

A Review on Edge Large Language Models: Design, Execution, and Applications

YUE ZHENG, Zhejiang University of Technology, China

YUHAO CHEN, Zhejiang University, China

BIN QIAN, Zhejiang University, China

XIUFANG SHI, Zhejiang University of Technology, China

YUANCHAO SHU*, Zhejiang University, China

JIMING CHEN*, Zhejiang University, China

Large language models (LLMs) have revolutionized natural language processing with their exceptional understanding, synthesizing, and reasoning capabilities. However, deploying LLMs on resource-constrained edge devices presents significant challenges due to computational limitations, memory constraints, and edge hardware heterogeneity. This survey provides a comprehensive overview of recent advancements in edge LLMs, covering the entire lifecycle — from resource-efficient model design and pre-deployment strategies to runtime inference optimizations. It also explores on-device applications across various domains. By synthesizing state-of-the-art techniques and identifying future research directions, this survey bridges the gap between the immense potential of LLMs and the constraints of edge computing.

CCS Concepts: • Computing methodologies → Artificial intelligence; • Computer systems organization → Embedded and cyber-physical systems; • Human-centered computing → Ubiquitous and mobile computing.

Additional Key Words and Phrases: edge computing, large language models, resource-efficient optimizations, on-device inference, LLM applications

ACM Reference Format:

Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. 2025. A Review on Edge Large Language Models: Design, Execution, and Applications. *ACM Comput. Surv.* 1, 1, Article 1 (January 2025), 36 pages. <https://doi.org/10.1145/3719664>

1 Introduction

Transformer-based large language models (LLMs) have made significant strides in recent years, reshaping the landscape of natural language processing (NLP). This rapid evolution has led to the emergence of several open-source LLMs, including Meta's LLaMA family [38, 176, 177], Google's Gemma [169, 170], and more recently, DeepSeek AI's DeepSeek series [58, 113]. The success of LLMs stems from their exceptional capabilities in natural

*Corresponding authors.

Authors' Contact Information: Yue Zheng, Zhejiang University of Technology, Hangzhou, China, zhengyue@zjut.edu.cn; Yuhao Chen, Zhejiang University, Hangzhou, China, csechenyh@zju.edu.cn; Bin Qian, Zhejiang University, Hangzhou, China, bin.qian0718@gmail.com; Xiufang Shi, Zhejiang University of Technology, Hangzhou, China, xiufangshi@zjut.edu.cn; Yuanchao Shu, Zhejiang University, Hangzhou, China, ycshu@zju.edu.cn; Jiming Chen, Zhejiang University, Hangzhou, China, cjm@zju.edu.cn.



Please use nonacm option or ACM Engage class to enable CC licenses

This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2025 Copyright held by the owner/author(s).

ACM 1557-7341/2025/1-ART1

<https://doi.org/10.1145/3719664>

language understanding, synthesizing, reasoning, and generation [35, 36], driving breakthroughs in applications like document summarization, question answering, and text reformulating [92, 97, 115]. These advancements have had profound implications across both academic and industrial domains, enabling the development of widely adopted tools like ChatGPT [135], Copilot [129] and Gemini [168]. The continued progress in LLMs underscores their transformative impact on artificial intelligence [23, 84, 194], human-computer interaction [61, 72, 87], and beyond.

While cloud-based deployment has traditionally supported LLMs' computational demands, there is a growing need to bring these models to resource-constrained edge devices, including personal agents [147, 194], office assistants [61, 168] and industrial Internet of Things (IoT) systems [76, 174]. Edge-based LLMs — executed directly on devices — offer key advantages: Firstly, local inference ensures faster responses and functionality without internet connectivity [19], critical for applications in robotics and autonomous systems [23, 31, 198]. Secondly, processing sensitive data on-device eliminates risks associated with cloud transmission [42, 168]. Lastly, on-device learning enables models to adapt to user-specific preferences and contexts [13, 86, 137, 143].

However, deploying LLMs on resource-constrained edge devices presents significant challenges. Firstly, **the computational and memory constraints** impose substantial limitations on LLM loading and inference. LLMs often consist of billions of parameters, resulting in massive memory footprints that exceed the RAM capacities of most edge devices [27]. For example, a LLaMA-2 [177] model with 7B parameters requires over 8GB of memory even in FP16 precision. Without compression techniques, edge devices risk latency spikes and memory overflow during model loading [112]. Moreover, the quadratic complexity of the self-attention mechanism with respect to sequence length exacerbates computational demands, creating severe throughput bottlenecks on edge Central Processing Units (CPUs), Graphics Processing Units (GPUs) or Neural Processing Units (NPUs) [156].

Secondly, **the heterogeneous nature of edge computing devices** complicates runtime inference optimizations. Edge devices range from smartphones with ARM CPUs and limited memory to IoT devices equipped with low-power chips. On mobile devices, frameworks like llama.cpp [49] and MLC LLM [171] optimize computational operators, while edge GPUs adopt approaches like vLLM [95] to alleviate memory bandwidth limitations and enhance throughput. Effective hardware-software co-design is critical to align workloads with hardware-specific capabilities. Additionally, the choice of hardware (e.g., CPUs, GPUs, or NPUs) and its integration with software frameworks directly affect inference efficiency, necessitating adaptable solutions tailored to diverse edge environments [225].

Lastly, **developing practical edge applications** remains challenging, particularly in bridging centralized LLM processing with distributed edge scenarios. In personal and enterprise applications, frameworks such as AutoDroid [194] and WebAgent [61] demonstrate the complexities of maintaining responsiveness and accuracy for task automation. For industrial systems like autonomous vehicles [23, 174], precise task prioritization and dynamic resource allocation are essential to balance LLM inference with real-time control processes. These domain-specific optimizations are vital to ensure LLMs meet real-world latency and reliability requirements on resource-constrained devices.

To address these challenges, we have devised a comprehensive optimization pipeline that integrates techniques across the entire lifecycle of edge-based LLM deployment, as illustrated in Fig. 1. Starting with pre-deployment methods such as quantization, pruning, and knowledge distillation, the pipeline enables the creation of compact, resource-efficient

models that preserve performance while reducing computational demands. These models are then deployed on edge devices, where runtime optimizations—spanning software-level strategies, hardware-level enhancements, and hardware-software co-design—ensure seamless adaptation to heterogeneous environments. Finally, the optimized models power a variety of on-device applications, from personal assistants to enterprise systems and industrial solutions, showcasing the practical impact of edge LLMs. This unified process effectively addresses key deployment challenges, demonstrating how offline compression and real-time optimizations together enable diverse real-world applications.

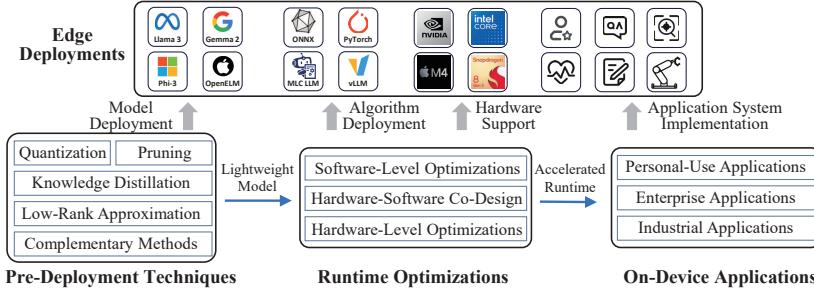


Fig. 1. Edge LLMs deployment, optimization and application pipeline.

Following this pipeline, this survey aims to provide a comprehensive exploration of the key areas involved in enabling LLMs on edge and mobile devices, including three crucial aspects, as summarized in Fig. 2. Specifically,

- **Offline Pre-Deployment Model Design Techniques.** It focuses on compression models to reduce size and ease deployment on edge devices. Traditional methods like quantization, pruning, knowledge distillation, and low-rank approximation face unique challenges with LLMs due to their scale, Transformer architecture, and diverse tasks [2, 77]. These challenges have motivated novel compression methods tailored for LLMs. Quantization reduces LLM size by representing weights and activations with fewer bits [112, 153]. Pruning removes unnecessary attention heads or other Transformer components, either structurally or unstructurally [94, 197]. Knowledge distillation transfers knowledge to smaller models [82, 110]. Low-rank approximation exploits matrix redundancy for efficient compression [69, 109]. Complementary methods, such as advanced pre-training strategies, data curation, and architectural optimization, further enhance compression effectiveness [127, 130, 169].
- **Online Runtime Inference Optimizations.** It introduces inference optimization techniques that improves the LLM performance on resource-constrained edge devices. Key strategies include software-level optimizations, hardware-software co-design, and hardware-level optimizations. Software-level optimizations encompass resource-aware scheduling strategies for cloud-edge collaboration [17, 161, 227], single-device inference scenarios [51, 154, 216], and lightweight frameworks for efficient memory management and tensor operations [95, 157]. Hardware-software co-design integrates software algorithms with specific hardware capabilities, facilitating efficient hardware profiling and enabling the implementation of hardware-aware inference algorithms [56, 186]. Hardware-level optimizations introduce commonly used edge hardware devices, highlighting their innovations in on-device LLM inference [172, 213].

- **On-Device LLM-Based Applications.** It showcases the practical impact of on-device LLMs across personal, enterprise, and industrial domains. In personal applications, they power AI assistants for tasks such as daily management, healthcare monitoring, and companionship, offering privacy-preserving and low-latency interactions [72, 119, 194]. In enterprise settings, on-device LLMs enhance productivity through message completion, meeting summarization, and secure local processing of sensitive data [105, 175, 234]. In industrial scenarios, they enable real-time and local processing capabilities like autonomous driving, fault localization, and anomaly detection, improving efficiency and safety in complex environments [54, 84, 174].

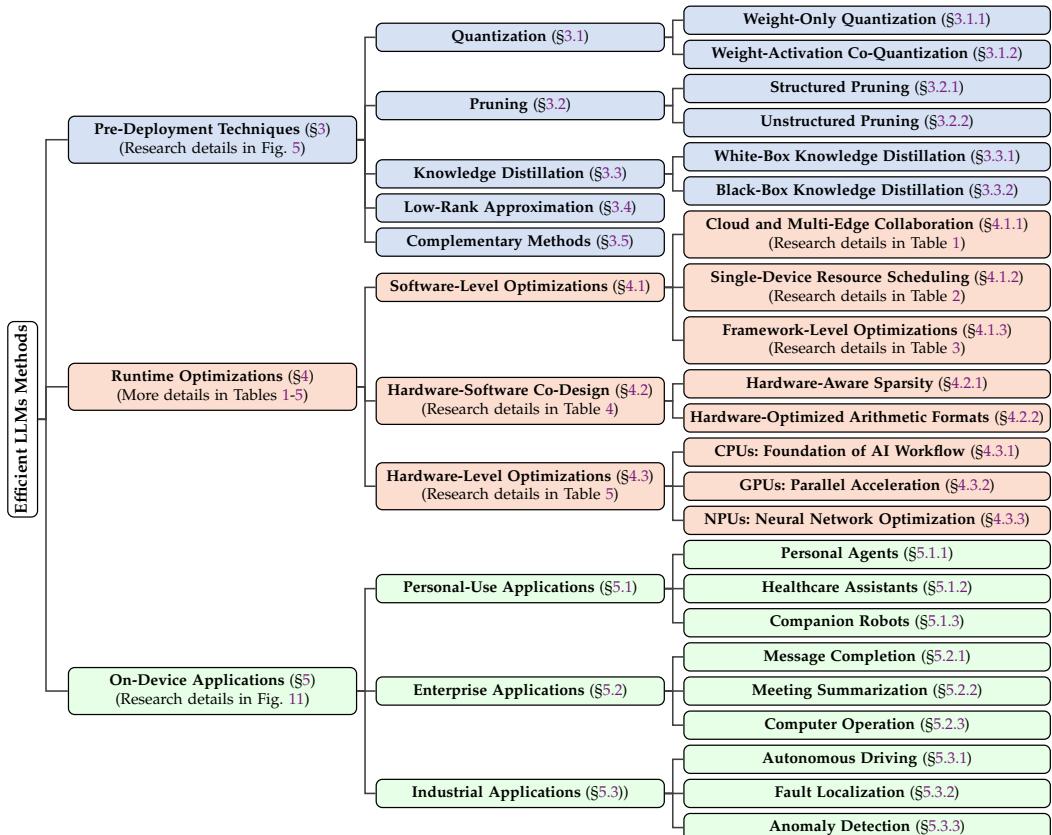


Fig. 2. A comprehensive overview of on-device LLMs, with detailed research categorized into: Pre-Deployment Techniques (§3, research details in Fig. 5), Runtime Optimizations (§5, details distributed across Tables 1-5), and On-Device Applications (§5, research details in Fig. 11). Each branch represents a key research direction with detailed methodology and implementation discussions in corresponding sections.

By employing these innovative techniques and methodologies, developers can harness the benefits of reduced model sizes and improved computational efficiency, facilitating the seamless integration of LLMs on edge devices. This advancement not only improves edge computing performance but also broadens the applicability of LLMs in resource-constrained environments, potentially revolutionizing the landscape of edge AI applications.

The remainder of this paper is structured as follows: Section 2 examines the widening gap between LLM complexity and edge device capabilities, reviews related work on efficient LLMs and edge computing, and analyzes research trends in on-device LLM optimizations, establishing the context for our survey. Section 3 and Section 4 provide a comprehensive examination of the state-of-the-art approaches for offline pre-deployment techniques and online runtime optimizations, respectively. Section 5 delves into the on-device applications of LLMs, highlighting their vast potential. Section 6 discusses future directions and open challenges in the field of on-device LLMs, while Section 7 concludes the survey, summarizing the key takeaways and insights gained.

2 Background and Related Work

The rapid advancement of LLMs and the increasing demand for edge computing have led to a growing interest in deploying these powerful AI models on resource-constrained devices [112]. However, this endeavor is hindered by a significant disparity between the computational complexity of LLMs and the capabilities of edge devices. Fig. 3 illustrates this widening gap, showing the evolution of estimated LLM pre-training FLOPs [18, 148] (measured in TFLOPs) compared to the AI performance of edge devices [30, 34, 145] (measured in TOPS) over time. The pre-training FLOPs for LLMs are estimated using the widely accepted heuristic $C \approx 6ND$, where N denotes the model's parameter count and D represents the total number of tokens used during pre-training [85].

This graph clearly illustrates the widening gap between the rapidly increasing computational complexity of LLMs and the relatively slow improvement in edge device capabilities. While LLMs have experienced a steep rise in estimated pre-training FLOPs, the AI performance of edge devices has improved at a much slower pace. This growing disparity underscores the critical need for researching efficient LLM deployment and implementation methods, which is precisely the focus of our survey.

While prior surveys in edge computing research have been conducted on efficient learning techniques on Deep Neural Network (DNN) architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) [98, 190], they have not adequately addressed the unique challenges posed by LLMs, including their larger model sizes and the complexities introduced by attention mechanisms. While research has addressed resource management [222] and security enhancements [192] in edge environments, these studies primarily focus on general deep learning, neglecting the specific needs of transformer-based LLMs in mobile edge computing.

Complementing these efforts, research in the realm of NLP has also made significant strides. Xu and McAuley [200] review methods for enhancing the efficiency of model compression and acceleration for pre-trained LLMs. Hedderich et al. [64] survey approaches for improving performance in low-resource NLP settings. Wan et al. [185] provide a comprehensive review of efficient LLMs research, organizing the literature into model-centric, data-centric, and framework-centric approaches. Treviso et al. [178] synthesize methods for conducting NLP under constraints of limited data, time, storage, or energy, emphasizing the trade-offs between performance and resource consumption. However, these works do not specifically address the challenges of deploying LLMs in edge environments. As a result, there is a critical need for focused research in this area.

Our survey uniquely provides a comprehensive analysis of LLMs in edge environments. The two most relevant surveys are Mobile Edge Intelligence for LLMs [144], which focuses primarily on collaborative resource management across different computing nodes, and

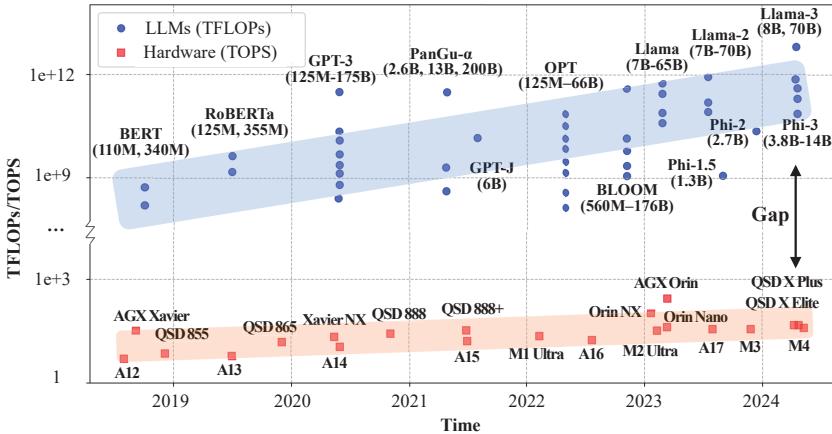


Fig. 3. LLMs (TFLOPs) vs Edge Devices (TOPS) Over Time. (QSD: Qualcomm Snapdragon.)

Personal LLM Agents [108], which explores the applications and scenarios of LLM assistants. However, the former does not address framework- and hardware-level optimizations for edge devices, and the latter lacks a systematic analysis of runtime optimizations on edge devices. To bridge this gap, we offer a holistic, top-down perspective on LLMs for edge devices, encompassing the entire optimization pipeline from offline pre-deployment model design techniques to online runtime inference optimizations and on-device LLM-based applications across various sectors. Our analysis covers model architectures, compression strategies, software-level optimizations, hardware-software co-design, and hardware enhancements for Transformer-based architectures at the edge. Additionally, we examine on-device application systems designed to maximize LLM performance under resource constraints. This multi-faceted approach distinguishes our survey, addressing challenges and solutions across the entire optimization pipeline for edge-deployed LLMs.

Fig. 4 shows the evolution of on-device LLM research from 2019 to 2024, categorized into pre-deployment techniques (blue), runtime optimizations (purple), and on-device applications (green). Offline pre-deployment techniques, such as quantization, pruning, knowledge distillation, and low-rank approximation have been consistently researched throughout the period. Online runtime optimizations, including software-level optimizations, hardware-software co-design, and hardware-level optimizations, have gained traction from 2022. The emergence of on-device applications for personal, enterprise, and industrial use cases is particularly notable in the latter half of the timeline, indicating a growing trend towards edge AI and mobile LLM deployments. This highlights the rapid advancement and diversification of LLM deployment for resource-constrained environments, emphasizing the growing importance of efficient on-device AI. Our survey provides a comprehensive analysis of these trends, offering a foundation for future research and practice in this field.

3 Offline Pre-Deployment Model Design Techniques

The proliferation of LLMs has engendered a burgeoning demand for their deployment on mobile and edge devices, driven by the imperative for enhanced privacy, reduced latency, and improved service availability in connectivity-constrained environments. This paradigm shift towards edge computing for LLMs. However, it presents significant challenges due to the LLMs' inherent computational complexity and substantial memory requirements [77]. Consequently, offline pre-deployment model design techniques have emerged

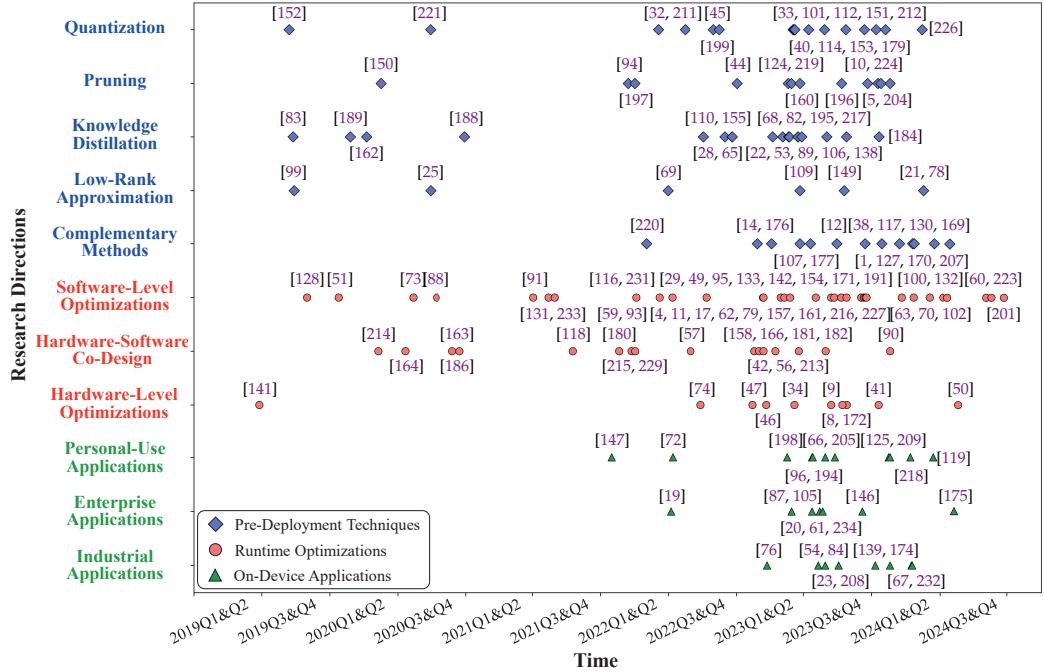


Fig. 4. Temporal Distribution of On-Device LLM Research.

as a pivotal strategy, aiming to substantially reduce the computational and memory footprint of LLMs while preserving their performance integrity. These techniques, applied prior to the models' deployment on target edge devices, facilitate efficient execution in resource-constrained environments.

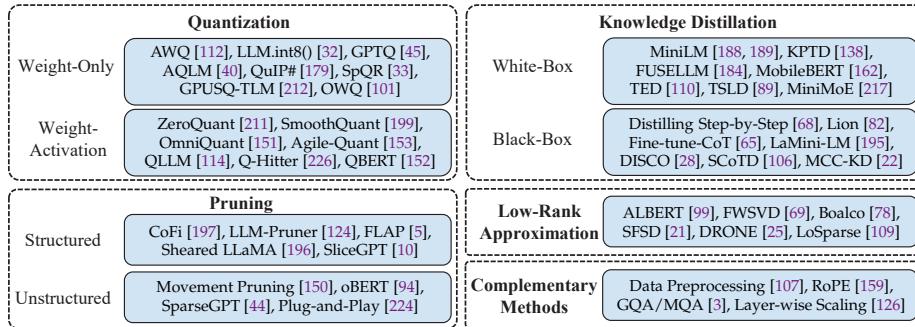


Fig. 5. An Overview of Offline Pre-deployment Model Design Techniques and Literature.

As illustrated in Fig. 5, these pre-deployment techniques encompass five primary categories: quantization, pruning, knowledge distillation, low-rank approximation, and complementary methods.

3.1 Quantization

Quantization is a compression technique that reduces the precision of numerical values in models, providing significant deployment advantages for edge devices. However, applying conventional methods to LLMs is challenging due to the architectural complexity of Transformer-based models, which rely heavily on attention mechanisms and high-dimensional representations [77]. These characteristics result in precision-sensitive tasks, and the high dynamic activation ranges in these models exacerbate quantization difficulties, often leading to performance degradation [15]. To address these challenges, specialized quantization methods have been developed, typically focused on two main areas:

- Weight quantization: Reduces the precision of model weights.
- Activation quantization: Reduces the precision of intermediate activations.

As shown in Fig. 6 (a), it can be categorized into weight-only quantization and weight-activation co-quantization.

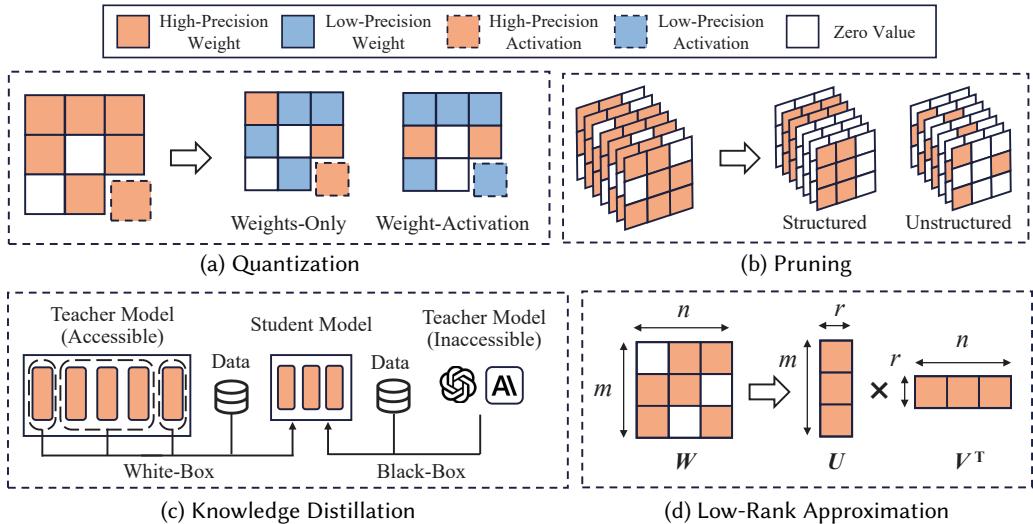


Fig. 6. Comparative Schematics of Offline Compression Methods for LLMs.

3.1.1 Weight-Only Quantization. Weight-only quantization reduces the precision of model weights from high-precision data types (e.g., 32-bit floating point) to lower-precision ones (e.g., 8-bit integers). This reduces memory usage and can speed up inference on resource-constrained devices. For instance, Dettmers et al. [32] propose LLM.int8(), which reduces memory requirements during inference without sacrificing performance. Similarly, Frantar et al. [45] introduce GPTQ, a post-training method that compresses LLM weights to lower bits, addressing layer-wise quantization challenges with the OBQ method [43]. Moreover, Lin et al. [112] propose AWQ, which preserves salient weights at high precision while quantizing others, optimizing computational complexity and energy consumption. Other weight-only quantization methods, such as AQMLM [40], QuIP# [179], and GPUSQ-TLM [212] also contribute to the advancements in this field by exploring different quantization strategies and optimization techniques.

Despite these advances, managing quantization errors in outlier weights remains a challenge. Techniques such as SpQR [33] and OWQ [101] address this issue by storing outlier

weights at higher precision, improving compression efficiency without compromising performance.

3.1.2 Weight-Activation Co-Quantization. While weight-only quantization offers benefits, it may leave activation values uncompressed. Weight-activation co-quantization, which quantifies both weights and activations, provides further compression. For example, ZeroQuant [211] combines group-wise weight and token-wise activation quantization. SmoothQuant [199] offering lossless 8-bit quantization by using per-channel scaling transformations. Furthermore, Agile-Quant [153] and Q-Hitter [226] employ activation-aware techniques to balance performance and real-time inference speed. In addition to the advancements discussed, other established technologies, including QBERT [152] and TernaryBERT [221] have also played a role in the development of weight-activation co-quantization for edge environments.

Despite these advancements, handling outlier issues in co-quantization remains a challenge. Methods like QLLM [114] and OmniQuant [151] address activation and weight outliers by leveraging adaptive calibration and learnable transformations, respectively.

In summary, quantization techniques for LLMs strike a balance between model compression, performance, and computational complexity. Weight-only methods, such as LLM.int8() [32], are ideal for quick deployment with moderate compression, while weight-activation co-quantization approaches like ZeroQuant [211] offer higher compression at the cost of increased complexity and potential accuracy loss.

3.2 Pruning

Pruning is a key technique for optimizing large language models (LLMs) by reducing the number of parameters, leading to smaller model sizes and faster inference. However, pruning in LLMs is challenging due to the complexity of their architecture and the varying significance of components such as attention heads. Conventional pruning methods, effective in CNNs, face limitations when applied to LLMs [104]. As shown in Fig. 6 (b), specialized pruning techniques for LLMs are generally categorized into structured and unstructured pruning, each with distinct trade-offs.

3.2.1 Structured Pruning. Structured pruning reduces the size of neural networks by removing entire structural components, such as neurons, channels, or layers. For example, CoFi [197] uses multiple pruning masks at different granularities to simultaneously remove layers and attention heads. LLM-Pruner [124] employs gradient-based pruning, removing non-critical units while preserving model performance, and incorporates LoRA [71] to recover performance post-pruning. LoRAPrune [219] leverages LoRA's weights and gradients for importance estimation. Sheared LLaMA [196] prunes specific layers and dimensions, using dynamic batch loading for domain-specific loss metrics, ideal for resource-constrained edge devices. SliceGPT [10] projects transformer block signal matrices onto principal components, removing redundant columns or rows to reduce size. FLAP [5] formulates importance metrics, enabling adaptive search for the optimal compressed model and implementing compensation mechanisms to mitigate performance loss.

3.2.2 Unstructured Pruning. Unstructured pruning removes individual weights or neurons, resulting in sparse models that are harder to optimize. Movement Pruning [150] adapts pruning decisions based on weight dynamics during fine-tuning, preserving important weights that exhibit significant movement. oBERT [94] introduces a second-order pruning

method supporting both unstructured and block pruning. SparseGPT [44] treats pruning as a sparse regression problem, allowing for one-shot pruning without retraining. Plug-and-Play [224] integrates activation-based importance to prune weights selectively, further improving pruning robustness in large-scale models. Wanda [160] prunes weights with the smallest magnitudes multiplied by the corresponding input activations, on a per-output basis. BESA [204] targets the overall pruning error with respect to individual transformer blocks and allocates layer-specific sparsity in a differentiable manner.

Overall, pruning techniques for LLMs offer various strategies for balancing size reduction and performance retention. Structured pruning methods like CoFi [197] and LLM-Pruner [124] provide controlled reductions, preserving architecture integrity, while unstructured methods such as Movement Pruning [150] and oBERT [94] offer greater flexibility but may result in irregular, sparse models. In practice, the choice of method depends on the specific deployment scenario, considering the trade-offs between model size, computational efficiency, and task performance.

3.3 Knowledge Distillation

Knowledge distillation transfers knowledge from complex teacher models to simpler student models to create computationally efficient alternatives without sacrificing performance. This process reduces model size, computational costs, and deployment requirements, while enhancing the student model’s diversity and stability. However, distilling knowledge from large teacher models, such as LLMs, remains challenging due to the difficulty in transferring internal representations and the complexity of attention mechanisms in Transformers [77]. As illustrated in Fig. 6 (c), distillation methods for LLMs are categorized into white-box and black-box approaches, each tailored to handle the scale and intricacies of LLMs.

3.3.1 White-Box Knowledge Distillation. White-box knowledge distillation leverages access to the teacher model’s architecture and parameters, using internal features and logits for knowledge transfer. MiniLM [189] extracts knowledge from the final Transformer [183] layer, alleviating the complexity of layer-to-layer mapping, while MiniLMv2 [188] extends this for task-agnostic compression. Other significant contributions include MobileBERT [162], and TinyBERT [83] collectively laying the foundation in this area.

To mitigate capacity mismatches between teacher and student models, techniques such as reverse Kullback-Leibler Divergence in MiniLLM [53] and layer-wise alignment in TED [110] improve the effectiveness of distillation. KPTD [138] and TSLD [89] further refine the process through entity-based transfer and token-scaled logit distillation, respectively. Mini-MoE [217] addresses the capacity gap by employing a Mixture-of-Experts (MoE) model, advancing the scalability of white-box distillation for LLMs. FUSELLM [184] integrates the capabilities of existing LLMs and transfers them into a single one, thereby elevating the capabilities of the target model. These advancements contribute to the evolving landscape of white-box knowledge distillation.

3.3.2 Black-Box Knowledge Distillation. Black-box knowledge distillation focuses solely on the outputs of the teacher model, bypassing internal model details. This approach is valuable when teacher models are proprietary or deployed via APIs. Chain-of-Thought (CoT) distillation aims to transfer reasoning abilities from LLMs to smaller student models [68, 106]. For example, Distilling Step-by-Step [68] uses LLM rationales to supervise task-specific models, improving dataset quality and model performance. Fine-tune-CoT [65] successfully transfers reasoning from large models (over 100B parameters) to students with as

few as 0.3B. SCoTD [106] further enhances performance in both supervised and few-shot tasks. Other methods for CoT distillation include Socratic CoT [155] and MCC-KD [22].

Black-box distillation also includes methods that focus on instruction-following, a crucial ability for LLMs in real-world applications. Lion [82] employs adversarial distillation to generate complex instructions, producing a 13B-parameter model comparable to ChatGPT [135]. DISCO [28] distills counterfactual data to enhance robustness, while LaMinILM [195] uses diverse instruction sets to compress large models effectively.

In conclusion, white-box methods, such as MiniLM [189], excel in scenarios where model internals are accessible. Conversely, black-box methods, such as Lion [82], are valuable in industrial or proprietary contexts where access to model internals is not accessible.

3.4 Low-Rank Approximation

Matrix factorization techniques, such as principal component analysis and regularized matrix factorization, have been pivotal in improving the generalization and interoperability of CNNs and RNNs. These methods reduce high-dimensional data to lower-dimensional spaces, enhancing model performance [77]. However, the large-scale parameters of Transformer-based LLMs pose challenges to traditional factorization approaches, as their computational complexity and unique structural elements, such as multi-head attention and feed-forward networks, require specialized adaptations [48]. To address these, low-rank approximation has emerged as a promising strategy within the Transformer framework [21, 25].

As illustrated in Fig. 6 (d), this technique approximates a high-dimensional matrix $W_{m \times n}$ with the product of two lower-rank matrices, $U_{m \times r}$ and $(V^T)_{r \times n}$, where r is much smaller than m and n . For example, ALBERT [99] applied low-rank approximation to vocabulary embeddings, decoupling hidden layer size from vocabulary size. FWSVD [69] enhances singular value decomposition by incorporating Fisher information to weight parameter importance, while DRONE [25] optimizes weight matrix compression by leveraging data distribution. Innovations such as LoSparse [109] introduce a method for separating coherent and incoherent neuron components, outperforming traditional pruning approaches. Additionally, inherent low-rank characteristics in LLMs have led to Bayesian optimization-based feature compression techniques [78]. SFSD [21] refine feature space approximation, and fast randomized algorithms using Gaussian sketches facilitate efficient low-rank factorization on consumer-grade hardware [149].

In summary, low-rank approximation effectively reduces parameters in LLMs, making it ideal for optimizing large-scale matrices like embedding layers and attention weights [206]. By minimizing redundancy, it significantly reduces storage and computation costs while maintaining performance.

3.5 Complementary Methods

Recent research in the field of LLMs has witnessed a paradigm shift towards developing innovative approaches specifically tailored for compact models, typically with around 10B parameters or less. As illustrated in Fig. 7, these technological advancements including:

- Data Preprocessing: Meticulous curation of high-quality training data [55].
- Grouped query and Multi-query Attention: Optimization of attention mechanisms [3].
- Rotary Position Embedding (RoPE): Advanced positional information encoding [159].
- Layer-wise Scaling: Strategic distribution of parameters across model layers [126].

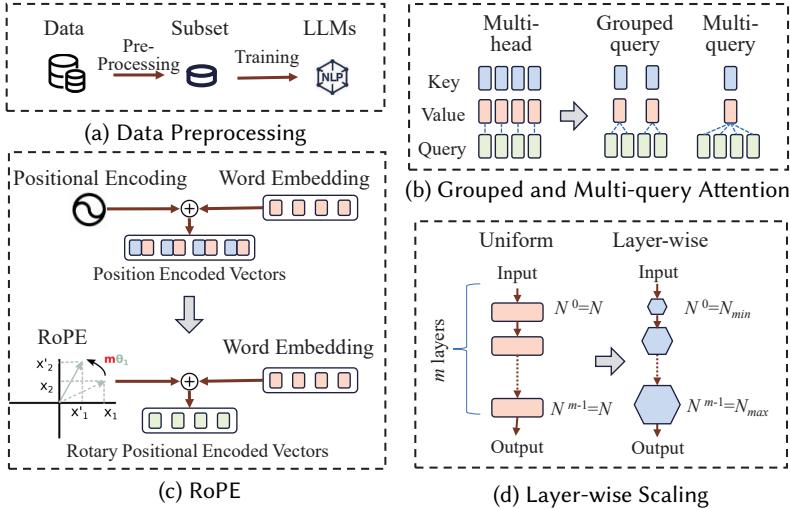


Fig. 7. Illustrations of complementary methods for designing efficient LLMs.

The integration of these techniques has resulted in compact LLMs to excel on resource-constrained edge devices, supporting high-performance AI applications in diverse scenarios. For instance, Meta’s LLaMA series [6, 38, 176, 177] have achieved remarkable performance gains through efficient pre-training on carefully curated datasets and leverages grouped query attention and RoPE to improve inference speed, reduce cache size, and expand context lengths. Similarly, Microsoft’s Phi series [1, 55, 130] showcases the impact of data preprocessing, using high-quality synthetic and filtered datasets to outperform larger models across benchmarks. Google’s Gemma models [169, 170] combine multi-query attention and RoPE to balance efficiency and accuracy, while Apple’s OpenELM family [127] employs layer-wise scaling and grouped query attention to optimize performance on low-powered devices. Other notable compact LLMs, such as Pythia [14], OPT [220], Qwen [12, 207], and MobileLLM [117] also benefit from similar optimizations, thereby reinforcing the growing potential of small-scale models for edge deployment.

While these methods significantly reduce parameter counts and resource demands, their development and optimization are predominantly executed by large organizations with substantial computational resources and domain expertise [127]. However, individual researchers can still benefit by using pre-trained compact LLMs from Hugging Face¹. These models, often with optimized weights and configurations, can be deployed on edge devices for efficient inference, lowering the entry barrier and enabling broader access to advanced AI capabilities in resource-limited settings [14, 176].

3.6 Comparative Analysis and Selection

The efficacy of offline pre-deployment techniques for LLMs depends on model architecture, deployment constraints, and performance requirements. The subsequent comparison provides an analysis to guide practitioners in selecting the most appropriate technique for their use case.

The selection of appropriate pre-deployment techniques should be informed by a comprehensive analysis of the deployment environment, hardware constraints, performance

¹Hugging Face: <https://huggingface.co/>

requirements, and LLM characteristics. For example, scenarios with strict memory constraints but moderate computational resources may benefit from weight-only quantization (e.g., LLM.int8() [32]) and structured pruning (e.g., CoFi [197]). Conversely, applications requiring maximum compression with some tolerance for accuracy loss might employ weight-activation co-quantization (e.g., SmoothQuant [199]) and aggressive unstructured pruning (e.g., Movement Pruning [150]). Knowledge distillation techniques such as MiniLM [189] or Lion [82] are suitable for transferring knowledge from larger to smaller models, with the choice between white-box or black-box approaches depending on teacher model accessibility. For models with large, redundant matrices, low-rank approximation methods like ALBERT [99] or LoSparse [109] can effectively reduce model size while preserving performance. In practice, combining multiple techniques often yields better results. ZeroQuant [211] demonstrates the efficacy of integrating quantization with knowledge distillation. NVIDIA's Nemotron-4 4B [134] exhibit high performance for on-device inference, which was pruned and distilled from Nemotron-4 15B [140]. DeepSeek-R1 [58] distills knowledge from compact models like Llama 3 [38], forming an edge-optimized 1.5B model further refined via large-scale reinforcement learning for mathematics, code generation, and complex reasoning, significantly enhancing inference capabilities.

When feasible, developing inherently efficient architectures such as LLaMA [176] or Gemma [169] may provide optimal performance within strict resource constraints. These approaches, typically developed by major tech companies with access to high-quality data and ample computing resources, demonstrate that careful architecture design and training strategies can yield highly efficient models without extensive post-training compression.

4 Online Runtime Inference Optimizations

We previously discussed offline optimization techniques in the pre-deployment phase, including model compression techniques and pre-training compact models for edge devices. This section shifts to runtime optimizations for efficient inference of LLMs directly on edge devices. As shown in Fig. 8, these optimizations are categorized into three areas: first, **Software-Level Optimizations**, which focuses on algorithmic strategies, resource scheduling, and framework optimizations independent of hardware; second, **Hardware-Software Co-Design**, which involves co-optimized solutions that leverage specific hardware features by adapting techniques such as sparsity and quantization to be hardware-friendly; and third, **Hardware-Level Optimizations**, which highlights innovations in hardware designed to improve the performance of LLMs.

4.1 Software-Level Optimizations

Software-level strategies can be categorized into Cloud and Multi-Edge Collaboration, Single-Device Resource Scheduling, and Framework-Level Optimizations, all of which focus on optimizing software algorithms, systems, frameworks or engines without modifications tailored to specific hardware.

4.1.1 Cloud and Multi-Edge Collaboration. Collaborative computing across cloud and edge devices plays a critical role in enhancing LLM performance by distributing computational workloads efficiently [37, 187]. Although it has seen extensive use in tasks like video analytics [7, 75, 80, 120, 122, 137, 225], multi-device and cloud-edge collaboration for LLM computation remains in its infancy and generally follows one of two approaches: split inference or speculative decoding, as illustrated in Fig. 9.

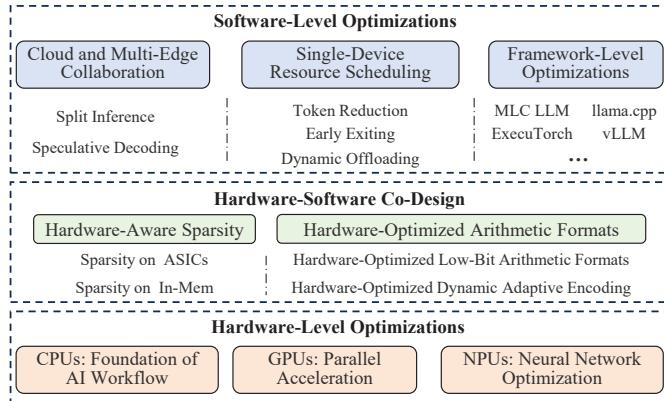


Fig. 8. A top-down view of runtime inference optimizations. (In-Mem: Computing/Processing in Memory.)

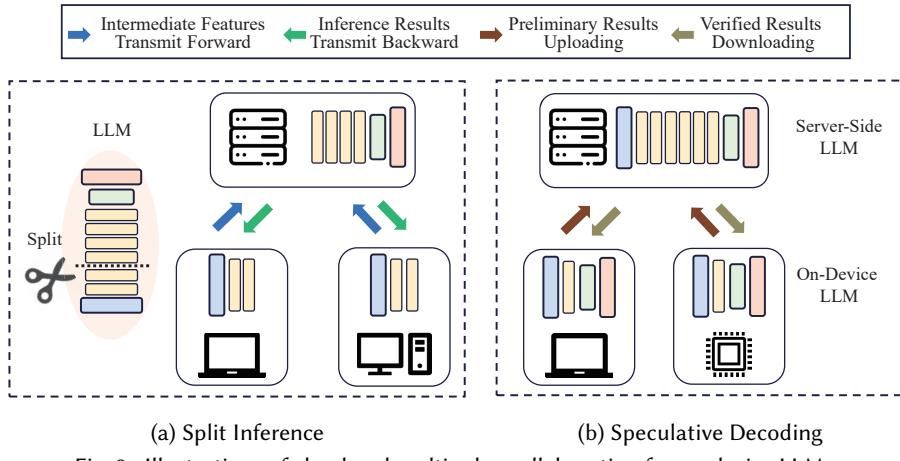


Fig. 9. Illustrations of cloud and multi-edge collaboration for on-device LLMs.

Split inference optimizes resource utilization and accelerates inference by partitioning computation across cloud and edge devices. PETALS [17] aggregates idle computational resources from diverse sources to ensure robust performance under dynamic network conditions. Voltage [70] improves throughput by distributing transformer layers across edge devices, achieving linear speed-ups through parallel computation. EASTER [60] addresses reliability challenges with adaptive partition strategies that maintain performance despite device failures. LinguaLinked [227] enhances split inference for mobile scenarios by aligning model segments with trusted device capabilities, enabling efficient data exchange. Approaches such as rewardless guidance [63] and Hepti [102] further optimize offloading decisions and dynamically adapt workloads to network and resource conditions. Building on these advancements, Zhang et al. [223] propose a tree-search framework to efficiently manage request batching and resource allocation, demonstrating the potential for adaptive and efficient cloud-edge collaboration.

Speculative decoding [103] addresses a key limitation of split inference, where the transmission of intermediate features often incurs significant communication overhead. In contrast, speculative decoding uploads preliminary results from edge devices and downloads only the verified results from a cloud-side LLM. This streamlined communication mechanism offers a practical alternative for scenarios where bandwidth or latency constraints are

critical. For example, SpecTr [161] uses lightweight models on edge devices to propose token drafts, enabling parallel validation by larger cloud models. Tabi [191] uses calibrated confidence scores to decide whether to upload tokens to the cloud for verification. Built atop speculative decoding, EdgeLLM [201] features a branch navigation and self-adaptive fallback strategy, enabling fast and accurate token generation.

To provide a concise overview of the advancements in cloud and multi-edge collaboration, Table 1 summarizes the primary literature. The table highlights the key methodologies, objectives, and application scenarios of each work, offering a comparative perspective on the progress in split inference and speculative decoding. By integrating these approaches, future research can address challenges such as latency, reliability, and resource constraints more effectively.

Table 1. Summary of the primary literature in cloud and multi-edge collaboration.

Techniques	Works	Features
Split Inference	PETALS [16]	Utilizes distributed idle computational resources to ensure robust performance under dynamic network conditions.
	Voltage [70]	Achieves linear speed-ups by distributing transformer layers across edge devices, leveraging parallel computation.
	EASTER [60]	Implements adaptive partition strategies to sustain performance despite device failures.
	LinguaLinked [227]	Optimizes split inference by aligning model segments with trusted mobile device capabilities for efficient data exchange.
	Hepti [102]	Dynamically adapts workload distribution and optimizes offloading decisions based on network and resource conditions.
Speculative Decoding	SpecTr [161]	Employs lightweight models on edge devices to generate token drafts, enabling parallel validation by larger cloud models.
	Tabi [191]	Utilizes calibrated confidence scores to decide token uploads for the cloud verification, balancing efficiency and accuracy.
	EdgeLLM [201]	Features a branch navigation and self-adaptive fallback strategy, enabling fast and accurate token generation.

4.1.2 Single-Device Resource Scheduling. Efficient scheduling for single-device inference builds upon a layered approach, where different techniques tackle optimization challenges from complementary perspectives. As illustrated in Fig. 10, recent advancements can be categorized into token reduction, early exiting, and dynamic offloading.

Token reduction focuses on reducing the computational load by selectively eliminating unnecessary tokens or inputs. For example, PoWER-BERT [51] eliminates non-critical word vectors. Length-Adaptive Transformer [88] adjusts input sequence lengths dynamically using a dropout variant called LengthDrop. LTP [91] prunes less significant tokens based on attention scores. Additionally, LLMLingua [79] compresses prompts iteratively at the token level, and AutoCompressors [29] reduce long context windows into more compact summary vectors. These techniques collectively aim to minimize the input size, ensuring that only the most important data is processed, thereby optimizing memory usage and reducing the computational burden on edge devices.

Early exiting terminates inference once a predefined confidence threshold is met, thereby reducing unnecessary computations during the model's forward pass. This technique can

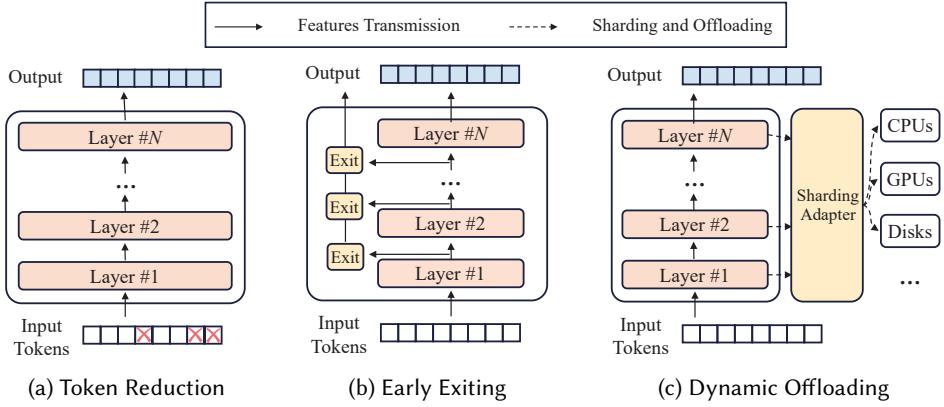


Fig. 10. Illustrations of single-device resource scheduling.

further benefit from the reduced input size. For example, PABEE [231] enhances the efficiency and robustness of pre-trained language models by integrating internal-classifiers at each layer. MPEE [93] combines horizontal and vertical early exit strategies for adaptive inference. FREE [11] proposes a shallow-deep module to synchronize the decoding process of the current token with previously stacked early-exit tokens. ConsistentEE [216] integrates reinforcement learning to determine the optimal early exit points, balancing inference speed with result accuracy. Techniques like LeeBERT [233] and FastBERT [116] enable adaptive computation via early exiting with layer-wise confidence evaluation, ensuring efficient resource utilization while maintaining output quality.

Dynamic offloading further optimizes resource utilization by distributing tasks across different processing units. Techniques like STI [59] introduce elastic pipelining, where model components are dynamically sharded and assigned to available resources. FlexGen [154] further refines this concept by leveraging a hybrid memory model, utilizing GPU, CPU, and disk memory with intelligent I/O scheduling for offloading computationally intensive tasks. LLM in a Flash [4] stores the model parameters in flash memory and dynamically loads them into DRAM for inference. They ensure balanced workloads, ensuring that edge devices can efficiently handle large-scale LLM inference with minimal bottlenecks.

Recent studies, such as MELTING Point [100], offer insights beyond traditional benchmarking by evaluating performance, memory, and energy demands across various model sizes and devices. These analyses reveal critical bottlenecks in computational efficiency, quality of experience, and accuracy, providing a foundation for advancements in algorithms and hardware. As summarized in Table 2, they enhance understanding of single-device resource scheduling optimization, addressing challenges in computational efficiency and resource allocation for edge inference tasks.

4.1.3 Framework-Level Optimizations. To meet the performance and portability demands of edge computing, framework-level optimizations focus on lightweight frameworks, libraries, and engines specifically designed. For instance, PyTorch, widely adopted for its flexibility and ecosystem support, has also extended its capabilities to edge computing with Execu-Torch [142], enabling efficient LLM inference through optimized execution plans for low-latency scenarios.

Table 2. Summary of the primary literature in single-device resource scheduling.

Techniques	Works	Features
Token Reduction	PoWER-BERT [51]	Eliminates non-critical word vectors to reduce computation while preserving accuracy.
	Length-Adaptive Transformer [88]	Dynamically adjusts input sequence lengths using a dropout-based technique called LengthDrop [88].
	LTP [91]	Prunes less significant tokens by leveraging attention scores to optimize inference efficiency.
	LLMLingua [79]	Compresses prompts iteratively at the token level to streamline input processing.
	AutoCompressors [29]	Reduces long context windows into more compact summary vectors for efficient processing.
Early Exiting	PABEE [231]	Improves the efficiency and robustness of pre-trained language models by integrating internal classifiers at each layer.
	MPEE [93]	Combines horizontal and vertical early exit strategies, enabling a more adaptive and efficient inference process.
	ConsistentEE [216]	Utilizes reinforcement learning to identify optimal early exit points, balancing inference speed with output accuracy.
	FREE [11]	Introduces a shallow-deep module to sync current token decoding with earlier early-exit tokens.
Dynamic Offloading	STI [59]	Implements elastic pipelining, dynamically sharding model components and assigning them to available resources.
	FlexGen [154]	Leverages a hybrid memory architecture, combining GPU, CPU, and disk memory with intelligent I/O scheduling to handle computationally intensive tasks.
	LLM in a Flash [4]	Stores model parameters in flash memory and dynamically loads them into DRAM for efficient inference.

Building on this, DNNFusion [131] optimizes mobile execution with advanced operator fusion, combining graph rewriting and fusion plan generation. SmartMem [132] reduces memory overhead by eliminating redundant layout transformations and selecting optimal memory layouts. PowerInfer [157] enhances runtime efficiency by preloading frequently activated neurons onto the GPU and processing less active neurons on the CPU to minimize memory and data transfer overhead. Furthermore, vLLM [95] introduces PagedAttention, inspired by virtual memory management, to segment attention key-value caches into blocks, enabling efficient memory sharing across sequences and requests. This approach improves memory efficiency while supporting quantization methods and optimized GPU kernels, making vLLM versatile for edge use.

The framework-level optimization ecosystem, as summarized in Table 3, includes tools tailored for LLM deployment on edge devices. The table categorizes frameworks by their descriptions, platform compatibility, and references to related works, helping readers identify suitable options and learn from previous implementations.

4.2 Hardware-Software Co-Design

Hardware-software co-design is a cross-disciplinary approach that integrates hardware and software optimization to enhance the performance, energy efficiency, and scalability of machine learning models. Unlike isolated optimization strategies, co-design aligns algorithmic innovations with hardware-specific capabilities, addressing the unique constraints

Table 3. Comparison of popular edge LLM frameworks, including examples of related works that have optimized or experimentally deployed these frameworks.

Name	Description	Mobile Support	Desktop Support	Related Works
MLC LLM [171]	Comprehensive LLM deployment framework leveraging machine learning compilation techniques.	✓	✓	OmniQuant [151], MobileLLM [117], MELTing Point [100].
llama.cpp [49]	Efficient C/C++ implementation for LLM inference.	✓	✓	EdgeLLM [201], MELTing Point [100].
ONNX Runtime [128]	A cross-platform machine-learning model accelerator, with a flexible interface to integrate hardware-specific libraries.	✓	✓	EASTER [60].
ExecuTorch [142]	On-device LLM across mobile and edge for PyTorch.	✓	✓	Llama 3.2 [6].
vLLM [95]	Utilize key-value cache stored in non-contiguous paged memory to efficiently inference.	✗	✓	AWQ [112], GPTQ [45], QLLM [114].
Neural Compressor [73]	State-of-the-art low-bit quantization and sparsity for run-time LLMs.	✗	✓	SmoothQuant [199], AWQ [112], GPTQ [45], SparseGPT [44].
TensorRT-LLM [133]	Advanced library for optimizing LLM inference powered by NVIDIA.	✗	✓	SmoothQuant [199], AWQ [112].
MLX [62]	Apple-developed array framework tailored for machine learning research on Apple silicon.	✗	✓	OpenELM [127].

of edge devices, such as limited power, memory, and computational resources. This synergy is essential for achieving high-efficiency inference in LLMs on resource-constrained platforms.

To provide a comprehensive overview of co-design optimizations, Table 4 compares software features and hardware platforms, highlighting the inference speedup and energy efficiency achieved by various approaches. Techniques are categorized into two primary areas: hardware-aware sparsity and hardware-optimized arithmetic formats. These classifications reflect different optimization focuses, enabling a clearer understanding of their respective contributions.

4.2.1 Hardware-Aware Sparsity. Hardware-aware sparsity focuses on integrating hardware-specific considerations into model design. This strategy can be further categorized into two primary approaches: sparsity on ASICs and sparsity on in-memory accelerators.

Sparsity on ASICs. ASIC-based accelerators achieve fine-grained control over model sparsity through custom circuit designs. Simulator-based methods explore early-stage designs: SpAtten [186] employs token pruning and a top-k ranking engine to prioritize token and head importance. Sanger [118] synergistically co-designs software that prunes attention matrices into dynamic structured patterns and hardware with a reconfigurable architecture. EdgeBERT [163] dynamically scales voltage and frequency, prunes networks, and quantizes floating points. TaskFusion [42] combines weight and activation sparsity with hardware-aware sub-task inference algorithms. AccelTran [182] enhances activation sparsity dynamically using a cycle-accurate simulator. Real-world tapeouts validate these designs: STP [165] optimizes latency and energy through mixed-precision computation and fine-grained power management. Similarly, C-Transformer [90] integrates spiking and non-spiking Transformers, achieving high sparsity and hardware utilization, illustrating the potential of hybrid architectures.

Sparsity on In-Memory Accelerators. In-memory accelerators address data movement challenges by embedding computation within memory arrays. Processing-in-Memory (PIM)

Table 4. Comparison of edge LLM-based co-design optimization techniques. (In-Mem: Computing/Processing in Memory. SW: Software. HW: Hardware.)

Type	Name	SW Features	HW Chips	Inference Speedup* or Energy Efficiency†
Hardware-Aware Sparsity	SpAtten[186]	Token-based sparsity and multi-head quantization.	ASIC 40nm simulator	$1095 \times^*$, $406 \times^t$ Jetson Nano GPU
	Sanger[118]	Threshold-based pruning prediction.	ASIC 55nm simulator	$1.47 \times^*$ SpAtten
	EdgeBERT[163]	Early exit prediction with pruning-aware optimizations	ASIC 12nm simulator	$53 \times^t$ Jetson Tegra X2 GPU
	TaskFusion [42]	Hardware-aware sub-task pruning with combined sparsity	ASIC 22nm simulator	$8.21 \times^*$, $19.83 \times^t$ Jetson Nano GPU
	AccelTran[182]	Activation sparsity via dynamic cycle simulation.	ASIC 14nm simulator	$280 \times^*$, $236 \times^t$ Apple M1 GPU
	4.60mm ² STP[165]	Entropy-based early exit and mixed-precision sparsity.	ASIC 12nm tapeout	18.1 TFLOPS/W (FP4) †
	C-Transformer[90]	Hybrid pruning for spiking/non-spiking Transformers.	ASIC 28nm tapeout	47.8 TOPS/W (INT8) †
	TransPIM[229]	Token-based dataflow optimization	In-Mem 65nm simulator	$114.9 \times^*$, $666.6 \times^t$ RTX 2080 Ti GPU
	X-Former[158]	Attention pruning with projection-based sparsity	In-Mem 32nm simulator	$19.6 \times^*$, $13 \times^t$ GTX 1060 GPU
	TranCIM[180]	Dynamic reconfiguration of streaming networks.	In-Mem 28nm tapeout	20.5 TOPS/W (INT8) †
	MulTCIM[181]	Token pruning with reshaped attention matrices.	In-Mem 28nm tapeout	101.1 TOPS/W (INT8) †
Hardware-Optimized Arithmetic Formats	GOBO[214]	Map weights to a dictionary of 3-/4-bit indexed representative floats.	ASIC 65nm simulator	$7 \times^*$, $3 \times^t$ Tensor Cores-based accelerator
	Mokey[215]	Quantizing values to 4-bit index counting instead of floating-point multiplications and accumulations.	ASIC 65nm simulator	$9 \times^t$ GOBO
	OliVe[56]	Afbfloat Format: An outlier-victim pair quantization strategy designed for sparse outlier distributions.	ASIC 22nm simulator	$4.5 \times^*$, $4 \times^t$ GOBO
	AdaptivFloat[164]	HFINT Format: Dynamically adjusts exponent bias across network layers to optimize precision.	ASIC 16nm simulator	$1.09 \times^t$ INT8-based accelerator
	ANT[57]	Flint Format: Adaptive bit-width selection between float and int.	ASIC 28nm simulator	$4 \times^*$, $3.33 \times^t$ AdaptivFloat

augments conventional memory hierarchies with compute units near memory, while Computing-in-Memory (CIM) embeds compute capabilities directly into memory cells for fine-grained operations. PIM-based sparsity designs like TransPIM [229] use token-based dataflows and high-bandwidth memory to minimize communication overhead, and X-Former [158] combines a software-level attention engine with nonvolatile memory and CMOS tiles. CIM-based sparsity designs, such as TranCIM [180], dynamically reconfigure streaming networks and bitline-transpose structures to reduce complexity. MulTCIM [181] addresses hybrid sparsity with runtime token pruning and attention matrix reshaping, improving scalability and efficiency.

Comparative Insights. ASIC accelerators excel in throughput optimization through fine-grained sparsity, whereas in-memory accelerators focus on reducing data movement with token-based designs. Together, these approaches demonstrate the transformative potential of hardware-aware sparsity in edge scenarios.

4.2.2 Hardware-Optimized Arithmetic Formats. Hardware-optimized arithmetic formats optimize inference performance by balancing computational accuracy with hardware efficiency. These formats employ two synergistic strategies: low-bit arithmetic and dynamic adaptive encoding.

Hardware-Optimized Low-Bit Arithmetic Formats. Low-bit arithmetic formats reduce the numerical precision of model parameters to fixed low-bit representations, tailored for specialized hardware accelerators. For example, GOBO [214] reduces 32-bit floating-point parameters to just 3 bits, significantly lowering power consumption. Mokey [215] quantizes parameters to 4-bit representations, enabling area- and energy-efficient hardware acceleration. OliVe [56] introduces an outlier-victim pair quantization approach, which sacrifices normal values to accommodate outliers, enabling more efficient memory alignment and boosting performance.

Hardware-Optimized Dynamic Adaptive Encoding. Dynamic adaptive encoding adjusts numerical precision based on runtime requirements, providing greater flexibility compared to fixed low-bit formats. AdaptivFloat [164] dynamically adjusts tensor-wide exponent biases to maintain accuracy at low bit widths. ANT [57] introduces a flexible adaptive data format, Flint, which combines floating-point and integer precision for low-bit quantization with minimal hardware overhead.

Comparative Insights. Low-bit formats offer energy efficiency and predictable performance but may sacrifice accuracy, whereas adaptive encoding provides precision flexibility at the cost of added complexity. Combining both strategies can yield robust and adaptable hardware-software designs for on-device LLMs.

In conclusion, hardware-software co-design represents a critical avenue for optimizing LLMs on edge devices. However, much of the research remains in the simulation phase, with limited real-world deployment on general-purpose heterogeneous platforms. By summarizing these approaches, we aim to highlight how tailored software optimizations can exploit hardware-specific features to outperform similar algorithms on existing hardware platforms. This discussion serves as a guide for future developments in deploying efficient and scalable LLMs in resource-constrained environments.

4.3 Hardware-Level Optimizations

The development of hardware architectures is essential for deploying LLMs on devices, enhancing inference speed and energy efficiency. Table 5 outlines prominent commercial hardware chips, emphasizing AI performance and related studies. The AI performance column highlights key metrics of System-on-Chips (SoCs), CPUs, and GPUs, while the Related Works column helps readers identify suitable hardware and explore relevant optimization strategies from previous implementations.

4.3.1 CPUs: Foundation of AI Workflow. CPUs remain fundamental in AI workflow management, providing essential flexibility and system-level integration. Recent advancements have expanded their LLM potential for LLM inference. For example, Arm Cortex A72 on Raspberry Pi 4B (8GB RAM) achieves low latency inference of LLaMA-7B [176] when paired with Agile-Quant [153]. Intel i9-13900k [74] enhanced by AQML [40], provides efficient inference for the LLaMA-2 model [177].

However, CPUs' architecture, optimized for sequential tasks, struggle with the parallel computations needed for LLM inference, especially in edge scenarios requiring real-time performance and energy efficiency. To address this, modern CPUs are often combined with

Table 5. Comparison of popular commercial hardware devices for on-device LLMs. (RPi: Raspberry Pi.)

Hardware	Chip	Type	AI Performance	Related Works
SoCs (CPU + iGPU + NPU)	Samsung Exynos	Mobile	Exynos 2400 equips with NPU and DSP AI Engine [41]	Llama 3.2 [6]
	Google Tensor	Mobile	Tensor G4 can run Gemini Nano [168] with multi-modality [50]	MELTING Point [100], Llama 3.2 [6], Gemini Nano [168], MobileLLM [117]
	Apple A	Mobile	A16 Bionic features 16-core Neural Engine with 17 TOPS [8]	MobileLLM [117], Llama 3.2 [6], MELTING Point [100], SpQR [33]
	Apple M	Laptop	M2 Ultra features a 32-core Neural Engine with 31.6 TOPS [9]	MELTING Point [100], LLM in a Flash [4]
	Qualcomm Snapdragon	Mobile	Snapdragon 8 Gen 3 achieves up to 20 tokens/sec for LLM inference [172]	EdgeLLM [201], MobileLLM [117], MobileBERT [162], Llama 3.2 [6], MELTING Point [100], Agile-Quant [153]
CPUs	Intel Core	Desktop	Intel i9-13900k combines Performance and Efficient cores with Deep Learning Boost [74]	SmoothQuant [199], EASTER [60], AQLM [40], Neural Compressor [73], FREE [11]
	ARM Cortex-A72 (on RPi 4/4B)	IoT	RPi 4B handles lightweight LLM inference efficiently with quantization and token pruning [153]	MELTING Point [100], Hepti [102], Agile-Quant [153]
GPUs	NVIDIA GeForce Laptop GPU	Laptop	RTX 4070 Laptop GPU achieves 321 TOPS with 8GB memory [46]	AWQ [112]
	NVIDIA GeForce Desktop GPU	Desktop	RTX 4090 offers 1321 TOPS with 24GB memory [47]	AWQ [112], SpQR [33], OWQ [101], AQLM [40], TensorRT-LLM [133], QuIP# [179], LLM in a Flash [4], Drone [25]
	NVIDIA Jetson Modules GPU	IoT	Jetson AGX Orin delivers 275 TOPS with 64GB memory [34]	EdgeLLM [201], MELTING Point [100], STI [59], Beyond the Cloud [223], TensorRT-LLM [133]

specialized accelerators (e.g., GPUs and NPUs) in SoCs like Apple’s M-series or A-series SoCs [8, 9], Google’s Tensor G4 [50], and Qualcomm Snapdragon chips [172]. These heterogeneous architectures aim to balance flexibility and performance by integrating general-purpose and parallel processing units. However, managing data flow and power consumption across these components remains a challenge.

4.3.2 GPUs: Parallel Acceleration. To address the inefficiency of CPUs, GPUs have been developed as highly parallel processors to accelerate computationally intensive tasks in edge computing. Equipped with hundreds or thousands of smaller cores, GPUs enable massive parallel computation for LLM inference. Modern edge GPUs, such as the NVIDIA Jetson series [34], feature specialized hardware such as Tensor Cores, optimized for matrix operations prevalent in LLM inference tasks. For example, Yuan et al. [213] demonstrates the feasibility of LLM inference on NVIDIA Jetson ORIN NX [34].

However, GPUs face significant power consumption challenges in edge scenarios, often necessitating hybrid models that distribute intensive computations between cloud and local resources. This strategy balances high-performance inference with the energy constraints of mobile and edge devices. Additionally, GPUs are frequently integrated with CPUs and NPUs in heterogeneous systems, complicating integration and resource management. Effective scheduling and task offloading among these processors are critical challenges in edge deployment, requiring advanced software frameworks to ensure efficient collaboration.

4.3.3 NPUs: Neural Network Optimization. NPUs are specialized accelerators designed to optimize neural network computations, offering substantial improvements in performance and energy efficiency for edge LLM inference. By employing low-precision arithmetic (e.g., INT8) and highly parallelized architectures, NPUs enable real-time inference with minimal

power consumption. Notable examples include Apple’s Neural Engine in the M-series or A-series chips [8, 9] and Qualcomm’s AI engines in Snapdragon processors [172], which boost on-device LLM capabilities and reduce cloud dependency.

However, NPUs face key limitations that restrict their broader applicability. They are optimized for a limited set of neural network operators, making them incompatible with many modern LLM architectures. This often forces fallback strategies like CPU-NPU co-processing, which can offset performance gains. Additionally, the rapid evolution of neural architectures further complicates matters, as NPUs struggle to keep up with model diversity and complexity, requiring extensive adaptations or failing to execute some models entirely [213]. In heterogeneous hardware environments, effective integration of NPUs with CPUs and GPUs is essential, as NPUs’ specialization may necessitate offloading tasks to general-purpose processors, complicating resource management.

In conclusion, online runtime optimizations enhance LLM performance on edge devices through software-level optimizations, hardware-software co-design, and hardware-level enhancements. These methods complement offline pre-deployment techniques, forming a holistic approach to on-device LLM optimization. Offline techniques reduce computational complexity and memory footprint via pre-training and fine-tuning, while runtime strategies focus on efficient resource utilization, dynamic adaptation, and scalability. Integrating both phases is crucial for high-performance LLMs on resource-constrained platforms.

5 On-Device LLM-Based Applications

Leveraging compact models and runtime optimizations, on-device LLMs enable efficient, low-latency, and privacy-preserving AI in edge environments. As shown in Fig. 11, on-device LLM-based application systems span personal, enterprise, and industrial domains.

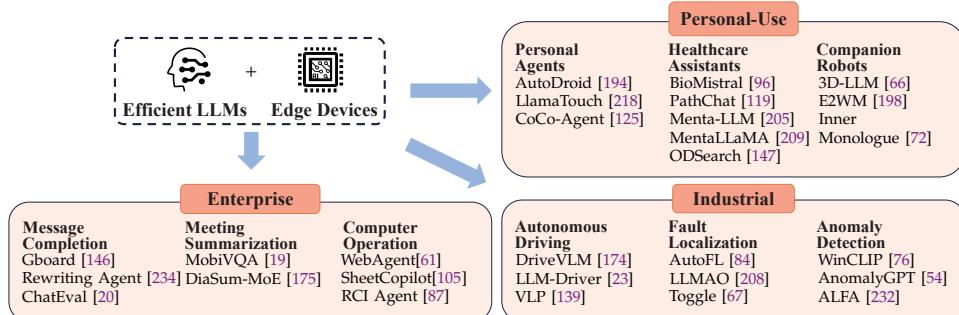


Fig. 11. Illustrations of on-device applications for LLMs.

5.1 Personal-Use Applications

On-device LLMs enable ubiquitous AI experiences in edge environments, driving advancements in personal assistants, healthcare assistants, and companion robots.

5.1.1 Personal Agents. On-device LLMs are transforming personal digital assistants by providing low-latency, privacy-preserving, and always available support tailored to individual needs. AutoDroid [194] automates daily tasks such as food ordering and health tracking, leveraging multi-granularity token pruning techniques to operate efficiently within the limited computational budget of mobile devices. Similarly, LlamaTouch [218] redefines mobile UI task automation evaluation, while CoCo-Agent [125] enhances GUI automation

for complex interactions with real-world environments. These advancements streamline personal productivity by automating routine tasks, including email management, photo editing, and scheduling [194].

5.1.2 Healthcare Assistants. In healthcare, on-device LLMs demonstrate versatility by providing domain-specific support in areas like clinical decision-making and mental health analysis. BioMistral [96], a 7B biomedical LLM, exemplifies the synergy between domain-specific fine-tuning and quantization to achieve low-latency inference. PathChat [119] integrates visual and natural language inputs through a multimodal projector module connecting a vision encoder to the LLaMA-2 model [177], transforming pathology education and research. Mental-LLM [205] and MentaLLaMA [209] focus on interpretable mental health analysis on social media platforms. On-device LLMs also enable wearable applications such as ODSearch [147], which provides a natural language interface for fitness tracker data, delivering near real-time search capabilities.

5.1.3 Companion Robots. On-device LLMs are transforming companion robots by enabling rapid, context-aware interactions and excelling in both verbal and non-verbal tasks. These robots can build connections through dialogue, negotiate with users, generate contextually appropriate responses, and execute actions requiring physical interaction. 3D-LLM [66] integrates 3D environment reasoning and planning, enabling robots to analyze spatial contexts and make informed decisions. E2WM [198] augments multiple compact LLMs (e.g., LLaMA [176] and OPT [220]) with embodied knowledge and skills, improving adaptability in dynamic environments. Inner Monologue [72] enhances these capabilities by allowing 1.3B InstructGPT [136] to reason over natural language feedback for better planning and execution in embodied tasks.

5.2 Enterprise Applications

On-device LLMs boost enterprise privacy and cost-efficiency, improving tasks like message completion, meeting summaries, and computer operations.

5.2.1 Message Completion. On-device LLMs streamline enterprise communication by generating context-aware responses, improving productivity through automated replies. For instance, Google's Gboard [146] integrates Gemini Nano [168] for real-time, context-sensitive suggestions, utilizing a cloud-edge framework where the cloud handles complex tasks and the edge manages lightweight interactions. Zhu et al. [234] propose a mobile-centric text rewriting LLM with efficient dataset generation and cascading mechanisms, while ChatEval [20] autonomously evaluates text quality, aligning closely with human judgment.

5.2.2 Meeting Summarization. In meetings, on-device LLMs are enabling automated summarization of key discussions and decisions, offering a privacy-preserving alternative to cloud-based solutions while maintaining high efficiency. For example, MobiVQA [19] delivers an efficient on-device visual question-answering system with attention-based early exit and question-aware pruning techniques. Tian et al. [175] propose a MoE framework for role-oriented dialogue summarization, enhancing contextual nuance in business settings.

5.2.3 Computer Operation. On-device LLMs facilitate natural language interfaces for computer operations, automating tasks and reducing user cognitive load. WebAgent [61] enables task execution on websites through real-time programming. SheetCopilot [105] simplifies spreadsheet interactions by translating natural language commands into actionable tasks. The RCI agent [87] uses LLMs to autonomously complete tasks on a computer using a keyboard and mouse, iteratively refining its output to improve performance. These systems leverage cloud-edge collaboration, with the cloud handling intensive planning and the edge executing lightweight, privacy-preserving operations.

5.3 Industrial Applications

On-device LLMs enhance industrial systems by enabling real-time analysis and decision-making in applications like autonomous driving, fault localization, and anomaly detection, while reducing network overhead and improving response times.

5.3.1 Autonomous Driving. On-device LLMs enhance autonomous driving by integrating linguistic understanding with navigation and decision-making. DriveVLM-Dual [174] uses a visual LLM to interpret urban environments and plan routes via natural language, efficiently deployed on NVIDIA Orin platforms [34]. LLM-Driver [23] leverages object-level vector inputs for explainable driving actions prediction. Similarly, VLP [139] strengthens both contextual understanding and memory foundation in self-driving systems. These advancements demonstrate the transformative potential of LLMs in autonomous driving.

5.3.2 Fault Localization. In software fault localization, LLM-based techniques offer significant improvements over traditional machine learning methods. AutoFL [84] generates bug explanations and identifies fault locations by navigating software repositories and mitigating LLM context length constraints. LLMAO [208] detects logic and security vulnerabilities without relying on extensive program analysis or test cases, outperforming deep learning-based methods. Toggle [67] employs tailored prompts and an adjustment module to localize and fix bugs at the token level, enabling fine-grained debugging.

5.3.3 Anomaly Detection. On-device LLMs also advance industrial anomaly detection by leveraging multi-modal capabilities and efficient feature aggregation [230]. WinCLIP [76] supports zero-shot and few-shot anomaly classification and segmentation using state word ensembles and prompt templates. AnomalyGPT [54] employs large vision-language models to identify anomalies directly, removing the need for manual threshold adjustments while showcasing strong few-shot learning performance. ALFA [232] addresses zero-shot visual anomaly detection by generating adaptive prompts, reducing semantic ambiguities, and fusing local pixel-level information for precise localization.

In summary, on-device LLMs use both offline and online optimization techniques to enable efficient, private, and responsive edge AI applications. Offline methods like quantization, pruning, and compact model design (e.g., Gemini Nano [168], BioMistral [96]) reduce model size and computational demands, making them suitable for resource-constrained devices. These are supported by runtime optimizations such as cloud-edge collaboration (e.g., WebAgent [61]), early exiting (e.g., MobiVQA [19]), and hardware acceleration (e.g., DriveVLM-Dual [174] on NVIDIA Orin [34]). Together, these strategies highlight the synergy between offline compression and runtime efficiency, driving innovation across personal, enterprise, and industrial applications.

6 Future Directions and Open Challenges

The rapid deployment of LLMs on mobile edge devices offers opportunities but faces key limitations. First, transitioning from centralized servers to decentralized edge nodes brings challenges due to device heterogeneity, including variations in computational power, communication latency, and fault tolerance [81, 120]. Second, many hardware-software co-design techniques show promise in simulations but often fail in real-world heterogeneous deployments [193]. Third, on-device LLMs struggle with complex scenarios requiring multi-hop reasoning, such as dynamic multi-agent interactions or real-time adaptation in personalized applications [26, 202, 203, 210]. These limitations hinder efficient inference and complicate the design of robust edge LLM-based systems. In the following, we explore key research directions and open challenges to address these limitations:

Compact LLM Architecture Development. Traditional Transformer-based models [183] are computationally intensive and impractical for edge devices due to their high memory and processing demands. To address edge device heterogeneity and reduce resource consumption, Mamba [52] introduces linear computational scaling through selective mechanisms, while Jamba [111] combines Transformer and Mamba layers for better adaptability. However, deploying these models on edge devices faces challenges [52, 111]: (1) their selective state space architecture demands substantial computational resources and memory bandwidth; (2) their recurrent nature inherently limits parallelization and hardware acceleration compared to transformers; and (3) optimizing them through quantization and pruning is complex due to interactions between selective state spaces and nonlinear components.

Innovative Edge-Cloud Collaboration. Cloud-edge collaboration can mitigate the limitation of insufficient memory and computing power of individual edge devices, which has seen progress with techniques like split inference [17] and speculative decoding [103]. However, distributed decoding strategies for edge deployment remain underexplored. Techniques such as disaggregated prefill and decoding [228], which separate tasks to optimize resource allocation, show potential for reducing inter-phase interference. However, to achieve efficient edge-cloud collaboration, it is crucial to address the challenges posed by fault tolerance and communication costs.

Heterogeneous Deployment with Simulation Techniques. Many hardware-software co-design approaches [56, 163, 186] for edge LLM inference have been evaluated in simulation or on specific ASIC prototypes, but the limitations are Cannot be used effectively on general-purpose hardware platforms. Future research should focus on improving scalability with heterogeneous hardware platforms, including CPUs, GPUs, and NPUs [213]. By optimizing for real-world scenarios, these methods can achieve faster, more energy-efficient LLM inference while addressing the challenges of dynamic workloads and diverse hardware configurations.

Graph-based LLM Development. On-device LLMs face significant limitations in handling graph-related inference tasks, which are critical for complex relationships and multi-hop reasoning in domains such as social networks, biology, and transportation. Recent approaches, including GraphRAG [39] for structured data extraction, GraphGPT [167] for graph interpretation through structural knowledge alignment, and GraphWiz [24] can address the multi-hop reasoning and complex scenarios limitation by extracting and interpreting graph-based data. However, the resource constraints of edge devices limit the scalability of these methods.

Multi-Agent Collaboration. While many on-device LLM applications, such as personal assistants [194, 218] and companion robots [66, 72], excel in single-agent tasks, they often struggle to operate effectively in dynamic and complex environments. To address this limitation, collaborative intelligence among heterogeneous agents leverages dynamic orchestration frameworks to enable multi-agent coordination, enhancing task performance beyond single-model systems [75, 123]. However, deploying such frameworks on edge devices remains challenging due to constraints in communication bandwidth, latency, and energy efficiency [122, 173].

Continual Learning and Personalization. On-device LLMs encounter significant challenges in real-time adaptation and personalization, especially under resource constraints. Continual learning provides a promising approach by enabling models to dynamically adapt while addressing issues such as catastrophic forgetting [13, 121, 225]. For example, Interactive Continual Learning [143] utilizes ongoing interactions to enhance model performance in complex scenarios. However, the limited parameter capacity of edge models restricts their ability to acquire new knowledge, and fine-tuning on-device demands substantial computational resources, complicating efficient adaptation [86, 137].

7 Conclusion

This survey provides a comprehensive review of recent advancements in enabling on-device LLMs, systematically exploring Offline Pre-Deployment Model Design Techniques, Online Runtime Inference Optimizations, and On-Device LLM-Based Applications. These components form a cohesive optimization pipeline: pre-deployment methods such as quantization and pruning create compact, efficient models; runtime techniques ensure adaptability and performance across heterogeneous environments; and diverse applications showcase the practical impact of edge LLMs. By addressing key challenges in efficiency and scalability, this survey offers valuable insights for researchers and practitioners, paving the way for accessible, sustainable AI solutions that unlock the full potential of LLMs.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 92467301, U23A20326, 62293511, and 62372414, the Key Research and Development Program of Zhejiang Province under Grant 2025C01061 and 2025C01012, and in part by the ZJUCSE-Enflame cloud and edge intelligence joint laboratory.

References

- [1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219* (2024).
- [2] Samira Abnar and Willem Zuidema. 2020. Quantifying Attention Flow in Transformers. *ACL* (2020).
- [3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *EMNLP* (2023).
- [4] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifar, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. Llm in a flash: Efficient large language model inference with limited memory. *ACL* (2024).
- [5] Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based Adaptive Structured Pruning for Large Language Models. *AAAI* (2024).
- [6] Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2024. Introducing Llama 3.2. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/. Accessed on December 5, 2024.

- [7] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. 2019. Demo: Video Analytics - Killer App for Edge Computing. In *ACM MobiSys*.
- [8] Apple. 2023. Apple debuts iPhone 15 and iPhone 15 Plus. <https://www.apple.com/newsroom/2023/09/apple-debuts-iPhone-15-and-iPhone-15-Plus>. Accessed on December 3, 2024.
- [9] Apple. 2023. Apple introduces M2 Ultra. <https://www.apple.com/newsroom/2023/06/apple-introduces-m2-ultra/>. Accessed on December 3, 2024.
- [10] Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefer, and James Hensman. 2024. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. *ICLR* (2024).
- [11] Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and Robust Early-Exiting Framework for Autoregressive Language Models with Synchronized Parallel Decoding. *EMNLP* (2023).
- [12] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).
- [13] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karinakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekyia: Continuous learning of video analytics models on edge compute servers. *USENIX NSDI* (2022).
- [14] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. *ICML* (2023).
- [15] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and Overcoming the Challenges of Efficient Transformer Quantization. *EMNLP* (2021).
- [16] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. 2023. Petals: Collaborative Inference and Fine-tuning of Large Models. *ACL* (2023).
- [17] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin Raffel. 2023. Distributed inference and fine-tuning of large language models over the internet. *NeurIPS* (2023).
- [18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS* (2020).
- [19] Qingqing Cao, Prerna Khanna, Nicholas D Lane, and Aruna Balasubramanian. 2022. MobiVQA: Efficient On-Device Visual Question Answering. *ACM UbiComp* (2022).
- [20] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. *ICLR* (2024).
- [21] Arnav Chavan, Nahush Lele, and Deepak Gupta. 2024. Surgical Feature-Space Decomposition of LLMs: Why, When and How? *ACL* (2024).
- [22] Hongzhan Chen, Siyue Wu, Xiaojun Quan, Rui Wang, Ming Yan, and Ji Zhang. 2023. MCC-KD: Multi-CoT Consistent Knowledge Distillation. *EMNLP* (2023).
- [23] Long Chen, Oleg Sinavski, Jan Hünermann, Alice Karnsund, Andrew James Willmott, Danny Birch, Daniel Maund, and Jamie Shotton. 2024. Driving with llms: Fusing object-level vector modality for explainable autonomous driving. *IEEE ICRA* (2024).
- [24] Nuo Chen, Yuhang Li, Jianheng Tang, and Jia Li. 2024. Graphwiz: An instruction-following language model for graph computational problems. *ACM SIGKDD* (2024), 353–364.
- [25] Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2021. DRONE: Data-aware Low-rank Compression for Large NLP Models. *NeurIPS* (2021).
- [26] Tao Chen, Yongjie Yang, Xiaoran Fan, Xiuzhen Guo, Jie Xiong, and Longfei Shangguan. 2024. Exploring the Feasibility of Remote Cardiac Auscultation Using Earphones. *ACM MobiCom* (2024).
- [27] Yuhao Chen, Yuxuan Yan, Qianqian Yang, Yuanchao Shu, Shibo He, and Jiming Chen. 2023. Confidant: Customizing Transformer-based LLMs via Collaborative Edge Training. *arXiv preprint arXiv:2311.13381* (2023).
- [28] Zeming Chen, Qiyue Gao, Antoine Bosselut, Ashish Sabharwal, and Kyle Richardson. 2023. DISCO: Distilling Counterfactuals with Large Language Models. *ACL* (2023).
- [29] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting Language Models to Compress Contexts. *EMNLP* (2023).
- [30] CPU-Monkey. 2024. AI Performance (NPU) CPU Benchmark List. https://www.cpu-monkey.com/en/cpu_benchmark-ai_benchmark. Accessed on July 18, 2024.
- [31] Yubin Dai, Bin Qian, Yangkun Liu, Yuxuan Yan, and Yuanchao Shu. 2025. Eros: Real-time Dense Mapping Made Easy on Mobile Devices. In *ACM HotMobile*.

- [32] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM. int8 () 8-bit matrix multiplication for transformers at scale. *NeurIPS* (2022).
- [33] Tim Dettmers, Ruslan A. Svirchevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefer, and Dan Alistarh. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *ICLR* (2023).
- [34] NVIDIA Developer. 2024. Jetson Modules, Support, Ecosystem, and Lineup. <https://developer.nvidia.com/embedded/jetson-modules>. Accessed on July 16, 2024.
- [35] Chenhe Dong, Yinghui Li, Haifan Gong, Miaoxin Chen, Junxin Li, Ying Shen, and Min Yang. 2022. A survey of natural language generation. *Comput. Surveys* (2022).
- [36] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *NeurIPS* (2019).
- [37] Jun Du, Tianyi Lin, Chunxiao Jiang, Qianqian Yang, C Faouzi Bader, and Zhu Han. 2024. Distributed Foundation Models for Multi-Modal Learning in 6G Wireless Networks. *IEEE Wireless Communications* (2024).
- [38] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).
- [39] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).
- [40] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme Compression of Large Language Models via Additive Quantization. *ICML* (2024).
- [41] Samsung Exynos. 2024. Exynos 2400 with 10-core CPU gets detailed after Galaxy S24 launch. <https://www.sammobile.com/news/exynos-2400-10-core-cpu-amd-rdna3-xclipse-940-gpu-specs-detailed/>. Accessed on December 11, 2024.
- [42] Zichen Fan, Qirui Zhang, Pierre Abillama, Sara Shoori, Changwoo Lee, David Blaauw, Hun-Seok Kim, and Dennis Sylvester. 2023. TaskFusion: An Efficient Transfer Learning Architecture with Dual Delta Sparsity for Multi-Task Natural Language Processing. *ACM/IEEE ISCA* (2023).
- [43] Elias Frantar and Dan Alistarh. 2022. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. *NeurIPS* (2022).
- [44] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *ICML* (2023).
- [45] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2022. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. *ICLR* (2022).
- [46] NVIDIA GeForce. 2023. Compare Gaming Laptops: GeForce RTX 40 Series. <https://www.nvidia.com/en-us/geforce/laptops/compare/>. Accessed on December 6, 2024.
- [47] NVIDIA GeForce. 2023. Compare GeForce Graphics Cards. <https://www.nvidia.com/en-us/geforce/graphics-cards/>. Accessed on December 6, 2024.
- [48] Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. 2021. Is Attention Better Than Matrix Decomposition? *ICLR* (2021).
- [49] Georgi Gerganov. 2023. llama.cpp. <https://github.com/ggerganov/llama.cpp>. Accessed on June 8, 2024.
- [50] Google. 2024. Google Tensor: the brains behind Pixel phones. https://store.google.com/intl/en_in/ideas/articles/google-tensor. Accessed on December 6, 2024.
- [51] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. *ICML* (2020).
- [52] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (2023).
- [53] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. MiniLLM: Knowledge Distillation of Large Language Models. *ICLR* (2023).
- [54] Zhaopeng Gu, Bingke Zhu, Guibo Zhu, Yingying Chen, Ming Tang, and Jinqiao Wang. 2024. Anomalygpt: Detecting industrial anomalies using large vision-language models. *AAAI* (2024).
- [55] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644* (2023).

- [56] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. *ACM/IEEE ISCA* (2023).
- [57] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2022. Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization. *IEEE/ACM MICRO* (2022).
- [58] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
- [59] Liwei Guo, Wonkyo Choe, and Felix Xiaozhu Lin. 2023. STI: Turbocharge NLP Inference at the Edge via Elastic Pipelining. *ACM ASPLOS* (2023).
- [60] Xiaotian Guo, Quan Jiang, Yixian Shen, Andy D Pimentel, and Todor Stefanov. 2024. EASTER: Learning to Split Transformers at the Edge Robustly. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).
- [61] Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. *ICLR* (2024).
- [62] Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. 2023. MLX: Efficient and flexible machine learning on Apple silicon. <https://github.com/ml-explore>. Accessed on June 8, 2024.
- [63] Ying He, Jingcheng Fang, F Richard Yu, and Victor C Leung. 2024. Large Language Models (LLMs) Inference Offloading and Resource Allocation in Cloud-Edge Computing: An Active Inference Approach. *IEEE Transactions on Mobile Computing* (2024).
- [64] Michael A. Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. 2021. A Survey on Recent Approaches for Natural Language Processing in Low-Resource Scenarios. *NAACL* (2021).
- [65] Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large Language Models Are Reasoning Teachers. *ACL* (2023).
- [66] Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 2023. 3d-llm: Injecting the 3d world into large language models. *NeurIPS* (2023).
- [67] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. 2024. A deep dive into large language models for automated bug localization and repair. *Proceedings of the ACM on Software Engineering* (2024).
- [68] Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. *Findings of ACL* (2023).
- [69] Yen-Chang Hsu, Ting Hua, Sungjen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language Model Compression with Weighted Low-Rank Factorization. *ICLR* (2022).
- [70] Chenghao Hu and Baochun Li. 2024. When the Edge Meets Transformers: Distributed Inference with Transformer Models. *IEEE ICDCS* (2024).
- [71] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *ICLR* (2021).
- [72] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2023. Inner Monologue: Embodied Reasoning through Planning with Language Models. *CoRL* (2023).
- [73] Intel. 2020. Intel Neural Compressor. <https://github.com/intel/neural-compressor>. Accessed on December 27, 2024.
- [74] Intel. 2022. Intel Launches 13th Gen Intel Core Processor Family Alongside New Intel Unison Solution. <https://www.intel.com/content/www/us/en/newsroom/news/13th-gen-core-launch.html>. Accessed on December 11, 2024.
- [75] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. *IEEE/ACM SEC* (2020).
- [76] Jongheon Jeong, Yang Zou, Taewan Kim, Dongqing Zhang, Avinash Ravichandran, and Onkar Dabeer. 2023. Winclip: Zero-/few-shot anomaly classification and segmentation. *IEEE/CVF CVPR* (2023).
- [77] Tianchu Ji, Shraddhan Jain, Michael Ferdman, Peter Milder, H Andrew Schwartz, and Niranjan Balasubramanian. 2021. On the Distribution, Sparsity, and Inference-time Quantization of Attention Values in Transformers. *ACL-IJCNLP* (2021).

- [78] Yixin Ji, Yang Xiang, Juntao Li, Wei Chen, Zhongyi Liu, Kehai Chen, and Min Zhang. 2024. Feature-Based Low-Rank Compression of Large Language Models via Bayesian Optimization. *Findings of EMNLP* (2024).
- [79] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. *EMNLP* (2023).
- [80] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A. Chien. 2019. Networked Cameras Are the New Big Data Clusters. In *ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*.
- [81] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible high-resolution object detection on edge devices with tunable latency. *ACM MobiCom* (2021).
- [82] Yuxin Jiang, Chunkit Chan, Mingyang Chen, and Wei Wang. 2023. Lion: Adversarial Distillation of Proprietary Large Language Models. *EMNLP* (2023).
- [83] Xiaochi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. *EMNLP* (2020).
- [84] Sungmin Kang, Gabin An, and Shin Yoo. 2024. A quantitative and qualitative evaluation of LLM-based explainable fault localization. *Proceedings of the ACM on Software Engineering* (2024).
- [85] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).
- [86] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. 2023. RECL: Responsive Resource-Efficient continuous learning for video analytics. *USENIX NSDI* (2023).
- [87] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *NeurIPS* (2024).
- [88] Gyuwan Kim and Kyunghyun Cho. 2021. Length-Adaptive Transformer: Train Once with Length Drop, Use Anytime with Search. *ACL-IJCNLP* (2021).
- [89] Minsoo Kim, Sihwa Lee, Janghwan Lee, Sukjin Hong, Du-Seong Chang, Wonyong Sung, and Jungwook Choi. 2023. Token-scaled logit distillation for ternary weight generative language models. *NeurIPS* (2023).
- [90] Sangyeob Kim, Sangjin Kim, Wooyoung Jo, Soyeon Kim, Seongyon Hong, and Hoi-Jun Yoo. 2024. 20.5 C-Transformer: A 2.6-18.1 μ J/Token Homogeneous DNN-Transformer/Spiking-Transformer Processor with Big-Little Network and Implicit Weight Generation for Large Language Models. *IEEE ISSCC* (2024).
- [91] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. *ACM SIGKDD* (2022).
- [92] Huan Yee Koh, Jiaxin Ju, Ming Liu, and Shirui Pan. 2022. An empirical survey on long document summarization: Datasets, models, and metrics. *Comput. Surveys* (2022).
- [93] Jun Kong, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2022. Accelerating Inference for Pretrained Language Models by Unified Multi-Perspective Early Exiting. *COLING* (2022).
- [94] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models. *EMNLP* (2022).
- [95] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. *ACM SOSP* (2023).
- [96] Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. 2024. Biomistrail: A collection of open-source pretrained large language models for medical domains. *Findings of ACL* (2024).
- [97] Huiyuan Lai and Malvina Nissim. 2024. A survey on automatic generation of figurative language: From rule-based systems to large language models. *Comput. Surveys* (2024).
- [98] Varsha S. Lalapura, J. Amudha, and Hariram Selvamuruga Satheesh. 2022. Recurrent Neural Networks for Edge Intelligence: A Survey. *Comput. Surveys* (2022).
- [99] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *ICLR* (2019).
- [100] Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. 2024. MELTING Point: Mobile Evaluation of Language Transformers. *ACM MobiCom* (2024).
- [101] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of Large Language Models. *AAAI* (2024).
- [102] Juhyeon Lee, Insung Bahk, Hoseung Kim, Sinjin Jeong, Suyeon Lee, and Donghyun Min. 2024. An Autonomous Parallelization of Transformer Model Inference on Heterogeneous Edge Devices. *ACM ICS*

- (2024).
- [103] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *ICML* (2023).
 - [104] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. 2020. Efficient Transformer-based Large Scale Language Representations using Hardware-friendly Block Structured Pruning. *EMNLP* (2020).
 - [105] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2024. SheetCopilot: Bringing software productivity to the next level through large language models. *NeurIPS* (2024).
 - [106] Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023. Symbolic Chain-of-Thought Distillation: Small Models Can Also “Think” Step-by-Step. *ACL* (2023).
 - [107] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463* (2023).
 - [108] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal ILM agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459* (2024).
 - [109] Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. LoSparse: Structured Compression of Large Language Models Based on Low-Rank and Sparse Approximation. *ICML* (2023).
 - [110] Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Less Is More: Task-aware Layer-wise Distillation for Language Model Compression. *ICML* (2023).
 - [111] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. 2024. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887* (2024).
 - [112] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *MLSys* (2024).
 - [113] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
 - [114] Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. 2023. QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models. *ICLR* (2023).
 - [115] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* (2023).
 - [116] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: A Self-distilling BERT with Adaptive Inference Time. *ACL* (2020).
 - [117] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024. MobileLm: Optimizing sub-billion parameter language models for on-device use cases. *ICML* (2024).
 - [118] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. *IEEE/ACM MICRO* (2021).
 - [119] Ming Y Lu, Bowen Chen, Drew FK Williamson, Richard J Chen, Melissa Zhao, Aaron K Chow, Kenji Ikemura, Ahrong Kim, Dimitra Pouli, Ankush Patel, et al. 2024. A multimodal generative AI copilot for human pathology. *Nature* (2024).
 - [120] Yan Lu, Shiqi Jiang, Ting Cao, and Yuanchao Shu. 2022. Turbo: Opportunistic enhancement for edge video analytics. *ACM SenSys* (2022).
 - [121] Yan Lu, Yuanchao Shu, Xu Tan, Yunxin Liu, Mengyu Zhou, Qi Chen, and Dan Pei. 2019. Collaborative Learning between Cloud and End Devices: An Empirical Study on Location Prediction. In *ACM/IEEE SEC*.
 - [122] Yan Lu, Zhun Zhong, and Yuanchao Shu. 2023. Multi-view domain adaptive object detection on camera networks. *AAAI* (2023).
 - [123] Ruilong Ma, Xiang Yang, Jingyu Wang, Qi Qi, Haifeng Sun, Jing Wang, Zirui Zhuang, and Jianxin Liao. 2024. HPipe: Large Language Model Pipeline Parallelism for Long Context on Heterogeneous Cost-effective Devices. *NAACL-HLT* (2024).
 - [124] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. *NeurIPS* (2023).

- [125] Xinbei Ma, Zhusong Zhang, and Hai Zhao. 2024. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. *Findings of ACL* (2024).
- [126] Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Delight: Deep and light-weight transformer. *ICLR* (2021).
- [127] Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. 2024. OpenELM: An Efficient Language Model Family with Open-source Training and Inference Framework. *arXiv preprint arXiv:2404.14619* (2024).
- [128] Microsoft. 2018. ONNX Runtime is a cross-platform inference and training machine-learning accelerator. <https://github.com/microsoft/ONNXRuntime>. Accessed on December 29, 2024.
- [129] Microsoft. 2023. Microsoft Copilot. <https://copilot.microsoft.com/>.
- [130] Sébastien Bubeck Mojan Javaheripi. 2023. Phi-2: The Surprising Power of Small Language Models. <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/>. Accessed on June 1, 2024.
- [131] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. 2021. DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion. *ACM PLDI* (2021).
- [132] Wei Niu, Md Musfiqur Rahman Sanim, Zhihao Shu, Jiexiong Guan, Xipeng Shen, Miao Yin, Gagan Agrawal, and Bin Ren. 2024. SmartMem: Layout Transformation Elimination and Adaptation for Efficient DNN Execution on Mobile. *ACM ASPLOS* (2024).
- [133] NVIDIA. 2023. TensorRT-LLM. <https://github.com/NVIDIA/TensorRT-LLM>. Accessed on June 9, 2024.
- [134] NVIDIA. 2024. Minitron-4B-Base. <https://huggingface.co/nvidia/Minitron-4B-Base>. Accessed on December 19, 2024.
- [135] OpenAI. 2022. Introducing ChatGPT. <https://openai.com/index/chatgpt>. Accessed on July 5, 2024.
- [136] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *NeurIPS* (2022).
- [137] Arathi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2023. Gemel: Model Merging for Memory-Efficient, Real-Time Video Analytics at the Edge. *USENIX NSDI* (2023).
- [138] Shankar Padmanabhan, Yasumasa Onoe, Michael Zhang, Greg Durrett, and Eunsol Choi. 2024. Propagating knowledge updates to lms through distillation. *NeurIPS* (2024).
- [139] Chenbin Pan, Burhaneddin Yaman, Tommaso Nesti, Abhirup Mallik, Alessandro G Allievi, Senem Velipasalar, and Liu Ren. 2024. VLP: Vision Language Planning for Autonomous Driving. *IEEE/CVF CVPR* (2024).
- [140] Jupinder Parmar, Shrimai Prabhumiye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, et al. 2024. Nemotron-4 15B Technical Report. *arXiv preprint arXiv:2402.16819* (2024).
- [141] Raspberry Pi. 2019. Raspberry Pi 4 on sale now. <https://www.raspberrypi.com/news/raspberry-pi-4-on-sale-now-from-£35>. Accessed on December 29, 2024.
- [142] Pytorch. 2023. ExecuTorch. <https://github.com/pytorch/executorch>. Accessed on December 12, 2024.
- [143] Biqing Qi, Xinquan Chen, Junqi Gao, Dong Li, Jianxing Liu, Ligang Wu, and Bowen Zhou. 2024. Interactive continual learning: Fast and slow thinking. *IEEE/CVF CVPR* (2024).
- [144] Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. 2024. Mobile Edge Intelligence for Large Language Models: A Contemporary Survey. *arXiv preprint arXiv:2407.18921* (2024).
- [145] Qualcomm. 2024. Snapdragon 8 Series Mobile Platforms | Qualcomm. <https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms>. Accessed on July 17, 2024.
- [146] Brian Rakowski. 2023. Pixel 8 Pro — the first smartphone with AI built in — is now running Gemini Nano, plus more AI updates coming to the Pixel portfolio. <https://blog.google/products/pixel/pixel-feature-drop-december-2023/>. Accessed on December 9, 2024.
- [147] Reza Rawassizadeh and Yi Rong. 2023. ODSearch: Fast and Resource Efficient On-device Natural Language Search for Fitness Trackers' Data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2023).
- [148] Yangjun Ruan, Chris J Maddison, and Tatsunori Hashimoto. 2024. Observational Scaling Laws and the Predictability of Language Model Performance. *arXiv preprint arXiv:2405.10938* (2024).

- [149] Rajarshi Saha, Varun Srivastava, and Mert Pilanci. 2023. Matrix Compression via Randomized Low Rank and Low Precision Factorization. *NeurIPS* (2023).
- [150] Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement Pruning: Adaptive Sparsity by Fine-Tuning. *NeurIPS* (2020).
- [151] Wensi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. *ICLR* (2023).
- [152] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *AAAI* (2020).
- [153] Xuan Shen, Peiyan Dong, Lei Lu, Zhenglun Kong, Zhengang Li, Ming Lin, Chao Wu, and Yanzhi Wang. 2024. Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge. *AAAI* (2024).
- [154] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. *ICML* (2023).
- [155] Kumar Sridhar, Alessandro Stolfo, and Mrinmaya Sachan. 2023. Distilling Reasoning Capabilities into Smaller Language Models. *ACL* (2023).
- [156] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. 2022. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proc. IEEE* (2022).
- [157] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. *ACM SOSP* (2024).
- [158] Shrihari Sridharan, Jacob R Stevens, Kaushik Roy, and Anand Raghunathan. 2023. X-former: In-memory acceleration of transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023).
- [159] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2024. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Elsevier Neurocomputing* (2024).
- [160] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A Simple and Effective Pruning Approach for Large Language Models. *ICLR* (2024).
- [161] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024. Spectr: Fast speculative decoding via optimal transport. *NeurIPS* (2024).
- [162] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: A Compact Task-Agnostic BERT for Resource-Limited Devices. *ACL* (2020).
- [163] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 2021. EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference. *IEEE/ACM MICRO* (2021).
- [164] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. 2020. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. *ACM/IEEE DAC* (2020).
- [165] Thierry Tambe, Jeff Zhang, Coleman Hooper, Tianyu Jia, Paul N Whatmough, Joseph Zuckerman, Maico Cassel Dos Santos, Erik Jens Loscalzo, Davide Giri, Kenneth Shepard, et al. 2023. 22.9 A 12nm 18.1 TFLOPs/W sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management. *IEEE ISSCC* (2023).
- [166] Thierry Tambe, Jeff Zhang, Coleman Hooper, Tianyu Jia, Paul N. Whatmough, Joseph Zuckerman, Maico Cassel Dos Santos, Erik Jens Loscalzo, Davide Giri, Kenneth Shepard, Luca Carloni, Alexander Rush, David Brooks, and Gu-Yeon Wei. 2023. 22.9 A 12nm 18.1TFLOPs/W Sparse Transformer Processor with Entropy-Based Early Exit, Mixed-Precision Predication and Fine-Grained Power Management. *IEEE ISSCC* (2023).
- [167] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. *ACM SIGIR* (2024), 491–500.
- [168] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricu, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).
- [169] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295* (2024).
- [170] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Huszenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118* (2024).
- [171] MLC Team. 2023. MLC LLM. <https://github.com/mlc-ai/mlc-llm>. Accessed on June 6, 2024.

- [172] Qualcomm Team. 2024. Snapdragon 8 Gen 3 Mobile Platform. <https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3>. Accessed on December 3, 2024.
- [173] Wenbin Tian, Chaojie Gu, Miao Guo, Shibo He, Jiawen Kang, Dusit Niyato, and Jiming Chen. 2024. Large-Scale Deterministic Networks: Architecture, Enabling Technologies, Case Study and Future Directions. *IEEE Network* (2024).
- [174] Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. 2024. Drivevlm: The convergence of autonomous driving and large vision-language models. *arXiv preprint arXiv:2402.12289* (2024).
- [175] Yuanhe Tian, Fei Xia, and Yan Song. 2024. Dialogue summarization with mixture of experts based on large language models. *ACL* (2024).
- [176] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [177] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [178] Marcos Treviso, Ji-Ung Lee, Tianchu Ji, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Colin Raffel, Pedro H. Martins, André F. T. Martins, Jessica Zosa Forde, Peter Milder, Edwin Simpson, Noam Slonim, Jesse Dodge, Emma Strubell, Niranjan Balasubramanian, Leon Derczynski, Iryna Gurevych, and Roy Schwartz. 2023. Efficient Methods for Natural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics* (2023).
- [179] Albert Tseng, Jerry Chee, Qingshao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. *ICML* (2024).
- [180] Fengbin Tu, Zihan Wu, Yiqi Wang, Ling Liang, Liu Liu, Yufei Ding, Leibo Liu, Shaojun Wei, Yuan Xie, and Shouyi Yin. 2022. A 28nm 15.59 μ J/token full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes. *IEEE ISSCC* (2022).
- [181] Fengbin Tu, Zihan Wu, Yiqi Wang, Weiwei Wu, Leibo Liu, Yang Hu, Shaojun Wei, and Shouyi Yin. 2023. 16.1 MuITCIM: A 28nm 2.24 μ J/Token Attention-Token-Bit Hybrid Sparse Digital CIM-Based Accelerator for Multimodal Transformers. *IEEE ISSCC* (2023).
- [182] Shikhar Tuli and Niraj K. Jha. 2023. AccelTran: A Sparsity-Aware Accelerator for Dynamic Inference With Transformers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).
- [183] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NeurIPS* (2017).
- [184] Fanqi Wan, Xiting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Knowledge fusion of large language models. *ICLR* (2024).
- [185] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. Efficient Large Language Models: A Survey. *Transactions on Machine Learning Research* (2024).
- [186] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. *IEEE HPCA* (2021).
- [187] Rongkai Wang, Yiyang Jing, Chaojie Gu, Shibo He, and Jiming Chen. 2025. End-to-End Multitarget Flexible Job Shop Scheduling With Deep Reinforcement Learning. *IEEE Internet of Things Journal* 12, 4 (2025), 4420–4434.
- [188] Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers. *ACL-IJCNLP* (2021).
- [189] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. *NeurIPS* (2020).
- [190] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* (2020).
- [191] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An efficient multi-level inference system for large language models. *ACM EuroSys* (2023).
- [192] Yingchao Wang, Chen Yang, Shulin Lan, Liehuang Zhu, and Yan Zhang. 2024. End-edge-cloud collaborative computing for deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* (2024).

- [193] Zichen Wang, Feng Yan, Tianyi Wang, Cong Wang, Yuanchao Shu, Peng Cheng, and Jiming Chen. 2025. Fed-DFA: Federated Distillation for Heterogeneous Model Fusion through the Adversarial Lens. *AAAI* (2025).
- [194] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered Task Automation in Android. *ACM MobiCom* (2024).
- [195] Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Aji. 2024. LaMini-LM: A Diverse Herd of Distilled Models from Large-Scale Instructions. *EACL* (2024).
- [196] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. *ICML* (2024).
- [197] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured Pruning Learns Compact and Accurate Models. *ACL* (2022).
- [198] Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. 2024. Language models meet world models: Embodied experiences enhance language models. *NeurIPS* (2024).
- [199] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. *ICML* (2023).
- [200] Canwen Xu and Julian McAuley. 2023. A survey on model compression and acceleration for pretrained language models. *AAAI* (2023).
- [201] Daliang Xu, Wangsong Yin, Hao Zhang, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. 2024. EdgeLLM: Fast On-device LLM Inference with Speculative Decoding. *IEEE Transactions on Mobile Computing* (2024).
- [202] Lilin Xu, Chaojie Gu, Rui Tan, Shibo He, and Jiming Chen. 2023. MESEN: Exploit Multimodal Data to Design Unimodal Human Activity Recognition with Few Labels. *ACM SenSys* (2023).
- [203] Lilin Xu, Keyi Wang, Chaojie Gu, Xiuzhen Guo, Shibo He, and Jiming Chen. 2024. GesturePrint: Enabling user identification for mmWave-based gesture recognition systems. *IEEE ICDCS* (2024).
- [204] Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. 2024. BESA: Pruning Large Language Models with Blockwise Parameter-Efficient Sparsity Allocation. *ICLR* (2024).
- [205] Xuhai Xu, Bingsheng Yao, Yuanzhe Dong, Saadia Gabriel, Hong Yu, James Hendler, Marzyeh Ghassemi, Anind K Dey, and Dakuo Wang. 2024. Mental-llm: Leveraging large language models for mental health prediction via online text data. *ACM UbiComp* (2024).
- [206] Yuxuan Yan, Qianqian Yang, Shunpu Tang, and Zhiguo Shi. 2024. Federa: Efficient fine-tuning of language models in federated learning leveraging weight decomposition. *arXiv preprint arXiv:2404.18848* (2024).
- [207] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671* (2024).
- [208] Aidan ZH Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. 2024. Large language models for test-free fault localization. *ACM/IEEE ICSE* (2024).
- [209] Kailai Yang, Tianlin Zhang, Ziyan Kuang, Qianqian Xie, Jimin Huang, and Sophia Ananiadou. 2024. MentalLLaMA: interpretable mental health analysis on social media with large language models. *ACM WWW* (2024).
- [210] Yongjie Yang, Tao Chen, Yujing Huang, Xiuzhen Guo, and Longfei Shangguan. 2024. MAF: Exploring Mobile Acoustic Field for Hand-to-Face Gesture Interactions. *ACM CHI* (2024).
- [211] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. *NeurIPS* (2022).
- [212] Chong Yu, Tao Chen, and Zhongxue Gan. 2023. Boost Transformer-based Language Models with GPU-Friendly Sparsity and Quantization. *Findings of ACL* (2023).
- [213] Jinliang Yuan, Chen Yang, Dongqi Cai, Shihe Wang, Xin Yuan, Zeling Zhang, Xiang Li, Dingge Zhang, Hanzi Mei, Xianqing Jia, et al. 2024. Mobile Foundation Model as Firmware. *ACM MobiCom* (2024).
- [214] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. *IEEE/ACM MICRO* (2020).
- [215] Ali Hadi Zadeh, Mostafa Mahmoud, Ameer Abdelhadi, and Andreas Moshovos. 2022. Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models. *ACM/IEEE ISCA* (2022).
- [216] Ziqian Zeng, Yihuai Hong, Hongliang Dai, Huiping Zhuang, and Cen Chen. 2024. ConsistentEE: A Consistent and Hardness-Guided Early Exiting Method for Accelerating Language Models Inference. *AAAI* (2024).

- [217] Chen Zhang, Yang Yang, Jiahao Liu, Jingang Wang, Yunsen Xian, Benyou Wang, and Dawei Song. 2023. Lifting the Curse of Capacity Gap in Distilling Language Models. *ACL* (2023).
- [218] Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024. LlamaTouch: A Faithful and Scalable Testbed for Mobile UI Automation Task Evaluation. *ACM UIST* (2024).
- [219] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2024. Lo-RAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning. *Findings of ACL* (2024).
- [220] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher De-wan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).
- [221] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware Ultra-low Bit BERT. *EMNLP* (2020).
- [222] Xiaojie Zhang and Saptarshi Debroy. 2023. Resource Management in Mobile Edge Computing: A Comprehensive Survey. *Comput. Surveys* (2023).
- [223] Xinyuan Zhang, Jiangtian Nie, Yudong Huang, Gaochang Xie, Zehui Xiong, Jiang Liu, Dusit Niyato, and Xuemin Sherman Shen. 2024. Beyond the Cloud: Edge Inference for Generative Large Language Models in Wireless Networks. *IEEE Transactions on Wireless Communications* (2024).
- [224] Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. 2024. Plug-and-play: An efficient post-training pruning method for large language models. *ICLR* (2024).
- [225] Yiwen Zhang, Xumiao Zhang, Ganesh Ananthanarayanan, Anand Iyer, Yuanchao Shu, Victor Bahl, Z Morley Mao, and Mosharaf Chowdhury. 2024. Vulcan: Automatic Query Planning for Live ML Analytics. *USENIX NSDI* (2024).
- [226] Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. 2024. Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache. *MLSys* (2024).
- [227] Junchen Zhao, Yurun Song, Ian Harris, Sangeetha Abdu Jyothi, et al. 2024. LinguaLinked: Distributed Large Language Model Inference on Mobile Devices. *ACL* (2024).
- [228] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. *USENIX OSDI* (2024).
- [229] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. Transpim: A memory-based acceleration via software-hardware co-design for transformer. *IEEE HPCA* (2022).
- [230] Qihang Zhou, Guansong Pang, Yu Tian, Shibo He, and Jiming Chen. 2024. AnomalyCLIP: Object-agnostic Prompt Learning for Zero-shot Anomaly Detection. *ICLR* (2024).
- [231] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT Loses Patience: Fast and Robust Inference with Early Exit. *NeurIPS* (2020).
- [232] Jiaqi Zhu, Shaofeng Cai, Fang Deng, Beng Chin Ooi, and Junran Wu. 2024. Do LLMs Understand Visual Anomalies? Uncovering LLM's Capabilities in Zero-shot Anomaly Detection. *ACM MM* (2024).
- [233] Wei Zhu. 2021. LeeBERT: Learned Early Exit for BERT with Cross-Level Optimization. *ACL* (2021).
- [234] Yun Zhu, Yinxiao Liu, Felix Stahlberg, Shankar Kumar, Yu-Hui Chen, Liangchen Luo, Lei Shu, Renjie Liu, Jindong Chen, and Lei Meng. 2024. Towards an On-device Agent for Text Rewriting. *Findings of NAACL* (2024).