A gain reconfigurable time difference amplifier with self-adaptive linearity control. *Analog Integrated Circuits and Signal Processing*, 107:435–449, 2021.

[229] Debendra Das Sharma. Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 5–12. IEEE, 2022.

[230] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, et al. A 1ynm 1.25 v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[231] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, et al. System architecture and software stack for gddr6-aim. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–25. IEEE, 2022.

[232] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024.

[233] Yu Huang, Chenzhuang Du, Zihui Xue, Xuanyao Chen, Hang Zhao, and Longbo Huang. What makes multi-modal learning better than single (provably). *Advances in Neural Information Processing Systems*, 34:10944–10956, 2021.

[234] Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. *arXiv preprint arXiv:2401.13601*, 2024.

[235] OpenCompass. Opencompass multi-modal leaderboard, 2025.

[236] Ilya Sutskever. Sequence to sequence learning with neural networks at neurips 2024, 2024.

[237] OpenAI. Introducing openai o1, 2024. https://openai.com/o1/.

[238] OpenAI. Openai o1-mini, September 2024.

[239] Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, et al. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*, 2024.

[240] David B Stewart, Donald E Schmitz, and Pradeep K Khosla. Implementing real-time robotic systems using chimera ii. In *1990 IEEE International Conference on Systems Engineering*, pages 252–257. IEEE, 1990.

[241] Ryan M Bena, Sushmit Hossain, Buyun Chen, Wei Wu, and Quan Nguyen. A hybrid quadratic programming framework for real-time embedded safety-critical control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3418–3424. IEEE, 2023.

[242] Ramin Mafi, Shahin Sirouspour, Behzad Mahdavikhah, Brian Moody, Kaveh Elizeh, Adam Kinsman, and Nicola Nicolici. A parallel computing platform for real-time haptic interaction with deformable bodies. *IEEE Transactions on Haptics*, 3(3):211–223, 2009.

[243] John Atkins and Hyunglae Lee. Mintnet: Rapid motion intention forecasting of coupled human-robot systems with simulation-to-real autoregressive neural networks. *IEEE Robotics and Automation Letters*, 2023.

[244] Henry Markram. The human brain project. *Scientific American*, 306(6):50–55, 2012.

[245] Pierre J Magistretti and Igor Allaman. A cellular perspective on brain energy metabolism and functional imaging. *Neuron*, 86(4):883–901, 2015.

[246] Acceleration Robotics. About tesla's optimus robot brain, a robotics hardware and software computer architecture perspective, 2022.

[247] NVIDIA. Nvidia jetson orin, 2024.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009