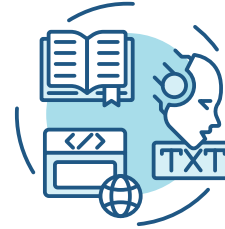




ENTITY RECOGNITION



LLMS



TEXT DATA COLLECTION



FINE-TUNING



TEXT SUMMARIZATION



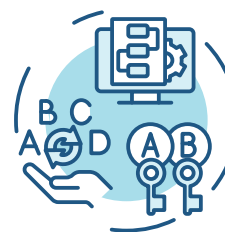
TEXT COMPLETION



TEXT DATA LABELING



CODE GENERATION



PROMPT ENGINEERING

IMAGE LICENSED BY INGRAM PUBLISHING

# 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 Li, *Fellow, IEEE*, and Yiran Chen, *Fellow, IEEE*

Digital Object Identifier 10.1109/MCAS.2024.3476008

Date of current version: 7 February 2025

## 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.

*Index Terms*—Large language model, hardware-software co-design.

## I. Introduction

The rapid advancement of large language models [1], [2], [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], [5], [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 350 GB 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 100 MB 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 reduce 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 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.

*The authors are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27705 USA.*

## 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:

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

*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  $\sigma$  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  $\mu$  and  $\sigma^2$  are the mean and variance of  $x$ , and  $\gamma$  and  $\beta$  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 1 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.

**Table 1.** 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
$2N$	$0 \sim 2N$	$4N \sim 12N$	$xNB$

### 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], [23], [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.

**Memory Optimization.** Zero Redundancy Optimizer (ZeRO) [9] and its subsequent works [25], [26], [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 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 for 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) a pretrained 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. SLoRA [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 Normal-Float data type with an innovative method called Double Quantization, and then back-propagate the gradients through the quantized weights with Paged Optimizers, can fine-tune a LLaMA 65B parameter model on a single 48 GB 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 pretrained language model (PLM) towards 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 [4] 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 [4], [42], [43], [44]. 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 [45], [46], [47], [48], [49], [50]. 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 [45], [51], [52] 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 [48], [50], [53].

**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 a RAG technique. Hierarchical Navigable Small World [54] (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 [55], 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 [56] 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 [57] 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 [58], on the other hand, finds a universal set of parameters to update for all

downstream tasks based on parameter magnitude. FISH Mask [59] 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 [60] and H100 [61], 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 [61], based on the mix-precision training technology [62].

**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 [63] 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 [64] is the first study to leverage host memory and storage as an extended memory hierarchy for LLM training, enhancing efficiency by addressing storage bandwidth bottle-necks 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 [65], Tensor-Dash [66], FAST [67], and Combricon-Q [68], including evaluations on Transformer-based models. However, these studies mainly targeted much smaller language models, like GNMT [69].

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 [70]. To optimize attention computation, FlashAttention [71], [72] integrates the entire process into a single operator, achieving 3× training speed up on GPT-2 model. FlashDecoding [73] and FlashDecoding++ [74] further enhance this by optimizing parallelism and introducing efficiency in SoftMax computation. For linear operations, TensorRT-LLM [75] employs a specialized GEMV implementation, while FlashDecoding++ [74] 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 [76], xFormers [77], and TensorRT-LLM [75], 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 [78] introduces PagedAttention to segment the KV cache into manageable blocks, enhancing memory management with 24× 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 [79] 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 [80].

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 [81] 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 [82], Sarathi-Serve [83] and LightLLM [84], 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<sup>3</sup> [85] predicts the upper limit of generation lengths, optimizing the initial memory allocation. Further improvements are seen in vLLM [78], 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 [84] refines this approach by allocating KV cache storage at the token level, maximizing space utilization, and minimizing waste. LLM in a Flash [80] 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 [78], [81], [84]. To combat this, FastServe [86] leverages a preemptive scheduling strategy that prioritizes requests based on their estimated completion time, thus improving throughput and reducing job completion times. Additionally, VTC [87] 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 [88] 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 [89] and TetrInfer [90] 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 [79], [80], [91], [92], [93] 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 [80], [91], [92] 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 [79], [93], 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 [80], [91], [93], [94], [95]. 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 [71] to memory-efficient strategies like vLLM [78] and scheduling optimizations such as FastServe [86], 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 [96] and [97] by Jordan and 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, [98] 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 [17], 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 [5] and Mixtral [99].

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:

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

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

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. [100] developed an LSTM model with 137B parameters, significantly enhancing model capacity with minimal computational overhead. Switch Transformer [101] extended this to a transformer-based MoE model with 1.6 T parameters, confirming MoE expansion follows the scaling law. GShard [18] efficiently implements large-scale MoE, expanding it by over 600B parameters. Clark et al. [102] 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 [103] improved transfer learning performance. Mixture-of-Depth (MoD) [104] explores skipping layers in transformer models. Xue et al. [105] and Riquelme et al. [106] develop vision transformer-based MoE models, with Riquelme achieving state-of-the-art performance with half the computation. Obando-Ceron et al. [107] constructs an MoE model for reinforcement learning, providing empirical evidence for scaling laws in this domain. MoEfication [108], [109] 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 [110] uses a linear assignment problem for balanced loading, while Hash Layer [111] uses predefined hash functions. StableMoE [112] stabilizes training by learning a balanced router function. Differentiable MoE architectures like Soft MoE [113] and Lory [114] 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 [115], M3ViT [116], and SiDA-MoE [117] introduce strategies to optimize throughput and expert caching. Analogy to MoE, FoE (Fusion of Experts) [118] 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) [119] and Group-Query Attention (GQA) [120] 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 techniques 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:

$$\text{SoftMax}(QK^T)V \approx \phi(Q)\phi(K)^TV; \quad (3)$$

where  $\phi(K)^TV$  is a multiplication between  $\mathbb{R}^{d \times n}$  and  $\mathbb{R}^{n \times d}$  and  $\phi(Q)\phi(K)^TV$  is a multiplication between  $\mathbb{R}^{n \times d}$  and  $\mathbb{R}^{d \times d}$ , taking  $O(nd^2)$  complexity. Performers [121] and RFA [122] employ the random feature projection as the feature map, while PolySketchFormer [123] 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 [124] is the first to explore the possibility of low-rank approximation of the attention matrix. LRT [125] then proposes to apply low-rank approximation on both the attention matrix and feed-forward layers. FLuRKA [126] 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 [127], [128] aims to speed up the decoding process for autoregressive 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 [129], [130], [131]. The other line of research aims to tackle the problem through the lens of algorithms, improving the alignment between LLMs and small models [132], [133], [134].

**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  $L$ ,  $\{h_t \in \mathbb{R}^{d_{\text{emb}}}\}_{t=1}^L$ , to a sequence of states  $\{x_t \in \mathbb{R}^{d_{\text{states}}}\}_{t=1}^L$  based on HiPPO theory [135]. 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:

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

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

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 [136], [137], [138], [139] and better computational architecture based on SSMs [138], [139], [140], [141], [145]. Specifically, LSSL [136] proposes to initialize matrix  $A$  via the optimal transition matrix proposed in [135]. Further, LSSL trains an SSM model through telescoping propagation equations 4, which can be computed efficiently through the Fast Fourier Transform. S4 [137] employs a diagonalized transition matrix  $A$  to enhance the computational efficiency. Later on, S5 [138] proposes to share the transition matrices across all input dimensions to boost the computational efficiency, while Mamba [139] and S4 [137] propose input dependent transition matrices that improve the model capability. Meanwhile, Mamba [139] and S4 [137] utilize a parallel scan technique that improves the computational efficiency of SSMs. MambaFormer [141] and Jamba [142] improve the SSM architecture by combining transformers into SSMs, where [141] use the SSM layer to replace the FFN layer in transformers and Jamba [142]

add four transformer layers to SSMs. Mamaba2 [140] proposes a new architecture based on State-Space Duality that achieves 2-8× speed up compared to Mamba [139].

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 [143]) 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], [144].

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 [145], which employs a transformer with queries to perform learned pooling, effectively replacing the MLP as the projector [146]. This approach can significantly reduce the computational demands of both training and inference. Another method is the algorithmic approach of PruMerge [147], 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 [148], [149], and [150], involves retraining the model to adapt to quantization noise. On the other hand, PTQ [148], [151], [152] 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 [153] 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) [154] optimizes low-bit weight-only quantization by protecting critical weights based on activation distributions, preserving model performance without backpropagation. QuaRot [155] achieves outlier-free 4-bit inference using randomized Hadamard transformations, efficiently handling activation quantization by removing outliers. QuIP [156] 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 [157] supports various bit-width quantizations by combining low-bit processing elements. OLAccel [158] and GOBO [158] quantizes outliers with higher precision, but these approaches often suffer from unaligned memory accesses, leading to additional overhead and limited computing speed. ANT [159] offers a fixed-length adaptive quantization framework, considering tensor distribution but overlooking outliers' importance. Mokey [160] 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 [161] 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) [162] uses a shared exponent for groups of values, enabling efficient dot product computations and higher arithmetic density compared to formats like Bfloat 16 or INT8, making it ideal for large-scale cloud deployments. FP6-LLM [163] 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 [164] 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() [165] 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 offer significant benefits for large language models, primarily by reducing memory bandwidth and capacity requirements, which in turn accelerates performance for memory-bound decoding process. As context lengths grow, weight quantization alone becomes insufficient, necessitating the quantization of 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 [166], [167], [168], [169], [170], [171], [172], [173], [174] 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 [175] 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 [176] prunes weights

into a structured N:M sparsity pattern, which can efficiently run on sparse tensor cores in GPUs [170]. SLOPE [177] applies N:M structured sparsity to both forward and backward passes, achieves significant speedups and memory savings compared to dense LLMs. PGF [178] 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 [179] 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) [172] 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) [180] 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 [200] utilizes a combination of local attention and global attention to specific tokens, while BigBird [201] 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 GEMM to SDDMM and SpMM. To efficiently perform these operations, sparse NVPIM [202] 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 [184], [187], [194], [197], [203]. SpAtten [203] 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 [184] 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 [197] 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. Reference [204] 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.

**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.**

#### ***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 2. 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 [181] 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 [188], 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 [205] improves transformer efficiency by dynamically adjusting attention mechanisms based on input complexity, achieving significant reductions in energy consumption. Finally, C-Transformer [195] 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 [184] introduces a detector for attention selection and utilizes a token-parallel data flow for sparse attention computation, enabling key/value reuse. Additionally, SPRINT [185] 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 [196] 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 [186] 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 [198] and NeuPIMs [199] 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.

**While numerous hardware and algorithmic accelerations have been proposed in academia, they are limited to certain applications and uses.**

For instance, DFX [182] 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 [206] 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 [207], [208], 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 [209], [210] differs

**Table 2.**  
**LLM accelerators for inference.**

Name	Platform	Model	Energy efficiency (TOPS/W)	Quantization/Sparsity	Year
TranCIM [181]	ASIC tapeout 28nm	BERT	20.5 (INT8)	Sparsity	2022
DFX [182]	FPGA	GPT-2			2022
X-Former [183]	PIM simulator 32 nm	BERT	13.44 (INT8)	-	2022
DOTA [184]	ASIC 22nm/simulator	GPT-2	-	Quantization/Sparsity	2022
SPRINT [185]	PIM simulator 32 nm	GPT-2/BERT	19.6x	Sparsity	2022
TransPIM [186]	PIM 65nm/simulator	GPT-2/BERT	666.6x RTX 2080Ti	-	2022
Mokey [160]	ASIC 65nm/simulator	BERT	9x GOBO (FP16)	Quantization	2022
LeOPArD [187]	ASIC 65nm/simulator	GPT-2/BERT	3x SpAtten	Quantization/Sparsity	2022
STP [188]	ASIC tapeout 12 nm	BERT	18.1 (FP4)	Quantization	2023
HAIMA [189]	PIM simulator 45 nm	BERT	-		2023
TF-MVP [190]	ASIC 28nm	BERT/GPT-2	0.48 (FP16)	Sparsity	2023
TiC-SAT [191]	gem5-X	BERT	-	-	2023
Transformer-OPU [192]	FPGA	BERT	-	-	2023
FACT [193]	ASIC 28nm	BERT	94.88x V100	Quantization/Sparsity	2023
TaskFusion [194]	ASIC 22nm/simulator	BERT	19.83x Jetson Nano	Sparsity	2023
OliVe [161]	ASIC 22nm/simulator	BERT/GPT-2/OPT	4 x GOBO	Quantization	2023
C-Transformer [195]	ASIC tapeout 28 nm	GPT-2	33.4 (INT8)	-	2024
SpecPIM [196]	PIM simulator	LLaMA/OPT	6.67x A100 (FP16)	-	2024
ASADI [197]	PIM simulator	BERT/GPT-2	- (FP32)	Sparsity	2024
AttAcc [198]	PIM simulator	LLaMA/GPT-3	2.67x DGX A100 (FP16)	-	2024
NeuPIMs [199]	PIM simulator 22nm	GPT-3	-	-	2024

**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.**

from their previous works, which were categorized as ASIC accelerators. NorthPole targets large-scale workload, comparing against Google TPUv4 [211], NVIDIA A100 & H100 AI processors. The performance achieved is a joint effort of both architecture and their SDK tool-chain. This software-assisted approach is also implemented in their work [212].

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 [213]. 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 better 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 [214] 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 [215]. Spikingformer demonstrated further innovation [216], incorporating spike-driven residual learning within Transformer-based SNNs. These advancements have been further augmented by the realization of C-Transformer [195], 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, 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 [217], recent work presented RIME [218], 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 [219] 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 are also research about methodologies of in-memory computing for transformers. TRex [220] propose 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 [221] deal with the noise of memristive crossbar, by clipping the Key and Value matrices of Transformers during inference, the impact of write noise effects are reduced. 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.

## Acknowledgment

This work was supported in part by the National Science Foundation under Grants 2328805 and 2112562 and in part by Army Research Office under Grant W911NF-23-2-0224. Corresponding author: Yiran Chen.



**Cong Guo** received the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2023, under the supervision of Prof. Jingwen Leng. He is currently a Post-Doctoral Fellow of electrical and computer engineering with Duke University. His research interests include computer architecture, high-performance computing, and AI accelerator design.



**Feng Cheng** received the B.Eng. degree in electrical engineering from the City University of Hong Kong, Hong Kong, China, in 2022. He is currently pursuing the 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 computer architecture, in-/near-memory computing, and deep learning accelerator design.



**Zhixu Du** received the B.Sc. degree in mathematics from The University of Hong Kong, Hong Kong, in 2021. He is currently pursuing the 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 efficiency, sparsity, fast inference of large models, mixture-of-experts, and federated learning.



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



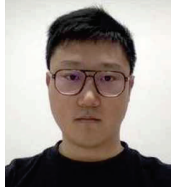
**Jonathan Ku** received the B.S. degree in electrical engineering and computer science from National Tsing Hua University in 2022. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Duke University, Durham, NC, USA. His research interests include VLSI design, hardware acceleration in cryptography, and machine learning.



**Shiyu Li** received the B.Eng. degree in automation from Tsinghua University, Beijing, China, in 2019, and the Ph.D. degree in computer engineering from Duke University in 2024, under the supervision of Prof. Yiran Chen. His research interests include computer architecture, algorithm-hardware co-design of deep learning systems, and near-data processing.



**Ziru Li** received the B.E. degree in electronic engineering from Tsinghua University, Beijing, China, in 2019, and the Ph.D. degree in electrical and computer engineering from Duke University, Durham, NC, USA, in 2024, under the supervision of Prof. Helen Li. His research interests mainly focus on integrated circuit design for advanced artificial intelligence algorithms.



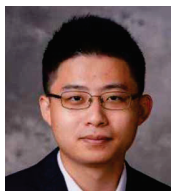
**Mingyuan Ma** received the B.E. degree in electronic engineering from Tsinghua University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Duke University, Durham, NC, USA, under the supervision of Prof. Yiran Chen. His research interests include computer architecture and machine learning systems.



**Tergel Molom-Ochir** received the B.S. degree in electrical engineering from the University of Massachusetts Amherst, Amherst, MA, USA, in 2023. He is currently pursuing the 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 circuit design, in-memory computing, AI accelerators, and non-volatile memories.



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



**Haoxuan Shan** received the B.S. degree in electrical and computer engineering from Shanghai Jiao Tong University and the B.S.E. degree in computer science from the University of Michigan in 2022. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, under the supervision of Prof. Yiran Chen. His research interests include computer architecture and algorithm-hardware co-design.

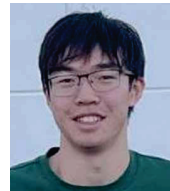


**Jingwei Sun** received the B.E. degree in electrical engineering from Wuhan University in 2019 and the M.S. degree in electrical and computer engineering from Duke University in 2021. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, under the supervision of Prof. Yiran

Chen. His research interests include machine learning systems and edge computing.



**Yitu Wang** received the B.Eng. degree in microelectronics from Fudan University, Shanghai, China, in 2020. He is currently pursuing the 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 near storage/memory architecture design for data-intensive applications and deep learning systems.



**Chiyue Wei** received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 2023. He is currently pursuing the 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 computer architecture and software-hardware co-design for machine learning.



**Xueying Wu** received the B.Eng. degree in microelectronics from Fudan University, Shanghai, China, in 2021. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, under the supervision of 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 the B.S. and M.S. degrees in computer science from the University of Arkansas, Fayetteville, AR, USA, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, under the supervision of Prof. Yiran Chen. He worked as a Software Engineer until 2021. His research interests include federated learning and medical image/video analysis.



**Hao Frank Yang** received the B.S. degree in telecommunication engineering from the Beijing University of Posts and Telecommunications and the University of London in 2017, and the Ph.D. degree in civil engineering

from the University of Washington. He was a Post-Doctoral Researcher at Duke University. He is currently an Assistant Professor with the Department of Civil and System Engineering, Johns Hopkins University. He has published over 20 top refereed journal articles and 45 conference proceedings, with three best paper awards. His research focuses on developing trustworthy machine learning and data science methods to improve the equity, safety, resilience, and sustainability of traffic systems.



**Jingyang Zhang** received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Duke University, under the supervision of 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 the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2021. He is currently pursuing the 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 electronic design automation, quantum computing, and high-performance computing.



**Qilin Zheng** received the B.Sc. degree from Peking University, the M.Sc. degree from KU Leuven, and the Ph.D. degree in electrical and computer engineering from Duke University in 2024. His research interests include machine learning accelerator, in-memory computing, and nonvolatile memory design.



**Guanglei Zhou** received the B.Eng. degree in electrical engineering from the City University of Hong Kong, Hong Kong, China, in 2020, and the M.A.Sc. degree in computer engineering from the University of Toronto, Toronto, ON, Canada, in 2022. He is currently pursuing the 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 computer architecture and electronic design automation.



**Hai (Helen) Li** (Fellow, IEEE) received the Ph.D. degree from Purdue University in 2004. She is currently the Clare Boothe Luce Professor and the Department Chair of the Electrical and Computer Engineering Department, Duke University. Her current research interests include neuro-morphic 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. She was a recipient of the NSF Career Award in 2012, the DARPA Young Faculty Award in 2013, the TUM-IAS Hans Fischer Fellowship from Germany in 2017, and the ELATE Fellowship in 2020. She received nine best paper awards and additional nine best paper nominations from international conferences. She is a Distinguished Lecturer of the IEEE CAS Society from 2018 to 2019 and a Distinguished Speaker of ACM from 2017 to 2020.



**Yiran Chen** (Fellow, IEEE) received the Ph.D. degree from Purdue University in 2005. He is currently the John Cocke Distinguished Professor of electrical and computer engineering with Duke University, the Director of the NSF AI Institute for Edge Computing Leveraging the Next-Generation Networks (Athena) and the NSF Industry-University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC), and the Co-Director of the Duke Center for Computational Evolutionary Intelligence (DCEI). He received 11 best paper awards, one 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 and the IEEE Computer Society Edward J. McCluskey Technical Achievement Award. He has been the Distinguished Lecturer of IEEE CEDA and CAS.

## References

- [1] A. Radford et al., "Improving language understanding by generative pre-training," 2018. [Online]. Available: <https://hayate-lab.com/wp-content/uploads/2023/05/43372bfa750340059ad87ac8e538c53b.pdf>
- [2] H. Touvron et al., "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
- [3] G. Team et al., "Gemini: A family of highly capable multimodal models," 2023, *arXiv:2312.11805*.
- [4] T. B. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.
- [5] OpenAI et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

- [6] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*, doi: <https://doi.org/10.48550/arXiv.2307.09288>
- [7] H. Liu et al., "Visual instruction tuning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 1–25.
- [8] K. He et al., "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [9] S. Rajbhandari et al., "ZeRO: Memory optimizations toward training trillion parameter models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Virtual Event, Atlanta, GA, USA, Nov. 2020, p. 20, doi: [10.1109/SC41405.2020.00024](https://doi.org/10.1109/SC41405.2020.00024).
- [10] A. Gholami et al., "AI and memory wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, May 2024.
- [11] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
- [12] A. Radford 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," 2016, *arXiv:1607.06450*.
- [14] J. Yang et al., "Harnessing the power of LLMs in practice: A survey on ChatGPT and beyond," *ACM Trans. Knowl. Discovery from Data*, vol. 18, no. 6, pp. 1–32, Jul. 2024.
- [15] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
- [16] W. X. Zhao et al., "A survey of large language models," 2023, *arXiv:2303.18223*.
- [17] J. Kaplan et al., "Scaling laws for neural language models," 2020, *arXiv:2001.08361*.
- [18] D. Lepikhin et al., "GShard: Scaling giant models with conditional computation and automatic sharding," 2020, *arXiv:2006.16668*.
- [19] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," 2020, *arXiv:2006.15704*.
- [20] M. Shoenybi et al., "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.
- [21] Q. Xu and Y. You, "An efficient 2D method for training super-large deep learning models," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2023, pp. 222–232.
- [22] A. Harlap et al., "PipeDream: Fast and efficient pipeline parallel DNN training," 2018, *arXiv:1806.03377*.
- [23] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 103–112.
- [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 et al., "Zero-offload: Democratizing billionscale model training," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Jul. 2021, pp. 551–564.
- [26] S. Rajbhandari et al., "Zero-infinity: breaking the GPU memory wall for extreme scale deep learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, St. Louis, MO, USA, Nov. 2021, p. 59, doi: [10.1145/3458817.3476205](https://doi.org/10.1145/3458817.3476205).
- [27] G. Wang et al., "ZeRO++: Extremely efficient collective communication for giant model training," 2023, *arXiv:2306.10209*.
- [28] T. Chen et al., "Training deep nets with sublinear memory cost," 2016, *arXiv:1604.06174*.
- [29] G. J. Chaitin et al., "Register allocation via coloring," *Comput. Lang.*, vol. 6, no. 1, pp. 47–57, Jan. 1981, doi: [10.1016/0096-0551\(81\)90048-5](https://doi.org/10.1016/0096-0551(81)90048-5).
- [30] J. Herrmann et al., "Optimal checkpointing for heterogeneous chains: How to train deep neural networks with limited memory," 2019, *arXiv:1911.13214*.
- [31] C. Guo et al., "GMLake: Efficient and transparent GPU memory defragmentation for large-scale DNN training with virtual memory stitching," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, vol. 2, Apr. 2024, pp. 450–466.
- [32] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.
- [33] R. Y. Aminabadi et al., "DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2022, pp. 1–15.
- [34] T. Wolf et al., "HuggingFace's transformers: State-of-the-art natural language processing," 2019, *arXiv:1910.03771*.
- [35] *TorchTune*. [Online]. Available: <https://pytorch.org/torchTune/stable/index.html>
- [36] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.
- [37] L. Hedegaard et al., "Structured pruning adapters," 2022, *arXiv:2211.10155*.
- [38] M. Zhang et al., "LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning," 2024, *arXiv:2305.18403*.
- [39] T. Dettmers et al., "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. 37th Conf. Neural Inf. Process. Syst.*, 2023, pp. 1–28. [Online]. Available: <https://openreview.net/forum?id=OUIFPHEgJU>
- [40] H. Guo et al., "LQ-LoRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024. [Online]. Available: <https://openreview.net/forum?id=xw29VvOMmU>
- [41] Y. Xu et al., "QA-LoRA: Quantization-aware low-rank adaptation of large language models," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024. [Online]. Available: <https://openreview.net/forum?id=WvFojccpo8>
- [42] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," 2020, *arXiv:2001.07676*.
- [43] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," 2020, *arXiv:2012.15723*.
- [44] T.-X. Sun et al., "Paradigm shift in natural language processing," *Mach. Intell. Res.*, vol. 19, no. 3, pp. 169–183, Jun. 2022.
- [45] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," 2021, *arXiv:2101.00190*.
- [46] K. Hambardzumyan, H. Khachatrian, and J. May, "WARP: Word-level adversarial ReProgramming," 2021, *arXiv:2101.00121*.
- [47] G. Qin and J. Eisner, "Learning how to ask: Querying LMs with mixtures of soft prompts," 2021, *arXiv:2104.06599*.
- [48] X. Liu et al., "GPT understands, too," 2023, *arXiv:2103.10385*.
- [49] Z. Zhong, D. Friedman, and D. Chen, "Factual probing is [MASK]: Learning vs. Learning to recall," 2021, *arXiv:2104.05240*.
- [50] X. Liu et al., "P-Tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," 2021, *arXiv:2110.07602*.
- [51] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," 2021, *arXiv:2104.08691*.
- [52] J. Sun et al., "FedBPT: Efficient federated black-box prompt tuning for large language models," 2023, *arXiv:2310.01467*.
- [53] X. Liu et al., "P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, vol. 2, 2022, p. 6168.
- [54] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.
- [55] S. J. Subramanya et al., "DiskANN: Fast accurate billion-point nearest neighbor search on a single node," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.
- [56] *Faiss*. [Online]. Available: <https://ai.meta.com/tools/faiss/>
- [57] D. Guo, A. Rush, and Y. Kim, "Parameter-efficient transfer learning with diff pruning," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics, 11th Int. Joint Conf. Natural Lang. Process.*, vol. 1, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, Aug. 2021, pp. 4884–4896. [Online]. Available: <https://aclanthology.org/2021.acl-long.378>
- [58] B. Liao, Y. Meng, and C. Monz, "Parameter-efficient fine-tuning without introducing new latency," in *Proc. 61st Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, ON, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4242–4260. [Online]. Available: <https://aclanthology.org/2023.acllong.233>
- [59] Y.-L. Sung, V. Nair, and C. Raffel, "Training neural networks with fixed sparse masks," in *Proc. Adv. Neural Inf. Process. Syst.*, A. Beygelzimer et al., Eds., 2021. [Online]. Available: <https://openreview.net/forum?id=Uwh-vHsw-x>
- [60] *NVIDIA A100 Tensor Core GPU Architecture*, NVIDIA, Santa Clara, CA, USA, 2020.
- [61] *NVIDIA H100 Tensor Core GPU Architecture*, NVIDIA, Santa Clara, CA, USA, 2022.
- [62] P. Micikevicius et al., "Mixed precision training," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.

- [63] Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, p. 3746, 1982.
- [64] H. Jang et al., "Smart-infinity: Fast large language model training using near-storage processing on a real system," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, vol. 12, Mar. 2024, pp. 345–360.
- [65] E. Qin et al., "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.
- [66] M. Mahmoud et al., "TensorDash: Exploiting sparsity to accelerate deep neural network training," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 781–795.
- [67] S. Q. Zhang, B. McDanel, and H. T. Kung, "FAST: DNN training under variable precision block floating point with stochastic rounding," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 846–860.
- [68] Y. Zhao et al., "Cambricon-Q: A hybrid architecture for efficient training," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 706–719.
- [69] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
- [70] Z. Zhou et al., "A survey on efficient inference for large language models," 2024, *arXiv:2404.14294*.
- [71] T. Dao et al., "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 16344–16359.
- [72] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," 2023, *arXiv:2307.08691*.
- [73] T. Dao et al. (2023). *Flashdecoding for Long-Context Inference*. [Online]. Available: <https://crfm.stanford.edu/2023/10/12/flashdecoding.html>
- [74] K. Hong et al., "FlashDecoding++: Faster large language model inference on GPUs," 2023, *arXiv:2311.01282*.
- [75] N. Vaidya, F. Oh, and N. Comly. (2023). *Optimizing Inference on Large Language Models With NVIDIA TensorRT-LLM, Now Publicly Available*. [Online]. Available: <https://github.com/NVIDIA/TensorRT-LLM>
- [76] Y. Zhai et al., "ByteTransformer: A high-performance transformer boosted for variable-length inputs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2023, pp. 344–355.
- [77] B. Lefaudeux et al. (2022). *xformers: A Modular and Hackable Transformer Modelling Library*. [Online]. Available: <https://github.com/facebookresearch/xformers>
- [78] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in *Proc. 29th Symp. Oper. Syst. Princ.*, Oct. 2023, pp. 611–626.
- [79] Y. Sheng et al., "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proc. Int. Conf. Mach. Learn. PMLR*, 2023, pp. 31094–31116.
- [80] K. Alizadeh et al., "LLM in a flash: Efficient large language model inference with limited memory," 2023, *arXiv:2312.11514*.
- [81] G.-I. Yu et al., "Orca: A distributed serving system for transformer-based generative models," in *Proc. 16th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Carlsbad, CA, USA: USENIX Association, 2022, pp. 521–538. [Online]. Available: <https://www.usenix.org/conference/osdi22/presentation/you>
- [82] A. Agrawal et al., "SARATHI: Efficient LLM inference by piggybacking decodes with chunked prefills," 2023, *arXiv:2308.16369*.
- [83] A. Agrawal et al., "Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve," 2024, *arXiv:2403.02310*.
- [84] ModelTC. (Feb. 2024). *Lightllm*. [Online]. Available: <https://github.com/ModelTC/lightllm/>
- [85] Y. Jin et al., "s3: Increasing GPU utilization during generative inference for higher throughput," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 18015–18027.
- [86] B. Wu et al., "Fast distributed inference serving for large language models," 2023, *arXiv:2305.05920*.
- [87] Y. Sheng et al., "Fairness in serving large language models," 2024, *arXiv:2401.00588*.
- [88] X. Miao et al., "SpotServe: Serving generative large language models on preemptible instances," 2023, *arXiv:2311.15566*.
- [89] P. Patel et al., "Splitwise: Efficient generative LLM inference using phase splitting," 2023, *arXiv:2311.18677*.
- [90] C. Hu et al., "Inference without interference: Disaggregate LLM inference for mixed downstream workloads," 2024, *arXiv:2401.11181*.
- [91] Z. Xue et al., "PowerInfer-2: Fast large language model inference on a smartphone," 2024, *arXiv:2406.06282*.
- [92] W. Yin et al., "LLM as a system service on mobile devices," 2024, *arXiv:2403.11805*.
- [93] Y. Song et al., "PowerInfer: Fast large language model serving with a consumer-grade GPU," 2023, *arXiv:2312.12456*.
- [94] Z. Liu et al., "Deja Vu: Contextual sparsity for efficient LLMs at inference time," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2023, pp. 22137–22176.
- [95] Y. Song et al., "Turbo sparse: Achieving LLM SOTA performance with minimal activated parameters," 2024, *arXiv:2406.05955*.
- [96] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, no. 2, pp. 181–214, Mar. 1994.
- [97] R. A. Jacobs et al., "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, Mar. 1991.
- [98] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," 2013, *arXiv:1312.4314*.
- [99] A. Q. Jiang et al., "Mixtral of experts," 2024, *arXiv:2401.04088*.
- [100] N. Shazeer et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2017, *arXiv:1701.06538*.
- [101] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, no. 120, pp. 1–39, 2022.
- [102] A. Clark et al., "Unified scaling laws for routed language models," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 4057–4086.
- [103] B. Zoph et al., "ST-MOE: Designing stable and transferable sparse expert models," 2022, *arXiv:2202.08906*.
- [104] D. Raposo et al., "Mixture-of-depths: Dynamically allocating compute in transformer-based language models," *arXiv:2404.02258*, 2024.
- [105] F. Xue et al., "Go wider instead of deeper," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 8, 2022, pp. 8779–8787.
- [106] C. Riquelme et al., "Scaling vision with sparse mixture of experts," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 8583–8595.
- [107] J. Obando-Ceron et al., "Mixtures of experts unlock parameter scaling for deep RL," 2024, *arXiv:2402.08609*.
- [108] Z. Zhang et al., "Moefication: Transformer feed-forward layers are mixtures of experts," 2021, *arXiv:2110.01786*.
- [109] T. Zhu et al., "LLaMA-MoE: Building mixture-of-experts from LLaMA with continual pre-training," 2024, *arXiv:2406.16554*.
- [110] M. Lewis et al., "Base layers: Simplifying training of large, sparse models," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 6265–6274.
- [111] S. Roller et al., "Hash layers for large sparse models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 17555–17566.
- [112] D. Dai et al., "StableMoE: Stable routing strategy for mixture of experts," 2022, *arXiv:2204.08396*.
- [113] J. Puigcerver et al., "From sparse to soft mixtures of experts," 2023, *arXiv:2308.00951*.
- [114] Z. Zhong et al., "Lory: Fully differentiable mixture-of-experts for autoregressive language model pre-training," 2024, *arXiv:2405.03133*.
- [115] L. Shen et al., "SE-MoE: A scalable and efficient mixture-of-experts distributed training and inference system," 2022, *arXiv:2205.10034*.
- [116] Z. Fan et al., "M3ViT: Mixture-of-experts vision transformer for efficient multitask learning with model-accelerator co-design," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 28441–28457.
- [117] Z. Du et al., "SiDA-MoE: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models," in *Proc. Mach. Learn. Syst.*, vol. 6, 2024, pp. 224–238.
- [118] H. Wang et al., "Fusing models with complementary expertise," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024. [Online]. Available: <https://openreview.net/forum?id=PhMrGCMIRL>
- [119] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019, *arXiv:1911.02150*.
- [120] J. Ainslie et al., "GQA: Training generalized multi-query transformer models from multi-head checkpoints," 2023, *arXiv:2305.13245*.
- [121] K. Choromanski et al., "Rethinking attention with performers," 2020, *arXiv:2009.14794*.
- [122] H. Peng et al., "Random feature attention," 2021, *arXiv:2103.02143*.
- [123] P. Kacham, V. Mirrokni, and P. Zhong, "PolySketchFormer: Fast transformers via sketching polynomial kernels," 2023, *arXiv:2310.01655*.
- [124] S. Wang et al., "Linformer: Self-attention with linear complexity," 2020, *arXiv:2006.04768*.

- [125] S. Zhang, N. Meng, and E. Y. Lam, "LRT: An efficient low-light restoration transformer for dark light field images," *IEEE Trans. Image Process.*, vol. 32, pp. 4314–4326, 2023.
- [126] A. Gupta et al., "FLuRKA: Fast and accurate unified low-rank & kernel attention," 2023, *arXiv:2306.15799*.
- [127] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2023, pp. 19274–19286.
- [128] S. Kim et al., "Speculative decoding with big little decoder," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 39236–39256.
- [129] Z. Sun et al., "SpecTr: Fast speculative decoding via optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–21.
- [130] X. Miao et al., "SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proc. 29th ACM Int. Conf. Architectural Support for Program. Lang. Oper. Syst. (ASPLOS)*, vol. 3. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 932–949, doi: 10.1145/3620666.3651335.
- [131] D. Xu et al., "LLMCad: Fast and scalable on-device large language model inference," 2023, *arXiv:2309.04255*.
- [132] Y. Zhou et al., "DistillSpec: Improving speculative decoding via knowledge distillation," 2023, *arXiv:2310.08461*.
- [133] X. Liu et al., "Online speculative decoding," 2023, *arXiv:2310.07177*.
- [134] J. Zhang et al., "Draft & verify: Lossless large language model acceleration via self-speculative decoding," 2023, *arXiv:2309.08168*.
- [135] A. Gu et al., "HIPPO: Recurrent memory with optimal polynomial projections," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1474–1487.
- [136] A. Gu et al., "Combining recurrent, convolutional, and continuous-time models with linear state space layers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 572–585.
- [137] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2021, *arXiv:2111.00396*.
- [138] J. T. H. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," 2022, *arXiv:2208.04933*.
- [139] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," 2023, *arXiv:2312.00752*.
- [140] T. Dao and A. Gu, "Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality," 2024, *arXiv:2405.21060*.
- [141] J. Park et al., "Can mamba learn how to learn? A comparative study on in-context learning tasks," 2024, *arXiv:2402.04248*.
- [142] O. Lieber et al., "Jamba: A hybrid transformer-mamba language model," 2024, *arXiv:2403.19887*.
- [143] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, M. Meila and T. Zhang, Eds., PMLR, Jul. 2021, pp. 8748–8763. [Online]. Available: <https://proceedings.mlr.press/v139/radford21a.html>
- [144] H. Liu et al. (Jan. 2024). *LLaVA-NeXT: Improved Reasoning, OCR, and World Knowledge*. [Online]. Available: <https://llava-vl.github.io/blog/2024-01-30-llava-next/>
- [145] A. Jaegle et al., "Perceiver: General perception with iterative attention," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 4651–4664.
- [146] H. Laurençon et al., "What matters when building vision-language models?" 2024, *arXiv:2405.02246*.
- [147] Y. Shang et al., "LLaVA-PruMerge: Adaptive token reduction for efficient large multimodal models," 2024, *arXiv:2403.15388*.
- [148] B. Jacob et al., "Quantization and training of neural networks for efficient Integer-Arithmetic-Only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
- [149] Z. Wang et al., "Learning channel-wise interactions for binary convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 568–577.
- [150] B. Zhuang et al., "Effective training of convolutional neural networks with low-bitwidth weights and activations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6140–6152, Oct. 2022.
- [151] S. Gupta et al., "Deep learning with limited numerical precision," in *Int. Conf. Mach. Learn.*, PMLR, 2015, pp. 1737–1746.
- [152] C. Guo et al., "SQuant: On-the-fly data-free quantization via diagonal Hessian approximation," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–18. [Online]. Available: <https://openreview.net/forum?id=JXhROKNZzOc>
- [153] G. Xiao et al., "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proc. 40th Int. Conf. Mach. Learn.*, 2023, pp. 1–13.
- [154] J. Lin et al., "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proc. MLSys*, 2024, pp. 1–14.
- [155] S. Ashkboos et al., "QuaRot: Outlier-free 4-Bit inference in rotated LLMs," 2024, *arXiv:2404.00456*.
- [156] J. Chee et al., "QulP: 2-bit quantization of large language models with guarantees," 2023, *arXiv:2307.13304*.
- [157] H. Sharma et al., "Bit fusion: bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 764–775.
- [158] 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 *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 811–824.
- [159] C. Guo et al., "ANT: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 1414–1433.
- [160] A. H. Zadeh et al., "Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA: Association for Computing Machinery, 2022, pp. 888–901, doi: 10.1145/3470496.3527438.
- [161] C. Guo et al., "OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*, vol. 36. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–15, doi: 10.1145/3579371.3589038.
- [162] B. Rouhani et al., "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2020, pp. 1–11.
- [163] H. Xia et al., "FP6-LLM: Efficiently serving large language models through FP6-centric algorithm-system co-design," 2024, *arXiv:2401.14112*.
- [164] S.-Y. Liu et al., "LLM-FP4: 4-bit floating-point quantized transformers," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Association for Computational Linguistics, 2023, pp. 592–605, doi: 10.18653/v1/2023.emnlp-main.39.
- [165] T. Dettmers et al., "GPT3.int8(): 8-bit matrix multiplication for transformers at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, S. Koyejo Eds., vol. 35. Red Hook, NY, USA: Curran Associates, Inc., 2022, pp. 30318–30332.
- [166] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
- [167] J. Albericio et al., "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 1–13, 2016.
- [168] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
- [169] X. Zhou et al., "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative Software/Hardware approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 15–28.
- [170] M. Zhu et al., "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern GPUs," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 359–371.
- [171] C. Guo et al., "Accelerating sparse DNN models without hardware-support via tile-wise sparsity," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2020, pp. 1–15.
- [172] Y. Wang et al., "Dual-side sparse tensor core," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1083–1095.
- [173] Y. Guan et al., "Fractal: Joint multi-level sparse pattern tuning of accuracy and performance for DNN pruning," in *Proc. 29th ACM Int. Conf. Architectural Support for Program. Lang. Oper. Syst.*, vol. 3, Apr. 2024, pp. 416–430.
- [174] C. Guo et al., "Accelerating sparse DNNs based on tiled GEMM," *IEEE Trans. Comput.*, vol. 73, no. 5, pp. 1275–1289, May 2024.
- [175] X. Ma, G. Fang, and X. Wang, "LLM-Pruner: On the structural pruning of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 21702–21720.
- [176] Y. Zhang et al., "Plug-and-play: An efficient post-training pruning method for large language models," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024, pp. 1–19.

- [177] M. Mozaffari et al., "SLoPe: Double-pruned sparse plus lazy low-rank adapter pretraining of LLMs," 2024, *arXiv:2405.16325*.
- [178] A. R. Bambhaniya et al., "Progressive gradient flow for robust N: M sparsity training in transformers," 2024, *arXiv:2402.04744*.
- [179] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2023, pp. 10323–10337.
- [180] G. Huang et al., "RM-STC: Row-merge dataflow inspired GPU sparse tensor core for energy-efficient sparse acceleration," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2023, pp. 338–352.
- [181] F. Tu et al., "A 28 nm 15.59  $\mu$ J/token full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2022, pp. 466–468.
- [182] S. Hong et al., "DFX: A low-latency multi-FPGA appliance for accelerating transformer-based text generation," in *Proc. IEEE Hot Chips 34 Symp. (HCS)*, Aug. 2022, pp. 1–17.
- [183] S. Sridharan et al., "X-former: In-memory acceleration of transformers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 8, pp. 1223–1233, Aug. 2023.
- [184] Z. Qu et al., "DOTA: Detect and omit weak attentions for scalable transformer acceleration," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, vol. 1703. New York, NY, USA: Association for Computing Machinery, Feb. 2022, pp. 14–26, doi: 10.1145/3503222.3507738.
- [185] A. Yazdanbakhsh et al., "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 744–762, doi: 10.1109/MICRO56248.2022.00059.
- [186] M. Zhou et al., "TransPIM: A memory-based acceleration via software-hardware co-design for transformer," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 1071–1085.
- [187] Z. Li et al., "Accelerating attention through gradient-based learned runtime pruning," in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 902–915, doi: 10.1145/3470496.3527423.
- [188] T. Tambe et al., "A 12 nm 18.1TFLOPs/W sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2023, pp. 342–344.
- [189] Y. Ding et al., "HAIMA: A hybrid SRAM and DRAM accelerator-in-Memory architecture for transformer," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2023, pp. 1–6.
- [190] E. Yoo et al., "TF-MVP: Novel sparsity-aware transformer accelerator with mixed-length vector pruning," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, vol. 32, Jul. 2023, pp. 1–6.
- [191] A. Amirshahi et al., "TiC-SAT: tightly-coupled systolic accelerator for transformers," in *Proc. 28th Asia South Pacific Design Autom. Conf.*, Jan. 2023, pp. 657–663.
- [192] Y. Bai et al., "Transformer-OPU: An FPGA-based overlay processor for transformer networks," in *Proc. IEEE 31st Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2023, pp. 221–221.
- [193] Y. Qin et al., "FACT: FFN-attention co-optimized transformer architecture with eager correlation prediction," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–14, doi: 10.1145/3579371.3589057.
- [194] Z. Fan et al., "TaskFusion: An efficient transfer learning architecture with dual delta sparsity for multi-task natural language processing," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–14, doi: 10.1145/3579371.3589040.
- [195] S. Kim et al., "20.5 C-transformer: A 2.6–18.1  $\mu$ J/token homogeneous DNN-transformer/spiking-transformer processor with big-little network and implicit weight generation for large language models," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 67, Feb. 2024, pp. 368–370.
- [196] C. Li et al., "SpecPIM: Accelerating speculative inference on PIM-enabled system via architecture-dataflow co-exploration," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, vol. 3. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 950–965, doi: 10.1145/3620666.3651352.
- [197] H. Li et al., "ASADI: Accelerating sparse attention using diagonal-based in-situ computing," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Los Alamitos, CA, USA: IEEE Comput. Soc., Mar. 2024, pp. 774–787, doi: 10.1109/hpca57654.2024.00065.
- [198] J. Park et al., "AttAcc! Unleashing the power of PIM for batched transformer-based generative model inference," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, vol. 2. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 103–119, doi: 10.1145/3620665.3640422.
- [199] G. Heo et al., "NeuPIMs: NPU-PIM heterogeneous acceleration for batched LLM inferencing," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, vol. 3. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 722–737, doi: 10.1145/3620666.3651380.
- [200] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020, *arXiv:2004.05150*.
- [201] M. Zaheer et al., "Big bird: Transformers for longer sequences," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 17283–17297.
- [202] Q. Zheng et al., "Accelerating sparse attention with a reconfigurable non-volatile processing-in-memory architecture," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2023, pp. 1–6.
- [203] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient sparse attention architecture with cascade token and head pruning," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 97–110.
- [204] S. B. Harma et al., "Effective interplay between sparsity and quantization: From theory to practice," 2024, *arXiv:2405.20935*.
- [205] A. Moitra, A. Bhattacharjee, and P. Panda, "PIVOT-input-aware path selection for energy-efficient ViT inference," 2024, *arXiv:2404.15185*.
- [206] S.-S. Park et al., "An LPDDR-based CXL-PNM platform for TCO-efficient inference of transformer-based large language models," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Mar. 2024, pp. 970–982.
- [207] S. Venkataramani et al., "RaPiD: AI accelerator for ultra-low precision training and inference," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 153–166.
- [208] S. K. Lee et al., "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 J. Solid-State Circuits*, vol. 57, no. 1, pp. 182–197, Jan. 2022.
- [209] A. S. Cassidy et al., "11.4 IBM NorthPole: An architecture for neural network inference with a 12 nm chip," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2024, pp. 214–215.
- [210] D. S. Modha et al., "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>
- [211] N. Jouppi et al., "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–14.
- [212] M. Kar et al., "A software-assisted peak current regulation scheme to improve power-limited inference performance in a 5 nm AI SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 67, Feb. 2024, pp. 254–256.
- [213] 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, May 2023.
- [214] Z. Zhou et al., "Spikformer: When spiking neural network meets transformer," 2022, *arXiv:2209.15425*.
- [215] Z. Wang et al., "Masked spiking transformer," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023, pp. 1761–1771.
- [216] X. Shi, Z. Hao, and Z. Yu, "SpikingResformer: Bridging ResNet and vision transformer in spiking neural networks," 2024, *arXiv:2403.14302*.
- [217] M. Imani et al., "FloatPIM: in-memory acceleration of deep neural network training with high precision," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 802–815.
- [218] Z. Lu et al., "An RRAM-based computing-in-memory architecture and its application in accelerating transformer inference," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 3, pp. 485–496, Mar. 2024.
- [219] L. Zhao et al., "RACE-IT: A reconfigurable analog CAM-crossbar engine for in-memory transformer acceleration," 2023, *arXiv:2312.06532*.
- [220] A. Moitra et al., "TRex-reusing vision Transformer's attention for efficient xbar-based computing," 2024, *arXiv:2408.12742*.
- [221] A. Bhattacharjee, A. Moitra, and P. Panda, "ClipFormer: Key-value clipping of transformers on memristive crossbars for write noise mitigation," 2024, *arXiv:2402.02586*.