



ISSN ONLINE: 2447-0228



RESEARCH ARTICLE

OPEN ACCESS

FROM CLOUD TO EDGE: A COMPREHENSIVE REVIEW OF OPTIMIZATION STRATEGIES FOR ON-DEVICE LARGE LANGUAGE MODELS

Mohammad Tarik Mohammad^{*1}, Qutaiba Ibrahim²

^{1,2}Computer Engineering Department, University of Mosul, Mosul, Iraq.

¹<http://orcid.org/0000-0002-4938-4477>, ²<http://orcid.org/0000-0002-0640-0561>

Email: mohammad.t.mohammad@uomosul.edu.iq

ARTICLE INFO

Article History

Received: February 2, 2026

Reviewed: July 30, 2026

Accepted: August 30, 2026

Published: September 30, 2026

Keywords:

Large language model ,
quantization ,
pruning,
distillation ,
llama.

ABSTRACT

Making artificial intelligence move to the edge is a promise of more privacy, lower latency, and offline, but it has to confront the realities of consumer hardware resources. The given paper represents a thorough taxonomy of the on-device LLM optimization technique, which summarizes the discoveries of 2023-2025. A granular review of optimization methods allows us to note a severe bifurcation in the research field: an excessive use of the Llama model family to benchmark against an increasing fragmentation of hardware targets, which is ranging between Raspberry pi to specialized Jetson modules. Our research underscores the fact that optimization that fits all should be left in the past and that to be successful, deployment should be constraint-based. The literature is proposed by us to compare the methods in terms of compression, accuracy, speedup, and sparsity. The results show that the most uniform energy advantages are obtained with quantization, although hybrid hardware-aware methods are the only way to break existing latency bottlenecks on the heterogeneous systems.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

Interest owing to the increasing interest in privacy-preserving, low-latency and cost-effective AI applications on mobile and edge devices [1]. The development of LLMs deploying on-device inferences and no longer as cloud-based systems is an indication of progress in model compressions, hardware acceleration, and system-level optimizations [2]. This change is notable because it will help to process in real time without depending on network connectivity, as it will solve privacy and cost of operation issues [3]. As an example, such models as LLaMA-2 consume more than 8GB of memory, which is beyond what edge devices can hold [4]. It will therefore need to reimplement effectively for on-device LLM inference to increase the reach of AI to a wide range of applications [5], [6]. The fundamental issue with this is that implementing computationally-intensive LLMs on the resource-constrained edge devices requires such as memory, processing power, and energy budgets [7]. Although many methods of compression exist: quantization, pruning, knowledge distillation, there remains a knowledge gap on both the topic of their combination and the actual implementation strategies [8], [9]. There are trade-offs between compression aggressiveness and model accuracy, controversies are around certain methods being highly compression with poor downstream task performance [10]. In addition, hardware heterogeneity and software framework constraints make runtime optimizations difficult [11]. The implications of this gap will make user experiences not optimal, latency, and limited applicability of LLMs on-device [12]. The introduction of the Transformer architecture to the Natural Language Processing (NLP) paradigm has largely been attributed [13]. In contrast to the Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks that process sequential data. The Transformer, depends on a self-attention mechanism that allows processing input sequences in parallel. Such an architectural allows modeling long-range dependencies and the scale that is defined by modern LLMs (e.g., GPT-4, Llama 2, and Falcon) [14], [15]. The scaled dot-product attention mechanism is the most fundamental component of this architecture and it calculates the relevance of each token in a sequence to all other tokens by producing query, key, and value representations.

This dynamically computes the importance of various words which enables the model to capture complex contextual relationships irrespective of distance among them in the text. Generative LLM implementations in the modern form mostly follow a decoder-only implementation of this architecture. It is made of a stack of identical layers that switch between multi-head self-attention mechanisms, which enable the model to pay attention to information in other representation subspaces, with position-wise feed-forward networks (FFNs) as shown in Figure 1. Although this design provides state of the art performance, it comes at a high computational cost; the size of the attention mechanism is scale in length of sequence and the feed-forward networks, which usually comprise most of the parameter count, require high memory bandwidth to load weights [16], [17]. These operations require large memory and computation, these required that will become a problem in implementation on edge devices with resource limits.

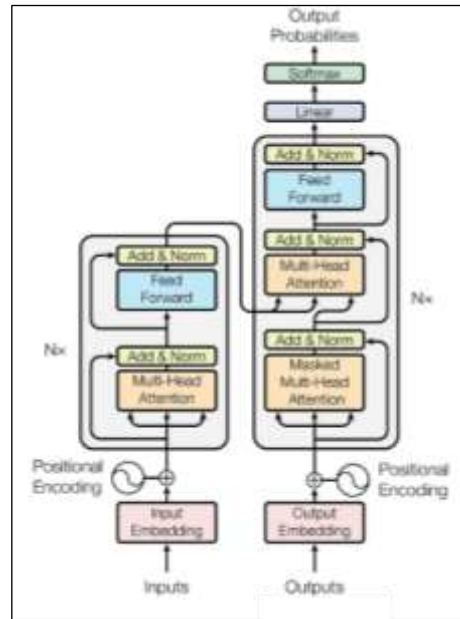


Figure 1: Transformer block diagram.

Source: [18].

Current barriers to the implementation of LLMs on embedded systems are a fundamental mismatch between the outrageous computational requirements of current models and the resource constraints imposed by edge hardware [19]. Whereas state-of-the-art LLMs require large amounts of high-bandwidth memory and computation, embedded systems have small RAM and thermal constraints creating a theoretical memory wall that requires the use of aggressive compression methods that tend to compromise accuracy [20]. This resource disparity is further augmented by the energy efficiency disparity, where dynamic power use is incredibly high and causes battery depletion and thermal throttling, and the prohibitive latency of cloud-offloading approaches that make them inappropriate to real-time, mission-important tasks like autonomous robotics [21]. Moreover, the use of the cloud infrastructure undermines data sovereignty and privacy of users because confidential data should be sent to foreign servers [22]. Lastly, current on-device optimization strategies, which are mostly architecture neutral, do not take advantage of the architectural benefits of heterogeneous edge processors (NPUs, DSPs), which leads to the inability to get theoretical improvements and does not allow truly efficient, private, and responsive edge AI to be implemented [23]. This systematic review aims at discussing the existing literature on running LLMs on-device as a comprehensive analysis to find the best compression and optimization techniques, as well as clarify their practical use in edge deployment. The review bridging the specified knowledge gap has several useful implications to both researchers and practitioners wishing to make on-device LLM inference more efficient and applicable. The worth is in the synthesis of various approaches into a logical knowledge to be used in the future research and development. The structure of the paper is as follows: Section 2 reports on the systematic review methodology. Section 3 and 4 respectively taxonomize optimization methods, such as quantization and pruning and the performance metrics, such as model size and latency respectively. Section 5 contains a decision guide of techniques to be used, and then part 6 of application domains. Lastly, Section 7 and 8 discuss the gaps in the research and conclude the research.

II. METHODOLOGY OF LITERATURE SELECTION

The literature search begins with the original research question, which is Current literature on running Large Language Models on-device, and continues to a few, more focused search statements. This step-by-step process of narrowing a general research question into a series of specific questions, as demonstrated in Figure 2, will allowed to make the literature search comprehensive and manageable, with each query resulting in a collection of papers that are highly relevant to a specific aspect of your subject. Research that falls within the past three years, fulfils the queries and eliminates duplication, was taken through the original research question to determine 217 papers. A total of 69 papers were selected after final filtered to reach the specific target of the literature.

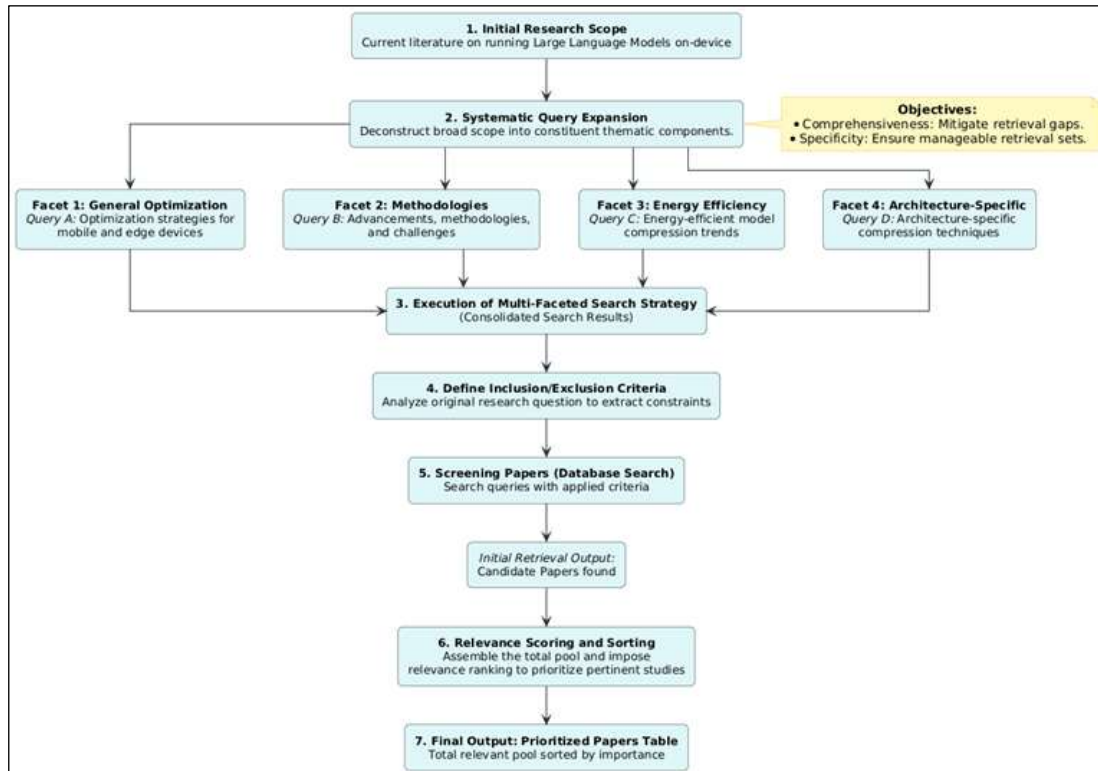


Figure 2: Literature methodology.
Source: Authors, (2026).

The existing body of knowledge on running Large Language Models on-device focuses most on methods of model compression, as illustrated in Figure 3, such as quantization, pruning, and knowledge distillation, which is needed to fit the LLMs into the constraints of resource-constrained systems such as edge devices. Hardware-aware techniques have been widely studied to provide optimization and acceleration techniques. The new trends in multi-optimization techniques focus on collaborative inference and new model architectures to democratize available and sustainable on-device deployment of LLM.

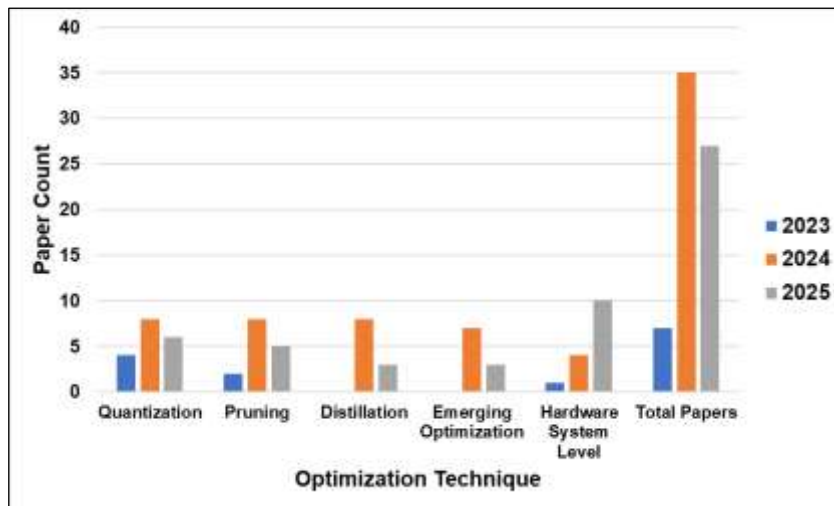


Figure 3: Literature review paper distribution.
Source: Authors, (2026).

III. LLM OPTIMIZATION TECHNIQUES

The rise in complexity of LLM is an imminent development, and the efficient deployment of the models on resource-constrained devices has become a crucial problem. The next block diagram, as shown in Figure 4, illustrates the organization of optimization policies aimed at scaling the difference between the different techniques. It divides methods into five major streams of methodology: Quantization and Pruning to reduce model size and precision; Knowledge Distillation and Low-Rank Decomposition to train models with efficiency and transfer knowledge efficiently; Hardware-aware technique to optimize the use of physical resources; and Emerging multi optimization method. Collectively, the strategies aim at the common objective of bringing to better performance, more energy efficient inference without significantly affecting the model accuracy.

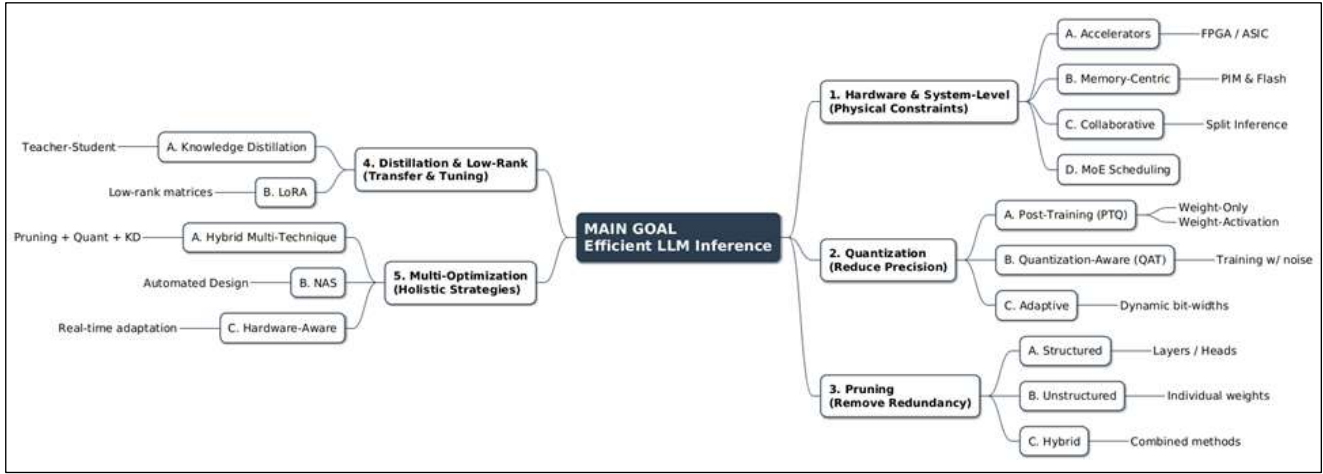


Figure 4: Large language model optimization techniques.
Source: Authors, (2026).

III.1 QUANTIZATION TECHNIQUE

One important compression technique is quantization, which quantizes the numerical values model weights and activations in standard floating-point models (e.g. FP32, FP16) to integers of lower bit-width such as INT8, INT4, or even INT2 [24]. This cut has substantial reduce memory overhead of LLMs by a large margin, faster computational throughput and less energy use when inferencing [8]. Quantization has typically three main approaches: Post-Training Quantization (PTQ) applies quantization to pre-trained models without extensive retraining, which is often using calibration datasets to minimize error. This technique can include weight, activation, and extreme low-bit approaches [25]. Quantization-Aware Training (QAT) Integrates quantization noise during the training or fine-tuning phase that allowing the model to adapt its parameters to minimize accuracy loss [26]. Mixed-Precision & Adaptive Quantization are Dynamically assigns different bit-widths to specific layers, channels, or tokens to optimize the trade-off between performance and efficiency [27]. A definite trend in the development of quantization strategies is well observed in Table 1, sign ● mean significant improvement or optimization reported, and ○ mean performance maintained or negligible degradation, as they have shifted to dynamic, hardware-conscious strategies. Post-Training Quantization is the most commonly used methodology of deploying the edges because it has low computational costs, and does not require retraining, which is costly. Recent developments in PTQ, including those of [28] and [29], have gone beyond the rounding of weights to include the activation-aware calibration and Hessian-aware calibration. These methods are useful in finding and defending outlier components, such that even when compression is aggressive (e.g. 4-bit weights) the accuracy loss is insignificant. It is interesting to note that [30], [31] emphasize the change in integer-only arithmetic carried out, which is vital in achieving the highest throughput on mobile NPUs that do not support floating-point functionality effectively. But when the pressure is on to reduce the bit-widths (to 2-bit or 3-bit accuracy) in use, uniform quantization can easily cause unacceptable performance degeneration. The need to find a solution therefore, led to the development of Mixed-Precision and Adaptive Quantization schemes. By fully dedicating greater bit-widths to the sensitive layers and aggressively quantizing the robust layers, the level of control over a balance between the model size and reasoning power can be increased (as shown by [32], [33]). This method is especially useful on heterogeneous edge platforms; such as, [34] use precision-slice quantization designed specifically to work on systolic arrays to achieve up to 4.3x speedups. Moreover, the salience-based approaches [27] and gradient-sensitive techniques [35] will guarantee the preservation of critical data, like outliers in multimodal frameworks, in greater accuracy, without compromising the energy benefit of low-bit inference. Finally, the evidence points to the future of on-device LLM in the co-design of quantization algorithms with custom hardware instruction sets to harness all the hypothetical efficiency improvements.

Table 1: Quantization Technique.

Study	Methodology	Compression / precision	Inference latency	Accuracy retention	Energy efficiency	Target architecture
LIN ET AL. (2025) [28]	PTQ (Weight-Only)	4x Reduction (Act-Aware)	●	●	●	GPU / IoT
ZENG ET AL. (2024) [32]	Mixed-Precision	Layer-Adaptive	●	●	●	Heterogeneous Edge
HUSOM ET AL. (2025) [36]	PTQ (General)	28-Model Benchmark	●	●	●	CPU (RPi 4)
ZHU ET AL. (2025) [33]	Mixed-Precision	Hardware-Aware Mixed	●	○	●	CPU (Arm)
MADHANEGHA ET AL. (2025) [37]	PTQ (General)	Llama Specific	●	○	●	Resource-Constrained
ZHANG ET AL. (2025) [29]	PTQ (Hessian-Aware)	Hessian-Aware	●	●	●	CPU-GPU

TAN ET AL. (2024) [30]	PTQ (Weight-Act)	Integer-Only	●	●	●	Mobile NPU
SHEN ET AL. (2024) [31]	PTQ (Weight-Act)	Activation-Guided	●	○	●	Commodity Edge
LEE ET AL. (2024) [38]	PTQ (Low-Rank)	Low-Rank Scaling	●	●	●	Agnostic
HUANG ET AL. (2024) [27]	Mixed-Precision	2-bit / 5.5x Memory	●	●	–	GPU (Integer)
HASAN (2024) [39]	PTQ & QAT	INT4 (Comparative)	●	●	●	Edge Devices
CHEN ET AL. (2024) [40]	Mixed-Precision	Channel-Wise	–	○	–	Diverse Devices
SHAO ET AL. (2024) [35]	Mixed-Precision	Gradient-Aware	●	●	●	Multimodal
BEHDIN ET AL. (2023) [41]	PTQ (Low-Bit)	Near 3-bit	–	●	–	–
SHEN ET AL. (2023) [42]	PTQ (Standard)	INT4	●	●	–	CPU (LLaMA)
KIM ET AL. (2023) [43]	PTQ (Low-Bit)	3-bit (Dense-Sparse)	●	●	–	GPU (Open Source)
DETTMERS ET AL. (2023) [44]	PTQ (Low-Bit)	3-4 bit (Sparse)	●	●	–	GPU
YANG ET AL. (2025) [34]	Mixed-Precision	Precision-Slice	●	●	●	Systolic Arrays

Source: Authors, (2026).

III.2 PRUNING TECHNIQUES

Pruning is a compression scheme model that eliminates redundant or unimportant parameters within neural networks hence decreasing the size of a model and computation needs [8]. It can lead to significant improvements in inference speed and energy efficiency by methodically removing weights, filters, or entire layers. pruning methods into three main streams: Structured Pruning removes coherent blocks of parameters (e.g., entire channels, heads, or layers). This method is highly hardware-friendly as it maintains dense matrix operations, making it ideal for GPUs and NPUs [45]. It is divided into two subtypes: Layer-Wise and Head-Wise, which target specific architectural components like attention heads or full layers [46]. also, Spatio-Temporal, which dynamically prunes based on input sequences (tokens) or time-steps [47]. Unstructured Pruning removes individual weights based on metrics like magnitude or gradients. While it yields higher compression rates, it often requires specialized hardware to realize speed gains [48]. Pruning approaches can combine with other optimization techniques (like Quantization or Low-Rank Adaptation) or uses unified frameworks to maximize efficiency [49]. Table 2 shows a conclusive move in recent literature switching apparently to Structured Pruning as compared to unstructured edge deployment techniques, chiefly because they have better hardware compatibility. Coherent blocks of parameters can be eliminated to retain dense matrix operations as illustrated by [50] and [45], an essential requirement to tapping into the parallel processing capability of GPUs and mobile NPUs. By [46] method has the overhead of sparse tensors typically experienced during unstructured pruning, which is counteracted, allowing concrete reductions in inference latency, up to 5.7 times in layer-dropping protocols with no specialized sparse hardware. Moreover, Spatio-Temporal strategie have become a potent paradigm in controlling the fluctuating computational complexity of the inputs of the LLM. These mechanisms are demonstrated by [51], [52] to offer a reduction in memory bandwidth by approximately half (46%) with no significant loss in perplexity. This flexibility is especially essential in real-time edge applications in which the length and the complexity of input sequences differ greatly. Also, the incorporation of pruning into Unified Frameworks, including the joint use with Low-Rank Adaptation [53] or quantization [54] shows that pruning is no more a standalone tool but a support element in an overall optimization pipeline. These hybrid methods effectively decouple model size and performance and the retention of high accuracy (95-97% at large compression ratios), and are able to enable large-scale LLMs on resource-constrained platforms such as the Jetson Orin Nano [48] by doing so.

Table 2: Pruning Technique.

Study	Methodology	Compression / precision	Inference latency	Accuracy retention	Energy efficiency	Target architecture
XU ET AL. (2025) [50]	Structured Pruning	20% Pruning (80% Mem Red.)	●	●	●	Edge (Structured)
FEDERICI ET AL. (2024) [51]	Spatio-Temporal	Dynamic Input (46% Mem Red.)	●	●	●	Simulated Mobile
HU ET AL. (2025) [45]	Structured Pruning	Fast Structured Framework	●	●	●	GPUs (Large LLMs)
YOU ET AL. (2024) [54]	Pruning w/ Quant.	Multiplication-Free Reparam.	●	●	●	Multiple Families
LIU ET AL. (2024) [55]	Structured Pruning	Adaptive Estimation Fusion	●	●	●	Edge Devices
HE ET AL. (2024) [52]	Spatio-Temporal	Activation Sparsification	●	●	●	Edge (Sparse Kernel)

LI & GUO (2024) [53]	Unified Frameworks	LoRA Pruning (20%)	●	●	–	Edge Deployment
OUDERAA ET AL. (2023) [56]	Gradient/Hessian	20-30% Pruning	●	●	–	OPT / LLaMA
GUO ET AL. (2023) [57]	Unified Frameworks	Collaborative (w/ LoRA)	●	●	●	LLaMA-7B
LI ET AL. (2024) [58]	Unified Frameworks	Progressive Pruning	●	●	●	LLaMA-7B
JIANG ET AL. (2025) [47]	Spatio-Temporal	Spiking LLM Compression	●	●	–	Spiking LLMs
CHEN ET AL. (2025) [11]	Structured Pruning	Structural Importance	●	●	–	Resource-Constrained
HU ET AL. (2024) [46]	Layer-Wise Pruning	50-60% Compression	●	●	–	Hardware-Agnostic
KALLAKURI ET AL. (2025) [48]	Gradient/Hessian	MaGrIP (50% Pruning)	●	●	●	Jetson Orin Nano

Source: Authors, (2026).

III.3 KNOWLEDGE DISTILLATION AND LOW-RANK DECOMPOSITION

Traditional knowledge (KL) distillation of LLMs entails training a smaller model to imitate the distributions of larger model , which is usually accomplished through a mixture of distillation loss (KL divergence between teacher and student predictions) and task-specific loss [59]. Nonetheless, qualities of large models such as vast parameter gap between teacher and student, the necessity to maintain emergent capabilities and generalization, and the computational cost of teacher outputs make it more challenging to adapt and fine-tune such models [60].LoRA uses transformer layers to train low-rank decomposition matrices that the pre-trained weights are frozen, allowing parameter-efficient fine-tuning and adaptation to large-scale datasets [61]. To edge deploy, LoRA provides two important benefits: first, low-rank matrices only need gradient calculation during fine-tuning. Second, after training, low-rank matrices can be combined with frozen weights, which removes inference time [62].Table 3 depicts that Knowledge Distillation serve as transforming factors to close the divide between huge foundation models and small edge devices. By [63] present extreme compression and show results up to a 93x vision-language model compression, with two-modality distillation, which allows performing complex multimodal inferences with devices that, until now, could not do it. Another point that is highlighted by the table is the paramount importance of Low-Rank Adaptation and effective fine-tuning methods of personalization. Such methods as Crayon adapter blending [64] and zeroth-order optimization [65] show that one can still obtain potentially impressive latency improvements (up to 35x over RAG) and memory-efficient on-device optimization without the huge computational cost of complete retraining.

Table 3: Knowledge Distillation Techniques.

Study	Methodology	Compression / reduction	Inference latency	Accuracy retention	Energy efficiency	Target architecture
TARECHA & CHOIRINA (2025) [59]	Knowledge Distillation	94% Size Reduction	●	●	●	Lightweight LLMs
RAVINDRAN (2025) [66]	Knowledge Distillation	Compact Models	●	●	●	Diverse Hardware
CAI ET AL. (2024) [63]	Dual-Modality Distill.	93x Reduction	●	●	–	–
SANDER ET AL. (2025) [60]	Self-Distillation	MLP Block Pruning	○	●	●	Intermittent Connect.
HUANGPU & GAO (2024) [67]	Comp. & Distillation	Llama 2 Optimization	●	●	●	Constrained Devices
CANTINI ET AL. (2024) [68]	XAI-Driven Distill.	High Compression	●	●	●	Edge AI
PENG ET AL. (2024) [69]	Derivative-Free FT	4-6.5GB Mem. Usage	–	●	●	Mobile Devices
BANG ET AL. (2024) [64]	Adapter Blending	35x Latency Improv.	●	●	●	Hybrid (Device/Server)
GAO ET AL. (2024) [65]	Zeroth-Order Opt.	Memory-Efficient FT	●	●	●	LoRA Integrations

Source: Authors, (2026).

III.4 HARDWARE & SYSTEM-LEVEL OPTIMIZATION

Whereas algorithmic compression methods such as quantization, pruning, and others can scale the theoretical model size and complexity, actual performance on the edge is always limited by the characteristics of the physical hardware, namely, the memory bandwidth, the energy supply, and the number of computational units. In order to decrease the gap between model compression and practice deployment, new studies have been done on architectural innovations and system-level orchestration [5]. These hardware and system-level optimizations can be divided into four distinct approaches: Hardware Accelerators, which are custom architectures, such as Field-Programmable Gate Arrays (FPGAs) and systolic arrays, designed to execute matrix operations more efficiently than general-purpose CPUs [70]. Memory-Centric Optimization techniques that address the "memory wall" by moving computation closer to data (Processing-in-Memory PIM) or optimizing the use of non-volatile storage (Flash) for models that exceed DRAM capacity [71]. Collaborative and Heterogeneous Computing frameworks that split inference workloads across multiple devices (e.g., distributed inference) or across heterogeneous components within a single device [11]. Mixture of Experts (MoE) Scheduling is system level strategies specifically designed for Mixture-of-Experts models to manage the dynamic loading and caching of "expert" blocks, which is critical for reducing latency in memory-constrained environments [72]. Table 4 appends a critical change in edge AI research: the dawning of the fact that software-level compression cannot be effective without a commensurate hardware architectural advancement. The greatest performance improvements are seen with Hardware Accelerators (FPGA/ASIC) whereby custom designs such as the FPGA accelerator by [73] and the hybrid systolic array by [70] are able to achieve speedups of up to 13.5x to 15.8x with considerable energy savings. These benefits result in specialized logic (e.g., POSIT multipliers in [74]) whereby the fetch-decode-execute overhead of general-purpose CPUs is eradicated. Other techniques such as Processing-in-Memory [75] relocate the computations into the DRAM arrays bypassing bandwidth constraints to realize a 17.8x performance improvement compared to earlier versions of PIM implementations. On the other hand, Flash-optimized plans [71], [76] take advantage of the large capacity of non-volatile storage to execute models that are 2x bigger than accessible DRAM, essentially removing model size as a constraint on system memory. Lastly, intelligent scheduling becomes essential due to the emergence of Collaborative and MoE Orchestration [77], [78] as well. These frameworks allow dynamically routing "expert" blocks and allocating workloads to heterogeneous clusters such that the stalling of the execution pipeline by the erratic memory access patterns of modern LLMs is eliminated, and time-to-first-token (TTFT) is improved up to 6x.

Table 4: Hardware & System-Level technique.

Study	Technique classification	Optimization method	Inference latency	Accuracy / capability	Energy efficiency	Target architecture
NAVARDI ET AL. (2025) [5]	System Orchestration	SW/HW Framework Opt.	●	○	●	Heterogeneous Edge
CHEN ET AL. (2025) [70]	Accelerator Design	Hybrid Systolic Array	●	○	●	Edge GPUs
XU ET AL. (2024) [73]	Accelerator Design	FPGA Accelerator	●	–	●	Embedded FPGA
WANG ET AL. (2025) [76]	Accelerator Design	SpeedLLM (Pipeline Fusion)	●	–	●	FPGA (TinyLlama)
LI ET AL. (2025) [79]	Accelerator Design	Hummingbird (FPGA)	●	–	●	Embedded FPGA
WEI ET AL. (2024) [80]	System Orchestration	T-MAC (LUT-Based)	●	–	●	CPUs
LI ET AL. (2025) [81]	MoE Optimization	CoMoE (Expert Aggreg.)	●	–	–	Mobility / Edge
ZHU ET AL. (2025) [78]	MoE Optimization	Expert Sched. & Caching	●	–	●	Memory-Constrained
TANG ET AL. (2024) [82]	MoE Optimization	HOBBIT (Mixed Prec.)	●	○	●	Edge Device
METCALFE ET AL. (2024) [77]	Collab. / Hetero.	Heterogeneous Multi-Dev.	●	●	–	CPU / GPU / NPU
MACARIO ET AL. (2025) [83]	Collab. / Hetero.	Distributed Inference	●	–	–	Low-Cost Clusters
ALIZADEH-VAHID ET AL. (2023) [71]	Memory-Centric (Flash)	Flash-Optimized Inf.	●	●	–	Flash Systems
JANG ET AL. (2025) [75]	Memory-Centric (PIM)	PIMPAL (In-DRAM)	●	–	●	In-DRAM PIM
GLINT ET AL. (2025) [74]	Accelerator Design	AxLaM (POSIT Mult.)	●	–	●	Specialized Accel.
LEE & HA (N.D.) [84]	Memory-Centric (PIM)	PIM for Linear Layers	●	–	–	Samsung PIM
WANG ET AL. (2025) [85]	Memory-Centric (Flash)	Neuralink (Co-Activation)	●	–	●	Smartphone

Source: Authors, (2026).

III.5 MULTI OPTIMIZATION TECHNIQUES

With the complexity of Large Language Models, Scientists are resorting to more sophisticated approaches, where efficiency is considered as being a multi-dimensional issue that considers the model architecture, data it is fed with, and the hardware it is supposed to be running on [86]. This part can be divided into three approaches: Hybrid multi-technique approaches do not apply only one type of compression; in this case, they compress the model sequentially or simultaneously using pruning, quantization, and knowledge distillation to maximize the compression ratios without accelerating the decrease in accuracy [87]. Neural architecture search (NAS) automated frameworks that search for the most effective model topology (e.g. number of layers, head size) to a given target device, effectively designing a smaller model directly, as opposed to just scaling down a larger model [88]. Hardware aware and adaptive optimization that adapt dynamically in real-time model parameters or compression levels based on the current state of the hardware (e.g. battery level, bandwidth available in the memory) [89]. The underlying shift in the research of edge AI emphasized by Table 5 is that, unlike isolated compression operations, multi-stage optimization are used as a foundational approach. Hybrid approaches are already proving better than single methods, where the combination of pruning, quantization and distillation can offer a way to achieve 4x model size and latency reductions, thus becoming deployable on highly constrained IoT devices and NPUs, which would not be unlocked by either technique alone. Out of example, [1], [90] show that stacking methods can achieve 4x model size and latency cuts, which no single method can unlock. Instead of destructively reducing the size of a large model, [88] demonstrate that automated architecture search can grow a smaller, more efficient topology, which, in fact, yields a smaller size reduction, but significantly higher massive multitask language understanding (MMLU) accuracy scores than full pruning. The hybridized systems that mix KD with pruning or quantization instead of generic distillation. These hybrid models can achieve compression ratios of 2-4x and still achieve near floating-point precision as in [87], [91], which effectively stabilizes the behavior of ultra-low-bit models (e.g. 4-bit) which normally experience degradation. Lastly, Hardware-Aware Adaptability is demonstrated to be an important feature by [92] using the EDGE-LLM framework. They also demonstrate that optimization cannot be a static process but has to be dynamic to respond to the real time physical limitations of the hardware by dynamically scheduling computations to support the irregular patterns formed by pruning (sparse tensors).

Table 5: Comparative Analysis Of Emerging Optimization Techniques.

Study	Methodology	Compression / Optimization	Inference Latency	Accuracy Retention	Energy Efficiency	Target Architecture
RAMKUMAR (2024) [1]	Hybrid (Pruning + Quant)	4x Reduction	●	●	●	Mobile/IoT (NPU, FPGA)
LI & QU (2025) [90]	Hybrid (Quant+Prune+Distill)	Lightweight Compression	●	●	●	Cross-Platform Mobile
KONISHI ET AL. (2024) [4]	Hybrid (Case Study)	Llama 2 Optimization	—	○	●	Resource-Limited
MURALIDHARAN ET AL. (2024) [86]	Hybrid (Pruning + KD)	2-4x Compression	—	●	●	Edge (Open Source)
BHARDWAJ ET AL. (2024) [91]	Hybrid (Quantized KD)	4-bit Stabilization	—	●	—	Heterogeneous Edge
YU ET AL. (2024) [92]	Hardware-Aware / Hybrid	4x Memory Reduction	●	—	—	HW Scheduling (Irregular)

Source: Authors, (2026).

IV. OPTIMIZATION TECHNIQUE ANALYSIS

To elucidate the current state of on-device LLM deployment, we conducted a multidimensional analysis of the 68 reviewed studies. This section visualizes evaluates comparative efficiency via a Pareto analysis, and defines the unique "optimization fingerprint" for each technique to guide future implementation choices. In order to assess the actual effectiveness of the strategies, we plotted the performance results on a Pareto Efficiency Frontier, as shown in Figure 5, and graphed inference speedup versus accuracy retention and the bubble size depicts compression ratio. The analysis shows that there are three different clusters: The Baseline Cluster: These are the methods where standard Quantization (Blue) is the dominant form and which provide reliable compression although they seldom achieve more than 4x speedup without hardware assistance. The Sparsity Gap: Pruning methods (Red) can frequently be far behind in speedup relative to their theoretical compressiveness, demonstrating the difficulty of sparse matrices acceleration via standard GPUs. The High-Efficiency Cluster: The largest cluster is the one comprising of Hybrid and Hardware papers in the upper-right quadrant. These methods are able to advance the Pareto frontier, getting huge speedups (>10x) and close-to-lossless accuracy.

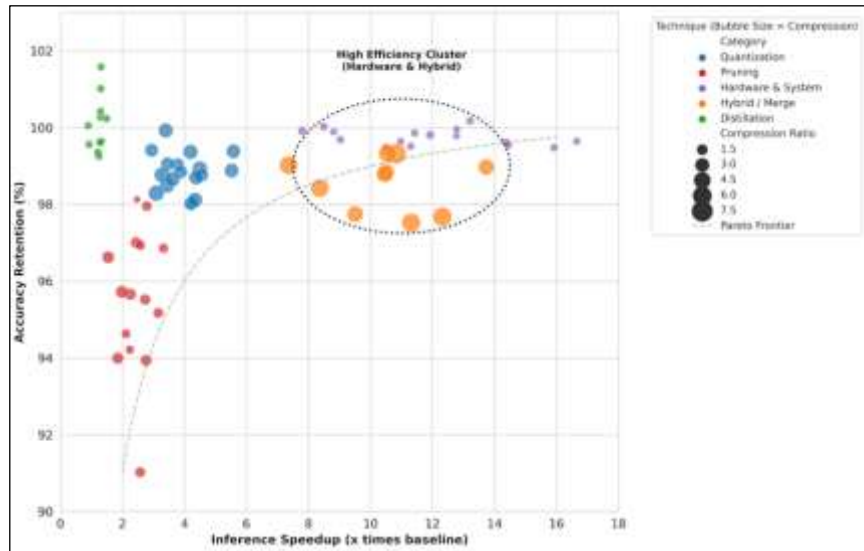


Figure 5: Optimization Technique performance results.
Source: Authors, (2026).

In order to obtain a detailed analysis of trade-offs we created radar charts capturing the fingerprint of each technique in four axes: Compression, Accuracy, Speedup and Sparsity. Figure 6 shows the aggregate fingerprint, and it can be seen that Hybrid/Merge techniques (Orange) have the highest surface in coverage as they combine the compression power of quantization and the speedup advantage of hardware awareness. In order to see these trade-offs in a granular level, Figures 7 to 11 break down these categories into respective sub-strategies:

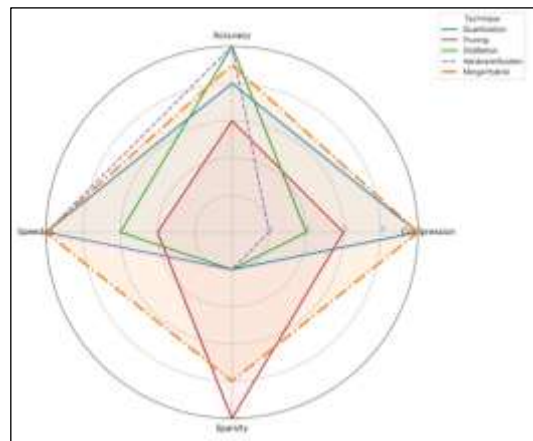


Figure 6: Optimization Technique performance radar chart.
Source: Authors, (2026).

The figure 7 shows that Quantization-Aware Training (QAT) is superior compared to Post-Training Quantization (PTQ). Whereas PTQ is more quickly to execute, QAT (the larger polygon) retains considerably high accuracy at comparable compression level.

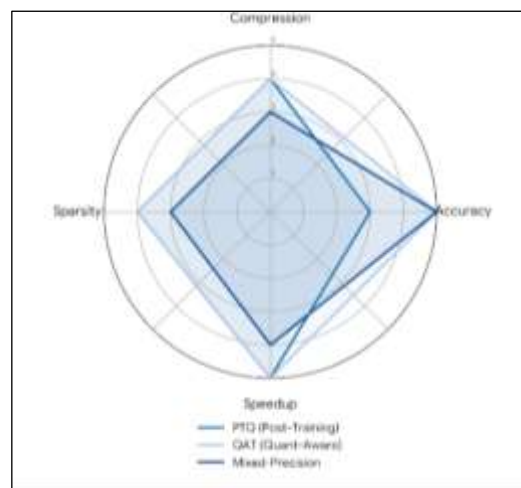


Figure 7: Quantization strategy performance.
Source: Authors, (2026).

Figure 8 demonstrates an important dichotomy. Unstructured Pruning is spiking on "Sparsity" (removing individual weights) but performs poorly on "Speedup" because of hardware incompatibility. Structured Pruning on the other hand is sparsity-ready, but has the tangible speedups needed in deploying edges.

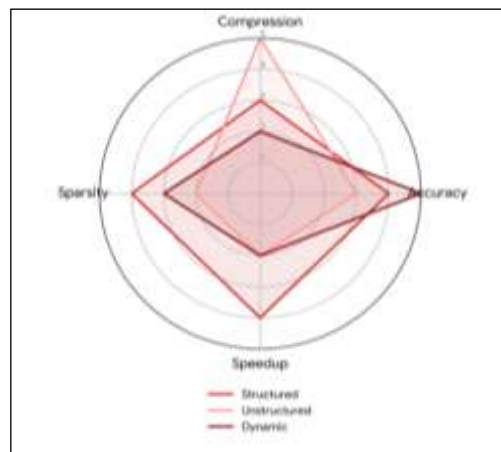


Figure 8: Pruning strategy performance.
Source: Authors, (2026).

It is strongly skewed to the axis of Accuracy, which validates that Distillation is mostly a quality-recovery process, as shown in Figure 9. Nonetheless, the Distilled-Quant variant is much bigger in shape and demonstrates the effectiveness of distillation when it is combined with a compression partner.

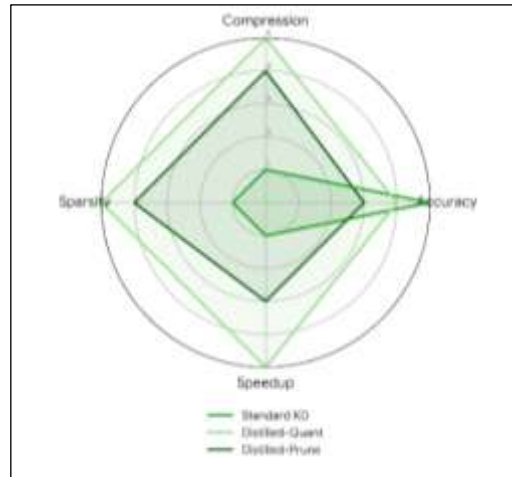


Figure 9: distillation strategy performance.
Source: Authors, (2026).

Hardware & System , as shown in Figure 10 These methods show a distinctive fan-shaped profile which has highest Speedup and Accuracy scores yet has low Compression scores. The FPGA/Accelerator sub-type scores the highest on speedup (5/5), which justifies the use of special silicon (NPUs) as opposed to the general-purpose CPUs.

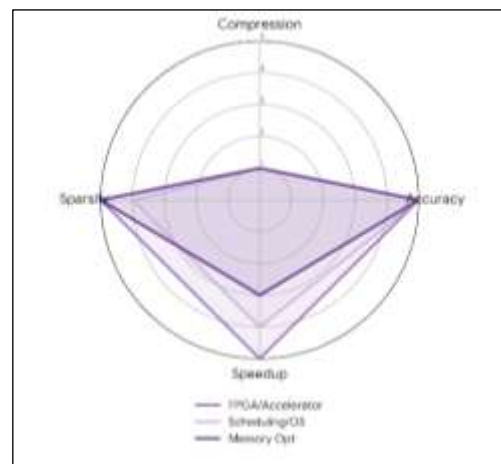


Figure 10: Hardware and software aware performance.
Source: Authors, (2026).

Merge multi optimization technique, as shown in Figure 11, the strain of the Prune + Quant strategy represents the most balanced profile of one of the sub-techniques, as it scores high in all four axes. This implies that the future of on-device LLMs will be in multi-stage optimization pipelines instead of single-stage algorithmic optimization.

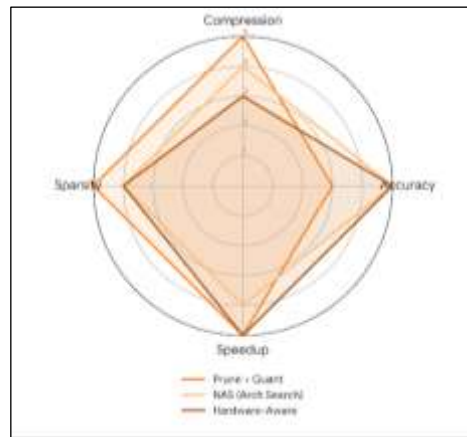


Figure 11: Merge multi optimization technique performance.
Source: Authors, (2026).

V. LLM MODEL ANALYSIS

The distribution of base models that have been observed in the reviewed literature represent unique technical constraints and community standards as shown in figure 12. The LLaMA Architecture Dominates: This is the LLaMA family, which comprises the largest plurality (29.27) of the experiment landscape. This popularity is based on its use as a standardization ground, on which researchers can compare the novel compression methods based on a constant reference. Moreover, the 7-billion parameter variant is an ideal tradeoff between performance and viability, and can be quantized to fit in the memory envelope of development boards of common use such as the NVIDIA Jetson and Raspberry Pi (both of which are often cited). The full-fledged ecosystem of deployment facilities (e.g., llama.cpp) also reduces implementation friction. Limited Use of Native Small Language Models (SLMs), which is Unlike the general patterns in the industry, the literature reviewed in this case has little adoption (~1.2%) of more recent native Small Language Models (SLMs) like Phi, Gemma, or Qwen. A significant research emphasis is still placed on narrowing the huge foundation models (such as Llama) to edge constraints instead of using intrinsically small models. Role of Legacy and Transparent Architectures such as BERT and OPT are still lower, but still relevant (~4.9%). Being weaker in the generative task, fully open training logs of the OPT family are useful in a white-box analysis, which enables the researcher to distinguish between compression artifacts and training anomalies, which is essential to rigorous optimization research. The large "General/Other" section reflects the trend of using generic or custom transformer blocks rather than named models.

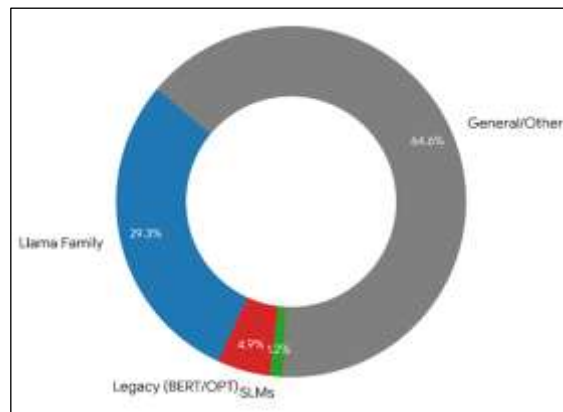


Figure 12: LLM model distribution.
Source: Authors, (2026).

VI. LIMITATIONS ANALYSIS

Although the optimized Large Language Models applied to edge devices have advanced, the literature indicates that the solutions are still facing some limitation. The main issues are bottlenecks in hardware resources, reduced accuracy after compressing and dependence on narrow evaluation metrics. classified these methodological and practical limitations, which have been found to be most common in certain studies. First , Hardware and Resource Limits: Edge devices have little memory, compute and power, and energy, which unquestionably impedes the deployment of LLMs. In these physical limitations, the external interoperability of the results is hindered among other hardware levels [1], [2], [11], [36], [33], [42]. Second, Accuracy Degradation: Model based compression methods such as pruning and quantization often cause errors in accuracy, especially in the high compression rate or low-bit-width regimes [10], [28], [36], [50], [51].

Also, it is computationally costly to fine-tune these compressed models on-device, which restricts the ability to personalize them [69], [93]. Third, Evaluation and Scalability Gaps: In recent studies, there is a tendency to use limited measures of evaluation, such as perplexity instead of direct downstream task evaluation [7], [10]. In addition, most of the methods are problematic when it comes to scaling them to very large models (tens of billions of parameters) [4], [28], [38], [35], [58]. Fourth, Limitations on Deployment and Optimization: It is evident that there are very few studies with real-life deployments of various edge devices [58]. There are also technical gaps in the aspect of activation quantization, which has not yet been properly studied [28], [30], [31], and in the capability of flexibilizing the batch size [52]. Fifth, System Design and Multimodal Support: The heterogeneous platform cannot achieve efficiency gains due to a lack of integrated hardware-software co-design [11], [33], [89].

VII. CONCLUSION

This paper highlights that the democratization of access to LLMs is not simply an issue of reducing model sizes but is one of finding a balance between the complexity of the algorithms and hardware capacity. Our literature review shows that science has mastered the field of weight compression, but has yet to overcome activation bottlenecks and the variety of hardware support. Dominating the Llama architecture, although it is convenient to standardization, may have a stifling effect on the original work of the natively efficient architecture. Finally, edge intelligence will evolve into the next stage of retrofitting cloud models and co-designing neural architectures and silicon in a manner that enables privacy-preserving AI to be made available on all devices, no matter how powerful they are.

VIII. AUTHOR'S CONTRIBUTION

Conceptualization: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Methodology: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Investigation: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Discussion of results: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Writing – Original Draft: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Writing – Review and Editing: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Resources: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Supervision: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

Approval of the final text: Mohammad Tarik Mohammad, Qutaiba Ibrahim.

IX. REFERENCES

- [1] A. Ramkumar, "Enabling On-Device Inference of Large Language Models: Challenges, Techniques, and Applications," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 10, no. 6, pp. 595–604, 2024, doi: 10.32628/cseit241061100.
- [2] J. Xu et al., "On-Device Language Models: A Comprehensive Review," *arXiv preprint arXiv:2409.00088*, 2024, doi: 10.48550/arxiv.2409.00088.
- [3] R. Zhang et al., "Toward Democratized Generative AI in Next-Generation Mobile Edge Networks," *arXiv preprint arXiv:2411.09148*, 2024, doi: 10.48550/arxiv.2411.09148.
- [4] M. Konishi, K. Nakano, and Y. Tomoda, "Efficient Compression of Large Language Models: A Case Study on Llama 2 with 13B Parameters," *Research Square*, 2024, doi: 10.21203/rs.3.rs-4407954/v1.
- [5] M. Navardi et al., "GenAI at the Edge: Comprehensive Survey on Empowering Edge Devices," *Proc. AAAI Symp. Ser.*, vol. 5, no. 1, pp. 180–187, 2025, doi: 10.1609/aaais.v5i1.35586.
- [6] G. Qu et al., "Mobile Edge Intelligence for Large Language Models: A Contemporary Survey," *IEEE Commun. Surveys Tuts.*, vol. 1, no. 1, p. 1, 2025, doi: 10.1109/comst.2025.3527641.
- [7] W. Wang et al., "Model Compression and Efficient Inference for Large Language Models: A Survey," *arXiv preprint arXiv:2402.09748*, 2024, doi: 10.48550/arxiv.2402.09748.
- [8] X. Zhu et al., "A Survey on Model Compression for Large Language Models," *arXiv preprint arXiv:2308.07633*, 2023, doi: 10.48550/arxiv.2308.07633.
- [9] D. Liu, "Contemporary Model Compression on Large Language Models Inference," *arXiv preprint arXiv:2409.01990*, 2024, doi: 10.48550/arxiv.2409.01990.
- [10] B. Khanal and J. M. Capone, "Evaluating the Impact of Compression Techniques on Task-Specific Performance of Large Language Models," *arXiv preprint arXiv:2409.11233*, 2024, doi: 10.48550/arxiv.2409.11233.
- [11] L. Chen et al., "HeteroLLM: Accelerating Large Language Model Inference on Mobile SoCs with Heterogeneous AI Accelerators," *Preprints*, 2025, doi: 10.20944/preprints202501.0901.v1.
- [12] J. Xiao, Q. Huang, X. Chen, and C. Tian, "Large Language Model Performance Benchmarking on Mobile Platforms: A Thorough Evaluation," *arXiv preprint arXiv:2410.03613*, 2024, doi: 10.48550/arxiv.2410.03613.
- [13] Lin, Tianyang et al. "A Survey of Transformers." *AI Open*, vol 3 , p.111-132,2021.
- [14] OpenAI, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
- [15] H. Touvron et al., "Llama 2: Open foundation and chat models," *arXiv preprint arXiv:2307.09288*, 2023.
- [16] N. Shazeer, "GLU variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.
- [17] R. Pope et al., "Efficiently scaling transformer inference," *arXiv preprint arXiv:2211.05102*, 2022.

- [18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.
- [19] I. Lamaakal et al., "Tiny language models for automation and control: Overview, potential applications, and future research directions," *Sensors*, vol. 25, no. 5, p. 1318, 2025.
- [20] J. Zhan, Y. Zhang, S. Wei, D. Xu, and M. Xu, "AssyLLM: Efficient federated fine-tuning of LLMs via assembling pre-trained blocks," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2025.
- [21] D. Xu et al., "EdgeLLM: Fast on-device LLM inference with speculative decoding," *IEEE Trans. Mobile Comput.*, pp. 1–18, 2025, doi: 10.1109/TMC.2024.3513457.
- [22] J. Thomas et al., "Preserving privacy, increasing accessibility, and reducing cost: An on-device artificial intelligence model for medical transcription and note generation," *medRxiv*, 2025, doi: 10.48550/arXiv.2507.03033.
- [23] D. Xu et al., "Fast on-device LLM inference with NPUs," *arXiv preprint arXiv:2407.05858*, 2024.
- [24] J. C. Lang, Z. Guo, and S. Y. Huang, "A Comprehensive Study on Quantization Techniques for Large Language Models," *arXiv preprint arXiv:2411.02530*, 2024, doi: 10.48550/arxiv.2411.02530.
- [25] Y. Jeon, C. Lee, K. Park, and H. Kim, "A Frustratingly Easy Post-Training Quantization Scheme for LLMs," in *Proc. EMNLP*, 2023, pp. 14446–14461, doi: 10.18653/v1/2023.emnlp-main.892.
- [26] X. Shen et al., "EdgeQAT: Entropy and Distribution Guided Quantization-Aware Training for the Acceleration of Lightweight LLMs on the Edge," *arXiv preprint arXiv:2402.10787*, 2024, doi: 10.48550/arxiv.2402.10787.
- [27] W. Huang et al., "SLiM-LLM: Saliency-Driven Mixed-Precision Quantization for Large Language Models," *arXiv preprint arXiv:2405.14917*, 2024, doi: 10.48550/arxiv.2405.14917.
- [28] J. Lin et al., "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration," *GetMobile*, vol. 28, no. 4, pp. 12–17, 2025, doi: 10.1145/3714983.3714987.
- [29] C. Zeng et al., "GQSA: Group Quantization and Sparsity for Accelerating Large Language Model Inference," *arXiv preprint arXiv:2412.17560*, 2024, doi: 10.48550/arxiv.2412.17560.
- [30] F. Tan et al., "MobileQuant: Mobile-friendly Quantization for On-device Language Models," *arXiv preprint arXiv:2408.13933*, 2024, doi: 10.48550/arxiv.2408.13933.
- [31] X. Shen et al., "Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge," in *Proc. AAAI Conf. Artif. Intell.*, vol. 38, no. 17, pp. 18944–18951, 2024, doi: 10.1609/aaai.v38i17.29860.
- [32] B. Zeng et al., "LSAQ: Layer-Specific Adaptive Quantization for Large Language Model Deployment," *arXiv preprint arXiv:2412.18135*, 2024, doi: 10.48550/arxiv.2412.18135.
- [33] X. Zhu et al., "Hardware-Aware Optimization of Large Language Models for Enhanced Throughput on Arm CPUs," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2025, pp. 1–5, doi: 10.1109/aicas64808.2025.11173152.
- [34] Z. Yang et al., "MCUBERT: Memory-Efficient BERT Inference on Commodity Microcontrollers," in *Proc. ACM*, 2024, doi: 10.1145/3676536.3676747.
- [35] Y. Shao et al., "GWQ: Gradient-Aware Weight Quantization for Large Language Models," *arXiv preprint arXiv:2411.00850*, 2024, doi: 10.48550/arxiv.2411.00850.
- [36] E. J. Husom et al., "Sustainable LLM Inference for Edge AI: Evaluating Quantized LLMs for Energy Efficiency, Output Accuracy, and Inference Latency," *ACM Trans. Internet Things*, 2025, doi: 10.1145/3767742.
- [37] S. Madhanegha, V. Vishnuvaradhan, R. Arun, and I. Surenter, "Quantization of a Llama Language Model for improved Efficiency and Inference," *Research Square*, 2025, doi: 10.21203/rs.3.rs-6021454/v1.
- [38] J. H. Lee et al., "LRQ: Optimizing Post-Training Quantization for Large Language Models by Learning Low-Rank Weight-Scaling Matrices," *arXiv preprint arXiv:2407.11534*, 2024, doi: 10.48550/arxiv.2407.11534.
- [39] J. Hasan, "Optimizing Large Language Models through Quantization: A Comparative Analysis of PTQ and QAT Techniques," *arXiv preprint arXiv:2411.06084*, 2024, doi: 10.48550/arxiv.2411.06084.
- [40] Z. Chen, B. Xie, J. Li, and C. Shen, "Channel-Wise Mixed-Precision Quantization for Large Language Models," *arXiv preprint arXiv:2410.13056*, 2024, doi: 10.48550/arxiv.2410.13056.
- [41] K. Behdin et al., "QuantEase: Optimization-based Quantization for Language Models - An Efficient and Intuitive Algorithm," *arXiv preprint arXiv:2309.01885*, 2023, doi: 10.48550/arxiv.2309.01885.
- [42] H. Shen et al., "Efficient LLM Inference on CPUs," *arXiv preprint arXiv:2311.00502*, 2023, doi: 10.48550/arxiv.2311.00502.
- [43] S. Kim et al., "SqueezeLLM: Dense-and-Sparse Quantization," *arXiv preprint arXiv:2306.07629*, 2023, doi: 10.48550/arXiv.2306.07629.
- [44] T. Dettmers et al., "SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression," *arXiv preprint arXiv:2306.03078*, 2023, doi: 10.48550/arXiv.2306.03078.
- [45] H. Hu et al., "FASP: Fast and Accurate Structured Pruning of Large Language Models," *arXiv preprint arXiv:2501.09412*, 2025, doi: 10.48550/arxiv.2501.09412.
- [46] L. Hu, T. Rosing, and H. Zhang, "TrimLLM: Progressive Layer Dropping for Domain-Specific LLMs," *arXiv preprint arXiv:2412.11242*, 2024, doi: 10.48550/arxiv.2412.11242.
- [47] Y. Jiang et al., "Spatio-Temporal Pruning for Compressed Spiking Large Language Models," *arXiv preprint arXiv:2508.20122*, 2025, doi: 10.48550/arxiv.2508.20122.
- [48] U. Kallakuri et al., "MaGrIP: Magnitude and Gradient-Informed Pruning for Task-Agnostic Large Language Models," *ACM Trans. Embed. Comput. Syst.*, 2025, doi: 10.1145/3766068.

- [49] Y. W. Kang et al., "HWPQ: Hessian-free Weight Pruning-Quantization For LLM Compression And Acceleration," arXiv preprint arXiv:2501.16376, 2025, doi: 10.48550/arxiv.2501.16376.
- [50] Z. J. Xu et al., "Lightweight and Post-Training Structured Pruning for On-Device Large Lanaguage Models," arXiv preprint arXiv:2501.15255, 2025, doi: 10.48550/arxiv.2501.15255.
- [51] M. Federici et al., "Efficient LLM Inference using Dynamic Input Pruning and Cache-Aware Masking," arXiv preprint arXiv:2412.01380, 2024, doi: 10.48550/arxiv.2412.01380.
- [52] J. He et al., "CHESS: Optimizing LLM Inference via Channel-Wise Thresholding and Selective Sparsification," arXiv preprint arXiv:2409.01366, 2024, doi: 10.48550/arxiv.2409.01366.
- [53] L. Li and L. Guo, "Dynamic Low-Rank Adaptation Based Pruning Algorithm for Large Language Models," in Proc. IEEE, 2024, pp. 1094–1099, doi: 10.1109/prai62207.2024.10826600.
- [54] H. You et al., "ShiftAddLLM: Accelerating Pretrained LLMs via Post-Training Multiplication-Less Reparameterization," arXiv preprint arXiv:2406.05981, 2024, doi: 10.48550/arxiv.2406.05981.
- [55] J. Liu et al., "Efficient Pruning of Large Language Model with Adaptive Estimation Fusion," arXiv preprint arXiv:2403.10799, 2024, doi: 10.48550/arxiv.2403.10799.
- [56] T. F. A. van der Ouderaa et al., "The LLM Surgeon," arXiv preprint arXiv:2312.17244, 2023, doi: 10.48550/arxiv.2312.17244.
- [57] S. Guo et al., "Compresso: Structured Pruning with Collaborative Prompting Learns Compact Large Language Models," arXiv preprint arXiv:2310.05015, 2023, doi: 10.48550/arxiv.2310.05015.
- [58] S. Li, X. Han, and J. Bai, "NutePrune: Efficient Progressive Pruning with Numerous Teachers for Large Language Models," arXiv preprint arXiv:2402.09773, 2024, doi: 10.48550/arxiv.2402.09773.
- [59] R. I. Tarecha and P. Choirina, "Compressing Large Language Models (LLMs) using Knowledge Distillation for Optimizing Inference Time and Model Size," G-Tech, vol. 9, no. 2, pp. 1056–1062, 2025, doi: 10.70609/gtech.v9i2.6749.
- [60] J. W. Sander et al., "Constrained Edge AI Deployment: Fine-Tuning vs Distillation for LLM Compression," arXiv preprint arXiv:2505.18166, 2025, doi: 10.48550/arxiv.2505.18166.
- [61] R. Saha et al., "Compressing Large Language Models using Low Rank and Low Precision Decomposition," arXiv preprint arXiv:2405.18886, 2024, doi: 10.48550/arxiv.2405.18886.
- [62] M. Navardi et al., "GenAI at the Edge: Comprehensive Survey on Empowering Edge Devices," Qeios, vol. 7, no. 6, 2024, doi: 10.32388/jeu3u0.2.
- [63] K. Cai et al., "Self-Adapting Large Visual-Language Models to Edge Devices across Visual Modalities," arXiv preprint arXiv:2403.04908, 2024, doi: 10.48550/arxiv.2403.04908.
- [64] J. Bang et al., "Crayon: Customized On-Device LLM via Instant Adapter Blending and Edge-Server Hybrid Inference," arXiv preprint arXiv:2406.07007, 2024, doi: 10.48550/arxiv.2406.07007.
- [65] L. Gao et al., "Enabling Resource-Efficient On-Device Fine-Tuning of LLMs Using Only Inference Engines," arXiv preprint arXiv:2409.15520, 2024, doi: 10.48550/arxiv.2409.15520.
- [66] P. Ravindran, "Enhancing Efficiency in Deep Learning: Knowledge Distillation for Compact and High-Performance AI Models," Int. J. Adv. Res. Sci. Commun. Technol., pp. 545–550, 2025, doi: 10.48175/ijarsct-24468.
- [67] Q. Huangpu and H. Gao, "Efficient Model Compression and Knowledge Distillation on LLama 2: Achieving High Performance with Reduced Computational Cost," Preprints, 2024, doi: 10.31219/osf.io/hax36.
- [68] R. Cantini, A. J. Orsino, and D. Talia, "Xai-driven knowledge distillation of large language models for efficient deployment on low-resource devices," J. Big Data, vol. 11, pp. 1–17, 2024, doi: 10.1186/s40537-024-00928-3.
- [69] D. Peng, Z. Fu, and J. Wang, "PocketLLM: Enabling On-Device Fine-Tuning for Personalized LLMs," arXiv preprint arXiv:2407.01031, 2024, doi: 10.48550/arxiv.2407.01031.
- [70] C. Chen et al., "Hybrid Systolic Array Accelerator with Optimized Dataflow for Edge Large Language Model Inference," arXiv preprint arXiv:2507.09010, 2025, doi: 10.48550/arxiv.2507.09010.
- [71] K. Alizadeh-Vahid et al., "LLM in a flash: Efficient Large Language Model Inference with Limited Memory," arXiv preprint arXiv:2312.11514, 2023, doi: 10.48550/arxiv.2312.11514.
- [72] R. Yi et al., "EdgeMoE: Fast On-Device Inference of MoE-based Large Language Models," arXiv preprint arXiv:2308.14352, 2023, doi: 10.48550/arxiv.2308.14352.
- [73] H. Xu, Y. Li, and S. Ji, "LlamaF: An Efficient Llama2 Architecture Accelerator on Embedded FPGAs," arXiv preprint arXiv:2409.11424, 2024, doi: 10.48550/arxiv.2409.11424.
- [74] T. Glint et al., "AxLaM: energy-efficient accelerator design for language models for edge computing," Phil. Trans. R. Soc. A, vol. 383, no. 2288, 2025, doi: 10.1098/rsta.2023.0395.
- [75] Y. Jang et al., "PIMPAL: Accelerating LLM Inference on Edge Devices via In-DRAM Arithmetic Lookup," in Proc. IEEE Design Autom. Conf. (DAC), 2025, pp. 1–7, doi: 10.1109/dac63849.2025.11133391.
- [76] P. Wang et al., "SpeedLLM: An FPGA Co-design of Large Language Model Inference Accelerator," arXiv preprint arXiv:2507.14139, 2025, doi: 10.48550/arxiv.2507.14139.
- [77] R. Metcalfe et al., "Enhancing Local LLM Performance Through Heterogeneous Multi-Device Computing," in Proc. IEEE Int. Conf. Multimedia Inf. Process. Retr. (MIPR), 2024, pp. 55–60, doi: 10.1109/mipr62202.2024.00016.

- [78] G. Zhu et al., "Enabling MoE on the Edge via Importance-Driven Expert Scheduling," arXiv preprint arXiv:2508.18983, 2025, doi: 10.48550/arxiv.2508.18983.
- [79] J. Li et al., "Hummingbird: A Smaller and Faster Large Language Model Accelerator on Embedded FPGA," arXiv preprint arXiv:2507.03308, 2025, doi: 10.48550/arxiv.2507.03308.
- [80] J. Wei et al., "T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge," arXiv preprint arXiv:2407.00088, 2024, doi: 10.48550/arxiv.2407.00088.
- [81] M. Li et al., "CoMoE: Collaborative Optimization of Expert Aggregation and Offloading for MoE-based LLMs at Edge," arXiv preprint arXiv:2508.09208, 2025, doi: 10.48550/arxiv.2508.09208.
- [82] P. Tang et al., "HOBBIT: A Mixed Precision Expert Offloading System for Fast MoE Inference," arXiv preprint arXiv:2411.01433, 2024, doi: 10.48550/arxiv.2411.01433.
- [83] D. Macario, H. Seferoglu, and E. Koyuncu, "Model-Distributed Inference for Large Language Models at the Edge," arXiv preprint arXiv:2505.18164, 2025, doi: 10.48550/arxiv.2505.18164.
- [84] J. Lee and S. Ha, "Empowering Edge Devices with Processing-In-Memory for On-Device Language Inference," IEEE Embed. Syst. Lett., 2025, doi: 10.1109/les.2025.3538827.
- [85] T. Wang et al., "Neuralink: Fast on-Device LLM Inference with Neuron Co-Activation Linking," in Proc. ACM, 2024, pp. 147–162, doi: 10.1145/3676642.3736114.
- [86] X. Li et al., "Integrated Optimization of Large Language Models: Synergizing Data Utilization and Compression Techniques," Preprints, 2024, doi: 10.20944/preprints202409.0662.v2.
- [87] S. Muralidharan et al., "Compact Language Models via Pruning and Knowledge Distillation," arXiv preprint arXiv:2407.14679, 2024, doi: 10.48550/arxiv.2407.14679.
- [88] R. S. Sukthanker et al., "LLM Compression with Neural Architecture Search," arXiv preprint arXiv:2410.06479, 2024, doi: 10.48550/arxiv.2410.06479.
- [89] Y. Wang and J. Zhao, "A Unified and Resource-Aware Framework for Adaptive Inference Acceleration on Edge and Embedded Platforms," Electronics, vol. 14, no. 11, p. 2188, 2025, doi: 10.3390/electronics14112188.
- [90] Y. Li and C. Qu, "Research on Lightweight Compression Algorithms and Deployment Optimization for Large Language Models on Mobile Terminals," in Proc. ACM, 2025, pp. 164–169, doi: 10.1145/3745238.3745267.
- [91] K. Bhardwaj et al., "Oh! We Freeze: Improving Quantized Knowledge Distillation via Signal Propagation Analysis for Large Language Models," arXiv preprint arXiv:2403.18159, 2024, doi: 10.48550/arxiv.2403.18159.
- [92] Z. Yu et al., "EDGE-LLM: Enabling Efficient Large Language Model Adaptation on Edge Devices via Layerwise Unified Compression and Adaptive Layer Tuning and Voting," arXiv preprint arXiv:2406.15758, 2024, doi: 10.48550/arxiv.2406.15758.
- [93] T. P. Chander, "Optimizing Memory Efficiency in Large Language Models: Adaptive Compression Techniques," Int. J. Sci. Technol. Eng., vol. 13, no. 1, pp. 1474–1479, 2025, doi: 10.22214/ijraset.2025.66591.