A Survey: Collaborative Hardware and Software Design in the Era of Large Language Models

Cong Guo, Feng Cheng, Zhixu Du, James Kiessling, Jonathan Ku, Shiyu Li, Ziru Li, Mingyuan Ma,
Tergel Molom-Ochir, Benjamin Morris, Haoxuan Shan, Jingwei Sun, Yitu Wang, Chiyue Wei, Xueying Wu,
Yuhao Wu, Hao Frank Yang, Jingyang Zhang, Junyao Zhang, Qilin Zheng, Guanglei Zhou,
Hai (Helen) Li, *Fellow, IEEE,* and Yiran Chen, *Fellow, IEEE*

(Invited Paper)

Abstract—The rapid development of large language models (LLMs) has significantly transformed the field of artificial intelligence, demonstrating remarkable capabilities in natural language processing and moving towards multi-modal functionality. These models are increasingly integrated into diverse applications, impacting both research and industry. However, their development and deployment present substantial challenges, including the need for extensive computational resources, high energy consumption, and complex software optimizations. Unlike traditional deep learning systems, LLMs require unique optimization strategies for training and inference, focusing on system-level efficiency. This paper surveys hardware and software co-design approaches specifically tailored to address the unique characteristics and constraints of large language models. This survey analyzes the challenges and impacts of LLMs on hardware and algorithm research, exploring algorithm optimization, hardware design, and system-level innovations. It aims to provide a comprehensive understanding of the trade-offs and considerations in LLM-centric computing systems, guiding future advancements in AI. Finally, we summarize the existing efforts in this space and outline future directions toward realizing production-grade co-design methodologies for the next generation of large language models and AI systems.

I. INTRODUCTION

The rapid advancement of large language models [1]–[3] (LLMs) has brought revolutionary change to the landscape of artificial intelligence (AI). These sophisticated models, leveraging vast amounts of data and significant computational power, have pushed the boundaries of what AI systems can achieve, demonstrating unprecedented capabilities in natural language understanding, generation, and interaction. Furthermore, LLMs are progressing by incorporating tasks beyond natural language processing, moving towards achieving multi-modal functionality. As LLMs become increasingly integrated into a wide range of applications—from chatbots [2], [4]–[6] and virtual assistants [5], [7] to complex decision-making systems—their impact on research and industry becomes increasingly profound.

Despite their success in various application fields, LLMs face unique challenges compared to CNN models, particularly in **training** and **inference**. Due to their vast number of parameters, often in the billions or even trillions, LLMs require significantly more memory during training. For example, training a model like GPT-3 [4], which has 175 billion parameters, demands around 350GB of GPU memory just for storing model parameters. In contrast, a typical CNN such as ResNet-50 [8], with 25 million parameters, requires only about 100MB of memory for weights. This vast difference in memory requirements makes training LLMs much more demanding. Solutions to address this include model parallelism, which splits the model across multiple devices to distribute memory usage; mixed-precision training, which reduces memory consumption by using lower-precision data types; and memory-efficient optimizers like DeepSpeed's ZeRO [9], which reduces the memory footprint during training.

In terms of inference, LLMs are inherently larger and require more computational power and memory than CNNs. This makes deploying LLMs significantly more resource-intensive. The autoregressive nature of LLMs also exacerbates the memory wall [10] problem because each token generated depends on all previously generated tokens, resulting in increased memory and computational requirements as the sequence length grows. This differs from convolutional neural networks (CNNs), where computations can be parallelized more efficiently. Furthermore, LLMs use key-value (KV) caches to store activations from previous tokens, speeding up subsequent token generation but also necessitating the storage of large amounts of activation data. As the sequence length increases, the KV cache grows linearly, posing significant memory management challenges, especially for longer contexts.

In addition to challenges, LLMs also offer unique opportunities for improved efficiencies. Unlike CNNs, which employ diverse operators, LLMs have similar architectures. This consistency allows for the custom-made implementation of operators specific to certain architectures or hyperparameters.

This survey aims to analyze the unique challenges posed by LLMs and their significant impact on research directions within both the hardware and algorithm communities. We examine existing works on algorithm optimization, hardware

Cong Guo, Feng Cheng, Zhixu Du, James Kiessling, Jonathan Ku, Shiyu Li, Ziru Li, Mingyuan Ma, Tergel Molom-Ochir, Benjamin Morris, Haoxuan Shan, Jingwei Sun, Yitu Wang, Chiyue Wei, Xueying Wu, Yuhao Wu, Jingyang Zhang, Junyao Zhang, Qilin Zheng, Guanglei Zhou, Hai (Helen) Li, and Yiran Chen are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, 27705, USA.

Hao Frank Yang is with the Department of Civil and Systems Engineering, Johns Hopkins University, Baltimore, Maryland, 21218, USA.

architecture design, and system-level innovations for LLMs. Through this survey, we strive to develop a comprehensive understanding of the intricate trade-offs and design considerations that govern the development of LLM-centric computing systems. By synthesizing the latest research findings and identifying emerging trends, we aim to pave the way for future breakthroughs in this rapidly evolving field, enabling the creation of more powerful and efficient artificial intelligence systems.

The structure of the remaining survey is as follows: Section II introduces the preliminary knowledge related to LLMs. In Section III, we examine the current best practices for LLM training. In Section IV, we discuss the latest hardware and software co-design techniques for LLM inference. Finally, Section V summarizes the main contributions of this survey.

II. PRELIMINARIES

Large Language Models (LLMs) leverage massive datasets and sophisticated architectures to understand, generate, and manipulate human language with unprecedented accuracy and fluency. The backbone of modern LLMs is the transformer architecture, which has revolutionized NLP by addressing the limitations of previous recurrent and convolutional models.

A. Transformer Architecture

The transformer architecture, introduced by Vaswani et al. in the paper "Attention is All You Need," [11] consists of an encoder-decoder structure. However, many LLMs, like GPT [1], [4], [5], [12] (Generative Pre-trained Transformer), use only the decoder part. The core innovation of the transformer is the multi-head self-attention mechanism [11] (MHSA), which enables the model to weigh the importance of different words in a sentence.

Linear Projection: In the MHSA block, input embeddings are first linearly projected into three different spaces to generate queries (Q), keys (K), and values (V). These projections are performed through learned linear transformations, which means that the input embeddings are multiplied by different weight matrices to produce Q, K, and V. Mathematically, this can be expressed as:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where X represents the input embeddings, and W_Q, W_K, W_V are the learned weight matrices for the queries, keys, and values, respectively. Each head in the multi-head attention mechanism independently performs this projection, enabling the model to participate in various parts of the input sequence and capture diverse relationships.

Self-Attention Mechanism: For each word in the input, attention scores are calculated using the following components:

- Query (Q): Represents the current word for which the attention score is being computed.
- Key (K): Represents all words in the input sequence.
- Value (V): Represents the actual values used to compute the weighted sum for the output.

The attention score for a pair of words is computed using the scaled dot product of the query and key, followed by a SoftMax function to obtain a probability distribution:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Here, d_k is the dimension of the key vectors, and the division by $\sqrt{d_k}$ is a scaling factor to ensure stable gradients. After that, the attention scores compute a weighted sum of the value vectors, resulting in the self-attention output.

Multi-Head Attention: To capture different types of relationships and dependencies, transformers use multi-head attention. This involves running multiple self-attention operations in parallel (each with different parameter sets) and then concatenating their outputs. This allows the model to jointly attend to information from different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{Attn}_1, \text{Attn}_2, \ldots, \text{Attn}_n)W_O$$

Feed-Forward Networks (FFN): After the attention mechanism, the output is passed through a feed-forward neural network (FFN). This network consists of multiple linear transformations with non-linear activations in between:

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2$$

Here, W_1, W_2, b_1, and b_2 are learned parameters, and σ is the activation function. The FFN is applied independently to each position in the sequence, allowing the model to learn complex representations.

Residual Connections and Layer Normalization: Each sub-layer (attention and FFN) in the transformer is wrapped with residual connections [8] and followed by layer normalization [13]. Residual connections help train deeper networks by allowing gradients to flow through the network directly. Layer normalization ensures that the input to each sub-layer has a stable distribution, which helps in faster convergence during training:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

where μ and σ^2 are the mean and variance of x, and γ and β are learned scale and shift parameters.

B. Scope of the Survey on Large Language Models

Based on previous research categorizations [14], we classify language models into three main types: Encoder-Decoder, Encoder-only, and Decoder-only models. All these models are based on the Transformer architecture [11]. Encoder-decoder and Encoder-only models are considered BERT-style models [15], while Decoder-only models are termed GPT-style models [1].

The term "large language model" lacks a precise definition and scope, leading to ongoing discussions in the field. For instance, Yang et al. [14] consider the BERT model as a "large" language model, yet their focus is predominantly on GPT-style models. Conversely, Zhao et al. [16] define BERT-style models as "small-scale language models."

This survey focuses on GPT-style models, particularly those with model sizes equal to or larger than GPT-2 [12], which contains 1.5 billion parameters. This focus is based on three primary reasons:

- Shift in Popularity: BERT models, especially Encoder-only variants, have gradually begun to fade away within the community [14]. The landscape changed significantly after 2021 with the introduction of transformative models like GPT-3 [4], which led to a surge in the adoption of decoder-only architectures. GPT-style models have since dominated the development of LLMs.
- Scaling Laws: Extensive research demonstrates that increasing model size substantially enhances LLM capabilities. In 2020, OpenAI's introduction of the "scaling law" [17] highlighted that model performance is strongly correlated with size. For example, GPT-3, with its 175 billion parameters, vastly outperforms BERT's 300 million parameters. The emphasis on "large" models is a defining characteristic of GPT-style models, resulting in significantly different hardware and software solutions compared to BERT-style models.
- Autoregressive Mechanism: GPT-style models employ an autoregressive mechanism, which has proven superior in few-shot and zero-shot scenarios. However, this mechanism also introduces significant hardware performance challenges, which will be discussed in Section IV.

The advent of LLMs has revolutionized natural language processing and artificial intelligence. However, these models come with significant challenges, particularly in terms of computational and memory requirements, making efficient deployment a critical concern. Strategies are proposed to address these challenges in the training and inference phases from both the software and hardware perspectives. This survey will focus on recent advancements in GPT-style models from aspects of the system, algorithm, and accelerator.

III. TRAINING

Training LLMs is a vital but both time- and resource-consuming step in their development. LLM training can be classified into two categories: 1) pretraining and 2) fine-tuning. Pretraining requires large datasets, many steps, and large batch sizes, making it very expensive. As reported in the literature [18], training a 600B model can take over 196,000 TPUv3 core hours. More optimized models require a much higher training cost. According to the study [6], with NVIDIA 80GB A100 GPU under 400W power consumption, it takes over 184,000 GPU hours for pretraining Llama2-7B and over 1,720,000 hours for Llama2-70B. The electricity cost alone for training all four variants of Llama2 amounts to approximately $158,000. Fine-tuning, on the other hand, can be performed with smaller datasets, fewer steps, and smaller batch sizes. The focus of this work is on the expensive pretraining step which will henceforth be referenced to as simply training.

At the scale of the LLM model size, both compute time and energy consumption per step are significant. Even marginal improvements in these areas could lead to substantial cost savings and reduced environmental impacts. To improve training performance, it is critical to optimize various types of parallelism. Data parallelism is still effective. However, as the model size scales and the system becomes more distributed, it becomes increasingly difficult to eke out performance gains from data parallelism alone. In addition, the peak performance of the hardware can limit the achievable data parallelism. To reduce energy consumption, the proposed framework must minimize data movement, and the supported hardware should be energy efficient. Coupled with performance and energy consumption challenges, LLMs have more stringent hardware requirements. First, it requires a large memory. Unlike inference, where only parameters will be stored, training needs parameters, gradients, optimizer states, and activations to be stored. Table I illustrates the relative size for each variable. Second, and correspondingly, LLMs require a higher memory bandwidth. As the size of the model increases, data movement becomes more intensive, leading to the need for high-bandwidth communication. This effect is even more pronounced when the system becomes distributed or when offloading techniques are applied, both of which incur increased data swapping.

To address these challenges, academics and industry have proposed many solutions, ranging from infrastructure to hardware and algorithms. In particular, collaboration between hardware and software design is critical to addressing these challenges. In the following subsections, we will discuss solutions at each level and the challenges they target to address, including system, algorithm, and accelerator.

A. Framework and System

In this subsection, we start by introducing different types of parallelism and popular distributed infrastructures. Then, offloading techniques, a powerful solution addressing the issue of not enough memory, will be discussed. Furthermore, rematerialization and LoRA will be illustrated. Finally, we will introduce existing popular frameworks for training.

Parallelism. With the increasing complexity of DNN models, distributed training has become essential, especially for LLMs. An example of data parallelism in this domain is illustrated by the PyTorch Distributed Data-Parallel (DDP) [19] feature. DDP duplicates the setup to process different data portions simultaneously and synchronizes after each training step. Model parallelism splits the model across multiple GPUs, with each GPU handling different stages. Model parallelism includes two categories: pipeline parallelism [18], [20], [21], which assigns individual layers to single GPUs, and tensor parallelism [22]–[24], which divides each tensor into chunks allocated to specific GPUs. In addition to traditional data and model parallelism, an emerging parallelism called fully sharded data parallelism (FSDP) is proposed in [9] for LLM training.

TABLE I
SIZE OF EACH TYPE OF DATA INVOLVED IN TRAINING FOR A MODEL WITH N PARAMETERS AND BATCH SIZE AS B. x IS A VARIABLE THAT DEPENDS ON THE MODEL ARCHITECTURE.

Parameters	Gradients	Optimizer States	Activations
$2N$	$0 \sim 2N$	$4N \sim 12N$	xNB

Memory Optimization. Zero Redundancy Optimizer (ZeRO) [9] and its subsequent works [25]–[27] have been proposed to alleviate the high GPU memory requirement in large model training. ZeRO focuses on reducing redundant copies of data on GPU. It proposes three main optimization stages that partition the optimizer states, gradients, and parameters accordingly. ZeRO-Offload [25] enables the training of even larger models by offloading optimizer states and optimizer updates to the CPU in order to strike a balance between accessibility and efficiency. ZeRO-Infinity [26] recognizes the much higher growth speed of model size than GPU memory and thus explores more methods that trade efficiency to enable the training of extremely large models. In addition to previous work, it incorporates NVMe memory for offloading for more storage space and offloads parameters, gradients, and activation checkpoints since their sizes can no longer be held on GPU as the model size grows. In addition, it manages the operations to reduce the buffer size requirement further. ZeRO++ [27] returns to the design of ZeRO and focuses more on communication efficiency for large clusters of GPUs. While offloading can facilitate the training of large models, it significantly increases communication overhead between the GPU and CPU. This is because the connection between these components, such as PCIe in most system setups, typically offers limited bandwidth compared to the GPU's peak performance and memory bandwidth. As a result, this bottleneck can lead to substantial slowdowns during training.

Rematerialization, also known as checkpointing or recomputation [28], is a technique used in training LLMs to manage memory usage more efficiently by trading off computational resources. During the forward pass, only a subset of intermediate activations is stored, with the selection based on a strategy that minimizes memory usage while maintaining manageable computational overhead. During the backward pass, the necessary intermediate activations that were not stored are recomputed on the fly from the previously stored activations. Combining recomputed activations with stored activations enables the gradient calculation necessary to update the model parameters. Rematerialization significantly reduces memory usage, allowing for training larger models or using larger batch sizes without exceeding hardware memory limits. The primary trade-off is the increased computational cost due to recomputation, which is often acceptable given the benefits of training more complex models. This approach is akin to register allocation via graph coloring [29], which seeks scheduling strategies to maximize the reuse of limited registers. Rematerialization has been implemented in PyTorch for homogeneous sequential networks [28], and more advanced versions [30] are modified for heterogeneous networks.

However, these memory reduction techniques, like recomputation and ZeRO, cause severe memory fragmentation. To address this, GMLake [31] employs a virtual memory stitching (VMS) mechanism to merge non-contiguous memory blocks, significantly reducing GPU memory usage for LLM fine-tuning. Transparent to DNN models and techniques, GMLake ensures the seamless execution of resource-intensive tasks.

Popular Frameworks. Besides being able to be implemented in the conventional PyTorch [19] and Tensorflow [32], there exist emerging frameworks which are specialized for LLM training like DeepSpeed [33], Hugging Face Transformer [34], Torchtune [35], Megatron-LM, etc. Most frameworks are integrated with the optimization techniques for LLM training mentioned above.

B. Algorithm and System Co-design

Full fine-tuning (fine-tuning all learnable parameters) of a pre-trained LLM for a specific downstream task is often infeasible or too costly. Full fine-tuning poses non-trivial challenges to the supporting system platforms due to its computationally-intensive and resource-demanding nature and can potentially hurt the generalizability of the pre-trained backbone model. Parameter Efficient Fine-Tuning, or PEFT, addresses this need by either introducing additional lightweight trainable modules or selectively adapting a small fraction of the original parameters. The family of adapter-based PEFT methods inserts extra trainable parameters strategically either within the frozen pre-trained transformer blocks or as attached components.

LoRA. LoRA, or Low-Rank Adaptation, is a technique designed to fine-tune large pre-trained models in a parameter-efficient manner [36]. Instead of updating all model parameters during adaptation, LoRA focuses on a lower-dimensional sub-space by applying a low-rank decomposition to the weight matrices. This involves updating pairs of smaller matrices that can approximate the necessary changes, significantly reducing the number of parameters to be fine-tuned. This approach makes the fine-tuning process faster and less memory intensive, which is particularly advantageous for large models. Despite reducing parameters, LoRA achieves competitive performance with traditional fine-tuning methods, making it an attractive option for adapting large pre-trained models to specific tasks without high computational costs.

Due to its simplicity and effectiveness, many LoRA-variant have been proposed to improve upon the vanilla LoRA. SPLoRA [37] and LoRAPrune [38] both leverage structured channel-pruning to remove groups of weights in order to increase the computational efficiency. QLoRA [39], a highly memory-efficient technique that first quantizes a pre-trained model into a novel 4-bit NormalFloat data type with an innovative method called Double Quantization and then back-propagate the gradients through the quantized weights with Paged Optimizers, can finetune a LLaMA 65B parameter model on a single 48GB GPU with similar performance as that of a 16-bit full-finetuned model. LQ-LoRA [40] further extends the quantization limit to sub-3 bits by decomposing each pre-trained matrix into a frozen quantized matrix and an adaptable low-rank matrix. QA-LoRA [41] tries to get the best of both quantization and adaptation by balancing the unequal freedom between them inherent in LoRA through group-wise operators.

Prompt-based Learning. Prompt-based learning has become increasingly prominent in the field of large language models (LLMs), primarily by leveraging minimal examples or specific cues to steer a pre-trained language model (PLM) toward generating the desired output. This approach marks a departure from traditional supervised learning, which relies

on extensive labeled data to train a model explicitly. The advent of OpenAI's GPT-3 [42] significantly advanced the exploration of prompt-based learning, demonstrating that the massive scale of GPT-3 enables the generation of relevant outputs with well-designed prompts without necessitating task-specific model fine-tuning. Despite this, manually crafted prompts often exhibit a performance discrepancy compared to fine-tuned models, as noted by multiple studies [42]–[45]. Recent advancements have shown that prompts need not be confined to natural language forms but can also be optimized in a continuous space using gradient descent, enhancing their efficiency [46]–[51]. In scenarios where only the continuous prompts are tuned—keeping the PLM's parameters unchanged—the training remains efficient while achieving comparable performance to full model tuning. The concept of prompt tuning [46], [52], [53] was introduced to fine-tune a continuous vector that is concatenated to the input embeddings, optimizing the prompt directly within the embedding space. Building on this, the p-tuning methodology was developed to further enhance performance by learning concrete prompts within the embedding space, a technique further refined in subsequent studies [49], [51], [54].

Retrieval-Augmented Generation. Retrieval-augmented generation (RAG) is a technique that enhances the capabilities of generative models (like large language models) by integrating external knowledge retrieval systems. RAG is a powerful technique for enhancing LLMs, allowing them to generate responses that are grounded in up-to-date, accurate information. By combining the strengths of retrieval systems and generative models, RAG ensures that large language models are more reliable, less prone to hallucination, and better suited for real-world applications that demand accuracy and specificity. Currently, approximate nearest neighbor search (ANNS), which retrieves the approximate nearest neighbors of a given query in the high-dimensional and large-scale vector database, has been widely used as an RAG technique. Hierarchical Navigable Small World [55] (HNSW) is a graph-based ANNS algorithm that organizes data points in multiple layers of proximity graphs, enabling fast searches by navigating through a hierarchical structure. DiskANN [56], on the other hand, is designed for handling large datasets that don't fit into memory by extending nearest neighbor search to disk-based systems, offering a balance between speed and memory efficiency. Faiss [57] is a popular library developed by Meta, providing highly optimized algorithms for similarity search, especially leveraging GPU acceleration for fast nearest-neighbor computations. These ANNS algorithms make it feasible to efficiently handle the retrieval tasks requiring high-dimensional similarity search in real-time.

Others. Other PEFT methods selectively update a subset of the pre-trained model weights during adaptation. Diff pruning [58] aims to learn an additive sparse mask that is applied to the frozen pre-trained weights for each task, effectively localizing task-specific weights to update. PaFi [59], on the other hand, finds a universal set of parameters to update for all downstream tasks based on parameter magnitude. FISH Mask [60] gauges the importance of each model parameter by estimating its' Fisher information, resulting in a binary mask that indicates

which parameters are crucial for the current task.

C. Accelerators for Training

LLM training and fine-tuning require substantial computational resources. Traditional CPUs, while versatile, are often insufficient for the massive parallel processing demands of LLMs. This has led to the adoption and innovation of specialized hardware accelerators designed to enhance the efficiency and speed of training and fine-tuning processes. For LLM, the extremely large volume of memory footprint makes the accelerator mainly focus on memory-centric optimizations, especially for memory compression.

GPU. GPUs (Graphics Processing Units) have become the cornerstone of modern deep learning infrastructure. Originally designed for rendering graphics, their highly parallel architecture makes them ideal for the matrix and tensor operations fundamental to training neural networks. GPUs, such as NVIDIA's A100 [61] and H100 [62], offer significant speedups in training times compared to CPUs, making them a popular choice for both academic research and industrial applications. For LLM, NVIDIA proposed a specific optimization for the training process, called Transformer Engine [62], based on the mix-precision training technology [63].

Transformer Engine. The NVIDIA Transformer Engine is designed to optimize the performance and efficiency of transformer-based models widely used in natural language processing and AI. It leverages mixed-precision techniques, combining FP16 (16-bit floating point) and FP32 (32-bit floating point) computations to maximize throughput and minimize memory usage without compromising model accuracy. Using Tensor Cores on NVIDIA GPUs, the Transformer Engine accelerates training and inference processes, enabling faster development and deployment of AI applications. This approach enhances computational efficiency and reduces costs and energy consumption in large-scale AI operations.

TPU. Tensor Processing Units (TPUs) are custom-built accelerators developed by Google specifically for machine learning tasks, particularly deep learning and large language models (LLMs). TPUs are designed to handle the vast computational requirements of training LLMs by providing high throughput and efficient performance. They utilize a systolic array architecture [64] to accelerate matrix multiplications, which are fundamental to neural network operations. TPUs support training and inference, with features such as high memory bandwidth and mixed-precision computation to enhance speed and efficiency. TPUs significantly reduce the time and cost of training large, complex models by offering scalable, high-performance computing.

Others. ASIC accelerator designs also aim to reduce the memory bottleneck in large language models (LLMs). Smart-Infinity [65] is the first study to leverage host memory and storage as an extended memory hierarchy for LLM training, enhancing efficiency by addressing storage bandwidth bottlenecks through near-storage processing. It performs parameter updates on custom near-storage accelerators, significantly reducing storage traffic. The system includes an efficient data transfer handler to manage system integration and overlap

data transfers with fixed memory consumption. Additionally, accelerator-assisted gradient compression/decompression improves scalability by reducing write traffic. Before this, many studies focused on efficient training based on sparsity and quantization, such as Sigma [66], TensorDash [67], FAST [68], and Combricon-Q [69], including evaluations on Transformer-based models. However, these studies mainly targeted much smaller language models, like GNMT [70].

Efficient training for large language models (LLMs) is a promising yet nascent field. Despite its potential, practical challenges and deployment difficulties have limited research, particularly in accelerator design. In the next section, our survey focus will shift to inference optimization studies, which present lower complexity and broader applicability.

IV. INFERENCE

LLMs are powerful and capable models, but deploying pre-trained LLMs is often difficult due to the models' exceptionally high resource usage requirements. In extreme cases, the largest models, such as LLaMa-70B, contain tens of billions of parameters, incurring massive computational and memory costs impractical for most consumer-grade devices. As such, much research has been performed to mitigate these bottlenecks, such as using dedicated accelerators, advanced model compression methods, and algorithmic advances. The following subsections offer discussions and key insights into these solutions.

A. LLM Inference System

A critical step for LLMs is their deployment on hardware devices, catering to both offline inference and online serving scenarios. Offline inference involves a single user with all requests initiated at the start, aiming to reduce inference latency by enhancing the model's forward process. In contrast, online serving handles asynchronous requests from multiple users, requiring optimized memory management and efficient batching and scheduling strategies to improve throughput. In addition, the increasing scale of LLMs generally necessitates deployment across multiple hardware devices, creating an intricate infrastructure. Consequently, system-level optimization has become a significant research focus. This section explores key optimization techniques.

1) Inference Engine: Inference engine optimizations for LLMs aim to accelerate the forward process, achieved through both fusion and non-fusion based techniques.

Operation fusion. Kernel fusion is a widely adopted optimization technique for LLM inference. It involves combining multiple operators or layers in the computation graph. This method enhances computational efficiency by reducing memory access, decreasing kernel launch overhead, and improving parallelism without data dependencies. Profile results indicate that attention and linear operations dominate LLM runtime, accounting for over 75% of total inference duration [71]. To optimize attention computation, FlashAttention [72], [73] integrates the entire process into a single operator, achieving $3\times$ training speed up on the GPT-2 model. FlashDecoding [74] and FlashDecoding++ [75] further enhance this by optimizing

parallelism and introducing efficiency in SoftMax computation. For linear operations, TensorRT-LLM [76] employs a specialized GEMV implementation, while FlashDecoding++ [75] adapts FlatGEMM for reduced dimensions, utilizing fine-grained tiling and double buffering to improve efficiency. Additional optimizations include the fusion of lightweight operations such as LayerNorm, SwiGLU, activation functions, and residual additions by frameworks like DeepSpeed [33], Byte-Transformer [77], xFormers [78], and TensorRT-LLM [76], which also uses a pattern-matching algorithm to identify potential fusions across various LLM architectures.

Memory Optimization. Beyond fusion, addressing the challenges posed by the dynamic sizes of input and output tokens during inference is crucial. Inspired by CPU virtual memory systems, vLLM [79] introduces PagedAttention to segment the KV cache into manageable blocks, enhancing memory management with $24\times$ higher throughput than HuggingFace Transformers [34]. When GPU memory is insufficient, techniques such as ZeRO-Inference by DeepSpeed [33] offload large model weights to CPU memory to improve performance by overlapping computation with weight fetching. Similarly, FlexGen [80] employs a linear programming-based strategy to optimize offloading across CPU, GPU, and disk spaces. The utilization of high-capacity flash memory for storing model parameters further demonstrates efficient inference by optimizing memory usage [81].

2) Online Serving: Optimizations in LLM serving systems are centered around effectively managing asynchronous requests to boost both throughput and responsiveness, utilizing strategies in dynamic batching, memory management, and Scheduling.

Batching Optimization. Efficient handling of variable request sizes is a primary concern in LLM serving. ORCA [82] introduces continuous batching or rolling batching, where new requests are dynamically batched as previous ones complete, optimizing the use of computational resources. This method is extended in Sarathi [83], Sarathi-Serve [84] and LightLLM [85], which employ a split-and-fuse technique to balance load across different processing stages, thereby minimizing response times and enhancing throughput.

Memory Management. Efficient memory usage is also crucial due to the extensive requirements of the KV cache, especially for lengthy context interactions. Traditional allocation strategies often lead to substantial memory waste. To address this, S^3 [86] predicts the upper limit of generation lengths, optimizing the initial memory allocation. Further improvements are seen in vLLM [79], which introduces a paged storage mechanism similar to that used in operating systems, allocating the largest possible contiguous space and mapping KV caches dynamically to reduce fragmentation. LightLLM [85] refines this approach by allocating KV cache storage at the token level, maximizing space utilization, and minimizing waste. LLM in a Flash [81] addresses the challenge of efficiently running large language models (LLMs) that exceed the available DRAM capacity by storing the model parameters in flash memory and dynamically loading them into DRAM as needed. When a new token is added, the system only needs to update a minimal number of neurons rather than

reloading all neurons.

Scheduling Strategy. Variability in request length can significantly impact scheduling efficiency. Traditional first-come-first-served approaches often lead to inefficient resource utilization, known as head-of-line blocking [79], [82], [85]. To combat this, FastServe [87] leverages a preemptive scheduling strategy that prioritizes requests based on their estimated completion time, thus improving throughput and reducing job completion times. Additionally, VTC [88] introduces a fairness-oriented scheduling model that adjusts resource allocation based on the workload of incoming requests, ensuring a balanced service across different users. Scheduling in Distributed architectures offers unique opportunities for scaling LLM services. SpotServe [89] addresses the challenges of using cloud-based preemptible GPU resources by implementing strategies for dynamic adjustment and state recovery, ensuring robust service continuity. Finally, techniques like those proposed in Splitwise [90] and TetriInfer [91] disaggregate compute-intensive prefilling from memory-intensive decoding processes, tailoring resource allocation to the specific demands of each stage.

Heterogeneous Computing. The significant computational and memory demands of large language model (LLM) inference typically require multiple high-end accelerators. However, driven by the growing need for latency-insensitive tasks, some studies [81], [92]–[95] explore high-throughput LLM inference using limited resources, such as a single GPU, edge devices, and mobile devices. The most critical challenge is data transfer due to insufficient memory capacity. There are typically two scenarios of data transfer: the first [81], [92], [93] is when model parameters and intermediate results need to be stored in storage (e.g., Flash memory) due to limited DRAM capacity, resulting in data transfer between DRAM and storage; the second scenario occurs when CPU and GPU cannot share memory [94], [95], requiring model parameters and intermediate results to be stored in host memory due to limited GPU memory, thus leading to data transfer between CPU and GPU. Reducing the cost of data transfer often becomes a primary consideration for optimizing LLMs on edge devices.

These studies can optimize the system from multiple angles to reduce data transfer and lower storage costs. Existing work has observed that retaining only a subset of effective activation values does not degrade model performance, and the sparsity pattern of activation values is predictable [81], [92], [95]–[97]. By leveraging the sparsity of activation values, only a subset of model parameters is needed for computation, significantly reducing data transfer and storage costs.

This section effectively delineates the optimization strategies for deploying large language models (LLMs) in both offline and online contexts, focusing on enhancing system performance through various techniques such as operation fusion, memory optimization, and dynamic batching. The outlined approaches, from kernel fusion like FlashAttention [72] to memory-efficient strategies like vLLM [79] and scheduling optimizations such as FastServe [87], reveal the depth of innovation aimed at improving the responsiveness and throughput of LLM systems. However, the practical implementation of these techniques often involves trade-offs between computational efficiency, memory usage, and response times. Real-world performance data would be invaluable in quantifying these trade-offs, offering a clearer perspective on the effectiveness of different strategies in varied deployment scenarios. Such data could guide in selecting the most appropriate optimizations based on specific requirements, such as latency constraints or hardware limitations, ensuring optimal performance tailored to the needs of diverse models or applications.

B. Algorithm for Efficient LLM

Faster inference is essential for large models, especially those with commercial potential. Different algorithms tailored for various aspects of large models have been proposed to improve the inference efficiency. In this subsection, we introduce several techniques that have greatly impacted the community. Specifically, *Mixture-of-Experts (MoE)* speeds up the feed-forward networks (FFNs), *Efficient Attention* speeds up the attention module, *Speculative Decoding* allows faster auto-regression and *Structured State Space Models (SSMs)* serve as an alternative to transformers that improve the computational efficiency over long sequences.

1) MoE: Mixture-of-experts (MoE) was first proposed in [98], [99] by Michael I. Jordan and Robert A. Jacobs more than three decades ago. The initial insight was to propose an architecture such that each expert handles a different subset of input data. Later on, [100] proposes to stack several neural-network-based MoE layers to make the model deeper with the rise of deep learning. Recently, the era of large models came under the guidance of the scaling law [101], asserting that the performance of the model demonstrates a predictive behavior as the model size increases. Now, MoE attracts more and more attention due to its scalability; that is, drastically increasing the number of parameters incurs little computational overhead. MoE offers a solution for fast inference of large models and has been employed by several famous LLMs, such as GPT-4 [102] and Mixtral [103].

An MoE layer consists of several expert models and a router function (or gating function in some literature). The router function will select experts for computation for each input entity. Specifically, let $g(\cdot)$ denote the router function and $\{f_i(\cdot)\}_{i=1}^E$ denote E experts. The output of an MoE layer is as follows:

$$M(\boldsymbol{x}) = \sum_{i\in\mathcal{I}} p_i(\boldsymbol{x})f_i(\boldsymbol{x}), \tag{1}$$

$$\text{where } p_i(\boldsymbol{x}) = \frac{\exp\{h_i(\boldsymbol{x})\}}{\sum_{j=1}^E \exp\{h_j(\boldsymbol{x})\}}. \tag{2}$$

Let M denote the Mixture of Experts (MoE) model and h the router function. Typically, h is a linear model that performs a linear classification over experts. We use p to denote the probability distribution over all experts. The set $\mathcal{I}$ signifies the selected experts; different choices for $\mathcal{I}$ yield various MoE techniques.

Many breakthroughs have been made in large language models (LLMs) using Mixture of Experts (MoE). Shazeer

et al. [104] developed an LSTM model with 137B parameters, significantly enhancing model capacity with minimal computational overhead. Switch Transformer [105] extended this to a transformer-based MoE model with 1.6T parameters, confirming MoE expansion follows the scaling law. GShard [18] efficiently implements large-scale MoE, expanding it by over 600B parameters. Clark et al. [106] investigated the scaling law for routing-based language models, deriving an Effective Parameter Count for scalable models.

Addressing instability in training large MoE models, ST-MoE [107] improved transfer learning performance. Mixture-of-Depth (MoD) [108] explores skipping layers in transformer models. Xue [109] and Riquelme [110] develop vision transformer-based MoE models, with Riquelme achieving state-of-the-art performance with half the computation. Obando [111] constructs an MoE model for reinforcement learning, providing empirical evidence for scaling laws in this domain. MoEfication [112], [113] converts dense models to MoE models by grouping weights in FFNs. Routing strategies have been explored to balance load and address training instability. Base Layer [114] uses a linear assignment problem for balanced loading, while Hash Layer [115] uses predefined hash functions. StableMoE [116] stabilizes training by learning a balanced router function. Differentiable MoE architectures like Soft MoE [117] and Lory [118] enhance stability by making operations differentiable. MoE models, which activate only some weights during each forward pass, improve GPU memory efficiency and throughput. SE-MoE [119], M3ViT [120], and SiDA-MoE [121] introduce strategies to optimize throughput and expert caching. The analogy to MoE, FoE (Fusion of Experts) [122] also aggregates knowledge from different experts, which can be pre-trained in their respective domains.

2) Efficient Attention: The bottleneck of transformers on computation is the attention scheme both in time and memory. Efforts have been made on algorithms towards efficient attention schemes that either speed up the inference or improve the memory usage. The main lines of research can be split into two categories, one is grouping the keys and values and the other is approximating the attention score either by kernel methods or low-rank methods.

Multi-Query Attention. Multi-Query Attention (MQA) [123] and Group-Query Attention (GQA) [124] improve the attention schemes by sharing the keys and values in multi-head attention. Specifically, in multi-head attention, each head possesses a pair of keys and values. MQA averages all the keys and values across all heads and shares the averaged keys and values for all heads. The proposed attention scheme significantly saves the memory bandwidth for loading keys and values and speeds up the decoding process. However, MQA may lead to performance degradation. GQA builds upon MQA and tackles the quality degradation by relaxing the sharing across heads. GQA defines a hyperparameter G that denotes the number of groups where, within each group, keys and values are averaged and shared. For example, GQA-1 reduces to MQA, and GQA-H reduces to multi-head attention with H heads.

Attention Approximation. Attention Approximation tech-

niques improve the efficiency of attention schemes by reducing the computation of the attention matrix from $O(n^2)$ to $O(n)$, where n is the sequence length. *Kernel-based methods* aim to design a kernel feature map $\phi \in \mathbb{R}^{n \times d}$, where d is the feature dimension. The formulation of kernel-based methods is as follows:

$$\text{SoftMax}(QK^T)V \approx \phi(Q)\phi(K)^T V, \tag{3}$$

where $\phi(K)^T V$ is a multiplication between $\mathbb{R}^{d \times n}$ and $\mathbb{R}^{n \times d}$ and $\phi(Q)(\phi(K)^T V)$ is a multiplication between $\mathbb{R}^{n \times d}$ and $\mathbb{R}^{d \times d}$, taking $O(nd^2)$ complexity. Performers [125] and RFA [126] employ the random feature projection as the feature map, while PolySketchFormer [127] exploits sketching techniques with polynomial functions. *Low-rank-based methods* aim to use low-rank matrix compression techniques to change $Q \in \mathbb{R}^{n \times d}$ and $K \in \mathbb{R}^{n \times d}$ to $\tilde{Q} \in \mathbb{R}^{k \times d}$ and $\tilde{K} \in \mathbb{R}^{k \times d}$, where k is a smaller number. Thus, the computational complexity for low-rank-based methods is $O(nk^2)$. Linformer [128] is the first to explore the possibility of low-rank approximation of the attention matrix. LRT [129] then proposes to apply low-rank approximation on both the attention matrix and feed-forward layers. FLuRKA [130] combines kernel-based methods and low-rank-based methods that first apply low-rank approximation and then apply kernel feature map on the low-rank $\tilde{Q}$ and $\tilde{K}$.

Speculative Decoding. Speculative decoding [131], [132] aims to speed up the decoding process for auto-regressive large language models (LLMs). The motivation comes from the observation that memory loading is a bottleneck in LLM inference, and models of smaller sizes can output the correct tokens while memory is efficient. Specifically, speculative decoding employs a small model to generate tokens, and the LLMs consistently evaluate the draft generated by the small model to decide whether to accept or reject the generation. Upon rejecting small models, a resampling from LLM will be performed. Inference with large language models is primarily constrained by heavy I/O, which acts as the bottleneck. Speculative decoding significantly reduces memory I/O during inference, leading to improvements in latency, though it potentially increases FLOPs.

Research on speculative decoding focuses on improving the acceptance rate from LLMs over models' generation. One line of research focuses on exploiting the computing units that ask small models to generate several candidates to be evaluated by LLMs in parallel [133]–[135]. The other line of research aims to tackle the problem through the lens of algorithms, improving the alignment between LLMs and small models [136]–[138].

SSMs. The State-Space Models (SSMs), which are efficient yet effective, serve as an alternative to the transformer. SSMs are especially good at long sequence tasks given their linear computation and memory compared to transformers. The key idea of SSMs is to compress the input sequence of length L, $\{h_t \in \mathbb{R}^{d_{emb}}\}_{t=1}^{t=L}$, to a sequence of states $\{x_t \in \mathbb{R}^{d_{states}}\}_{t=1}^{t=L}$ based on HiPPO theory [139]. Compared to transformers, where the attention score is computed between every two embeddings in the sequence, SSMs compress all the up to t embedding vectors in the sequence to the state x_t and performs

the prediction only based on the state by the following formulas:

$$x_t = Ax_{t-1} + Bh_t, \qquad (4)$$

$$y_t = Cx_t, \qquad (5)$$

where x is the state, h is the input sequence, and A, B and C represent the transition matrices. SSMs enjoy linear computation and memory since at each round of propagation, the next token interacts with the states only instead of all previous tokens.

Based upon the foundational architecture, the mainstream research focuses on better parametrization on transition matrices [140]–[143] and better computational architecture based on SSMs [142]–[146]. Specifically, LSSL [140] proposes to initialize matrix A via the optimal transition matrix proposed in [139]. Further, LSSL trains an SSM model through telescoping propagation equations 4, which can be computed efficiently through the Fast Fourier Transform. S4 [141] employs a diagonalized transition matrix A to enhance the computational efficiency. Later on, S5 [142] proposes to share the transition matrices across all input dimensions to boost the computational efficiency, while Mamba [143] and S4 [141] propose input dependent transition matrices that improve the model capability. Meanwhile, Mamba [143] and S4 [141] utilize a parallel scan technique that improves the computational efficiency of SSMs. MambaFormer [145] and Jamba [146] improve the SSM architecture by combining transformers into SSMs, where [145] use the SSM layer to replace the FFN layer in transformers and Jamba [146] add four transformer layers to SSMs. Mamaba2 [144] proposes a new architecture based on State-Space Duality that achieves 2-8× speed up compared to Mamba [143].

3) Multi-modal LLMs.: Advancements in text-only LLMs have paved the way for the rapid development of *multi-modal* LLMs [7], which can process visual inputs such as images and videos. The construction of these multi-modal LLMs follows a well-established recipe. Initially, a vision encoder (e.g., CLIP [147]) is used to encode the visual input into a sequence of embeddings. These embeddings are then passed through a projector, such as a multi-layer perceptron (MLP), to align them with the embedding space of the originally text-only LLM. Once aligned, the vision embeddings are concatenated with the text embeddings in an autoregressive manner, enabling the LLM to process and generate outputs based on both modalities. However, the integration of vision tokens/embeddings introduces a significant computational burden compared to text-only LLMs. This is primarily due to the large number of vision tokens—often numbering in the hundreds or thousands—required to represent the visual input comprehensively [7], [148].

To mitigate the increased computational cost associated with multi-modal LLMs, several methods have been developed to prune the number of vision tokens. One such method involves the use of the Perceiver module [149], which employs a transformer with queries to perform learned pooling, effectively replacing the MLP as the projector [150]. This approach can significantly reduce the computational demands of both training and inference. Another method is the algorithmic approach of PruMerge [151], which selectively retains important vision tokens while merging the less significant ones. Despite these advancements, there remains considerable potential for further development in this area, as systematic explorations are still relatively limited. We believe that continued research and innovation will yield even more efficient techniques for handling vision tokens in multi-modal LLMs.

C. Compression Methods and Accelerators

The immense size of LLMs creates significant challenges for deployment, both due to the computational complexity as well as resource availability requirements. Significant research has been performed to strategically compress LLMs in order to mitigate these bottlenecks while preserving the capabilities of the model, increasing inference efficiency while continuing to scale down the required resources needed to execute. Model compression can be categorized into four main methods: quantization, pruning, knowledge distillation, and low-rank factorization.

1) Quantization: Quantization is a highly effective method for reducing the size and computational demands of deep neural network (DNN) models. There are two primary quantization techniques: quantization-aware training (QAT) and post-training quantization (PTQ). QAT, as discussed in [152]–[154], involves retraining the model to adapt to quantization noise. On the other hand, PTQ [152], [155], [156] converts a floating-point model to a lower-bit model without requiring training data, making it particularly suitable for large-scale language models.

Quantization Algorithm. Innovative quantization methods have significantly enhanced the efficiency and performance of LLMs. SmoothQuant [157] enables 8-bit weight and activation quantization (W8A8) by migrating quantization difficulty from activations to weights through per-channel scaling, reducing activation outliers and maintaining accuracy. AWQ (Activation-aware Weight Quantization) [158] optimizes low-bit weight-only quantization by protecting critical weights based on activation distributions, preserving model performance without backpropagation. QuaRot [159] achieves outlier-free 4-bit inference using randomized Hadamard transformations, efficiently handling activation quantization by removing outliers. QuIP [160] facilitates 2-bit quantization through incoherence processing, leveraging incoherent weight and Hessian matrices, and using adaptive rounding to minimize quantization error, supported by theoretical analysis.

Quantization Accelerator. To improve the accuracy of quantized DNN models, numerous studies have proposed new architecture designs based on advanced quantization techniques. BitFusion [161] supports various bit-width quantizations by combining low-bit processing elements. OLAccel [162] and GOBO [162] quantizes outliers with higher precision, but these approaches often suffer from unaligned memory accesses, leading to additional overhead and limited computing speed. ANT [163] offers a fixed-length adaptive quantization framework, considering tensor distribution but overlooking outliers' importance. Mokey [164] uses narrow

fixed-point inference for transformer models by converting values to 4-bit indices into dictionaries of 16-bit fixed-point centroids, improving hardware efficiency without fine-tuning. OliVe [165] employs an outlier-aware quantization method using an outlier-victim pair mechanism to address quantization challenges, reducing hardware overhead and aligning memory access, enabling efficient 4-bit quantization for weights and activations. These methods collectively advance the deployment of quantized LLMs in resource-constrained environments by improving performance, reducing memory usage, and maintaining model accuracy.

New Data Type. Many studies also focus on designing new numeric types with reduced precision to improve model compression and efficiency. Microsoft Floating Point (MSFP) [166] uses a shared exponent for groups of values, enabling efficient dot product computations and higher arithmetic density compared to formats like Bfloat16 or INT8, making it ideal for large-scale cloud deployments. FP6-LLM [167] introduces a 6-bit floating-point format that leverages TC-FPx, a GPU kernel design, to reduce inference costs and improve performance for large language models (LLMs). LLM-FP4 [168] utilizes 4-bit floating-point quantization, optimizing exponent bits and clipping ranges, achieving minimal accuracy loss while enabling efficient deployment in resource-constrained environments. LLM.int8() [169] enables efficient 8-bit matrix multiplication by combining vector-wise quantization and mixed-precision decomposition, maintaining accuracy for models up to 175B parameters and reducing memory usage, facilitating inference on large models using consumer-grade GPUs.

Quantization offers significant benefits for large language models, primarily by reducing memory bandwidth and capacity requirements, which in turn accelerates performance for the memory-bound decoding process. As context lengths grow, weight quantization alone becomes insufficient, necessitating the quantization of the KV cache to maintain efficiency. However, it's important to note that pushing quantization to extremely low bit widths may not always yield proportional improvements due to the increased overhead in decoding operations.

2) Sparsity: Sparsity, which involves setting parts of the weights or activations to zero, is a commonly used technique for compressing neural networks. By efficiently skipping these zeros during inference, sparsification reduces computational complexity, memory occupancy, and bandwidth requirements. In LLMs, sparsification is applied to weights in fully connected layers and activations in attention scores, leading to two main techniques: weight pruning and sparse attention.

Weight Pruning. Weight pruning [170]–[178] reduces the number of parameters by removing less important weights. This process identifies and zeros out weights that have minimal impact on the model's performance, effectively compressing the model and making it more efficient. To leverage the benefits of common deep learning accelerators optimized for dense and regular workloads, structured pruning is employed to remove weights in a regular manner (e.g., removing entire channels or layers). The LLM Pruner [179] identifies group structures in LLM weights and prunes whole groups, creating a regularly pruned weight matrix, which allows the pruned LLM to run efficiently on GPUs. The Plug-and-Play [180] prunes weights into a structured N:M sparsity pattern, which can efficiently run on sparse tensor cores in GPUs [174]. SLOPE [181] applies N:M structured sparsity to both forward and backward passes and achieves significant speedups and memory savings compared to dense LLMs. PGF [182] further explored sparse LLM training recipes at high sparsity level, by introducing progressive gradient flow techniques for N:M structured sparsity in transformers, it outperform existing methods on terms of model accuracy. While structured pruning aligns well with modern GPU requirements, achieving a high compression ratio and maintaining good model performance simultaneously is challenging. Unstructured pruning, on the other hand, offers higher flexibility, allowing for better compression ratios and model performance. SparseGPT [183] achieves up to 50% weight sparsity in GPT models through optimal partial updates and adaptive masked selection. However, while this method reduces memory usage, it may not efficiently reduce computational complexity on common deep learning accelerators optimized for dense workloads. Unstructured pruning requires customized hardware support to efficiently utilize sparsity.

To this end, several hardware accelerators have been proposed to efficiently process the sparse matrix multiplication resulting from unstructured weight pruning. For instance, the Dual-Side Sparse Tensor Core (DS-STC) [176] modifies tensor core architecture on GPUs to support dual-side sparse matrix multiplication with arbitrary sparsity, outperforming NVIDIA's sparse tensor core on pruned BERT models. DS-STC changes the dataflow of the tensor core from inner-product to outer-product, making it more suitable for arbitrary sparse computation. Meanwhile, the Row-Merge Sparse Tensor Core (RM-STC) [184] proposes using row merge dataflow for dual-side sparse matrix multiplication, further improving upon DS-STC to achieve high efficiency across all levels of sparsity and reducing the hardware overhead in the design.

Sparse Attention. Sparse attention is applied to the Multi-Headed Self Attention (MHSA) module in transformers. By limiting the tokens that attend to each other and ignoring the computation of certain attention scores, the complexity and memory access of MHSA are reduced. The sparsity pattern of sparse attention can be defined online or offline, diverging into static sparse attention, which is agnostic to the input data, and dynamic sparse attention, which depends on the input.

Static sparse attention applies pre-defined attention masks to set the corresponding attention scores to zero during inference. The static sparse pattern usually includes local, global, and random attention. In local attention, tokens attend only to their neighbors within a fixed window. In global attention, certain tokens attend to all other tokens, regardless of their position. In random attention, tokens attend to a set of random tokens, covering various types of dependencies. Longformer [204] utilizes a combination of local attention and global attention to specific tokens, while BigBird [205] further adds random attention on top of local and global attention, demonstrating its ability to encompass all sequence-to-sequence functions. Static sparse attention changes the operations in MHSA from

TABLE II
LLM ACCELERATORS FOR INFERENCE.

Name	Platform	Model	Energy efficiency (TOPS/W)	Quantization/Sparsity	Year
TranCIM [185]	ASIC tapeout 28nm	BERT	20.5 (INT8)	Sparsity	2022
DFX [186]	FPGA	GPT-2			2022
X-Former [187]	PIM simulator 32nm	BERT	13.44 (INT8)	-	2022
DOTA [188]	ASIC 22nm/simulator	GPT-2	-	Quantization/Sparsity	2022
SPRINT [189]	PIM simulator 32nm	GPT-2/BERT	19.6x	Sparsity	2022
TransPIM [190]	PIM 65nm/simulator	GPT-2/BERT	666.6x RTX 2080Ti	-	2022
Mokey [164]	ASIC 65nm/simulator	BERT	9x GOBO (FP16)	Quantization	2022
LeOPArd [191]	ASIC 65nm/simulator	GPT-2/BERT	3x SpAtten	Quantization/Sparsity	2022
STP [192]	ASIC tapeout 12nm	BERT	18.1 (FP4)	Quantization	2023
HAIMA [193]	PIM simulator 45nm	BERT	-	-	2023
TF-MVP [194]	ASIC 28nm	BERT/GPT-2	0.48 (FP16)	Sparsity	2023
TiC-SAT [195]	gem5-X	BERT	-	-	2023
Transformer-OPU [196]	FPGA	BERT	-	-	2023
FACT [197]	ASIC 28nm	BERT	94.88x V100	Quantization/Sparsity	2023
TaskFusion [198]	ASIC 22nm/simulator	BERT	19.83x Jetson Nano	Sparsity	2023
OliVe [165]	ASIC 22nm/simulator	BERT/GPT-2/OPT	4x GOBO	Quantization	2023
C-Transformer [199]	ASIC tapeout 28nm	GPT-2	33.4 (INT8)	-	2024
SpecPIM [200]	PIM simulator	LLaMA/OPT	6.67x A100 (FP16)	-	2024
ASADI [201]	PIM simulator	BERT/GPT-2	- (FP32)	Sparsity	2024
AttAcc [202]	PIM simulator	LLaMA/GPT-3	2.67x DGX A100 (FP16)	-	2024
NeuPIMs [203]	PIM simulator 22nm	GPT-3	-	-	2024

GEMM to SDDMM and SpMM. To efficiently perform these operations, sparse NVPIM [206] is proposed to efficiently map sparse attention on processing-in-memory architecture.

For dynamic sparse attention, it removes activations in the attention map according to the value of activations, requiring real-time detection of activations. Algorithm and hardware co-design is often used to efficiently determine the sparsity pattern and compute the sparse attention [188], [191], [198], [201], [207]. SpAtten [207] measures the cumulative importance of tokens or heads and prunes the tokens or heads on the fly. The entire token or head is eliminated to preserve a structured sparsity pattern, making computation easier. SpAtten also proposes a parallel top-k engine to identify the sparse pattern. DOTA [188] proposes a lightweight detector to omit weak attention score during runtime, inducing a finer-grained sparsity compared with SpAtten and introduces a reconfigurable matrix multiplication unit to cope with the dynamic sparsity pattern. ASADI [201] introduces a new sparse matrix computation paradigm tailored for the DIA format in self-attention tasks, supported by a highly parallel in-situ computing architecture.

In summary, sparsification in LLMs, through techniques such as weight pruning and sparse attention, enhances efficiency and reduces computational complexity. However, unlike quantization, the efficiency gains from sparsity are not straightforward and require careful hardware considerations to achieve significant improvements. Unstructured sparsity offers good compression ratios and maintains accuracy, but it necessitates dedicated hardware designs. While proposed solutions for unstructured sparsity are effective, they inevitably introduce additional hardware overhead to manage irregularities. Consequently, in current practices for efficient LLM processing, structured sparsity is often favored. It introduces a degree of regularity that allows for more efficient parallel processing, striking a balance between performance gains and hardware cost. Sparsity and quantization are usually combined

to compress LLMs in practice. [208] provides both theoretical and empirical evidence on the optimal way to combine sparsity and quantization in deep neural networks, offering valuable insights for model compression and efficient deployment of large language models.

D. Accelerators for Inference

The use of LLMs for complex language tasks is exceptionally data- and computation-intensive. As a result, there is a strong need for energy-efficient, dedicated processors to minimize these costs, especially on power-constrained edge devices. The solutions to achieving this goal and boosting the efficiency of LLM inference involves advancements to both hardware and algorithms, and this research is summarized in Table II. Some of these accelerators, which were introduced in the previous subsection, focus on compression techniques such as sparsity and quantization. Here, we will present some representative accelerators.

Hardware Acceleration. On the hardware side, numerous research efforts have been focused on investigating how to take the advantages of novel architectures to minimize costly data movement and enhance computational parallelism. For instance, TranCIM [185] follows the non von-Neumann compute-in-memory (CIM) architecture. The digital SRAM-based Bitline-Transpose-CIM macro is introduced to process multiply-accumulate (MAC) operations. By performing MAC operations locally within the SRAM array, CIM macros eliminate excessive and costly data transference for intermediate data. In the TranCIM macros, the SRAM arrays store the weight matrices and take in the input vectors along the bitlines in the same direction. This avoids the need for transpose buffers on the output side to transpose the generated self-attention matrices. Additionally, all the bitlines in each array are activated simultaneously to perform MACs on different

weights and inputs, thus improving the computation parallelism. CIM macros that execute different matrix multiplications work in a pipeline manner for further efficiency improvement.

Algorithm Acceleration. On the algorithm side, optimizing the LLM computation paradigm with compression techniques and the removal of redundant computations enables the LLM processors to achieve both higher efficiency and better hardware utilization. For example, TranCIM supports dynamically selecting dense attention patterns for computation to fully leverage the sparsity in the workload. Another work, STP [192], exploits the entropy information of input patterns as the criteria to dynamically reconfigure data paths and skip computations of subsequent layers if necessary. This entropy information is also used to customize the local power supply and clock frequency. These techniques lead to a boost in both the throughput and the energy efficiency of the processor with only marginal accuracy loss. PIVOT [209] improves transformer efficiency by dynamically adjusting attention mechanisms based on input complexity, achieving significant reductions in energy consumption. Finally, C-Transformer [199] incorporates conventional LLMs with spiking LLMs and executes workloads in both spiking and non-spiking domains to achieve high sparsity as well as high hardware utilization.

Architecture Design. Researchers have also proposed accelerators to optimize LLM inference at the architectural level. Various studies propose architecture designs that facilitate the execution of sparse attention graphs by skipping unnecessary connections. Specifically, DOTA [188] introduces a detector for attention selection and utilizes a token-parallel data flow for sparse attention computation, enabling key/value reuse. Additionally, SPRINT [189] computes attention scores approximately and prunes low attention scores using lightweight analog thresholding circuitry within the processing element (PE) arrays.

Some studies leverage speculative decoding to accelerate LLM inference. In speculative decoding, a small draft model generates multiple draft tokens, which are later verified in parallel by the target LLM. Meanwhile, SpecPIM [200] finds the optimal resource allocation design through design space exploration, considering the algorithmic and architectural heterogeneity of the draft model and the target LLM.

In addition to digital accelerators, there are efforts to accelerate LLM inference using Processing-In-Memory (PIM) architectures. TransPIM [190] introduces a token-based dataflow for Transformer-based models, which avoids costly inter-layer data movements. Observing that PIM accelerators are more efficient for GEMV computations compared to commercial accelerators like GPUs and TPUs, and that batched decoding alleviates the memory-bound issue of LLM inference on GPUs to some extent, AttAcc [202] and NeuPIMs [203] propose heterogeneous xPU/PIM systems for batched LLM inference. These systems accelerate the attention mechanism on PIM accelerators while assigning other computations to xPUs.

Beyond off-loading scenarios, some studies focus on accelerating LLM inference through distributed systems. For instance, DFX [186] employs model parallelism and an efficient network within a multi-FPGA system, resulting in minimal data synchronization between FPGAs. In another study, the authors of CXL-PNM [210] introduce a processing near memory (PNM) platform using Compute Express Link (CXL), leveraging both model parallelism and data parallelism for workload partitioning.

E. Industry-led AI accelerators

While numerous hardware and algorithmic accelerations have been proposed in academia, they are limited to certain applications and uses. In the scope of this paper, we are focusing primarily on LLM models such as BERT and GPT-2. This is quite different than industry applications since the target is not only LLMs, but all AI workloads in general. As a result, architecture design must consider more specific requirements. IBM presented RaPiD [211], [212], which is a fabricated AI accelerator chip designed for ultra-low precision training and inference. It supports a spectrum of precisions, including the lowest 2-bit fixed point. By utilizing precision scaling, performance and energy improvements are achieved in AI workloads ranging from VGG-16 to BERT. The latest work from IBM, NorthPole [213], [214] differs from their previous works, which were categorized as ASIC accelerators. NorthPole targets large-scale workload, comparing against Google TPUv4 [215], NVIDIA A100 & H100 AI processors. The performance achieved is a joint effort of both architecture and their SDK toolchain. This software-assisted approach is also implemented in their work [216].

At the same time, Microsoft announced their first in-house AI accelerator, Azure Maia 100, to facilitate their cloud-based AI workloads. It is designed for scalability and sustainability through end-to-end system optimization. It is equipped with a fully custom network protocol and a comprehensive AI framework environment. Cerebras, known for wafer-scale computing, provided a guide for software-hardware co-design for deep learning [217]. It is clear that LLMs have been developing rapidly from 100 million parameters in BERT to 175 billion parameters in GPT-3 in just a few years. To keep up with the growth of extreme-scale ML models, they proposed a new chip architecture that is wafer-sized.

While numerous LLM accelerators have been proposed in academia, they are mostly for inference and are restricted to certain models. In other words, they are not generalized for different workloads. However, they are able to leverage the unique datapath or observations found in a specific workload to accelerate the computation and achieve power efficiency at the same time. Industry-led AI accelerators, on the other hand, focus on a different perspective. While these accelerations are appreciated, they aren't the general case. Whether it is for edge or cloud-based AI computation, customer targets are very diverse, so accelerators have to be designed to take all AI workloads into consideration. This includes not only inference but also efficient training. Even though the approach for academia and industry is different, both share their critical goal of accelerating LLMs to improve accuracy, power efficiency, latency, and scalability.

F. Other optimizations

Spiking Transformers. Merging biologically plausible structures, Spiking Transformers have emerged as an innovative approach to integrating Spiking Neural Networks (SNNs) with Transformer architectures. Spiking Transformers have achieved notable advancements in both performance and energy efficiency. Spikformer [218] was the first to implement spiking self-attention in Transformers. It introduced Spikformer, pioneering Spiking Self Attention (SSA) blocks to eliminate resource-intensive multiplications and SoftMax operations. Following this, Masked Spiking Transformers were realized, utilizing Random Spike Masking (RSM) to reduce redundant spikes effectively [219]. Spikingformer demonstrated further innovation [220], incorporating spike-driven residual learning within Transformer-based SNNs. These advancements have been further augmented by the realization of C-Transformer [199], a processor designed to accelerate Spiking Transformer operations and LLMs. It accelerates Spiking Transformers by integrating a Hybrid Multiplication-Accumulation Unit (HMAU), which does accumulation for spiking operations. The Output Spike Speculation Unit (OSSU) further enhances efficiency by speculating the output spikes, making the architecture ideal for accelerating spiking neural network operation.

Emerging Accelerator Designs. The following notable studies contribute unique methodologies to the field of in-memory computing [221]–[230], specifically showcasing innovations that optimize the efficiency and performance of Transformer models through unique approaches. Building on FloatPIM's demonstration of the feasibility of high-precision in-memory acceleration of DNN training [228], recent work presented RIME [223], an RRAM-based in-memory floating-point computation architecture aimed at accelerating Transformer inference. RIME employs single-cycle NOR, NAND, and innovative minority (Min3) logic functions within RRAM cells to perform high-precision floating-point operations directly in memory. Its novel Min3-based adder enables 32-bit floating-point multiplication with minimal cycle count, area and energy consumption. RACE-IT [230] is a reconfigurable analog content-addressable memory and crossbar engine designed for in-memory Transformer acceleration. The core innovation is the Compute-ACAM unit, which performs various non-matrix-vector-multiplication operations within the analog domain using analog content-addressable memories (CAMs), significantly improving computation efficiency. There is also research about methodologies of in-memory computing for transformers. TReX [221] proposes a novel approach to optimize Transformers for In-Memory Computing architectures by reusing attention blocks, leading to significant improvements in energy efficiency and area utilization while maintaining high accuracy. ClipFormer [224] deals with the noise of the memristive crossbar by clipping the KV matrices of Transformers during inference. These studies highlight recent advancements in in-memory computing architectures for Transformer models.

V. CONCLUSION

This survey has examined the multifaceted challenges and opportunities associated with LLMs. We have delved into various aspects of LLM training, inference, and system integration, highlighting the need for specialized hardware and software solutions. By synthesizing the latest research, we aim to comprehensively understand the trade-offs and design considerations crucial for developing efficient LLM-centric computing systems. As LLMs continue to evolve and integrate into diverse applications, future research must focus on optimizing their performance and sustainability. This will involve advancing the current methodologies and developing new strategies to enhance their efficiency and practicality. Ultimately, the insights gained from this survey can pave the way for future breakthroughs, enabling the creation of more powerful, efficient, and sustainable LLM systems.

REFERENCES

[1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.09288

[7] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: memory optimizations toward training trillion parameter models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*. IEEE/ACM, 2020, p. 20. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00024

[10] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *IEEE Micro*, 2024.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[13] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[14] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, pp. 1–32, 2024.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[16] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[18] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[19] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[20] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[21] Q. Xu and Y. You, "An efficient 2d method for training super-large deep learning models," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 222–232.

[22] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[23] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[24] M. S. Johnstone and P. R. Wilson, "The memory fragmentation problem: Solved?" *ACM Sigplan Notices*, vol. 34, no. 3, pp. 26–36, 1998.

[25] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "Zero-offload: Democratizing billion-scale model training," in *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. USENIX Association, 2021, pp. 551–564.

[26] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: breaking the GPU memory wall for extreme scale deep learning," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*. ACM, 2021, p. 59. [Online]. Available: https://doi.org/10.1145/3458817.3476205

[27] G. Wang, H. Qin, S. A. Jacobs, C. Holmes, S. Rajbhandari, O. Ruwase, F. Yan, L. Yang, and Y. He, "Zero++: Extremely efficient collective communication for giant model training," *CoRR*, vol. abs/2306.10209, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.10209

[28] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[29] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein, "Register allocation via coloring," *Comput. Lang.*, vol. 6, no. 1, pp. 47–57, 1981. [Online]. Available: https://doi.org/10.1016/0096-0551(81)90048-5

[30] J. Herrmann, O. Beaumont, L. Eyraud-Dubois, J. Hermann, A. Joly, and A. Shilova, "Optimal checkpointing for heterogeneous chains: how to train deep neural networks with limited memory," *arXiv preprint arXiv:1911.13214*, 2019.

[31] C. Guo, R. Zhang, J. Xu, J. Leng, Z. Liu, Z. Huang, M. Guo, H. Wu, S. Zhao, J. Zhao *et al.*, "Gmlake: Efficient and transparent gpu memory defragmentation for large-scale dnn training with virtual memory stitching," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 450–466.

[32] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[33] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented

[34] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[35] "Torchtune," https://pytorch.org/torchtune/stable/index.html.

[36] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[37] L. Hedegaard, A. Alok, J. Jose, and A. Iosifidis, "Structured pruning adapters," *arXiv preprint arXiv:2211.10155*, 2022.

[38] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, and B. Zhuang, "Loraprune: Pruning meets low-rank parameter-efficient fine-tuning," 2024. [Online]. Available: https://arxiv.org/abs/2305.18403

[39] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=OUIFPHEgJU

[40] H. Guo, P. Greengard, E. Xing, and Y. Kim, "LQ-loRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=xw29VvOMmU

[41] Y. Xu, L. Xie, X. Gu, X. Chen, H. Chang, H. Zhang, Z. Chen, X. ZHANG, and Q. Tian, "QA-loRA: Quantization-aware low-rank adaptation of large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=WvFoJccpo8

[42] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[43] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," *arXiv preprint arXiv:2001.07676*, 2020.

[44] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," *arXiv preprint arXiv:2012.15723*, 2020.

[45] T.-X. Sun, X.-Y. Liu, X.-P. Qiu, and X.-J. Huang, "Paradigm shift in natural language processing," *Machine Intelligence Research*, vol. 19, no. 3, pp. 169–183, 2022.

[46] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[47] K. Hambardzumyan, H. Khachatrian, and J. May, "Warp: Word-level adversarial reprogramming," *arXiv preprint arXiv:2101.00121*, 2021.

[48] G. Qin and J. Eisner, "Learning how to ask: Querying lms with mixtures of soft prompts," *arXiv preprint arXiv:2104.06599*, 2021.

[49] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "Gpt understands, too," *AI Open*, 2023.

[50] Z. Zhong, D. Friedman, and D. Chen, "Factual probing is [mask]: Learning vs. learning to recall," *arXiv preprint arXiv:2104.05240*, 2021.

[51] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," *arXiv preprint arXiv:2110.07602*, 2021.

[52] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[53] J. Sun, Z. Xu, H. Yin, D. Yang, D. Xu, Y. Chen, and H. R. Roth, "Fedbpt: Efficient federated black-box prompt tuning for large language models," *arXiv preprint arXiv:2310.01467*, 2023.

[54] X. Liu, K. Ji, Y. Fu, W. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2022, pp. 61–68.

[55] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[56] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnaswamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[57] "Faiss," https://ai.meta.com/tools/faiss/.

scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[58] D. Guo, A. Rush, and Y. Kim, "Parameter-efficient transfer learning with diff pruning," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 4884–4896. [Online]. Available: https://aclanthology.org/2021.acl-long.378

[59] B. Liao, Y. Meng, and C. Monz, "Parameter-efficient fine-tuning without introducing new latency," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4242–4260. [Online]. Available: https://aclanthology.org/2023.acl-long.233

[60] Y.-L. Sung, V. Nair, and C. Raffel, "Training neural networks with fixed sparse masks," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=Uwh-v1HSw-x

[61] NVIDIA, "Nvidia a100 tensor core gpu architecture," Tech. Rep., 2020.

[62] ——, "Nvidia h100 tensor core gpu architecture," Tech. Rep., 2022.

[63] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[64] Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[65] H. Jang, J. Song, J. Jung, J. Park, Y. Kim, and J. Lee, "Smart-infinity: Fast large language model training using near-storage processing on a real system," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 345–360.

[66] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[67] M. Mahmoud, I. Edo, A. H. Zadeh, O. M. Awad, G. Pekhimenko, J. Albericio, and A. Moshovos, "Tensordash: Exploiting sparsity to accelerate deep neural network training," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 781–795.

[68] S. Q. Zhang, B. McDanel, and H. Kung, "Fast: Dnn training under variable precision block floating point with stochastic rounding," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 846–860.

[69] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang *et al.*, "Cambricon-q: A hybrid architecture for efficient training," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 706–719.

[70] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[71] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[72] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

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

[74] T. Dao, D. Haziza, F. Massa, and G. Sizov, "Flash-decoding for long-context inference," [Online], 2023, https://crfm.stanford.edu/2023/10/12/flashdecoding.html.

[75] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, and Y. Wang, "Flashdecoding++: Faster large language model inference on gpus," 2024.

[76] N. Vaidya, F. Oh, and N. Comly, "Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available," [Online], 2023, https://github.com/NVIDIA/TensorRT-LLM.

[77] Y. Zhai, C. Jiang, L. Wang, X. Jia, S. Zhang, Z. Chen, X. Liu, and Y. Zhu, "Bytetransformer: A high-performance transformer boosted for variable-length inputs," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 344–355.

[78] B. Lefaudeux, F. Massa, D. Liskovich, W. Xiong, V. Caggiano, S. Naren, M. Xu, J. Hu, M. Tintore, S. Zhang, P. Labatut, D. Haziza, L. Wehrstedt, J. Reizenstein, and G. Sizov, "xformers: A modular and hackable transformer modelling library," https://github.com/facebookresearch/xformers, 2022.

[79] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[80] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[81] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," *arXiv preprint arXiv:2312.11514*, 2023.

[82] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu

[83] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills," *arXiv preprint arXiv:2308.16369*, 2023.

[84] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, , and R. Ramjee, "Taming throughput-latency tradeoff in llm inference with sarathi-serve," *arXiv preprint arXiv:2403.02310*, 2024.

[85] ModelTC, "Lightllm," February 2024. [Online]. Available: https://github.com/ModelTC/lightllm/

[86] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "*s*3: Increasing gpu utilization during generative inference for higher throughput," *Advances in Neural Information Processing Systems*, vol. 36, pp. 18 015–18 027, 2023.

[87] B. Wu, Y. Zhong, Z. Zhang, G. Huang, X. Liu, and X. Jin, "Fast distributed inference serving for large language models," *arXiv preprint arXiv:2305.05920*, 2023.

[88] Y. Sheng, S. Cao, D. Li, B. Zhu, Z. Li, and D. Zhuo, "Fairness in serving large language models," *arXiv preprint arXiv:2401.00588*, 2024.

[89] X. Miao, C. Shi, J. Duan, X. Xi, D. Lin, B. Cui, and Z. Jia, "Spotserve: Serving generative large language models on preemptible instances," *arXiv preprint arXiv:2311.15566*, 2023.

[90] P. Patel, E. Choukse, C. Zhang, Íñigo Goiri, A. Shah, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," *arXiv preprint arXiv:2311.18677*, 2023.

[91] C. Hu, H. Huang, L. Xu, X. Chen, J. Xu, S. Chen, H. Feng, C. Wang, S. Wang, Y. Bao, N. Sun, and Y. Shan, "Inference without interference: Disaggregate llm inference for mixed downstream workloads," *arXiv preprint arXiv:2401.11181*, 2024.

[92] Z. Xue, Y. Song, Z. Mi, L. Chen, Y. Xia, and H. Chen, "Powerinfer-2: Fast large language model inference on a smartphone," *arXiv preprint arXiv:2406.06282*, 2024.

[93] W. Yin, M. Xu, Y. Li, and X. Liu, "Llm as a system service on mobile devices," *arXiv preprint arXiv:2403.11805*, 2024.

[94] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[95] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," *arXiv preprint arXiv:2312.12456*, 2023.

[96] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re *et al.*, "Deja vu: Contextual sparsity for efficient llms at inference time," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 137–22 176.

[97] Y. Song, H. Xie, Z. Zhang, B. Wen, L. Ma, Z. Mi, and H. Chen, "Turbo sparse: Achieving llm sota performance with minimal activated parameters," *arXiv preprint arXiv:2406.05955*, 2024.

[98] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[99] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[100] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," *arXiv preprint arXiv:1312.4314*, 2013.

[101] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[102] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[103] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[104] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[105] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[106] A. Clark, D. de Las Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud *et al.*, "Unified scaling laws for routed language models," in *International conference on machine learning*. PMLR, 2022, pp. 4057–4086.

[107] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, and W. Fedus, "St-moe: Designing stable and transferable sparse expert models," *arXiv preprint arXiv:2202.08906*, 2022.

[108] D. Raposo, S. Ritter, B. Richards, T. Lillicrap, P. C. Humphreys, and A. Santoro, "Mixture-of-depths: Dynamically allocating compute in transformer-based language models," *arXiv preprint arXiv:2404.02258*, 2024.

[109] F. Xue, Z. Shi, F. Wei, Y. Lou, Y. Liu, and Y. You, "Go wider instead of deeper," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8779–8787.

[110] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby, "Scaling vision with sparse mixture of experts," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8583–8595, 2021.

[111] J. Obando-Ceron, G. Sokar, T. Willi, C. Lyle, J. Farebrother, J. Foerster, G. K. Dziugaite, D. Precup, and P. S. Castro, "Mixtures of experts unlock parameter scaling for deep rl," *arXiv preprint arXiv:2402.08609*, 2024.

[112] Z. Zhang, Y. Lin, Z. Liu, P. Li, M. Sun, and J. Zhou, "Moefication: Transformer feed-forward layers are mixtures of experts," *arXiv preprint arXiv:2110.01786*, 2021.

[113] T. Zhu, X. Qu, D. Dong, J. Ruan, J. Tong, C. He, and Y. Cheng, "Llama-moe: Building mixture-of-experts from llama with continual pre-training," *arXiv preprint arXiv:2406.16554*, 2024.

[114] M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer, "Base layers: Simplifying training of large, sparse models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6265–6274.

[115] S. Roller, S. Sukhbaatar, J. Weston *et al.*, "Hash layers for large sparse models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 555–17 566, 2021.

[116] D. Dai, L. Dong, S. Ma, B. Zheng, Z. Sui, B. Chang, and F. Wei, "Stablemoe: Stable routing strategy for mixture of experts," *arXiv preprint arXiv:2204.08396*, 2022.

[117] J. Puigcerver, C. Riquelme, B. Mustafa, and N. Houlsby, "From sparse to soft mixtures of experts," *arXiv preprint arXiv:2308.00951*, 2023.

[118] Z. Zhong, M. Xia, D. Chen, and M. Lewis, "Lory: Fully differentiable mixture-of-experts for autoregressive language model pre-training," *arXiv preprint arXiv:2405.03133*, 2024.

[119] L. Shen, Z. Wu, W. Gong, H. Hao, Y. Bai, H. Wu, X. Wu, J. Bian, H. Xiong, D. Yu *et al.*, "Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system," *arXiv preprint arXiv:2205.10034*, 2022.

[120] Z. Fan, R. Sarkar, Z. Jiang, T. Chen, K. Zou, Y. Cheng, C. Hao, Z. Wang *et al.*, "M³vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 441–28 457, 2022.

[121] Z. Du, S. Li, Y. Wu, X. Jiang, J. Sun, Q. Zheng, Y. Wu, A. Li, H. Li, and Y. Chen, "Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 224–238, 2024.

[122] H. Wang, F. M. Polo, Y. Sun, S. Kundu, E. Xing, and M. Yurochkin, "Fusing models with complementary expertise," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=PhMrGCMIRL

[123] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *arXiv preprint arXiv:1911.02150*, 2019.

[124] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[125] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[126] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. A. Smith, and L. Kong, "Random feature attention," *arXiv preprint arXiv:2103.02143*, 2021.

[127] P. Kacham, V. Mirrokni, and P. Zhong, "Polysketchformer: Fast transformers via sketches for polynomial kernels," *arXiv preprint arXiv:2310.01655*, 2023.

[128] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[129] S. Zhang, N. Meng, and E. Y. Lam, "Lrt: an efficient low-light restoration transformer for dark light field images," *IEEE Transactions on Image Processing*, 2023.

[130] A. Gupta, Y. Yuan, Y. Zhou, and C. Mendis, "Flurka: Fast fused low-rank & kernel attention," *arXiv preprint arXiv:2306.15799*, 2023.

[131] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 274–19 286.

[132] S. Kim, K. Mangalam, S. Moon, J. Malik, M. W. Mahoney, A. Gholami, and K. Keutzer, "Speculative decoding with big little decoder," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[133] Z. Sun, A. T. Suresh, J. H. Ro, A. Beirami, H. Jain, and F. Yu, "Spectr: Fast speculative decoding via optimal transport," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[134] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia, "Specinfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 932–949. [Online]. Available: https://doi.org/10.1145/3620666.3651335

[135] D. Xu, W. Yin, X. Jin, Y. Zhang, S. Wei, M. Xu, and X. Liu, "Llmcad: Fast and scalable on-device large language model inference," *arXiv preprint arXiv:2309.04255*, 2023.

[136] Y. Zhou, K. Lyu, A. S. Rawat, A. K. Menon, A. Rostamizadeh, S. Kumar, J.-F. Kagy, and R. Agarwal, "Distillspec: Improving speculative decoding via knowledge distillation," *arXiv preprint arXiv:2310.08461*, 2023.

[137] X. Liu, L. Hu, P. Bailis, I. Stoica, Z. Deng, A. Cheung, and H. Zhang, "Online speculative decoding," *arXiv preprint arXiv:2310.07177*, 2023.

[138] J. Zhang, J. Wang, H. Li, L. Shou, K. Chen, G. Chen, and S. Mehrotra, "Draft & verify: Lossless large language model acceleration via self-speculative decoding," *arXiv preprint arXiv:2309.08168*, 2023.

[139] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *Advances in neural information processing systems*, vol. 33, pp. 1474–1487, 2020.

[140] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.

[141] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[142] J. T. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," *arXiv preprint arXiv:2208.04933*, 2022.

[143] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[144] T. Dao and A. Gu, "Transformers are ssms: Generalized models and efficient algorithms through structured state space duality," *arXiv preprint arXiv:2405.21060*, 2024.

[145] J. Park, J. Park, Z. Xiong, N. Lee, J. Cho, S. Oymak, K. Lee, and D. Papailiopoulos, "Can mamba learn how to learn? a comparative study on in-context learning tasks," *arXiv preprint arXiv:2402.04248*, 2024.

[146] O. Lieber, B. Lenz, H. Bata, G. Cohen, J. Osin, I. Dalmedigos, E. Safahi, S. Meirom, Y. Belinkov, S. Shalev-Shwartz *et al.*,

"Jamba: A hybrid transformer-mamba language model," *arXiv preprint arXiv:2403.19887*, 2024.

[147] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[148] H. Liu, C. Li, Y. Li, B. Li, Y. Zhang, S. Shen, and Y. J. Lee, "Llava-next: Improved reasoning, ocr, and world knowledge," January 2024. [Online]. Available: https://llava-vl.github.io/blog/2024-01-30-llava-next/

[149] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira, "Perceiver: General perception with iterative attention," in *International conference on machine learning*. PMLR, 2021, pp. 4651–4664.

[150] H. Laurençon, L. Tronchon, M. Cord, and V. Sanh, "What matters when building vision-language models?" 2024.

[151] Y. Shang, M. Cai, B. Xu, Y. J. Lee, and Y. Yan, "Llava-prumerge: Adaptive token reduction for efficient large multimodal models," *arXiv preprint arXiv:2403.15388*, 2024.

[152] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[153] Z. Wang, J. Lu, C. Tao, J. Zhou, and Q. Tian, "Learning channel-wise interactions for binary convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 568–577.

[154] B. Zhuang, M. Tan, J. Liu, L. Liu, I. Reid, and C. Shen, "Effective training of convolutional neural networks with low-bitwidth weights and activations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[155] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[156] C. Guo, Y. Qiu, J. Leng, X. Gao, C. Zhang, Y. Liu, F. Yang, Y. Zhu, and M. Guo, "SQuant: On-the-fly data-free quantization via diagonal hessian approximation," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=JXhROKNZzOc

[157] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[158] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," in *MLSys*, 2024.

[159] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," 2024. [Online]. Available: https://arxiv.org/abs/2404.00456

[160] J. Chee, Y. Cai, V. Kuleshov, and C. D. Sa, "Quip: 2-bit quantization of large language models with guarantees," 2024. [Online]. Available: https://arxiv.org/abs/2307.13304

[161] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Es-maeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[162] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[163] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[164] A. H. Zadeh, M. Mahmoud, A. Abdelhadi, and A. Moshovos, "Mokey: enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York,

NY, USA: Association for Computing Machinery, 2022, p. 888–901. [Online]. Available: https://doi.org/10.1145/3470496.3527438

[165] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589038

[166] B. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. C. Koppaka, X. Song, S. Som, K. Das, S. Tiwary, S. Reinhardt, S. Lanka, E. Chung, and D. Burger, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[167] H. Xia, Z. Zheng, X. Wu, S. Chen, Z. Yao, S. Youn, A. Bakhtiari, M. Wyatt, D. Zhuang, Z. Zhou, O. Ruwase, Y. He, and S. L. Song, "Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design," 2024. [Online]. Available: https://arxiv.org/abs/2401.14112

[168] S.-y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng, "Llm-fp4: 4-bit floating-point quantized transformers," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023. [Online]. Available: http://dx.doi.org/10.18653/v1/2023.emnlp-main.39

[169] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3.int8(): 8-bit matrix multiplication for transformers at scale," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 30 318–30 332.

[170] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[171] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[172] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[173] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 15–28.

[174] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 359–371.

[175] C. Guo, B. Y. Hsueh, J. Leng, Y. Qiu, Y. Guan, Z. Wang, X. Jia, X. Li, M. Guo, and Y. Zhu, "Accelerating sparse dnn models without hardware-support via tile-wise sparsity," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[176] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, "Dual-side sparse tensor core," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1083–1095.

[177] Y. Guan, C. Yu, Y. Zhou, J. Leng, C. Li, and M. Guo, "Fractal: Joint multi-level sparse pattern tuning of accuracy and performance for dnn pruning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 416–430.

[178] C. Guo, F. Xue, J. Leng, Y. Qiu, Y. Guan, W. Cui, Q. Chen, and M. Guo, "Accelerating sparse dnns based on tiled gemm," *IEEE Transactions on Computers*, 2024.

[179] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.

[180] Y. Zhang, H. Bai, H. Lin, J. Zhao, L. Hou, and C. V. Cannistraci, "Plug-and-play: An efficient post-training pruning method for large language models," in *The Twelfth International Conference on Learning Representations*, 2024.

[181] M. Mozaffari, A. Yazdanbakhsh, Z. Zhang, and M. M. Dehnavi, "Slope: Double-pruned sparse plus lazy low-rank adapter pretraining of llms," *arXiv preprint arXiv:2405.16325*, 2024.

[182] A. R. Bambhaniya, A. Yazdanbakhsh, S. Subramanian, S.-C. Kao, S. Agrawal, U. Evci, and T. Krishna, "Progressive gradient flow for robust n: M sparsity training in transformers," *arXiv preprint arXiv:2402.04744*, 2024.

[183] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 323–10 337.

[184] G. Huang, Z. Wang, P.-A. Tsai, C. Zhang, Y. Ding, and Y. Xie, "Rm-stc: Row-merge dataflow inspired gpu sparse tensor core for energy-efficient sparse acceleration," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 338–352.

[185] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "A 28nm 15.59 μj/token full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 466–468.

[186] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–17.

[187] S. Sridharan, J. R. Stevens, K. Roy, and A. Raghunathan, "X-former: In-memory acceleration of transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 8, pp. 1223–1233, 2023.

[188] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 14–26. [Online]. Available: https://doi.org/10.1145/3503222.3507738

[189] A. Yazdanbakhsh, A. Moradifirouzabadi, Z. Li, and M. Kang, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '22. IEEE Press, 2023, p. 744–762. [Online]. Available: https://doi.org/10.1109/MICRO56248.2022.00059

[190] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071–1085.

[191] Z. Li, S. Ghodrati, A. Yazdanbakhsh, H. Esmaeilzadeh, and M. Kang, "Accelerating attention through gradient-based learned runtime pruning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 902–915. [Online]. Available: https://doi.org/10.1145/3470496.3527423

[192] T. Tambe, J. Zhang, C. Hooper, T. Jia, P. N. Whatmough, J. Zuckerman, M. C. Dos Santos, E. J. Loscalzo, D. Giri, K. Shepard *et al.*, "22.9 a 12nm 18.1 tflops/w sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 342–344.

[193] Y. Ding, C. Liu, M. Duan, W. Chang, K. Li, and K. Li, "Haima: A hybrid sram and dram accelerator-in-memory architecture for transformer," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[194] E. Yoo, G. Park, J. G. Min, S. Jung Kwon, B. Park, D. Lee, and Y. Lee, "Tf-mvp: Novel sparsity-aware transformer accelerator with mixed-length vector pruning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[195] A. Amirshahi, J. A. Harrison Klein, G. Ansaloni, and D. Atienza, "Tic-sat: Tightly-coupled systolic accelerator for transformers," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023, pp. 657–663.

[196] Y. Bai, H. Zhou, K. Zhao, J. Chen, J. Yu, and K. Wang, "Transformer-opu: An fpga-based overlay processor for transformer networks," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 221–221.

[197] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589057

[198] Z. Fan, Q. Zhang, P. Abillama, S. Shoouri, C. Lee, D. Blaauw, H.-S. Kim, and D. Sylvester, "Taskfusion: An efficient transfer learning architecture with dual delta sparsity for multi-task natural language processing," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589040

[199] S. Kim, S. Kim, W. Jo, S. Kim, S. Hong, and H.-J. 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)*, vol. 67. IEEE, 2024, pp. 368–370.

[200] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. 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*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 950–965. [Online]. Available: https://doi.org/10.1145/3620666.3651352

[201] H. Li, Z. Li, Z. Bai, and T. Mitra, "Asadi: Accelerating sparse attention using diagonal-based in-situ computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2024, pp. 774–787. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA57654.2024.00065

[202] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. 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*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 103–119. [Online]. Available: https://doi.org/10.1145/3620665.3640422

[203] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. 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*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 722–737. [Online]. Available: https://doi.org/10.1145/3620666.3651380

[204] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[205] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[206] Q. Zheng, S. Li, Y. Wang, Z. Li, Y. Chen, and H. H. Li, "Accelerating sparse attention with a reconfigurable non-volatile processing-in-memory architecture," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[207] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.

[208] S. B. Harma, A. Chakraborty, E. Kostenok, D. Mishin, D. Ha, B. Falsafi, M. Jaggi, M. Liu, Y. Oh, S. Subramanian *et al.*, "Effective interplay between sparsity and quantization: From theory to practice," *arXiv preprint arXiv:2405.20935*, 2024.

[209] A. Moitra, A. Bhattacharjee, and P. Panda, "Pivot-input-aware path selection for energy-efficient vit inference," *arXiv preprint arXiv:2404.15185*, 2024.

[210] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo, N. Kim, Y. Lee, H. Kim, Y. Kwon, J. Kim, J. Lee, Y. Cho, Y. Tai, J. Cho, H. Song, J. H. Ahn, and N. S. Kim, "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)*, 2024, pp. 970–982.

[211] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran, Y. Li, E. Ogawa, K. Ishizaki, H. Inoue, M. Schaal, M. Serrano, J. Choi, X. Sun, N. Wang, C.-Y. Chen, A. Allain, J. Bonano, N. Cao, R. Casatuta, M. Cohen, B. Fleischer, M. Guillorn, H. Haynie, J. Jung, M. Kang, K.-h. Kim,

S. Koswatta, S. Lee, M. Lutz, S. Mueller, J. Oh, A. Ranjan, Z. Ren, S. Rider, K. Schelm, M. Scheuermann, J. Silberman, J. Yang, V. Zalani, X. Zhang, C. Zhou, M. Ziegler, V. Shah, M. Ohara, P.-F. Lu, B. Curran, S. Shukla, L. Chang, and K. Gopalakrishnan, "Rapid: Ai accelerator for ultra-low precision training and inference," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 153–166.

[212] S. K. Lee, A. Agrawal, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen, S. M. Mueller, J. Oh, M. Lutz, J. Jung, S. Koswatta, C. Zhou, V. Zalani, M. Kar, J. Bonanno, R. Casatuta, C.-Y. Chen, J. Choi, H. Haynie, A. Herbert, R. Jain, K.-H. Kim, Y. Li, Z. Ren, S. Rider, M. Schaal, K. Schelm, M. R. Scheuermann, X. Sun, H. Tran, N. Wang, W. Wang, X. Zhang, V. Shah, B. Curran, V. Srinivasan, P.-F. Lu, S. Shukla, K. Gopalakrishnan, and L. Chang, "A 7-nm four-core mixed-precision ai chip with 26.2-tflops hybrid-fp8 training, 104.9-tops int4 inference, and workload-aware throttling," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 182–197, 2022.

[213] A. S. Cassidy, J. V. Arthur, F. Akopyan, A. Andreopoulos, R. Appuswamy, P. Datta, M. V. Debole, S. K. Esser, C. O. Otero, J. Sawada, B. Taba, A. Amir, D. Bablani, P. J. Carlson, M. D. Flickner, R. Gandhasri, G. J. Garreau, M. Ito, J. L. Klamo, J. A. Kusnitz, N. J. McClatchey, J. L. McKinstry, Y. Nakamura, T. K. Nayak, W. P. Risk, K. Schleupen, B. Shaw, J. Sivagnaname, D. F. Smith, I. Terrizzano, T. Ueda, and D. Modha, "11.4 ibm northpole: An architecture for neural network inference with a 12nm chip," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 214–215.

[214] D. S. Modha, F. Akopyan, A. Andreopoulos, R. Appuswamy, J. V. Arthur, A. S. Cassidy, P. Datta, M. V. DeBole, S. K. Esser, C. O. Otero, J. Sawada, B. Taba, A. Amir, D. Bablani, P. J. Carlson, M. D. Flickner, R. Gandhasri, G. J. Garreau, M. Ito, J. L. Klamo, J. A. Kusnitz, N. J. McClatchey, J. L. McKinstry, Y. Nakamura, T. K. Nayak, W. P. Risk, K. Schleupen, B. Shaw, J. Sivagnaname, D. F. Smith, I. Terrizzano, and T. Ueda, "Neural inference at the frontier of energy, space, and time," *Science*, vol. 382, no. 6668, pp. 329–335, 2023. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.adh1174

[215] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[216] M. Kar, J. Silberman, S. Venkataramani, B. Fleischer, J. Rubin, J. Lancaster, S. Lee, M. Cohen, M. Ziegler, N. Cao, S. Woodward, A. Agrawal, C. Zhou, P. Chatarasi, T. Gooding, M. Guillorn, B. Hekmatshoartabari, P. Jacob, R. Jain, S. Jain, J. Jung, K.-H. Kim, S. Koswatta, M. Lutz, A. Mannari, A. Mathew, I. Nair, A. Ranjan, Z. Ren, S. Rider, T. Roewer, D. Satterfield, M. Schaal, S. Sen, G. Tellez, H. Tran, W. Wang, V. Zalani, J. Zhang, X. Zhang, V. Shah, R. Senger, A. Kumar, P.-F. Lu, and L. Chang, "14.1 a software-assisted peak current regulation scheme to improve power-limited inference performance in a 5nm ai soc," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 254–256.

[217] S. Lie, "Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning," *IEEE Micro*, vol. 43, no. 3, pp. 18–30, 2023.

[218] Z. Zhou, Y. Zhu, C. He, Y. Wang, S. Yan, Y. Tian, and L. Yuan, "Spikformer: When spiking neural network meets transformer," 2022. [Online]. Available: https://arxiv.org/abs/2209.15425

[219] Z. Wang, Y. Fang, J. Cao, Q. Zhang, Z. Wang, and R. Xu, "Masked spiking transformer," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 1761–1771.

[220] X. Shi, Z. Hao, and Z. Yu, "Spikingresformer: Bridging resnet and vision transformer in spiking neural networks," 2024. [Online]. Available: https://arxiv.org/abs/2403.14302

[221] A. Moitra, A. Bhattacharjee, Y. Kim, and P. Panda, "Trex-reusing vision transformer's attention for efficient xbar-based computing," *arXiv preprint arXiv:2408.12742*, 2024.

[222] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415640

[223] Z. Lu, X. Wang, M. T. Arafin, H. Yang, Z. Liu, J. Zhang, and G. Qu, "An rram-based computing-in-memory architecture and its application in accelerating transformer inference," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 3, pp. 485–496, 2024.

[224] A. Bhattacharjee, A. Moitra, and P. Panda, "Clipformer: Key-value clipping of transformers on memristive crossbars for write noise mitigation," *arXiv preprint arXiv:2402.02586*, 2024.

[225] B. Yan, J.-L. Hsu, P.-C. Yu, C.-C. Lee, Y. Zhang, W. Yue, G. Mei, Y. Yang, Y. Yang, H. Li, Y. Chen, and R. Huang, "A 1.041-mb/mm2 27.38-tops/w signed-int8 dynamic-logic-based adc-less sram compute-in-memory macro in 28nm with reconfigurable bitwise operation for ai and embedded applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 188–190.

[226] R. Linderman, Q. Wu, G. Rose, H. Li, Y. Chen, and M. Hu, "Apparatus for performing matrix vector multiplication approximation using crossbar arrays of resistive memory devices," Oct. 6 2015, uS Patent 9,152,827.

[227] B. Liu, M. Hu, H. Li, Z.-H. Mao, Y. Chen, T. Huang, and W. Zhang, "Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2463209.2488741

[228] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 802–815.

[229] H. Liu, X. Wang, Y. Lu, and Y. Chen, "High density reconfigurable spin torque non-volatile memory," Apr. 15 2010, uS Patent App. 12/251,788.

[230] L. Zhao, L. Buonanno, R. M. Roth, S. Serebryakov, A. Gajjar, J. Moon, J. Ignowski, and G. Pedretti, "Race-it: A reconfigurable analog cam-crossbar engine for in-memory transformer acceleration," 2023. [Online]. Available: https://arxiv.org/abs/2312.06532

Cong Guo received his Ph.D. degree in Computer Science from Shanghai Jiao Tong University, Shanghai, China, in 2023 under the supervision of Prof. Jingwen Leng. He is now a Postdoctoral Fellow of Electrical and Computer Engineering at Duke University. His research interests include computer architecture, high-performance computing, and AI accelerator design.

Feng Cheng received his B.Eng. degree in Electrical Engineering from City University of Hong Kong, Hong Kong, China, in 2022. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interests include computer architecture, in-/near- memory computing and deep learning accelerator design.

Zhixu Du received his B.Sc. degree in Mathematics from The University of Hong Kong, Hong Kong, in 2021. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interests include efficiency, sparsity, fast inference of large models, Mixture-of-Experts, and Federated Learning.

James Kiessling received his B.S. degree in Computer Engineering from the University of Rhode Island, Kingston, RI, USA, in 2018. He worked as a firmware engineer until 2023, and is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, under the supervision of Prof. Yiran Chen. His research interests include efficient deep learning for edge devices, software-hardware co-design for machine learning, and electronic design automation.

Tergel Molom-Ochir received his B.S. degree in Electrical Engineering from the University of Massachusetts Amherst, MA, USA, in 2023. He is currently pursuing a Ph.D. degree in the Department of Electrical and Computer Engineering at Duke University, Durham, NC, USA, under the supervision of Prof. Yiran Chen. His research interests include circuit design, in-memory computing, AI accelerators, and non-volatile memories.

Jonathan Ku received the B.S. degree in Electrical Engineering and Computer Science from National Tsing Hua University in 2022. He is currently a Ph.D. student in Electrical and Computer Engineering at Duke University, Durham, NC, USA. His research interests include VLSI design, hardware acceleration in cryptography and machine learning.

Benjamin Morris received his B.S. degrees in Computer Science and Biomedical Engineering from North Carolina State University, Raleigh, NC in 2022. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Helen Li. His research interests include computer architecture, algorithm-hardware co-design of data-intensive applications, and near- or in-memory computing.

Shiyu Li received the Ph.D. degree in Computer Engineering from Duke University in 2024 under the guidance of Prof. Yiran Chen. Previously, he received the B.Eng. degree in automation from Tsinghua University, Beijing, China, in 2019. His research interests include computer architecture, algorithm-hardware co-design of deep learning systems, and near-data processing.
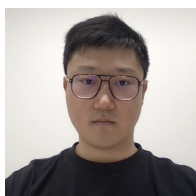
Haoxuan Shan received his B.S. degree in Electrical and Computer Engineering from Shanghai Jiao Tong University and B.S.E. degree in Computer Science from University of Michigan in 2022. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering at Duke University, supervised by Prof. Yiran Chen. His research interests include computer architecture and algorithm-hardware co-design.

Ziru Li received the Bachelor of Engineering (2019) in Electronic Engineering from Tsinghua University, Beijing, China. He received his Ph.D. degree (2024) in Electrical and Computer Engineering at Duke University, Durham, NC, USA, supervised by Prof. Helen Li. His research interests mainly focus on integrated circuit design for advanced artificial intelligence algorithms.
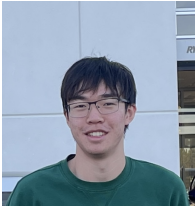
Jingwei Sun received his B.E. degree in Electrical Engineering from Wuhan University in 2019 and M.S. degree in Electrical and Computer Engineering from Duke University in 2021. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering at Duke University, supervised by Prof. Yiran Chen. His research interests include machine learning systems and edge computing.

Mingyuan Ma received the Bachelor of Engineering (2020) in Electronic Engineering from Tsinghua University, Beijing, China. He is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interests mainly focus on computer architecture and machine learning system.

Yitu Wang received the B.Eng. degree in Micoelectronics from Fudan University, Shanghai, China, in 2020. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interests focus on the near storage/memory architecture design for data-intensive applications and deep learning system.

Chiyue Wei received his B.Eng. degree in Electronic Engineering from Tsinghua University, Beijing, China, in 2023. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interest include computer architecture and software-hardware co-design for machine learning.
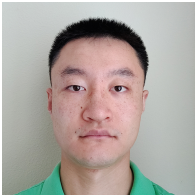
Xueying Wu received her B.Eng. degree in Microelectronics from Fudan University, Shanghai, China, in 2021. She is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Hai (Helen) Li. Her research interests include computer architecture, neural network compression, and hardware-software co-design of Processing-in-Memory systems.

Yuhao Wu received his B.S. and M.S. in Computer Science from the University of Arkansas, Fayetteville, AR, USA, in 2015. He worked as a software engineer until 2021, and is currently pursuing a Ph.D. degree in the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, under the supervision of Prof. Yiran Chen. His research interests include Federated Learning and medical image/video analysis.

Dr. Hao (Frank) Yang received B.S. degrees in Telecommunication Engineering from Beijing University of Posts and Telecommunications and the University of London in 2017, the Ph.D. degree in Civil Engineering from the University of Washington, and finished his post-doc at Duke University. He is currently an assistant professor at Johns Hopkins University, Department of Civil and System Engineering. His research focuses on developing trustworthy machine learning and data science methods to improve the equity, safety, resilience, and sustainability of traffic systems. Dr. Yang has published over 20 top refereed journal papers and 45 conference proceedings, with three best paper awards.

Jingyang Zhang received his B.Eng. degree in Electronic Engineering from Tsinghua University, Beijing, China, in 2019. Since then, he has been pursuing the Ph.D. degree at Dept. of Electrical and Computer Engineering at Duke University, supervised by Dr. Yiran Chen and Dr. Hai (Helen) Li. His research spans from robustness of deep learning-based vision systems to (more recently) generative AI and multi-modal LLMs.

Junyao Zhang received his M.S. in Electrical Engineering from University of Southern California, Los Angeles, USA in 2021. He is currently pursuing a Ph.D. degree in the Department of Electrical and Computer Engineering at Duke University, Durham, USA, under the supervision of Prof. Yiran Chen. His research interests include electronic design automation, quantum computing and high-performance computing.
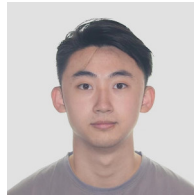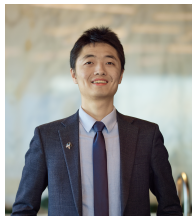
Qilin Zheng received his Ph.D degree (2024) in Electrical and Computer Engineering at Duke University. Before that, he received his M.Sc and B.Sc degree from KU Leuven and Peking University, respectively. His research interests include machine learning accelerator, in-memory computing and non-volatile memory design.

Guanglei Zhou received his B.Eng. degree in Electrical Engineering from City University of Hong Kong, Hong Kong, China, in 2020 and a M.A.Sc degree in Computer Engineering from University of Toronto, Toronto, Canada in 2022. He is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supervised by Prof. Yiran Chen. His research interests include computer architecture and electronic design automation.
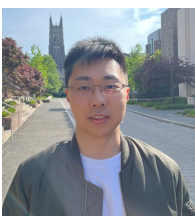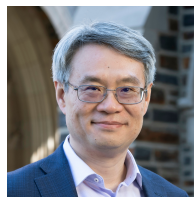
Hai (Helen) Li (M'08-SM'16-F'19) received the Ph.D. degree from Purdue University in 2004. Dr. Li is currently the Clare Boothe Luce Professor and Department Chair of the Electrical and Computer Engineering Department at Duke University. Her current research interests include neuromorphic circuits and systems for brain-inspired computing, machine learning acceleration and trustworthy AI, conventional and emerging memory design and architecture, and software and hardware co-design. Dr. Li is a recipient of the NSF Career Award (2012), DARPA Young Faculty Award (2013), TUM-IAS Hans Fischer Fellowship from Germany (2017), and ELATE Fellowship (2020). She received 9 best paper awards and additional 9 best paper nominations from international conferences. Dr. Li is a Distinguished Lecturer of the IEEE CAS Society (2018-2019) and a Distinguished Speaker of ACM (2017-2020).

Yiran Chen received the Ph.D. degree from Purdue University in 2005. He is now the John Cocke Distinguished Professor of Electrical and Computer Engineering at Duke University and serving as the director of the NSF AI Institute for Edge Computing Leveraging the Next-generation Networks (Athena), the NSF Industry-University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC), and the co-director of Duke Center for Computational Evolutionary Intelligence (DCEI). He received 11 best paper awards, 1 best poster award, and 15 best paper nominations from international conferences and workshops. He received numerous awards for his technical contributions and professional services such as the IEEE CASS Charles A. Desoer Technical Achievement Award, the IEEE Computer Society Edward J. McCluskey Technical Achievement Award, etc. He has been the distinguished lecturer of IEEE CEDA and CAS.