Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# On-Device Large Language Models: A Survey of Model Compression and System Optimization

**Wanyi Chen**

berryccc0904@gmail.com

Soochow University

**Junhao Wang**

Xi'an Jiaotong University

**Yiwei Zhang**

The University of Sydney

**Yufan Shi**

University of California, San Diego

**Tianyi Jiang**

Beijing Jiaotong University

**Shengxian Zhou**

Shandong University

**Chenxu Wu**

University of Science and Technology of China

**Andi Zhang**

University of Manchester

**Chenyue Zhou**

Nanyang Technological University

**Minxuan Wang**

Renmin University of China

**Xinyu Liu**

Hong Kong University of Science and Technology

**Xiaoshuai Hao**

Beijing Academy of Artificial Intelligence

**Yinan Wu**

Tsinghua University

**Yichen Li**

Huazhong University of Science and Technology

**Yuwei Hu**

Renmin University of China

**Zhao Cao**

Renmin University of China
**Yang Lu**
Xiamen University
**Mengke Li**
Shenzhen University
**Yanbiao Ma**
Renmin University of China
**Zhiwu Lu**
Renmin University of China
**Jungong Han**
Tsinghua University
**Yike Guo**
Hong Kong University of Science and Technology

**Research Article**

**Keywords:** Large language models, model compression, quantization, pruning, knowledge distillation, KV cache, low rank adaptation, hybrid methods, on device and edge inference, ALEM evaluation

**Posted Date:** November 21st, 2025

**DOI:** https://doi.org/10.21203/rs.3.rs-7975734/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

[1]

# On-Device Large Language Models: A Survey of Model Compression and System Optimization

Wanyi Chen[1], Junhao Wang[2], Yiwei Zhang[3], Yufan Shi[4], Tianyi Jiang[5], Shengxian Zhou[6], Chenxu Wu[7], Andi Zhang[8], Chenyue Zhou[9], Minxuan Wang[10], Xinyu Liu[11], Xiaoshuai Hao[12], Yinan Wu[13], Yichen Li[14], Yuwei Hu[15], Zhao Cao[15], Yang Lu[16], Mengke Li[17], Yanbiao Ma[15*], Zhiwu Lu[15*], Jungong Han[13*], Yike Guo[11], Wanyi Chen[1*]

[1]School of Computer Science, Soochow University, China.
[2]Xi'an Jiaotong University, China.
[3]The University of Sydney, Sydney, Australia.
[4]University of California, San Diego, USA.
[5]Beijing Jiaotong University, China.
[6]Shandong University, China.
[7]University of Science and Technology of China, China.
[8]The University of Manchester, Manchester, U.K..
[9]Nanyang Technological University, China.
[10]Renmin University of China, China.
[11]The Hong Kong University of Science and Technology, China.
[12]Beijing Academy of Artificial Intelligence, China.
[13]Tsinghua University, China.
[14]School of Computer Science and Technology, Huazhong University of Science and Technology, China.
[15]Gaoling School of Artificial Intelligence, Renmin University of China, China.
[16]Xiamen University, China.
[17]Shenzhen University, China.

*Corresponding author(s). E-mail(s): ybma1998@ruc.edu.cn; luzhiwu@ruc.edu.cn; jghan@tsinghua.edu.cn; berryccc0904@gmail.com;
Contributing authors: 20255227068@stu.suda.edu.cn; wangjunhao@stu.xjtu.edu.cn; Yaff0377@uni.sydney.edu.au; yus120@ucsd.edu; tianyijiang0219@gmail.com; 202400450123@mail.sdu.edu.cn; wuchenxu@mail.ustc.edu.cn; az381@cantab.ac.uk; CHENYUE001@e.ntu.edu.sg; wangmx2025@ruc.edu.cn; xliugd@connect.ust.hk; haoxiaoshuai714@163.com; yinanwu@ieee.org; ycli0204@hust.edu.cn; huyuweiyisui@ruc.edu.cn; caozhao@ruc.edu.cn; luyang@xmu.edu.cn; mengkeli@szu.edu.cn; yikeguo@ust.hk;

**Abstract**

Large language models are increasingly deployed on device and at the edge, where memory capacity, bandwidth, latency, and privacy requirements dominate system behavior. This survey systematizes the end side stack from algorithms to systems. On the model side, we present a clear taxonomy of quantization, pruning, knowledge distillation, low rank adaptation, and hybrid pipelines, explaining where representative methods belong and how they compose. On the system side, we link these techniques to inference frameworks, compiler and runtime optimizations, kernel fusion, and explicit management of the KV cache. We further propose a unified ALEM protocol, namely Accuracy, Latency, Energy, and Memory, and instantiate it on representative models from 1 to 4 billion parameters to reveal practical trade offs: apply quantization first for memory and time to first token, pair structured pruning with mergeable low rank compensation, and treat the KV cache as a first class subsystem through paging, compression, and eviction. Finally, we outline open problems and directions, including a unified

low bit pipeline that couples transform, calibration, and kernel fusion, joint search over structured pruning and distillation, and train and serve unification that collapses sparse, quantized, and low rank parameters into inference ready weights. The goal is a practical bridge from algorithmic compression to resource aligned and reliable on device and edge deployment.https://github.com/LumosJiang/Awesome-On-Device-LLMs: a repository hosting the complete references for Sections 3–4.

**Keywords:** Large language models, model compression, quantization, pruning, knowledge distillation,KV cache,low rank adaptation, hybrid methods, on device and edge inference, ALEM evaluation

# 1 Introduction

Large language models (LLMs) have advanced rapidly with coupled growth in data scale, model capacity, and compute, exemplified by GPT-3 [1], LLaMA [2], GPT-4 [3], and the recent Qwen-3 [4]. Beyond open-ended generation, they now demonstrate reliable semantic understanding, instruction following, and code synthesis and debugging; recent systems extend these abilities to multimodal inputs and long-context reasoning. In parallel, open model families designed for efficient training and inference on commodity hardware, such as OpenELM [5], Phi-3 [6], and Gemma 2 [7], broaden the balance between capability and efficiency for practical deployment. For on-device use, MobileLLM [8] shows that sub-billion-parameter models can be tailored to real mobile workloads, while DeepSeek-R1 [9] advances step-by-step reasoning via reinforcement learning. A systems perspective further argues that mobile and edge execution is a natural home for many generative AI experiences when latency, bandwidth, and privacy dominate [10].

In practice, LLMs are already used for intelligent personal assistants, enterprise knowledge-base question answering, and creative content generation. Beyond these uses, deployments are moving into settings with tight real-time, reliability, and privacy requirements, such as in-vehicle voice interaction for driver assistance, edge side perception and decision making in industrial IoT, and portable medical devices. This trajectory is reinforced by device-oriented modeling such as MobileLLM and by strong general-purpose baselines such as Qwen-3 that broaden multilingual and tool-use capabilities; in parallel, reinforcement-learning based reasoning as in DeepSeek-R1 raises expectations for on-device assistants that must plan and justify decisions.

Recent systems studies [11], [12], [13] quantify the main obstacles to broad adoption, which can be broadly categorized into two groups: ***(1) prohibitive on-device resource requirements and (2) the inherent limitations of cloud-centric architectures.*** Regarding on-device resources, the memory and computational footprint is the primary barrier. Under standard 32-bit floating-point precision, a 7-billion-parameter model occupies about 28,GB of storage, while a 175-billion-parameter model can require hundreds of gigabytes, far exceeding the capacity of typical end devices. This pressure is exacerbated during inference, as activation tensors and the key–value (KV) cache required for autoregressive decoding consume substantial additional memory and I/O bandwidth, directly increasing latency and energy consumption. The cloud-centric model, while powerful, introduces significant operational and privacy risks. Sending potentially sensitive user content off-device raises fundamental privacy concerns. Furthermore, service stability is often undermined by network jitter in mobile and edge environments, leading to an unreliable user experience. Finally, the recurring costs of network bandwidth and cloud compute challenge the economic sustainability of scaling such services to billions of users. These pressures collectively motivate on-device and edge deployment, running inference locally on smartphones, wearables, IoT gateways, or embedded controllers, which offers tighter privacy boundaries, stable low-latency responses under weak connectivity, and reduced dependence on cloud infrastructure.

To mitigate the gap between LLM scale and end-side budgets, recent work has largely progressed along two intertwined threads: ***model-level optimizations*** and ***system-level optimization***. At the model level, techniques like quantization, pruning, knowledge distillation, and low-rank factorization have moved from isolated use to hybrid compression that strategically combines them. For instance, quantization methods such as FlatQuant [14], GuidedQuant [15], and ResQ [16] enable robust W4A4 deployment by mitigating outliers; KV cache optimization like ThinK [17] reduces long-context overhead; and in parallel, pruning approaches (Probe Pruning [18], RotPruner [19]) support both one-shot trimming and runtime adjustment. Collectively, these techniques redefine the trade-off between accuracy, latency, and memory within tight device constraints.

Complementing model-side advances, systems innovations span compilers, runtimes, and hardware integration. **Efficient attention/decoding kernels** (e.g., FlashAttention [20], FlashDecoding++ [21]) reorganize memory access to ease I/O bottlenecks; **paged KV management** (e.g., vLLM's PagedAttention [22]) increases effective capacity and throughput under fragmented memory, which is crucial for long contexts; and **compiler-level optimization** (PyTorch 2.0 TorchDynamo/Inductor [23], Hidet's task mapping [24]) captures dynamic graphs and emits specialized kernels for hot paths. Building on these pillars, recent work shows a trend toward deeper integration. For instance, integrating FlexAttention with

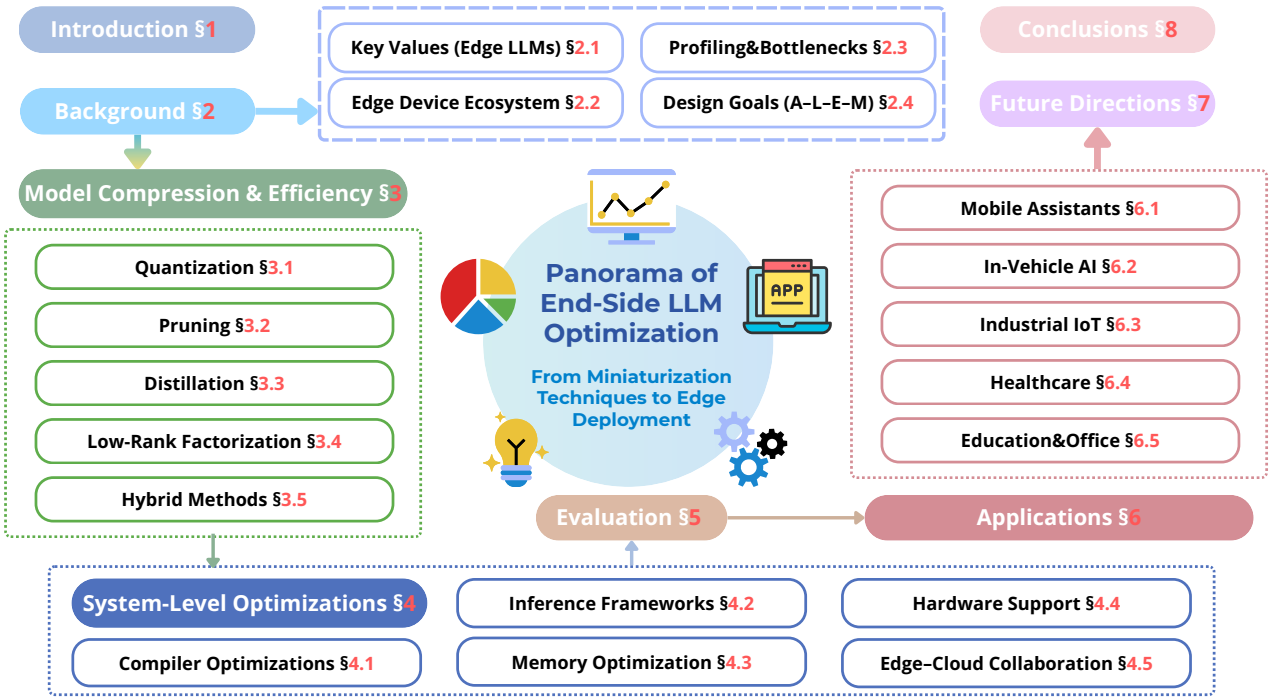


**Fig. 1**: Overview of the survey.

PagedAttention combines shape-specialized compilation and online kernel adaptation to stabilize long-context throughput [25]. In parallel, runtime-adaptive compression [26], unified multi-GPU scheduling for training and inference [27], and ultra-low-bit (1–2 bit) general-purpose kernels [28] further narrow the gap between algorithms, compilers, and hardware.

As illustrated in Fig. 1, the survey follows a pipeline from model miniaturization to edge deployment. Sec. 2 defines end-side devices, profiles hardware bottlenecks, and clarifies the core value of on-device deployment together with A–L–E–M metrics. Sec. 2.4 reviews five technique families—quantization (Sec. 3.1), pruning (Sec. 3.2), knowledge distillation (Sec. 3.3), low-rank factorization (Sec. 3.4), and hybrid methods (Sec. 3.5)—covering motivations, mechanics, and limitations. System-level optimizations are detailed in Sec. 4.1 to Sec. 4.5: compiler optimizations (Sec. 4.1), inference frameworks (Sec. 4.2), memory optimization (Sec. 4.3), hardware support (Sec. 4.4), and edge–cloud collaboration (Sec. 4.5). We then present evaluation methodology and benchmarks in Sec. 5, survey applications across mobile, in-vehicle, industrial IoT, healthcare, and office in Sec. 6, discuss open challenges and future trends in Sec. 7, and conclude in Sec. 8. Our goal is to provide a structured reference for end-side deployment, enabling reusable decisions across the model–system–hardware spectrum.

# 2 Background and Motivation

This section establishes the motivation for on-device LLMs by first outlining their core value propositions. It then provides an analysis of the heterogeneous hardware ecosystem where these models operate, identifies the critical performance bottlenecks that arise from this hardware, and finally, introduces a comprehensive framework for their evaluation.

## 2.1 Core Values of On-Device Deployment

On-device LLMs deliver distinct advantages in privacy, latency, robustness, and cost efficiency:

- **Privacy and compliance**: Local inference enables data minimization, purpose limitation, and controlled retention, facilitating adherence to the General Data Protection Regulation (GDPR) and the U.S. Health Insurance Portability and Accountability Act (HIPAA) [29], [30].
- **Latency and availability**: Eliminating cloud round trips stabilizes sub-second interaction; for voice and real-time applications, the International Telecommunication Union–Telecommunication Standardization Sector (ITU-T) Recommendation G.114 suggests $\leq$ 150 ms one-way latency for acceptable quality [31]. Platform roadmaps emphasize "on-device, low-latency, offline-capable" AI stacks [32].
- **Offline robustness**: Functionality persists under weak or intermittent connectivity (e.g., industrial workshops, remote areas) for summarization, automatic speech recognition (ASR) $\rightarrow$ natural language understanding (NLU), and domain question answering (QA) [32].
- **Cost and energy efficiency**: Shifting inference from cloud to edge reduces long-run infrastructure cost; standardized edge benchmarks (e.g., MLPerf Mobile Inference Benchmark [33]) enable like-for-like comparisons of latency, throughput, and energy.

## 2.2 Edge Device Ecosystem

Edge devices span a heterogeneous family of network edge platforms that perform local ingestion, computation, storage, and communication, including smartphones, wearables, industrial IoT controllers, in vehicle systems, and proximal micro edge nodes (e.g., cloudlets[34], micro data centers [35]). Common attributes include proximity to data sources, hardware heterogeneity, strict energy and thermal envelopes, and real time performance demands. Recent mobile system on chip (SoC) platforms, with unified memory, low and mixed precision accelerators, and high speed on chip interconnects, have made on-device LLM inference practical [32]. In terms of deployment models, systems fall into three patterns: on-device only, device with edge collaboration, and edge with cloud collaboration; the choice determines how privacy boundaries, latency targets, and resource budgets are enforced end to end.

## 2.3 Fundamental Bottlenecks in On-Device Inference

The defining challenge for on-device LLM inference is not a lack of computational power, but rather a confrontation with the fundamental limits of the memory subsystem, a phenomenon widely known as the "memory wall." This bottleneck manifests in two distinct yet interrelated forms: static memory capacity and dynamic memory bandwidth.

First, the sheer size of multi-billion parameter models presents a **static capacity bottleneck**. A 7-billion parameter model, for instance, requires approximately 28 GB of storage in its native precision, a figure that far exceeds the available DRAM on the majority of consumer edge devices. This fundamental mismatch dictates that model compression is not merely an optimization but a prerequisite for feasibility.

However, for models that can fit into memory, a more critical **dynamic bandwidth bottleneck** emerges, which directly governs the token generation rate. The root cause lies in the operational nature of autoregressive decoding, which can be dissected into two phases with starkly different performance characteristics. The initial prefill phase, where the input prompt is processed in parallel, is typically compute-bound and can effectively utilize the device's processing units. In stark contrast, the subsequent token generation phase is sequential and entirely constrained by memory bandwidth. For every single token generated, the model's entire set of parameters—often gigabytes of data—must be read from the relatively slow off-chip DRAM into the on-chip SRAM and compute units. This repetitive, high-volume data movement leaves the powerful computational cores idle for a significant portion of the time, making the speed of memory access, rather than floating-point operations, the primary determinant of end-to-end performance.

Consequently, the most impactful system-level optimizations are precisely those designed to alleviate this intense pressure on the memory subsystem. These strategies can be viewed as direct countermeasures to the identified bottlenecks. I/O-aware kernels like FlashAttention [20] attack the bandwidth issue at the algorithmic level by minimizing data transfers between DRAM and SRAM during the attention calculation. At the same time, techniques such as low-bit quantization [36, 37] and architectural modifications like Grouped-Query Attention (GQA) [38] reduce the total volume of data (for weights and the KV cache, respectively) that must be moved. Finally, advanced compilation frameworks [23, 24] optimize the entire execution graph to reduce overhead and generate hardware-specific code that maximizes memory efficiency.

## 2.4 Design Goals and Metrics: A–L–E–M

Given the complex interplay between model capabilities and hardware constraints, evaluating the efficacy of an on-device LLM system requires a comprehensive approach. The research community has largely converged on a set of core objectives that together define the performance boundaries. For clarity within this survey, we organize these standard criteria along four dimensions, referred to as A-L-E-M. **Accuracy (A)** remains the foundational metric, evaluated using benchmarks such as MMLU [39], BIG-bench Hard (BBH) [40], and MT-Bench [41]. **Latency (L)** is measured by Time-To-First-Token (TTFT) and the steady-state generation rate, expressed in tokens per second. **Energy (E)** consumption is quantified by per-token or per-task energy usage (e.g., Joules/token), often measured under protocols like the MLPerf Mobile Inference Benchmark [33]. Finally, the **Memory (M)** footprint is evaluated based on peak DRAM usage, with emphasis on how the KV cache scales with sequence length. This behavior is significantly influenced by kernels like FlashAttention [20] and memory managers like PagedAttention [22].

Achieving an optimal balance among these A-L-E-M dimensions is the central challenge. To manage performance and cost, hybrid strategies are emerging, such as edge-first execution with cloud fallback or shared speculative decoding between a compact device model and a larger server model [13]. Ultimately, delivering stable low latency and high energy efficiency under strict device constraints necessitates the co-design of model-level techniques (e.g., quantization, pruning) with system-level mechanisms (e.g., compilers, KV cache policies), a theme that will be explored throughout this survey.

# 3 Model-Level Optimizations

Edge-oriented LLM miniaturization centers on five families: quantization, pruning, knowledge distillation,

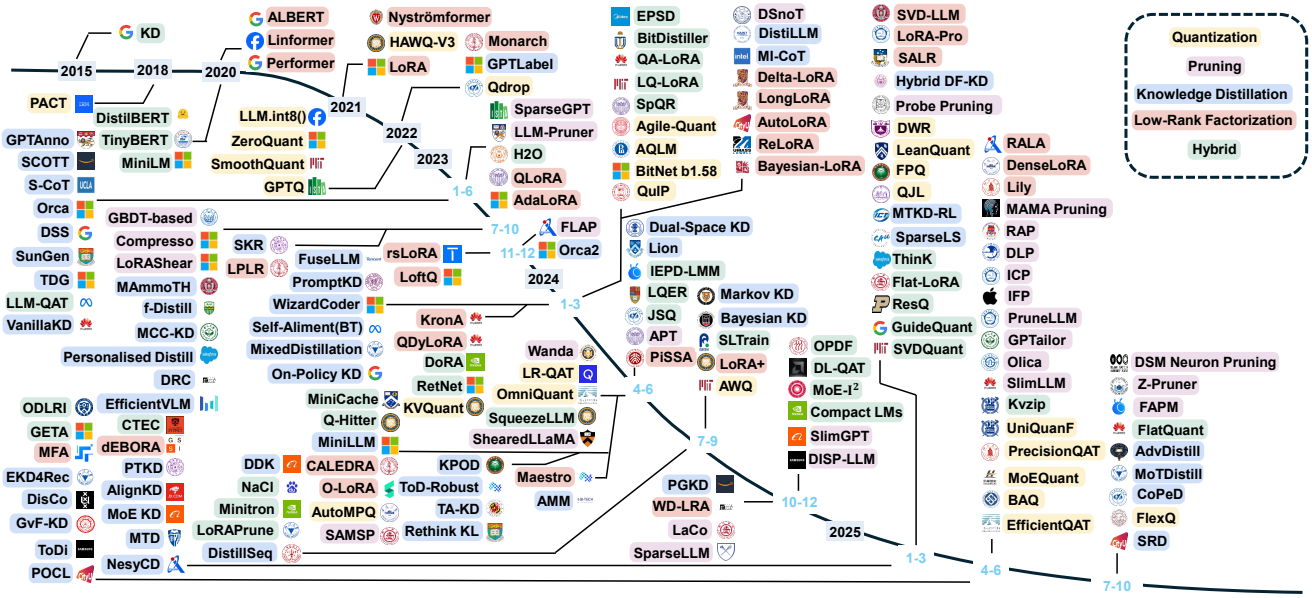


**Fig. 2**: Timeline of representative advancements in LLM compression and efficiency techniques (2015–2025). Methods are categorized into five families: quantization, pruning, knowledge distillation, low-rank factorization, and hybrid approaches, highlighting the evolution and interplay of these strategies over time. The figure organizes developments chronologically and includes logos or icons for major contributors where applicable.

low-rank factorization, and hybrid methods, as chronicled in the timeline of key developments shown in Fig. 2. These act, respectively, on precision, structure, supervision, and factorization. Together they target the dominant edge costs (parameter, activation, and KV memory; memory bandwidth and I/O traffic; and compute) while preserving task accuracy under the A–L–E–M criteria. We proceed in the following order: quantization (Sec. 3.1), pruning (Sec. 3.2), distillation (Sec. 3.3), low-rank (Sec. 3.4), and hybrids (Sec. 3.5).

## 3.1 Quantization

### 3.1.1 Background

Quantization is the process of mapping high-precision numerical formats, such as 32-bit floating-point (FP32), to lower-bit representations, typically 8-bit or 4-bit integers (INT8/INT4). Its primary goals are to reduce the model's memory footprint and to accelerate inference by leveraging hardware support for fast low-precision arithmetic. Earlier studies in computer vision and medium-scale natural language processing showed that models can work effectively at four- to eight-bit precision. Typical examples include the Learned Step-Size Quantization method [42] and the Adaptive Rounding method [43]. These approaches showed that quantization can be done without retraining and inspired further research on applying it to LLMs. However, LLMs introduce additional challenges due to their vast parameter size, long-context attention, and heavy memory demands.

When quantization is applied to LLMs, several new challenges emerge. **(i) Activation outliers.** Sharp activation distributions in attention layers often contain outlier values that are difficult to quantize accurately. These outliers may cause unstable outputs, and methods such as LLM.INT8 [44] and SmoothQuant [36] address this issue by protecting sensitive channels or shifting activation ranges. **(ii) KV cache memory.** During autoregressive decoding, the KV cache stores hidden states for each token and can occupy a large portion of memory. Quantizing the KV cache helps extend context length and lower runtime cost [45], [46], [47]. **(iii) Real-world efficiency.** Even if quantization reduces numerical precision, the speedup depends on optimized kernel implementations, compiler support, and scheduling at inference time [22], [48]. **(iv) Cross-hardware consistency.** Maintaining stable performance across graphics processing units (GPUs), central processing units (CPUs), and edge accelerators remains a challenge [49], [50].

To address these issues, research on quantization for LLMs can be roughly divided into three main categories: **(i) Post-training quantization**, which calibrates weights and activations after model training without updating parameters; **(ii) Quantization-aware training**, which incorporates quantization effects during training so that the model learns to adapt to low-bit arithmetic; and **(iii) Ultra-low-bit and hybrid schemes**, which combine structural or codebook-based techniques to achieve compression below four bits while maintaining stability. The following subsections (Sec.3.1.2–Sec.3.1.4) examine these categories in detail. Table 1 and Figure 3 provide an overview and timeline of representative methods in the field.

### 3.1.2 Post-Training Quantization

Post training quantization applies quantization after pretraining with a small calibration set and no weight updates. It is widely adopted because it avoids full retraining and fits existing inference stacks.

**Table 1**: An overview of representative methods for model quantization.

| Method | Venue | Highlight |
|---|---|---|
| **Post-Training Quantization** | | |
| LLM.int8 [44] | NeurIPS'22 | Detects outlier feature channels and keeps them in high precision (FP16), while quantizing others to INT8, enabling GPT-3-scale inference on a single node. |
| ZeroQuant [51] | NeurIPS'22 | End-to-end PTQ framework with fine-grained hardware-friendly quantization and layer-wise knowledge distillation. |
| GPTQ [49] | ICLR'23 | One-shot weight quantization using approximate second-order information for accurate 3-/4-bit quantization. |
| SmoothQuant [36] | ICML'23 | Migrates quantization difficulty from activations to weights by smoothing activation outliers, enabling W8A8 quantization. |
| AWQ [37] | MLSys'24 | Activation-aware weight quantization; protects ∼1% salient weights to maintain accuracy for 4-bit models. |
| OmniQuant [50] | ICLR'24 | Unified calibration with weight clipping, learned step-size, and cross-architecture consistency. |
| FPTQuant [52] | arXiv'25 | Introduces function-preserving transforms (pre-RoPE, value, MLP scaling, dynamic scaling) to reshape activation distributions without changing model behavior, enabling accurate static INT4 quantization. |
| FlexQ [53] | arXiv'25 | Post-training INT6 quantization with layer-wise sensitivity adaptation and custom GPU kernels (W6A6/W6A8), balancing accuracy and efficiency. |
| KVQuant [54] | arXiv'25 | Focuses on KV cache quantization for ultra-long context LLM inference (up to 10M tokens); proposes hierarchical quantization and mixed-precision allocation for memory-efficient deployment. |
| QJL [55] | AAAI'25 | 1-bit Quantized Johnson–Lindenstrauss Transform for KV cache quantization, eliminating overhead while achieving long-context compression comparable to high-bit methods. |
| **Quantization-Aware Training** | | |
| PACT [56] | ICLR'18 | Introduces a learnable activation clipping parameter to minimize quantization error. |
| HAWQ-V3 [57] | arXiv'21 | Hessian-guided bit allocation with dyadic grid for ultra-low-bit training stability. |
| LR-QAT [58] | arXiv'24 | Low-rank adapters with activation checkpointing for feasible QAT on 7B-scale models. |
| AutoMPQ [59] | TETCI'24 | Automatic mixed-precision quantization framework using few-shot sampling and adapter-based bitwidth search; integrates seamlessly with QAT for efficient precision allocation. |
| BitDistiller [60] | ACL'24 | QAT with self-distillation for sub-4-bit LLMs; combines asymmetric quantization, clipping, and a confidence-aware KL objective to achieve strong 3- and 2-bit accuracy with modest data/compute. |
| EfficientQAT [61] | arXiv'25 | Proposes an efficient QAT pipeline for LLMs with block-wise training followed by end-to-end quantization parameter optimization, significantly reducing training cost while maintaining accuracy. |
| PrecisionQAT [62] | arXiv'25 | Employs learnable adaptive modules to dynamically adjust quantization step sizes and introduces non-uniform quantization schemes to better fit diverse activation distributions. |
| FPQ [63] | arXiv'25 | Stabilizes quantization-aware training by introducing feature perturbation and implicit Hessian regularization, improving convergence and robustness of QAT. |
| **Ultra-low bits** | | |
| SqueezeLLM [64] | ICML'24 | Sensitivity-driven dense-and-sparse quantization achieving robust 3-bit models. |
| AQLM [65] | arXiv'24 | Additive quantization with learnable codebooks for 2–4-bit extreme compression. |
| QuIP [66] | ICML'24 | Uses incoherent rotations and E8 lattice codebooks for better quantization robustness. |
| BitNet b1.58 [67] | arXiv'24 | Explores ternary parameterization $\{-1, 0, 1\}$ for ∼1.58-bit LLMs. |
| LeanQuant [68] | ICLR'25 | Loss-error-aware quantization grid preserves inverse-Hessian outliers, enabling accurate 2–4-bit quantization for large models on limited GPUs. |
| DWR [69] | AAAI'25 | Recovers information from discarded weights during quantization to boost low-bit accuracy without extra cost. |
| UniQuanF [70] | ACL'25 | Unifies uniform and binary-coding quantization (UQ + BCQ) for accurate LLM compression, yielding state-of-the-art results on GSM8K and reasoning benchmarks. |

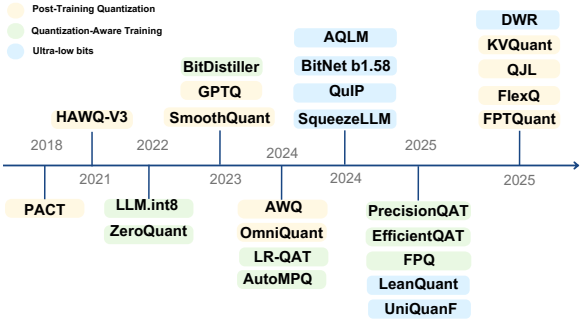


**Fig. 3**: Timeline of quantization methods (2018–2025).

Outlier aware eight bit inference keeps a tiny fraction of channels in higher precision while quantizing the rest, as in LLM.INT8 [44]. Cross layer scaling that moves activation variation onto weights enables stable eight bit weights and eight bit activations in SmoothQuant [36]. End to end PTQ pipelines combine fine grained quantization with local distillation for GPT style models in ZeroQuant [51]. Error compensated column wise approximation supports three to four bit weight only quantization in GPTQ [49]. Protecting roughly one percent salient channels improves four bit robustness in AWQ [37]. Channel reorder and cluster strategies target three bit activations in RPTQ [71]. Architecture agnostic calibration aims for consistent performance across LLaMA, OPT, and Falcon in OmniQuant [50]. Function-preserving transforms (FPTQuant [52]) generalize activation scaling into mathematically equivalent mappings, enabling accurate 4-bit quantization without retraining overhead. From a hardware co-design perspective, FlexQ [53] optimizes INT6 quantization with layer-wise sensitivity adaptation and fused GPU kernels, achieving balanced accuracy and throughput. Furthermore, extending PTQ beyond model weights, KVQuant [54] targets the KV cache and achieves up to ten million context length inference through hierarchical cache quantization. Complementary to this, QJL [55] proposes a 1-bit quantized Johnson–Lindenstrauss (JL) transform for KV cache quantization, which eliminates overhead and maintains near-lossless accuracy, offering an ultra-efficient compression path for long-context inference.

Complementary components include loss aware calibration to match task loss [72], bit split with stitching for tighter reconstruction [73], randomized dropping to

stabilize ultra low bit settings [74], and outlier suppression through equivalent shifting or scaling [75]. Additive codebook based quantization explores the extreme two to three bit regime with competitive accuracy [65]. In practice, realized gains depend on operator support and memory bandwidth, so PTQ results should be reported together with latency, memory, and kernel details [48].

### 3.1.3 Quantization-Aware Training

Quantization-aware training (QAT) incorporates quantization effects into the training loop so that the model adapts to low-bit arithmetic. Although more expensive computationally than PTQ, QAT provides the most consistent accuracy at extreme bit rates (e.g., 2–3 bit for weights or activations) on large models.

A first research direction makes QAT feasible under realistic resource limits. LR-QAT [58] introduces low-rank auxiliaries with activation checkpointing, enabling QAT on multi-billion-parameter models with commodity GPUs. EfficientQAT [61] organizes block-wise updates with step-size optimization, achieving joint 2-bit W+A quantization on large LLMs with modest zero-shot degradation. Complementary studies analyze mixed-precision setups (e.g., W4A8) and propose techniques to prevent underflow and overflow during training [76]. Complementary to this, PrecisionQAT [62] employs learnable adaptive modules for step-size control and non-uniform quantization to better match activation distributions. Pushing to sub-4-bit regimes, BitDistiller [60] couples QAT with self-distillation using asymmetric quantization, clipping, and a confidence-aware KL loss, delivering strong 3-bit and 2-bit results with modest data and compute.

A second direction develops theoretical and algorithmic stability foundations. HAWQ-V3 [57] exploits curvature information to guide bit allocation, achieving sensitivity-aware precision assignment at low bit widths. On the activation side, PACT [56] introduces a learnable clipping function to constrain activation distributions and reduce quantization noise. Recent work such as FPQ [63] introduces feature perturbation and implicit Hessian regularization to stabilize QAT convergence under extreme bit constraints. Earlier methods such as LSQ and I-BERT remain standard initialization or auxiliary constraints in modern QAT workflows [42], [77].

From a systems viewpoint, effective QAT aligns numerical objectives with runtime constraints. Contemporary pipelines couple data-free or lightly supervised training with architectural heuristics (e.g., per-channel scaling, range regularization) and memory-aware objectives such as KV cache quantization. Automated mixed-precision frameworks like AutoMPQ [59] further integrate few-shot bitwidth search into QAT, reducing manual tuning and enabling adaptive precision allocation across layers. Evaluation studies emphasize that accuracy, latency, memory footprint, and kernel support must be considered jointly, since training gains matter only if they yield real inference speedups [48]. Overall, QAT complements PTQ by enabling aggressive low-bit quantization with controlled accuracy loss, laying the foundation for hybrid and mixed-precision schemes discussed next.

### 3.1.4 Ultra-Low Bits and Hybrid Schemes

Below 4 bit precision, neither PTQ nor QAT alone is sufficient for stable performance. To overcome this, research converges on two complementary strategies.

**The first stratefy** emphasizes outlier handling and hybrid parameterization. SqueezeLLM [64] applies sensitivity-driven nonuniform allocation with dense–sparse hybrids to protect abnormal weights at 3 bit, while SpQR [78] extends the GPTQ [49] family by retaining outlier weights as high-precision sparse entries, combining error compensation with sparsity for near-lossless accuracy.

**The second strategy** focuses on structural and algorithmic design. AQLM [65] and QuIP [66] use additive codebooks, Hadamard rotations, and lattice-based transforms to stabilize 2–3 bit regimes across GPUs and CPUs, while BAQ [79] formulates bit allocation as an optimization problem to balance robustness and efficiency. LeanQuant [68] adopts a loss-error-aware quantization grid that preserves salient weight information and scales effectively to large LLMs. Building upon hybrid coding, UniQuanF [70] unifies uniform and binary-coding quantization (UQ + BCQ) to achieve accurate compression under 2–4 bit settings, outperforming prior methods on reasoning benchmarks such as GSM8K. For constrained devices, Agile-Quant [80] combines activation-aware quantization with token pruning to achieve up to 2.55× speedups. At the extreme, BitNet b1.58 [67] explores ∼1.58 bit training near the binary limit. DWR [69] further recycles discarded weights to recover quantization information and improves 4-bit robustness without additional memory cost. Classical refinements such as Bit-Split and Stitching [73] reappear as lightweight structural tools for aggressive compression.

In summary, ultra-low-bit quantization advances mainly along two routes: (i) outlier-aware sparse hybrids for robustness, and (ii) codebook-based structural designs for hardware efficiency. Together, they are making sub-4 bit quantization increasingly actionable for both cloud and edge deployment.

### 3.1.5 Open Problems and Outlook

Despite rapid progress, several open problems persist, directly reflecting the core challenges of LLM quantization. The joint quantization of weights and activations below 4-bits remains fragile, especially for complex reasoning tasks. The high computational cost of QAT continues to be a barrier for models at the 10B+ scale, even with recent efficiency improvements. Furthermore, robustly compressing the KV cache for extremely long contexts lacks standardized evaluation

protocols, making direct comparisons between methods like KVQuant [54] and WKVQuant [46] difficult. Finally, ensuring consistent performance when transferring a quantization recipe across different model families and hardware backends remains a major goal.

Looking ahead, the field is progressing along several key trajectories to address these issues. At the theoretical level, a primary goal is to establish a unified view that explains why structural techniques like Hadamard rotations [66] and additive codebooks [65] so effectively stabilize extreme quantization. From a systems integration perspective, progress hinges on expanding native low-bit kernel support across mainstream serving ecosystems. This includes integration into paged-attention frameworks like vLLM [22] and compiler backends such as TensorRT and ONNX Runtime, which is critical for aligning theoretical gains with practical inference speedups [48]. Furthermore, the frontier is moving toward cross-technology co-design, where quantization is combined with sparsification, pruning, and distillation under hardware-aware scheduling. This holistic approach, exemplified by frameworks like SqueezeLLM [64], SpQR [78], aims to yield predictable trade-offs across the A-L-E-M dimensions. The rise of Mixture-of-Experts (MoE) models also presents an emerging challenge, requiring specialized calibration strategies to handle sparse expert routing, a direction explored by MoEQuant [81]. As these advances mature, quantization is poised to remain a foundational pillar of efficient LLM inference.

## 3.2 Pruning

### 3.2.1 Background

Pruning is a fundamental compression strategy that eliminates redundant parameters to improve inference efficiency. For LLMs, whose immense scale makes retraining prohibitively expensive, the conventional "prune-and-fine-tune" paradigm has become impractical. This has catalyzed a shift in pruning research, moving away from iterative retraining toward highly efficient, often training-free, one-shot methods. The new generation of pruning techniques is designed to be computationally lean, architecturally aware of heterogeneous layer sensitivities, and structured for real-world hardware acceleration. Contemporary research thus frames pruning not merely as parameter removal, but as a sophisticated optimization problem.

Methodologically, pruning techniques are distinguished by the granularity of the components they eliminate. **Unstructured pruning** removes individual weights, an approach that offers the highest compression potential but results in irregular sparsity patterns requiring specialized hardware or software for acceleration. In contrast, **structured pruning** removes entire architectural units (such as neurons, attention heads, or even layers), yielding smaller, dense models that can be efficiently executed on standard hardware. A prominent trend in contemporary research is to integrate both paradigms within adaptive frameworks, combining the interpretability and deployment efficiency of structured sparsity with the flexibility of its unstructured counterpart. These developments define an emerging roadmap for LLM pruning toward methods that are not only accurate and compact but also hardware-efficient and easily deployable. The following subsections (Sec.3.2.2 & Sec.3.2.3) examine these categories in detail, with representative methods summarized in Table 2 and Figure 4.

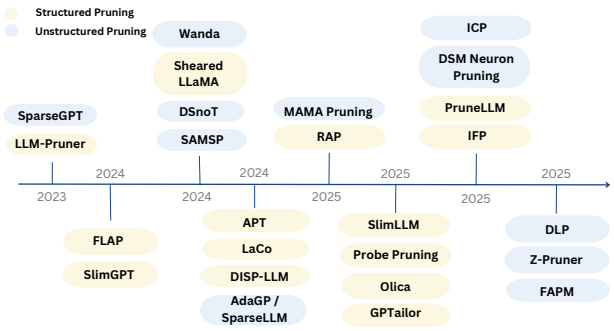


**Fig. 4**: Timeline and Genealogy Chart of Pruning Methods (2023–2025).

### 3.2.2 Structured Pruning

Structured pruning removes architectural units such as neurons or channels, attention heads, and even entire layers, which exposes dense compute on existing hardware. Early work centers on identifying important sub units and adding lightweight recovery. LLM-Pruner [82] establishes a baseline with task agnostic graph analysis that prunes coupled structures and uses LoRA for fast recovery, maintaining most of the original model capacity after moderate pruning. To avoid additional adaptation, FLAP [83] proposes a fluctuation based criterion derived from activation variability and applies bias compensation, enabling retraining-free pruning at high sparsity levels with significant efficiency improvments. SlimLLM [89] integrates Pearson similarity for attention heads and principal component analysis, abbreviated as PCA, for feed forward network channels into a holistic importance framework, assigns non uniform pruning ratios across layers, and uses a linear regression recovery step to restore accuracy. Extending this line, Olica [90] and SlimGPT [84] further reduce reliance on retraining using PCA and optimal brain surgeon arguments, achieving more stable one shot pruning behavior compared to LLM-Pruner. Recent advances improve both the criteria for selection and the practical realization of masks: SPAP [107] provides a penalty-based alternating optimization that decides which units to preserve, managing the trade-off between sparsity and model fidelity; PFN [108] derives masks from ICA-identified modules to preserve global function; in a different approach, MaskPrune

**Table 2**: An overview of representative methods for model pruning.

| Method | Venue / Source | Highlight |
|---|---|---|
| **Structured Pruning** | | |
| LLM-Pruner [82] | NeurIPS'23 | Task-agnostic graph analysis; prune coupled structures; lightweight LoRA recovery; ∼95% perf. kept after 20% pruning on LLaMA-7B. |
| FLAP [83] | AAAI'24 | Fluctuation-based importance from activation variability; bias compensation; up to 50% pruning with ≤66% speedup on LLaMA-7B. |
| SlimGPT [84] | NeurIPS'24 | OBS-based structured pruning with minimal retraining. |
| ShearedLLaMA [85] | ICLR'24 | Prune-to-target architecture; 1.3B student from LLaMA2-7B surpasses TinyLlama-1.1B. |
| APT [86] | ICML'24 | Adaptive pruning + adaptive tuning; removes task-irrelevant structures; retains ≥98% perf. with 60% parameter reduction (RoBERTa/T5). |
| LaCo [87] | EMNLP'24 | Layer collapse: merge consecutive layers into a single "front" layer; 25–30% pruning with ∼80% perf. kept. |
| DISP-LLM [88] | NeurIPS'24 | Dimension-independent importance score for heterogeneous units under a unified scale. |
| SlimLLM [89] | ICML'25 | Holistic importance (Pearson similarity & PCA over heads/channels); linear-regression recovery. |
| Olica [90] | ICML'25 | Retraining-free one-shot structured pruning via PCA analysis; strong one-shot performance. |
| GPTailor [91] | arXiv'25 | Cut&stitch layers from a pool of base/fine-tuned models; 25% pruning on Llama2-13B keeps 97.3% baseline, outperforming LaCo. |
| PruneLLM [92] | ICML'25 | Information-transfer metrics allocate non-uniform budgets across blocks and within blocks (attn/MLP); better accuracy–sparsity trade-offs than uniform ratios. |
| IFP [93] | ICML'25 | Prompt-/task-conditioned structured masks (FFN rows/cols); reduces TTFB and speeds up end-to-end decoding. |
| Probe Pruning [94] | ICLR'25 | Runtime: light probe obtains batch-specific states, then derives per-batch structured masks (heads/channels) for decoding. |
| RAP [95] | arXiv'25 | Runtime RL agent adjusts structured pruning (heads/FFN/layers) under parameter+KV cache budgets. |
| **Unstructured Pruning** | | |
| SparseGPT [96] | ICML'23 | OBS-style second-order compensation; maintains quality at 50–60% sparsity even for OPT-175B. |
| Wanda [97] | ICLR'24 | Magnitude × input-norm criterion; similar quality to SparseGPT but orders-of-magnitude faster (e.g., 5.6 s vs. ∼1350 s on LLaMA-65B). |
| DSnoT [98] | ICLR'24 | Training-free prune–grow updates at fixed sparsity; large PPL drop from Wanda-pruned LLaMA-7B; much faster than LoRA tuning. |
| SAMSP [99] | ICASSP'24 | One-shot, sensitivity-aware mixed sparsity (OBS/OBD fusion); improves perplexity & zero-shot accuracy over SparseGPT. |
| AdaGP / SparseLLM [100] | NeurIPS'24 | Global pruning as coordinated subproblems with auxiliary variables; large gains at high sparsity. |
| DLP [101] | ICML'25 | median-based relative importance distribution for non-uniform layer-wise sparsity. |
| Z-Pruner [102] | IEEE AICCSA'25 | Z-score statistics to amplify inter-weight differences; strong zero-shot accuracy at 50% sparsity. |
| MAMA Pruning [103] | arXiv'25 | Movement+magnitude signals with intra-layer redistribution. |
| FAPM [104] | EMNLP'25 | Forgetting-aware pruning along task vectors; 99.67% downstream accuracy with only 0.25% forgetting. |
| DSM Neuron Pruning [105] | COLM'25 | Integrated gradients to remove detrimental neurons; +4.78% MMLU on LLaMA2-7B. |
| ICP [106] | CVPR'25 | Sliding-window, block-wise immediate compensation + sparsity rearrangement. |

[109] learns binary masks that enforce per-layer uniformity in head counts and FFN width, simplifying dense inference and post-pruning training while maintaining competitive performance.

Beyond improving pruning signals and recovery mechanisms, a second thread reinterprets pruning as a means of architectural reshaping. LoRAShear [110] transfers information from pruned structures into surviving ones during progressive training, using pretraining data and instruction data to recover performance dynamically. ShearedLLaMA [85] reframes pruning as target architecture shaping, yielding a compact version of the original model through structured reorganization, facilitating efficient deployment without extensive retraining. APT [86] combines adaptive pruning with adaptive tuning to remove task irrelevant structures and expand essential modules, enabling effective compression across diverse architectures such as RoBERTa and T5. LaCo [87] collapses consecutive layers into a single front layer, reducing depth while preserving topology and retaining architectures with minimal model performance degradation. Compresso [111] uses collaborative prompting, allowing the model itself to participate in pruning decisions. This co-learning framework therefore enhances the efficiency without compromising comprehension ability. GPTailor [91] cuts and stitches layers from a pool of base and fine tuned models to assemble a compact high performing composite. It represents a direction of cross-model structural fusion, offering an approach to architecture-level pruning for the efficient model assembly.

At the decision making level, methods also study how to formulate the pruning policy itself. DISP LLM [88] introduces a dimension independent importance score that places heterogeneous components on a unified scale for fair comparison. Accuracy predictors then automate sparsity assignment. For instance, a predictor based on gradient boosted decision trees [112] searches pruning configurations and demonstrates consistent improvements in language modelling and reasoning benchmarks compared to uniform baselines. Moving from learned predictors to information-driven allocation, PruneLLM [92] allocates sparsity through information transfer, with non-uniform budgets allocated across blocks, followed by within-block splits to attention and MLP, therefore achieving a lower perplexity than previous fluctuation-based approaches,

such as FLAP. Furthermore, EvoP [113] builds a diverse calibration set via clustering-based sampling, then runs an evolutionary search over the large layer-wise pruning pattern space under a sparsity budget to identify robust pruning configurations. Together, these advances treat pruning as an adaptive decision problem and push structured pruning from pure deletion toward constructive model design.

In contrast to static policies which are fixed at deployment time, the final line of work proposes strategies that adapt the retained structure at inference time to each request and to runtime budgets. IFP [93] introduces prompt-conditioned mask over FFN rows and columns that dynamically adapt computation to each input, considerably reducing latency while maintaining efficiency comparable to larger dense models. Unlike IFP, Probe Pruning [94] runs a tiny probe to get batch-specific hidden states and derives a per-batch structured mask for decoding. It achieves notably better language modelling quality than static pruning baseline under equivalent sparsity level. In addition, RAP [95] treats pruning as runtime control: it employs a lightweight RL agent that adapts block-level pruning during decoding from runtime signals (seq length, batch size, and parameter/KV-memory) to stay within budget.

### 3.2.3 Unstructured Pruning

Unstructured pruning removes individual weights rather than whole units, which enables higher sparsity and fine-grained compression but leads to irregular patterns. Early work imports second-order ideas into LLMs to preserve accuracy at large sparsity levels. SparseGPT [96] adapts optimal brain surgeon style updates to identify prunable weights and compensate the remaining ones, maintaining strong performance at high sparsity level on large-scale models. Because SparseGPT is still computationally heavy, Wanda [97] proposes a simple criterion that combines weight magnitude with the norm of the corresponding input activations; it preserves similar accuracy while being orders of magnitude faster. To mitigate pruning error accumulation at medium–high sparsity level, ICP [106] introduces a novel sliding-window block-wise compensation step together with simple sparsity rearrangement. It improves accuracy without increasing memory usage, outperforming prior methods such as SparseGPT and Wanda. AdaGP and SparseLLM [100] formulate global pruning as coordinated subproblems with auxiliary variables, achieving greater robustness and lower perplexity at extreme sparsity levels. At the layer allocation level, DLP [101] applies median-based relative importance distribution to assign non-uniform sparsity across layers, yielding more balanced pruning and improved model quality compared with a more simplistic uniform allocation baseline.

Moving toward finer grained selection, Z Pruner [102] amplifies inter-weight differences with Z score and sustains strong zero-shot accuracy at 50% sparsity. To overcome static masks, DSnoT [98] performs training-free prune and grow updates at a fixed sparsity, boosting sparse LLM quality under high sparsity levels with efficiency far surpassing LoRA-based fine-tuning. While Z Pruner strengthens static discrimination and DSnoT introduces dynamic masks, both still rely on relatively simple importance signals. Building on SparseGPT, SAMSP [99] fuses optimal brain surgeon and optimal brain damage signals and allocates mixed sparsity according to Hessian sensitivity, improving both perplexity and zero-shot accuracy over SparseGPT. Furthermore, MAMA [103] integrates movement and magnitude signals to identify prunable weights, then redistributes low-importance weight values within each layer to reinforce critical connections. Combined with distillation, it yields consistent yet modest improvements over a simple magnitude-only pruning baseline.

Recent works also broaden pruning objectives beyond compression. FAPM [104] prunes along task vectors to mitigate catastrophic forgetting during continual tuning, preserving strong downstream accuracy while minimizing forgetting. DSM Neuron Pruning [105] applies integrated gradients to identify and remove detrimental neurons, yielding measurable improvements in reasoning and generalization benchmarks such as MMLU. Together, these results illustrate a significant and ongoing a shift from pruning only for size reduction toward pruning as a tool to reduce forgetting and enhance generalization.

### 3.2.4 Open Problems and Outlook

Despite rapid progress from iterative prune-retrain to efficient one-shot pruning, key challenges remain. First, fully automated, non-uniform sparsity schedules are central: accuracy predictors such as the GBDT-based predictor [112] and "one-for-all" pruning models like UniCuCo [114] promise on-demand compression with minimal search cost. Second, pruning objectives should extend beyond size and latency to capability shaping: neuron-level pruning for robustness and generalization as in DSM Neuron Pruning [105], and forgetting-aware pruning for continual learning as in FAPM [104]; assembly approaches such as GPTailor [91] re-interpret pruning as constructive model design. Third, current evaluations over-rely on perplexity and a handful of downstream tasks [115]; future research should adopt more comprehensive benchmark suites, incorporating reasoning, creativity, and safety dimensions to assess LLM compression impacts [115], [116]. Finally, the integration of pruning with quantization and distillation represents a pragmatic and increasingly vital approach towards achieving enhanced compression ratios and improved hardware efficiency at the edge scale.

## 3.3 Knowledge Distillation

### 3.3.1 Background

As LLMs scale to hundreds of billions of parameters, their inference and storage costs create major challenges for real-world deployment. Among compression techniques, Knowledge Distillation (KD) [117] effectively transfers knowledge from a large teacher to a compact student, enabling efficient use in latency- and resource-constrained environments such as mobile devices [118] and industrial edge systems [119]. Unlike pruning or low-rank factorization, KD preserves representational and behavioral knowledge, achieving strong compression with minimal accuracy loss. **Early KD** mainly matched a teacher's final softmax outputs [120], [121], focusing on outcome imitation but overlooking the intermediate reasoning processes crucial to multi-step inference in LLMs.

To overcome these limitations, modern KD evolves toward richer and adaptive supervision paradigms. **(i) Process-based KD** (Sec. 3.3.2) transfers chain-of-reasoning traces to teach procedural understanding; **(ii) Uncertainty-aware KD** (Sec. 3.3.3) calibrates confidence to handle signal reliability; **(iii) Multi-teacher KD** (Sec. 3.3.4) fuses complementary expertise for robust cross-domain generalization; **(iv) Dynamic and Adaptive Strategies** (Sec. 3.3.5) dynamically adjust supervision strength, temperature, or teacher–student alignment according to training progress and task difficulty; and **(v) Task-specific KD** (Sec. 3.3.6) tailors guidance for dialogue, coding, and domain adaptation.Collectively, these developments mark a shift from static prediction alignment to dynamic knowledge transfer, laying the foundation for scalable and interpretable model compression in next-generation LLMs. Representative methods are summarized in Table 3 and Figure 5.

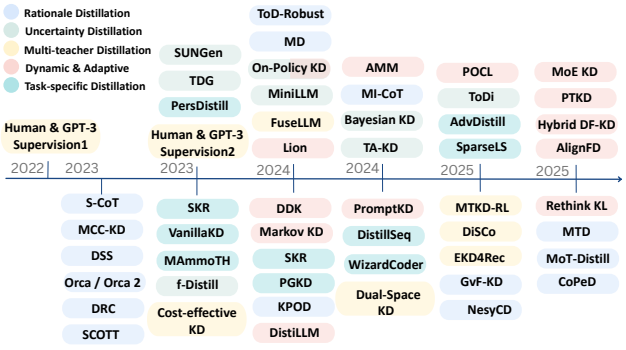


**Fig. 5**: Timeline of Knowledge Distillation Methods (2022–2025).

### 3.3.2 Process-based Distillation

Process-based distillation extends classical KD by focusing on how a model derives an answer rather than simply reproducing final outputs. LLMs contain intermediate reasoning traces, or Chain-of-Thought (CoT), that support multi-step inference. Foundational work such as Distilling Step-by-Step (DSS) [122] showed that transferring step-level reasoning enables small students to outperform larger few-shot teachers like PaLM-540B, highlighting that learning reasoning dynamics can be more beneficial than imitating outcomes. Building on this, Orca [123] and Orca2[124] applied "explanation tuning," integrating GPT-4 reasoning demonstrations into LLaMA-based students to strengthen logical coherence and interpretability. However, directly imitating natural-language CoTs often introduces redundancy and ambiguity. Symbolic CoT Distillation (S-CoT) [125] addresses this by transforming reasoning into symbolic sequences, while Mixed Distillation (MD) [129] combines natural-language and program-of-thought reasoning to balance expressive and structural features. Meanwhile, frameworks such as MCC-KD [126], SCOTT [127], and MoT-Distill [132] promote cross-path consistency among reasoning trajectories, and CoPeD [133] introduces correctness-aware filtering to retain only high-confidence reasoning chains. Together, these methods mark a transition from simple imitation toward structured reasoning transfer.

Recent studies further enhance this paradigm through symbolic integration and adaptive optimization. KPOD [130] uses token-wise weighting and progressive training to identify crucial reasoning steps, while NesyCD [134] combines neural and symbolic reasoning by assigning general cognitive tasks to neural modules and domain-specific logic to knowledge bases. On the optimization side, MI-CoT [131] redefines reasoning distillation with an information-theoretic objective that aligns rationales and predictions, and GvF-KD [135] highlights the trade-off between reasoning fidelity and generalization, motivating adaptive guidance weighting. Beyond procedural reasoning, MTD [136] condenses model knowledge into interpretable microtheories, enriching conceptual clarity, while the safety-oriented STM [176] improves reasoning stability through gradual adaptation. Overall, these advances—from symbolic to information-theoretic and safety-focused distillation—illustrate a shift toward cognitive reconstruction, enabling compact models to reason transparently and effectively like their large scale teachers.

### 3.3.3 Uncertainty-aware KD

Traditional KD assumes uniform teacher reliability, yet large models often mix high- and low-confidence outputs. Blindly imitating uncertain signals risks amplifying noise. To counter this, *MiniLLM* [140] highlights high-confidence modes via reverse KL regularization, while *ToDi* [145] introduces token-wise dynamic weighting between forward and reverse KL to down-weight uncertain tokens. *f-Distill* [137] unifies these designs under an $f$-divergence view, explaining their sensitivity to confidence bias. Optimization- and data-driven approaches such as *SUNGen* [138] and *Targeted Data Generation(TDG)* [139] adaptively reweight

**Table 3**: An overview of representative methods for knowledge distillation.

| Method | Venue | Highlight |
|---|---|---|
| **Rationale-based Distillation** | | |
| DSS [122] | ACL'23 | CoT supervision; small student can match or surpass larger models. |
| Orca / Orca2 [123, 124] | arXiv'23 | Transfer GPT-4 style reasoning to LLaMA-2 7B/13B models. |
| S-CoT [125] | ACL'23 | Symbolizes natural-language CoT to stabilize training process. |
| MCC-KD [126] | EMNLP'23 | Multiple reasoning chains enforce consistency during student learning. |
| SCOTT [127] | ACL'23 | Distill CoT trajectories to transfer complex reasoning skills. |
| DRC [128] | ACL'23 | Multi-path reasoning reduces bias from single reasoning chain. |
| MD [129] | EMNLP'24 | Mix CoT and PoT traces to strengthen reasoning transfer. |
| KPOD [130] | ICML'24 | Use keypoints to guide progressive CoT distillation steps. |
| MI-CoT [131] | ACL'24 | Maximize mutual information between teacher and student CoT. |
| MoT-Distill [132] | arXiv'25 | Merge teacher-specific branches for robust CoT transfer. |
| CoPeD [133] | arXiv'25 | Distill only high-confidence reasoning traces from teacher. |
| NesyCD [134] | AAAI'25 | Collaborative distillation between neural and symbolic components. |
| GvF-KD [135] | ACL'25 | Investigate generalization vs fidelity paradox in knowledge distillation. |
| MTD [136] | ICLR'25 | Distill topical knowledge into structured Microtheories for QA. |
| **Uncertainty-aware KD** | | |
| f-Distill [137] | ACL'23 | Formalize KD as f-divergence optimization to handle uncertainty. |
| SUNGen [138] | ICLR'23 | Bi-level optimization to reweight synthetic training samples. |
| TDG [139] | ACL'23 | Use GC/IC metrics to guide subgroup-focused data augmentation. |
| MiniLLM [140] | ICLR'24 | Reverse-KL emphasizes high-confidence outputs and avoids noise. |
| On-Policy KD [141] | ICLR'24 | Teacher corrects student generated tokens to reduce gap. |
| ToD-Robust [142] | ACL'24 | Study robustness of KD and mitigate performance degradation. |
| TA-KD [143] | ACL'24 | Teaching assistant model corrects imperfect teachers for students. |
| Bayesian KD [144] | ICML'24 | Bayesian knowledge distillation with uncertainty quantification for reliability. |
| ToDi [145] | arXiv'25 | Token-wise control of KL divergence via fine-grained adjustment. |
| **Multi-teacher Distillation** | | |
| Human + GPT-3 expert supervision 1 [146] | ACL'22 | Collaboration with GPT-3 reduces manual labeling costs. |
| Human + GPT-3 expert supervision 2 [147] | EMNLP'23 | GPT-3 acts as data annotator for NLP tasks. |
| FuseLLM [148] | ICLR'24 | Align heterogeneous teacher outputs using probabilistic fusion. |
| MTKD-RL [149] | AAAI'25 | Robust teachers defend against adversarial input attacks. |
| DiSCo [150] | SIGIR'25 | LLM KD for efficient sparse retrieval in conversational search. |
| EKD4Rec [151] | WWW'25 | Transfer knowledge from ensemble LLMs to sequential recommenders. |
| **Dynamic and Adaptive Strategies** | | |
| SAKD [152] | TIP'22 | Adjust distillation dynamically based on sample and feature importance. |
| Cost-effective KD [153] | ACL'23 | Focus on KD strategies under limited computational resources. |
| On-Policy KD [141] | ICLR'24 | Student self-generates; teacher corrects to reduce exposure bias. |
| Lion [154] | EMNLP'24 | Imitate–discriminate–regenerate adversarial learning loop. |
| PromptKD [155] | CVPR'24 | Adapt teacher outputs to student using prompt engineering. |
| Dual-Space KD [156] | EMNLP'24 | Distill knowledge in dual spaces for holistic capture. |
| DistiLLM [157] | ICML'24 | Streamlined protocol for efficient LLM distillation. |
| AMM [158] | NeurIPS'24 | Align distributions using adversarial learning and moment matching. |
| DDK [159] | NeurIPS'24 | Distill domain knowledge for domain-specific LLM efficiency. |
| Markov KD [160] | ECCV'24 | Use Markov processes to address "nasty teacher" distillation. |
| POCL [161] | arXiv'25 | Curriculum KD: progress from simple to complex tasks. |
| Rethink KL [162] | COLING'25 | Re-evaluate KL divergence loss for optimal knowledge transfer. |
| MoE KD [163] | ICLR'25 | Employ mixture-of-experts for efficient student distillation. |
| Hybrid DF-KD [164] | AAAI'25 | Combine hybrid data-free methods for knowledge transfer. |
| AlignFD [165] | ACL'25 | Align features across layers beyond logits for effective KD. |
| PTKD [166] | ACL'25 | Explore systematic KD strategies during LLM pretraining. |
| **Task-specific and Foundations** | | |
| MAmmoTH [167] | arXiv'23 | Parallel CoT and programmatic reasoning for math generalist. |
| PersDistill [168] | EMNLP'23 | Error-correction distillation improves HumanEval performance. |
| VanillaKD [169] | NeurIPS'23 | Examine scaling behavior and effectiveness of vanilla KD. |
| SKR [170] | EMNLP'23 | Self-knowledge-driven KD; retrieval under uncertainty only. |
| DistillSeq [171] | ISSTA'24 | Safety-alignment testing using knowledge distillation techniques. |
| WizardCoder [172] | ICLR'24 | Programming CoT evolution improves code generation quality. |
| PGKD [173] | EMNLP'24 | Performance-guided KD enhances large-scale text classification. |
| AdvDistill [174] | arXiv'25 | Reward-guided distillation allows student to surpass teacher. |
| SparseLS [175] | ACL'25 | Reduce KD compute with sparse-logit caching and sampling. |

samples and focus training on informative regions. In addition, *On-Policy Distillation* [141] mitigates train–inference gaps by involving student-generated trajectories. Collectively, these techniques evolve KD from suppressing uncertainty toward exploiting it for more reliable supervision.

Recent work extends this to probabilistic and structured formulations. *Bayesian KD (BKD)* [144] models the teacher as a prior over student parameters, offering principled uncertainty quantification, while *Teaching-Assistant-in-the-Loop (TA-KD)* [143] introduces an auxiliary evaluator that reconciles teacher confidence,

student self-consistency, and assistant scoring to filter unreliable supervision. *ToD-Robust* [142] further validates these ideas in multilingual speech recognition, where distilling robust knowledge from Whisper-large-v2 improves recognition across diverse dialects. Complementing earlier designs such as *ToDi* [145] and *CoPeD* [133], these advances outline a clear trajectory toward robust uncertainty-aware KD, enabling students to learn not only what to imitate but how confidently to do so.

### 3.3.4 Multi-teacher Distillation

Using a single teacher often limits knowledge coverage and introduces domain bias. Multi-teacher distillation overcomes this by combining complementary expertise from multiple teachers to provide more balanced supervision. For robustness, MTKD-RL [149] employs specialized teachers against diverse attacks, allowing students to gain generalized resilience without adversarial exposure. FuseLLM [148] aligns probabilistic outputs from several LLMs to fuse heterogeneous knowledge and improve calibration, while DiSCo [150] distills conversational retrieval signals from multiple teachers to boost in-domain and out-of-domain performance. These works demonstrate a shift from robustness-driven specialization to collaborative knowledge fusion that enhances generalization.

Beyond robustness and retrieval, multi-teacher setups also extend to reasoning and recommendation. EKD4Rec [151] transfers knowledge from multiple LLM recommenders into compact models by averaging confident predictions, improving efficiency and accuracy trade-offs. Compared with FuseLLM and DiSCo, such task-oriented fusion shows the flexibility of multi-teacher design. Human-in-the-loop methods [146], [147] further refine aggregated supervision with expert feedback to reduce synthetic bias. Altogether, multi-teacher distillation is evolving into a collective intelligence framework where diverse teachers promote robustness, adaptability, and balanced knowledge transfer.

### 3.3.5 Dynamic and Adaptive Strategies

Traditional KD often uses fixed supervision, overlooking that a student's learning ability changes over time. To adapt to this dynamic process, curriculum-based and interaction-driven strategies have emerged. POCL [161] adopts progressive overload training from simple to complex tasks, while [177] stabilizes learning through data scheduling. Dynamic interaction approaches further improve adaptability. On-Policy Distillation [141] lets the student generate responses that the teacher corrects. Lion [154] introduces an imitate–discriminate–regenerate cycle, and PromptKD [155] adapts prompts to reduce exposure bias. Large-scale methods such as DistiLLM [157] use skew KL objectives with adaptive off-policy reuse, while Adversarial Moment-Matching Distillation(AMM) [158] aligns behavioral moments through adversarial training. Spot-Adaptive KD(SAKD) [152] further refines supervision by choosing optimal teacher layers per sample. Together, these works transform KD into a progressive and interactive process where guidance evolves with student capability.

Beyond curricula and feedback, adaptive KD also focuses on improving alignment across feature spaces, losses, and domains. Dual-Space KD [156] aligns teacher–student representations through cross-model attention, aiding cross-architecture and multilingual transfer. Adaptive KL (AKL) [162] dynamically balances divergence terms, while Markov KD [160] uses probabilistic transitions to handle resistant teachers. In multimodal and domain settings, LLaVA-MoD [163] employs a progressive MoE pipeline, Pre-training Distillation [166] optimizes distillation during pre-training, and DDK [159] re-weights domain samples adaptively. Hybrid Data-Free KD [164] combines real and synthetic data for stable supervision, Cost-Effective KD [153] simplifies losses for efficiency, and Beyond Logits [165] extends KD to feature dynamics. Collectively, these methods redefine KD as a dynamic, self-adjusting process adapting supervision to model growth and domain context.

### 3.3.6 Task-specific KD

Task-specific KD targets specialized reasoning domains where generic distillation cannot capture domain-dependent expertise. In mathematical reasoning, MAmmoTH [167] shows how parallel distillation of chain-of-thought and programmatic reasoning builds mathematical generalist students, while AdvDistill [174] uses reward-guided data selection to adjust supervision by confidence and outcome signals, enabling students to outperform teachers on challenging benchmarks such as GSM8K. For code generation, WizardCoder [172] employs evolutionary instruction optimization to transfer structured reasoning, and Personalised Distillation [168] adapts training to each student's individual error patterns. In safety and alignment, DistillSeq [171] ensures responsible generation with low computational cost, while VanillaKD [169] revisits scaling laws, showing competitive results with minimal architectural change. These works reveal a trend from static compression to adaptive, task-aware supervision, aligning reasoning, coding logic, and alignment goals with evolving student capability.

Efficiency oriented research complements this adaptation by optimizing cost and scalability. Sparse Logit Sampling [175] records only the most informative logits to cut training storage and computation with little accuracy loss. SKR [170] adds self knowledge retrieval, activating external data only under uncertainty, while PGKD [173] scales KD into large scale industrial classification through a performance guided active learning loop that emphasizes hard examples and efficient feedback. Conceptually related to AdvDistill, PGKD extends reward guided supervision to continuous performance adaptation, achieving large efficiency gains in inference and deployment cost. Together, these studies

demonstrate how task specific and performance guided KD integrate domain reasoning precision with industrial practicality, shaping KD into a core approach for specialized and cost effective model development.

### 3.3.7 Open Problems and Outlook

Despite significant progress in reasoning, uncertainty, and task-specific KD, key challenges remain. Robust and secure distillation is a main focus, as models still risk inheriting backdoors from untrusted teachers. RobustKD [178] mitigates this through feature variance alignment to preserve both safety and accuracy. Structural alignment is another frontier. Neural Collapse inspired KD [179] transfers teacher feature geometry to improve generalization, while Multi-Level Optimal Transport [180] aligns token and sequence representations for cross-architecture transfer. Efficiency principles are also being formalized, as Law of Capacity Gap [181] reveals a linear scaling relation between teacher and student capacity, guiding efficient teacher selection.

KD is also expanding across modalities and toward autonomous learning. RDRec [182] transfers rationale-level knowledge in recommendation models, Align-KD [183] optimizes cross-modal feature alignment, and Agent of Thoughts Distillation [184] compresses temporal reasoning for video understanding. Meanwhile, teacher-free approaches such as Tailored Coordinate System (TCS) [185] enable self-distillation within a learned subspace. Together, these lines—robust detoxification, geometric alignment, scaling efficiency, cross-modal transfer, and autonomous learning—signal KD's transition from static mimicry to a self-evolving, interpretable, and resilient framework spanning safety, efficiency, and multi-domain reasoning.

## 3.4 Low-Rank Factorization

### 3.4.1 Background

Low rank factorization targets the linear and projection layers that dominate compute and memory traffic in LLMS. A weight matrix $W \in \mathbb{R}^{m\times n}$ is approximated as $W \approx AB^\top$ with rank $r \ll \min\{m,n\}$, which replaces one large matrix multiplication with two smaller ones and can reduce MAC operations, parameter count, and memory movement. When low rank updates are merged before serving, the execution graph stays dense and adds no extra latency at inference time [186]. Low rank also complements quantization in deployment pipelines, since parameter reduction and low bit formats address different bottlenecks [187]. These properties are attractive under on-device budgets where memory and bandwidth are tight and long context workloads are common.

When low rank factorization is applied to LLMs, several new challenges emerge. **(i) Kernel realism.** Fewer FLOPs do not guarantee lower latency; speedups depend on fused kernels, compiler passes, and the runtime scheduler on the target device. **(ii) Rank selection and stability.** Per layer ranks need to reflect spectral variation across blocks and remain stable across tasks, otherwise quality degrades under aggressive compression. **(iii) Interaction with quantization.** Coordination with INT4 or INT8 is required to avoid error compounding at high compression ratios. **(iv) Bandwidth and the KV cache.** Under long context and streaming, the KV cache and memory bandwidth can dominate so arithmetic savings may be overshadowed unless kernels and caching are engineered accordingly. **(v) Merge policy at deployment.** Keeping low rank updates external simplifies adaptation but can add memory or latency, while merging before deployment changes time to first token and steady state throughput.

At a high level we organize the discussion by when low rank is applied and what it targets. Training time low rank with parameter efficient updates learns compact adapters during adaptation and merges them before deployment (Sec. 3.4.2). Post training low rank applies layerwise decomposition and truncation under data or time limits, typically with light calibration and in coordination with low bit formats (Sec. 3.4.3). Architectural low rank redesigns attention or state updates so that complexity scales with sequence length and bandwidth is improved when the KV cache dominates (Sec. 3.4.4). For orientation, Fig. 6 provides a compact timeline, and Tab. 4 summarizes representative variants and their emphasis.

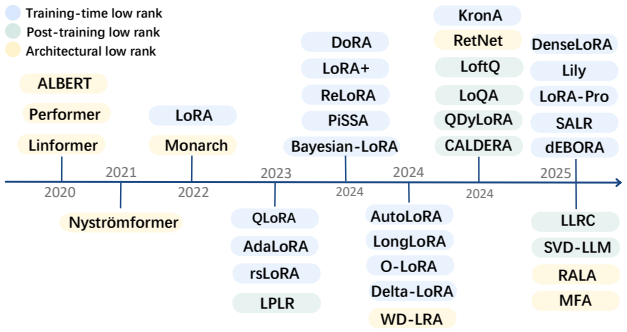


**Fig. 6**: Timeline of Low-Rank Factorization Methods (2020–2025).

### 3.4.2 Training Time Low Rank with PEFT

PEFT learns compact low rank updates on a frozen backbone and merges them before inference so the served graph stays dense and latency neutral; the canonical form is LoRA [186], $W \approx W_0 + BA$ with modest rank. In practice, three pressures shape the design: first, tight memory and throughput budgets; second, stability and optimization ceilings; third, layer and task heterogeneity under a fixed adapter budget. For the first, placing adapters on a low bit base as in QLoRA [187] reduces optimizer and state memory without changing the served graph; periodic reinitialization as proposed in ReLoRA [190] shortens

**Table 4**: An overview of representative methods for low-rank factorization .

| Method | Venue | Highlight |
|---|---|---|
| **Training Time Low Rank with PEFT** | | |
| LoRA [186] | ICLR'22 | Low-rank adapter path; mergeable before serving for zero latency overhead. |
| AdaLoRA [188] | ICLR'23 | Importance-aware adaptive rank allocation under a fixed budget. |
| QLoRA [187] | NeurIPS'23 | 4-bit finetuning (NF4 + double-quant + paged optimizer). |
| rsLoRA [189] | arXiv'23 | Rank-stabilization scaling for robust adapters. |
| ReLoRA [190] | ICLR'24 | Train high-rank nets via low-rank updates to save memory/time. |
| LongLoRA [191] | ICLR'24 | Efficient finetuning for long-context settings. |
| Bayesian-LoRA [192] | ICLR'24 | Uncertainty-aware low-rank adaptation. |
| LoRA+ [193] | ICML'24 | Learning-rate/scale rebalancing for stable LoRA training. |
| DoRA [194] | ICML'24 | Magnitude–direction decoupling; update direction only; merge with no runtime cost. |
| AutoLoRA [195] | NAACL'24 | Automatic rank search. |
| PiSSA [196] | NeurIPS'24 | Principal-component/SVD-based initialization. |
| O-LoRA [197] | arXiv'24 | Orthogonality constraints to reduce interference. |
| Delta-LoRA [198] | arXiv'24 | Residual-style low-rank updates. |
| KronA [199] | CVPR'24 | Kronecker-factorized adapters. |
| dEBORA [200] | ICLR'25 | Bilevel PEFT with per-layer rank selection; no implicit diff; away-step Frank–Wolfe; identifiability. |
| SALR [201] | ICLR'25 | Plug-in sine activation to boost effective rank at the same parameter budget (ViT/LLM/NeRF/3D). |
| LoRA-Pro [202] | ICLR'25 | Optimization view; re-weight adapter gradients to approximate full-FT low-rank gradient; consistent gains. |
| Lily [203] | ACL'25 | Cross-layer interconnected PEFT: locally shared $A$, globally shared $B$ with data-dependent routing; higher-rank $\Delta W$ with equal/fewer params (input-conditioned, typically non-mergeable). |
| DenseLoRA [204] | ACL'25 | Dense low-rank adapter + representation compressor; 0.01% params reaches 83.8% on LLaMA3-8B. |
| **Post Training Low Rank** | | |
| LPLR [205] | NeurIPS'23 | Randomized mappings for low-precision × low-rank compression. |
| LoftQ [206] | ICLR'24 | Quantization-aware low-rank initialization/co-design. |
| CALDERA [207] | NeurIPS'24 | Joint low-precision + low-rank ($W \approx Q+LR$); strong at $\leq$2.5 bits/param. |
| QDyLoRA [208] | arXiv'24 | Dynamic rank with quantization for stability at low bits. |
| LLRC [209] | ICLR'25 | Differentiable rank selection; learn per-layer $r$ with few samples. |
| SVD-LLM [210] | ICLR'25 | Truncation-aware whitening + sequential low-rank update; strong at high compression ratios. |
| **Architectural Low Rank and Linear Attention** | | |
| ALBERT [211] | ICLR'20 | Factorized embeddings and parameter sharing. |
| Linformer [212] | ICML'20 | Low-rank projection for linear-time/space self-attention. |
| Performer [213] | NeurIPS'20 | Kernel random features (FAVOR+) for linear attention. |
| Nyströmformer [214] | AAAI'21 | Nyström approximation to attention. |
| Monarch [215] | ICML'22 | Expressive structured low-rank matrices. |
| RetNet [216] | ICML'24 | Retention units; parallel training; near-$O(1)$ state at inference, low KV bandwidth. |
| Maestro [217] | ICML'24 | Trainable low-rank decomposition (LoD) to uncover architectural low-rank structure; fewer params with strong accuracy; works with quant/distillation. |
| WD-LRA [218] | NeurIPS'24 | L2/weight decay induces low-rank attention matrices; theoretical & empirical caution for regularization strength. |
| RALA [219] | CVPR'25 | Rank-augmented linear attention; keeps $O(n)$ while closing gap to Softmax (vision evidence). |
| MFA [220] | ACL'25 | Low-rank multi-matrix factorization in QK; reuse K as V; ∼56–93.7% KV cache reduction. |

training time at fixed rank; when extreme parameter efficiency is required, DenseLoRA [204] replaces the two matrix adapter with a dense low rank module plus a shared compressor, while structured factors such as KronA [199] trade rank for expressivity under the same budget. For the second, stable initialization and scale control, as seen in PiSSA [196] and LoRA Plus [193], and direction-only updates from DoRA [194] improve convergence without harming mergeability; from an optimization perspective, LoRA Pro [202] reweights adapter gradients to better match full finetuning signals, lifting the performance ceiling. For the third, uniform ranks misalign with layer sensitivities: AdaLoRA [188] allocates ranks under a fixed budget, AutoLoRA [195] searches ranks automatically, and dEBORA [200] formulates per layer rank selection as an efficient bilevel program with the Frank Wolfe method and identifiability guarantees.

At a constant parameter budget, accuracy can be improved further by increasing the effective rank, for example via sinusoidal modulation in SALR [201]. Cross layer interconnected experts, as proposed by Lily [203], enhance capacity with locally shared $A$ and globally shared $B$, but are typically input conditioned and thus nonmergeable at serve time, trading a small online overhead for quality. A practical recipe is to begin with a mergeable LoRA [186] or DoRA [194] baseline, switch to QLoRA [187] when memory is the bottleneck, adopt PiSSA [196] and LoRA Plus [193] for stability or LoRA Pro [202] when under-optimized, and use AdaLoRA [188], AutoLoRA [195], and dEBORA [200] for rank allocation across layers.

### 3.4.3 Post Training Low Rank

When full finetuning is impractical, post training low rank treats compression as a short calibration problem and exports dense weights for serving. Recent progress can be organized along two axes. First, how the low rank structure is obtained: by direct decomposition with truncation, where SVD LLM [210] links singular values to loss through truncation aware whitening and compensates with sequential low rank updates; by adding a low rank residual to a quantized backbone so that $W \approx Q + LR$, a technique used in CALDERA [207] to stabilize very low bit operations; or by randomized mappings co-designed for low precision and low rank, as in LPLR [205]. LoftQ [206] integrates quantization into the low rank initialization loop to improve low bit stability at the source. Second, how rank is allocated: fixed budgets are simple but misaligned with layer heterogeneity; dynamic rank, as seen

in QDyLoRA [208], adapts to data and calibration difficulty; and differentiable rank selection, demonstrated by LLRC [209], learns the per layer $r_\ell$ end to end from few samples.

A practical recipe is to start from a robust eight or four bit baseline with quantization aware initialization using LoftQ [206]; for moderate compression, couple fixed truncation with the sequential updates of SVD LLM [210] to recover accuracy; under aggressive ratios or at most two point five bits per parameter, prefer the residualized form $Q + LR$ in CALDERA [207]; when layers differ markedly, allocate rank adaptively with QDyLoRA [208] or learn it differentiably with LLRC [209].

### 3.4.4 Architectural Low Rank and Linear Attention

When long context or streaming inference is limited by KV cache traffic rather than dense matrix multiplications, changing the computation graph is often more effective than per layer factorization. The common goal is to replace quadratic softmax attention with linear time or near constant state updates while preserving modeling power. Concretely, the attention map can be compressed before it is formed by projecting keys and values to a low rank subspace, as in Linformer [212]; softmax can be kernelized to enable associativity and single pass accumulation, as in Performer [213]; or a structured subset can be sampled to approximate the full map, as in Nyströmformer [214]. Alternatively, explicit attention maps can be avoided by maintaining a recurrent state that is close to constant size at inference through retention units, as in RetNet [216]. These mechanisms address the same bandwidth bottleneck through different approximations: projection, kernel features, sampling, or recurrent state.

A persistent challenge is the accuracy gap to softmax at long range. Recent work narrows this gap by raising the effective rank inside linear pipelines: Rank-Augmented Linear Attention (RALA) [219] increases representational capacity while keeping linear complexity; and Multi-matrix Factorization Attention (MFA) [220] expands head capacity in the query-key pathway and reuses keys as values to reduce cache usage. Orthogonal structural tools further cut footprint or expose low rank inductive bias: Monarch [215] constrains matrices to expressive structured low rank, ALBERT [211] factorizes embeddings and shares parameters across layers, and Maestro [217] learns a trainable low rank decomposition that integrates with quantization and distillation. A complementary caution is that standard L2 or weight decay can implicitly push attention matrix products toward low rank with measurable accuracy impact, as shown by WD LRA [218]. In summary, architectural low rank offers a principled path to bandwidth aware scaling, and rank augmentation together with factorization helps mitigate the classic accuracy trade off in linear attention.

### 3.4.5 Open Problems and Outlook

Open issues in low-rank factorization align with the three intervention points in the model pipeline: training time, post training, and architectural design. **Training time PEFT** After adapters are merged, a core question is where to allocate rank. Layer sensitivities differ widely, so a fixed rank wastes capacity; data-driven or differentiable allocation helps, as demonstrated by AdaLoRA [188] and LLRC [209], but still lacks evidence of transfer across tasks. A second tension concerns expressivity and mergeability: input-conditioned cross-layer schemes like Lily [203] increase effective rank but complicate truly zero-overhead serving, whereas mergeable stabilizers such as PiSSA [196], LoRA+ [193], and DoRA [194] keep the execution graph clean. **Post training factorization** The main failure mode is instability under aggressive compression, where truncation and quantization errors accumulate. Co-designed pipelines help, including quantization-aware initialization as in LoftQ [206], a quantized backbone corrected by a low-rank residual as in CALDERA [207], and truncation-aware SVD with compensatory updates as in SVD-LLM [210]; yet robustness with tiny calibration sets and under distribution shift remains open. **Architectural design** Linear-time attention still trails softmax at long range. The current trend is to raise effective rank within linear pipelines to close the gap while keeping linear complexity, as seen in RALA [219] and MFA [220]. A complementary caution is that standard weight decay can implicitly push attention products toward low rank with measurable accuracy impact, which, as shown by the analysis in [218], calls for rank-aware regularization in attention blocks. Priorities include transferable rank assignment that turns AdaLoRA and LLRC style signals into deployment policies, compression that remains stable at no more than four bit with minimal calibration (e.g., LoftQ, CALDERA, SVD-LLM), and architectural rank augmentation that preserves linear complexity while approaching softmax accuracy (e.g., RALA, MFA).

## 3.5 Hybrid Methods

### 3.5.1 Background

Hybrid compression combines two or more model side techniques such as quantization [221], pruning [222], [223], low rank [188], [209], and distillation [117]. The goal is to push compression beyond what any single method can achieve while preserving accuracy and delivering end-to-end gains under constrained on-device budgets where memory, bandwidth, and long context workloads are common.

When hybrid compression is applied to LLMs, several design questions arise, each motivating one family below. **(i) Turning parameter savings into throughput.** Low bit formats reduce bytes, but real speedups depend on operator coverage and kernel shape; structured or block sparsity aligns compute so savings appear on hardware, a principle detailed

in foundational sparsity research [223]. This motivates quantization with sparsity. **(ii) Stabilizing very low bits under spectral concentration.** A few dominant directions make INT4 brittle; a small low-rank path can capture these directions while the residual runs at low bit, leveraging principles from parameter-efficient adaptation methods [188], [209]. This motivates quantization with low rank. **(iii) Recovering semantics after structural removal.** Pruning exposes compute but can drop important subspaces; compact low-rank updates are a light way to restore them under tight budgets, a recovery concept explored in the context of post-pruning adaptation [223]. This motivates pruning with low rank. **(iv) Making sub–4–bit calibration trainable.** Post training calibration alone is fragile at ultra low bit; a teacher signal, as introduced by knowledge distillation [117], turns it into a learnable objective and improves stability, a technique now widely adopted in quantization [221]. This motivates quantization with distillation. **(v) Repairing task competence after pruning.** Structured or magnitude pruning often erases task–specific behavior; distillation provides a mechanism to restore function [117] while preserving the sparse structure, a combination pioneered in early work like Deep Compression [222]. This motivates distillation with pruning. **(vi) Increasing student capacity only while learning.** Small students can underfit the teacher; a technique demonstrated in [224] uses temporary low-rank over-parameterization to add degrees of freedom during training, which are then merged away for serving. This motivates distillation with low rank.

At a high level, we organize hybrids by how two techniques interact inside a pipeline, pointing to subsections that detail each family. **(i) Quantization with sparsity.** Aligns low bit formats with structured or semi-structured masks so that parameter savings become throughput (Sec. 3.5.2). **(ii) Quantization with low rank.** Uses factorized updates to absorb directions that destabilize INT4 while keeping dense execution after merge (Sec. 3.5.3). **(iii) Pruning with low rank.** Pairs capacity removal with compact adapters to recover accuracy under tight budgets (Sec. 3.5.4). **(iv) Quantization with distillation.** Uses teachers to stabilize low bit calibration and close the quality gap (Sec. 3.5.5). **(v) Distillation with pruning.** Restores semantics after structured or magnitude pruning (Sec. 3.5.6). **(vi) Distillation with low rank.** Guides adapter learning and improves transfer at small parameter budgets (Sec. 3.5.7). For orientation, Fig. 7 gives a compact timeline, and Tab. 5 summarizes representative instances and their emphasis.

### 3.5.2 Quantization and Sparsity

Quantization reduces bytes moved, while sparsity lowers effective multiplies; together they target the two dominant costs in transformer inference. Used alone, each has failure modes: at very low bitwidths, outlier

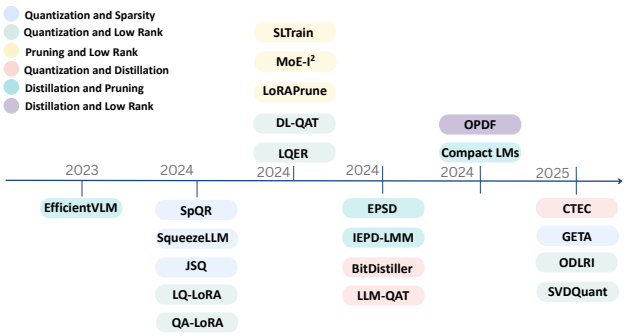


**Fig. 7**: Timeline of Hybrid Methods (2023–2025).

channels can inflate scales and destabilize calibration [221], and pruning can remove hard-to-recover capacity, especially with small calibration sets [222], [223]. A hybrid pipeline turns these into design levers: reserve precision or density where sensitivity is high, compress aggressively elsewhere, and calibrate the two knobs so one covers the other's weakness. Recent methods make this concrete. SpQR [78] preserves outliers explicitly while quantizing the remainder, stabilizing low-bit accuracy. SqueezeLLM [64] allocates bitwidth and sparsity by measured sensitivity, mixing dense and sparse quantization across layers. JSQ [225] unifies sparsification and quantization with a shared sparsity metric and adds an activation editor to suppress outliers harmful to low-bit inference. GETA [226] automates the co-optimization with quantization-aware training and structured masks, enforcing layerwise bit constraints and simplifying engineering. In practice, block or group sparsity is preferred so zeros translate to throughput on real kernels [223], and short QAT or light distillation after calibration helps when pushing to sub-4-bit regimes [221].

### 3.5.3 Quantization and Low Rank

Quantization cuts memory traffic, yet very low bits are brittle when a few directions dominate the spectrum of a layer. Low rank updates complement this by capturing the dominant directions with a tiny set of trainable parameters, leaving the remainder to a low bit path. Two practical integration patterns have emerged. The first decomposes each pretrained matrix into a high precision low rank term plus a quantized residual, then updates only the small low rank part during adaptation. This yields strong quality under tight memory budgets and keeps the runtime graph friendly to kernels. LQ-LoRA [227] demonstrates this template and further selects per layer bitwidth and block size under a global budget through an integer program, with a data aware variant that weights reconstruction by Fisher information. QA-LoRA [228] follows the same spirit but makes the adaptation quantization aware through group wise operators, so that weights are quantized during finetuning and the adapters are merged back without loss in accuracy.

**Table 5**: An overview of representative methods for hybrid model compression.

| Method | Venue | Highlight |
|---|---|---|
| **Quantization and Sparsity** | | |
| SpQR [78] | ICLR'24 | Sparse plus quantized representation with outlier preservation for low-bit accuracy. |
| SqueezeLLM [64] | ICML'24 | Sensitivity-driven allocation mixing dense and sparse quantization for stable low bits. |
| JSQ [225] | ICML'24 | Joint sparsification and quantization with a unified sparsity metric and a search based activation editor to handle outliers; reports up to 7.96× compute reduction on LLaMA. |
| GETA [226] | CVPR'25 | Automatic joint structured pruning with quantization-aware training via a quantization-aware dependency graph, partially projected SGD enforcing per-layer bit-width constraints, and an interpretable joint learning strategy; competitive on CNNs and Transformers. |
| **Quantization and Low Rank** | | |
| LQ-LoRA [227] | ICLR'24 | Decompose weights into a high-precision low-rank branch plus a quantized branch; update only the low-rank branch during finetuning; ILP to pick per-matrix bitwidth and block size; Fisher-weighted variant. |
| QA-LoRA [228] | ICLR'24 | Quantization-aware LoRA with group-wise operators: quantize weights during finetuning (e.g., INT4), update low-rank adapters, then merge into a quantized model without accuracy loss. |
| LQER [229] | ICML'24 | Low-rank reconstruction of quantization error for near-lossless W4A8 without distillation or grid search; activation-induced scaling; about 1.36× lower hardware resource use. |
| DL-QAT [230] | EMNLP'24 | Low-rank QAT with group-specific scales and LoRA-style updates in the quantization space; $<1\%$ trainable params. |
| ODLRI [231] | ACL'25 | Outlier-driven low-rank initialization under $W \approx Q+LR$; assigns the low-rank term to activation-sensitive outliers, mitigating quantization harm and improving low-bit perplexity and zero-shot accuracy. |
| SVDQuant [232] | ICLR'25 | 4-bit quantization with a high-precision low-rank branch that absorbs outliers via SVD; residual handled by low-bit path; fused kernels in Nunchaku cut activation movement; large speedups on SDXL, PixArt-Σ, and FLUX.1. |
| **Pruning and Low Rank** | | |
| LoRAPrune [233] | ACL'24 | Structured pruning with low-rank compensation to recover accuracy under slimming. |
| MoE-I$^2$ [234] | EMNLP'24 | MoE-oriented pipeline combining inter-expert pruning and intra-expert low-rank decomposition; preserves zero-shot accuracy. |
| SLTrain [235] | NeurIPS'24 | Sparse plus low-rank parameterization during pretraining; reduces memory and pairs with quantized updates. |
| **Quantization and Distillation** | | |
| LLM-QAT [236] | ACL'24 | Data-free quantization-aware training with self-distillation; supports KV cache quantization. |
| BitDistiller [60] | ACL'24 | QAT with self-distillation for sub-4-bit LLMs; asymmetric quantization and clipping plus a confidence-aware KL objective; strong 3-bit and 2-bit results with lower data and compute. |
| CTEC [237] | AAAI'25 | Quantized diffusion with distillation: cross-timestep error correction to counter error propagation, improving low-bit image quality on LSUN and ImageNet. |
| **Distillation and Pruning** | | |
| EfficientVLM [238] | ACL'23 | Distillation then modality-adaptive pruning for vision–language models; ∼2.2× speedup with high retention. |
| EPSD [239] | AAAI'24 | Early pruning combined with self distillation; preserves distillable weights to keep pruned nets trainable and improves KD efficiency; strong results on CIFAR and ImageNet scale. |
| IEPD-LMM [240] | WWW'24 | Iterative pruning with task-tailored distillation for LMMs; deployed at Ant Group with latency cut from 700 ms to 90 ms and large energy savings. |
| Compact LMs [241] | NeurIPS'24 | Best-practice guide combining depth, width, head and MLP pruning with knowledge distillation; 2–4× compression. |
| **Distillation and Low Rank** | | |
| OPDF [224] | NeurIPS'24 | Over-parameterized distillation via tensor decomposition: scales student parameters during training without increasing inference cost; competitive KD performance across NLP and CV. |

A second line uses low rank structure to repair or redistribute quantization error after calibration. LQER [229] reconstructs the quantization error with a compact low rank module guided by activation induced scaling, reaching near lossless W4A8 while avoiding grid search and iterative distillation. DL-QAT [230] shows that quantization aware training can be lightweight by training less than one percent of parameters through group specific scales and LoRA style updates inside the quantized space, which lifts low bit downstream accuracy. Role assignment between the two components also matters. ODLRI [231] initializes the low rank path to absorb activation sensitive outliers so the quantized term can use smaller scales with lower error, improving perplexity and zero shot accuracy at low bits. Beyond language models, SVDQuant [232] applies the same principle to diffusion models. It shifts outliers into a high precision low rank branch obtained by singular value decomposition and leaves the residual to a four bit path. A co designed engine fuses the low rank kernels into the quantized path to avoid extra activations, delivering large speedups while preserving image quality.

### 3.5.4 Pruning and Low Rank

Pruning removes channels, heads, or blocks to expose computational savings, but aggressive slimming can drop important directions and destabilize optimization. To counter this, low rank updates offer a light mechanism to restore the most valuable subspaces with a small parameter and compute budget. A practical recipe is to prune first for structure and then attach a compact low rank compensator so that accuracy recovers while the graph remains kernel friendly. LoRAPrune [233] exemplifies this strategy by pairing structured pruning with a learned low rank correction that recovers quality at target sparsity and keeps the deployment graph simple. In architectures with mixture of experts, capacity is concentrated in a subset of experts, which makes indiscriminate pruning risky. MoE–I$^2$ [234] addresses this by pruning at the expert level using importance signals and then applying low

rank decomposition inside the remaining experts, preserving zero shot accuracy while reducing model size and inference cost.

A complementary viewpoint is to bake sparsity and low rank structure into the model during pretraining so that compression is not an afterthought. SLTrain [235] follows this route with a sparse plus low rank parameterization that lowers memory traffic and also composes cleanly with later quantized updates, demonstrating that early factorization can make downstream compression more stable. Across these designs, the key choices are where to impose structure for hardware alignment, how to set per layer ranks to avoid underfitting or wasted capacity, and how to schedule light retraining so that the low rank path restores pruned semantics without reintroducing heavy compute.

### 3.5.5 Quantization and Distillation

Quantization reduces memory and bandwidth, but at very low bit levels the induced noise can destabilize optimization and degrade task accuracy. Distillation provides a stabilizing signal by training the low bit student to match a higher fidelity teacher or its own full precision behavior, often turning brittle post training quantization into a trainable objective. In practice, effective hybrids decide three things up front: the source of soft targets, the scheduling between quantization loss and distillation loss, and how to handle long context artifacts such as the KV cache. LLM-QAT [236] exemplifies this recipe with a data free pipeline that couples quantization aware training and self distillation, and extends the path to KV cache quantization so that gains persist under long contexts. When pushing below four bits, stabilization becomes harder; BitDistiller [60] advances this frontier by pairing asymmetric quantization and clipping with a confidence aware divergence objective under self distillation, delivering strong two and three bit results with modest data and compute budgets.

The same idea transfers to iterative generative models where quantization errors can accumulate across steps. Rather than only matching a teacher at each step, CTEC [237] teaches a quantized diffusion model to correct the previous step's error while learning the current target, reducing cross timestep error propagation and improving image quality at low precision. Across tasks, these methods point to a common pattern: use distillation to absorb quantization noise where it matters, choose targets that reflect deployment bottlenecks, and schedule loss weights to maintain stability as bitwidth drops.

### 3.5.6 Distillation and Pruning

Pruning removes parameters to meet latency and memory targets, but aggressive removal can erase task specific competence. Distillation supplies a soft supervisory signal that steers the pruned network back toward the teacher's function, turning a capacity reduction step into an optimization problem with semantic guidance. Two design choices dominate outcomes: when to distill relative to pruning, and how to allocate sparsity across layers and modalities. A distill-then-prune workflow helps produce compact, task-agnostic backbones that tolerate later structure removal; for instance, EfficientVLM [238] shows that in vision-language models, modality-adaptive pruning after pretraining distillation yields about a twofold speedup with high retention. Early in the training process, EPSD [239] reduces cost by identifying and preserving weights that are most amenable to distillation, improving the trainability of sparse students without a heavy teacher pipeline.

Under deployment constraints, iterative schemes such as IEPD-LMM [240] alternate structure search with task-tailored distillation so that each pruning step is corrected before the next, enabling large latency reductions in production workloads. For general-purpose language models, Compact LMs [241] provides a set of best practices combining depth, width, attention, and MLP pruning with carefully chosen distillation targets to achieve two to four times compression while using only a small fraction of the original data, demonstrating that the pairing scales to billion-parameter regimes. Together, these results support a simple causal story: distillation restores the function that pruning perturbs, while pruning provides the structural freedom that distillation alone cannot efficiently discover.

### 3.5.7 Distillation and Low Rank

Distillation transfers behavior from a strong teacher to a compact student, yet a small student may lack the capacity needed to match the teacher signals during training. To solve this, low rank parameterization offers a pragmatic remedy: introduce additional degrees of freedom during optimization and collapse them away for inference so that the student learns richer transformations without paying a permanent cost. A representative design, exemplified by OPDF [224], over-parameterizes the student via tensor decomposition during distillation, which increases the effective capacity while preserving the target footprint at serve time. By scaling the student parameters only for the learning phase and then folding the factors back, the method narrows the fit gap that plain distillation often leaves in deep language and vision stacks.

In practice, outcomes hinge on three choices. First, where to insert low rank factors, for example in attention projections or feedforward blocks, since placement governs gradient flow and expressivity during the student match. Second, how to schedule the rank over training, which balances stability against overfitting to teacher logits. Third, how to merge before serving so that the final model retains the compact compute and memory profile. Evidence from this approach of over-parameterized distillation shows competitive results across natural language and vision tasks while

keeping inference cost unchanged, indicating that temporary capacity can materially improve student quality without violating deployment budgets.

#### 3.5.8 Open Problems and Outlook

Despite recent progress, two issues still limit hybrid compression at aggressive ratios. The first is stability. Very low bit quantization remains sensitive to outliers and layer heterogeneity, even when combined with low rank compensation or distillation [221]. High sparsity can also remove capacity that light fine tuning does not recover, a long-standing challenge in the field [222], [223]. Designs that assign clear roles to each component help (for example, by letting the low rank path absorb activation-sensitive directions while the quantized path carries the residual), but truly general and robust recipes that transfer across model families are not yet settled.

The second is system alignment. Real gains depend on kernel coverage for low-bit graphs and I/O-aware attention mechanisms, such as FlashAttention [20], that reduce data movement. Long context introduces a bandwidth ceiling from the KV cache, which calls for both efficient runtimes with paging, exemplified by vLLM [22], and direct compression of the cache itself. Methods like KVQuant [54], WKVQuant [46], and 1-bit schemes [47] aim to preserve accuracy at serve time. Consistent evaluation with metrics like time-to-first-token, tokens-per-second, peak memory, and energy would make results comparable, and more automation for per-layer bitwidth, rank, and sparsity schedules would ease adoption in practice, as highlighted in recent surveys on LLM inference systems [242], [243].

# 4 System-level Optimizations for Edge Inference

This chapter explains how to turn model-side compression in Sec. 2.4 into real device speedups by aligning five threads: compiler optimizations in Sec. 4.1 covering graph simplification, autoscheduling and code generation, kernel fusion, and KV aware compilation for compressed models; inference frameworks in Sec. 4.2 comparing MLC LLM, llama.cpp, TensorFlow Lite, and ONNX Runtime while contrasting execution providers and delegates with ahead-of-time compilation and reviewing low bit and heterogeneous back ends; memory optimization in Sec. 4.3 spanning mixed precision and checkpointing during training, KV cache quantization, compression and paging during inference, plus sharding and offloading across tiers; hardware support in Sec. 4.4 across CPU, GPU, NPU, FPGA, ASIC, and processing in memory, mapping low bit formats, sparsity, and two GEMM low rank updates to kernel-friendly execution; and edge and cloud collaboration in Sec. 4.5 including partition and offloading, speculative decoding, distributed experts, cross tier caching, and privacy preserving pipelines. Together these threads compose with quantization, pruning, low rank methods, and KV management to deliver practical gains in accuracy, latency, energy, and memory.

## 4.1 Compiler Optimizations

### 4.1.1 Background

Deploying LLMs on edge and near-edge devices requires navigating severe constraints on computation, memory, and power. Compilers serve as a critical bridge in this process, translating abstract model representations into efficient executables tailored for specific hardware [244]. Their primary role is to systematically transform compressed model graphs, incorporating techniques like quantization, pruning, and low-rank factorization, into optimized code that can run effectively on a diverse landscape of heterogeneous hardware, from CPUs and GPUs to specialized accelerators. This translation is not a monolithic process but an optimization pipeline composed of multiple stages. An effective compiler stack must first understand the semantics of model compression to preserve its benefits. It then explores a vast search space of possible execution plans to balance computation with memory access. Finally, it generates specialized machine code that maximizes the capabilities of the underlying hardware. As LLM workloads become more dynamic, particularly with growing context windows, compilers also face the complex challenge of managing runtime phenomena like the KV cache. However, bridging the gap between algorithmic potential and performance on edge devices remains a significant challenge. The diversity of hardware backends, the dynamic nature of modern workloads, and the ever-present memory bandwidth bottleneck create persistent obstacles that can limit realized efficiency [243]. These issues necessitate a deeper look into the compiler's internal mechanisms, its historical evolution, and the open problems that define its future. This section will explore these dimensions by examining the layered technique stack that powers modern compilers (Sec.4.1.2), tracing their evolution from static optimizers to dynamic co-designers (Sec.4.1.3), and outlining the key open problems and future outlook (Sec. 4.1.4).

### 4.1.2 Layered Technique Stack

The compiler optimization pipeline is structured in layers, each with a distinct responsibility. At the front-end and intermediate representation (IR) layer, the primary goal is to make model compression semantics visible to the compiler. This is achieved through expressive IRs like Relay/Relax [245] and MLIR dialects [246]. These frameworks represent quantization scales, sparsity patterns, and low-rank factorizations as integral attributes. This design is a core principle of frameworks like TVM [244] and enables subsequent optimization passes to safely fuse, reorder, or rematerialize operations. Downstream, the middle-end layer translates these structured graphs into hardware-specific execution plans. This process is largely driven by

auto-schedulers, which search for optimal strategies in tiling, vectorization, and thread mapping. Early systems like AutoTVM [247] used template-guided search, while more recent approaches like Ansor [248], MetaSchedule [249], and Hidet [24] automate this exploration. The search is guided by learned cost models that evaluate the trade-offs between computation and memory traffic, a concept foundational to both auto-scheduling [250] and modern kernel-authoring frameworks like Triton [251]. Finally, the back-end layer translates these plans into highly optimized code, focusing on co-optimizing operators and memory management. For critical operations, IO-aware kernels are designed to minimize data movement, even at the cost of recomputation. Prominent examples include FlashAttention [20], its successors FlashAttention-2 [252] and FlashAttention-3 [253], and the decoding-focused FlashDecoding++[21]. For state management, techniques directly address the KV cache bottleneck, such as the virtual memory approach in PagedAttention[22] or the compression-focused designs of PyramidKV [254] and KVQuant [54]. This layered dependency is causal; sustaining end-to-end throughput requires tight integration across all layers, a challenge addressed by large-scale inference systems that co-design scheduling and memory management [255], [256].

### 4.1.3 Evolution from Static Graphs to Dynamic Codesign

The evolution of compiler frameworks reflects a shift from static graph compilation toward runtime co-adaptation under growing memory pressure and workload variability. From 2018 to 2021, systems such as TVM [244], MLIR [246], and Glow [257] standardized multi-level intermediate representations and operator fusion, enabling full-pipeline graph lowering across hardware targets. As model scales expanded and devices diversified, hardware-aware auto-scheduling extended this paradigm: AutoTVM [247] introduced template-driven search, Ansor [248] automated the exploration process, MetaSchedule [249] incorporated learning-based cost modeling, and Hidet [24] unified thread, warp, and block mappings [250], [251]. These frameworks collectively replaced manual kernel tuning with adaptable scheduling that balances tiling, locality, and portability.

Since 2022, long-context and online workloads have accelerated a shift toward dynamic full-stack codesign, where compiler optimization co-evolves with runtime execution. IO-aware kernels such as FlashAttention [20], FlashAttention2 [252], FlashAttention3 [253], and FlashDecoding++ [21] fuse attention and decoding to minimize memory traffic. Their designs inspired PagedAttention [22], which unifies paged KV management with continuous batching, and its extension to FlexAttention [25] applies multi-version specialization to reduce KV fragmentation and latency. Compiler–runtime hybrids deepen adaptivity: PyTorch Inductor performs JIT specialization [23], BOLT couples graphs with fused kernels [258], and RAP [26] dynamically adjusts compression and cache budgets. FLASH [259] modularizes speculative decoding, while ultra-low-bit CPU kernels [28] broaden compilation targets. At the system level, LeMix [260] coordinates training and inference scheduling, eLLM [261] enhances long-context stability through elastic memory, and surveys [243], [242] consolidate emerging runtime standards. As KV traffic intensifies and quantization deepens, compilers increasingly rely on memory-centric cost models and runtime multi-versioning to balance efficiency and adaptability across dynamic workloads.

### 4.1.4 Open Problems and Outlook

Despite advances in kernel fusion and memory planning, compiler performance still lags behind algorithmic efficiency due to three coupled limits. Dynamic shapes and variability introduce runtime branching and tensor-rank changes from low-rank adapters, sparsity, and variable sequence lengths, breaking static fusion and cache reuse. Multi-version compilation with online specialization, used by PyTorch Inductor [23], Hidet [24], and BOLT [258], mitigates this issue, as shown by FlexAttention's integration with PagedAttention [25]. Uneven backend support for quantization (W4A8/W4A4), sparsity, and low-rank GEMM causes fragmentation across devices. TVM [244], MLIR [246], and Triton [251] aim to unify lowering, while autoschedulers such as AutoTVM [247], Ansor [248], and MetaSchedule [249] help but leave gaps across CPU, GPU, NPU, and SoC. Memory hierarchy and KV bandwidth limit long-context inference: without IO-aware kernels like FlashAttention [20], [252], or paging via PagedAttention [22], long sequences erode quantization and sparsity gains. Elastic memory [261] and hierarchical caching, combined with KV systems such as PyramidKV [254] and KVQuant [54], offer relief. Finally, the proliferation of backends inflates portability and maintenance costs. MLIR dialects [246], Glow-style lowering [257], and Hidet mapping [24] improve reuse, while runtime compression [26], unified scheduling [260], and low-bit CPU kernels [28] must coevolve to avoid new silos.

Compiler research now requires tighter integration of IR design, backend mapping, and memory systems. *Near term*, embed compression metadata (quantization scales, sparsity masks, low-rank factors) directly in a Relay-style IR [245], and adopt multi-version compilation with online specialization. Graph passes should integrate IO-aware attention and KV paging [20], [22]. *Mid term*, standardize precision and sparsity metadata across compilation stages, and train cost models that consider bandwidth and residency, building on AutoTVM, MetaSchedule, and Hidet [247], [249], [24], with Triton [251] for kernel authoring. *Long term*, manage the KV path as an elastic memory subsystem with hierarchical caches [254], [261], and evaluate throughput and latency under long-context inference as key

compilation metrics aligned with recent surveys [243], [242].

## 4.2 Inference Frameworks

### 4.2.1 Background

In engineering practice, the **inference framework** serves as the software abstraction that converts compressed models into executable graphs on heterogeneous devices, seeking a balance between performance and portability. As model compression and hardware diversity expand, frameworks become crucial for bridging algorithmic optimizations with real deployment constraints.

On edge hardware, limited memory, bandwidth, and kernel coverage create strong demands for unified execution environments. To meet these constraints, in particular, inference frameworks focus on three key capabilities: (i) encoding **compression semantics** such as **low-bit quantization**, **sparsity**, and **low-rank fusion** to fully exploit model compactness; (ii) coordinating **data flow** and scheduling around **attention** and **KV hotspots** to maximize memory locality and minimize transfer latency; and (iii) maintaining consistent **back-end performance** across CPU, GPU, and NPU targets. Together, these mechanisms form an integrated optimization layer that bridges algorithmic compression with system-level efficiency, enabling scalable and portable inference across heterogeneous edge–cloud hardware landscapes (Sec. 4.2.2 & 4.2.3).

### 4.2.2 Archetypes and trade-offs

Compiler-centric pipelines (MLCLLM) adopt a structured compilation flow built on TVM [244], lowering models through Relay and multi-level IRs [246] to produce TensorIR with auto-scheduled kernels guided by learned cost models [262], [263], [264]. Preserving quantization, pruning, and low-rank metadata as IR annotations enables the autoscheduler to explore memory-efficient tiling and fusion, improving TTFT and throughput under limited bandwidth. These pipelines excel at ahead-of-time specialization and metadata-aware optimization but are constrained by compilation latency and immature backends for emerging ISAs. Manual-kernel frameworks such as *llama.cpp* take an opposite approach, using compact C/C++ implementations with handcrafted GGML loops [265], [266], [267]. Tailored for specific ISAs (x86, ARM, GPU), they exploit intrinsics and fused kernels for strong latency and energy efficiency on W4/W5 variants, though portability and update agility remain limited by manual development effort.

HAL-based delegates and execution-provider architectures extend these optimizations through modular offloading and unified acceleration. TensorFlow Lite employs Delegates to partition workloads across NNAPI, GPU, DSP, CoreML, and vendor NPUs [268], [269], [270], while ONNX Runtime adopts the `.onnx` format with Execution Providers (EPs) for TensorRT, DirectML, and QNN [271]. Both prioritize integration simplicity and cross-platform portability, now supporting 4-bit and FP4 quantization via GPTQ [49], AWQ [37], and FP4 interfaces [272]. Under ALEM evaluation, compiler-centric pipelines provide high specialization from IR metadata, manual-kernel paths deliver low-latency static optimization, and Delegate/EP ecosystems balance efficiency and flexibility for mobile or IoT deployment. Integer inference [273], GPTQ [49], AWQ [37], and QuIP [274] remain stable baselines across these design paradigms.

### 4.2.3 Open Problems and Outlook

Inference frameworks are converging across three fronts yet still face open issues in unifying abstraction, adaptability, and multimodal scheduling. Compiler stacks increasingly expose interfaces for handcrafted kernels, manual implementations integrate JIT and graph capture, and meta-runtimes unify heterogeneous backends through shared APIs [275], [271]. This convergence improves flexibility but introduces complexity in dependency management and backend compatibility. Meanwhile, low-bit and on-device adaptation methods have matured, with GPTQ and AWQ establishing stable W4 baselines [49], [37] and new FP4 pathways emerging [272]. Although combining PEFT adapters and merged low-rank updates [186] reduces runtime overhead, maintaining accuracy consistency across hardware remains unresolved. At the system level, multimodal and edge-intensive workloads require tight scheduling among CPU, GPU, NPU, VPU, and DSP pipelines [276], [277], [268]. Current frameworks embed compression semantics in IR, align tensor formats with kernel layouts, and manage KV quantization and eviction [265], [266], [267], yet joint optimization of TTFT, throughput, and energy remains difficult. No existing stack reconciles peak performance, portability, and engineering simplicity [278]; thus selecting an effective archetype remains a co-design trade-off guided by hardware capacity and product demands [279], [280].

## 4.3 Memory Optimization

### 4.3.1 Background

Memory consumption, stemming from parameters, activations, optimizer states, and inference-time KV caches, is one of the primary bottlenecks in scaling large language models (LLMs) and deploying them on constrained hardware. Unlike Section 3, which focuses on architectural compression, this section surveys system- and runtime-level techniques that aim to reduce memory overhead during training, inference, and deployment.

The rapid expansion of LLMs has pushed memory capacity and bandwidth to their limits. Memory arises mainly from four components: **(i) parameters**, **(ii) activations**, **(iii) optimizer states**, and **(iv) KV caches** [281], [282]. Each component predominantly dominates a distinct computation stage. Parameters persist throughout training and inference, activations

peak during backpropagation, optimizer states enlarge with adaptive algorithms, and KV caches are primarily associated with inference and grow rapidly with sequence length, though certain streaming or online-distillation training setups may also maintain KV states. Together, these elements restrict scalability and efficient deployment across devices.

Reducing this overhead is crucial for sustainable LLM development. Early pruning and sparsity studies [222], [223] demonstrated that memory compression improves both storage and runtime efficiency. More recent efforts combine algorithmic innovation with system-level co-design, forming three major categories: **(i) training-time optimization**, **(ii) inference-time compression and caching**, and **(iii) system-level and hardware-aware approaches**. The following subsections (Sec. 4.3.2–Sec. 4.3.4) examine these directions in detail. Table 1 and Figure 3 summarize representative techniques and their historical evolution.

### 4.3.2 Training-Time Memory Optimization

Training consumes the majority of memory resources because it stores multiple model states, optimizer parameters, and intermediate activations. Mixed-precision training using 16-bit floating-point formats (FP16 or BF16) [283], [284] reduces memory footprint while exploiting tensor cores. Building upon this, COAT (ICLR 2025) [285] compresses optimizer states and activations under FP8 via Dynamic Range Expansion and Mixed-Granularity Activation Quantization, cutting end-to-end memory versus BF16.

Low-precision arithmetic alone cannot address activation memory. **Gradient checkpointing** [286] recomputes selected intermediate tensors during backpropagation to save memory. Compiler-assisted **rematerialization** [287], [288] further refines this trade-off by scheduling recomputation automatically. Parameter-efficient tuning methods, including LoRA [186], low-rank factorization [58], and Quantization-Aware Training (QAT) [61], reduce trainable parameters and gradient memory during fine-tuning, but do not substantially reduce inference-time memory unless merged back into weights.

For large-scale models exceeding single-device capacity, distributed memory partitioning methods such as ZeRO [289], ZeRO-Offload [290], and Fully Sharded Data Parallel (FSDP) [291] distribute model states across GPUs, CPUs, and NVMe. These approaches minimize peak training memory by partitioning parameters, gradients, and optimizer states, while later system-level scheduling frameworks extend these ideas to heterogeneous device hierarchies. Although originally developed for datacenter-scale training, these ideas have inspired lightweight offloading designs for edge and on-device fine-tuning. Collectively, these techniques demonstrate that memory efficiency arises from the synergy of precision scaling, recomputation scheduling, and distributed sharding.

### 4.3.3 Inference-Time Compression and Cache Management

During inference, memory usage is dominated by static weights and dynamic KV caches. Quantization serves as a primary strategy for weight compression. Early integer-only inference [273] established the foundation for reduced precision. Subsequent works such as LLM INT8 [44] and HAWQ V3 [57] improved accuracy retention by adaptive scaling. Recent PTQ/QAT methods (e.g., GPTQ, AWQ, and SmoothQuant) enable per-channel or block-wise quantization that better preserves accuracy in large models, complementing integer-only and adaptive-scaling baselines. However, even with compressed weights, the KV cache remains a major bottleneck as sequence length grows.

Paging-based methods like PagedAttention [22] treat the cache as a collection of virtual memory pages, dynamically swapping entries to manage context. Compression-oriented methods including KVQuant [292] and KIVI [293] use low-bit encoding to shrink cache storage. While KVQuant demonstrates up to 1M–10M context feasibility under specific GPU setups, practical throughput remains bounded by memory bandwidth, paging overhead (e.g., PagedAttention), and scheduler design. Structural innovations such as cross-layer KV sharing [294], Ring Attention [295], and StreamingLLM [296] redesign the attention mechanism to reuse or discard redundant states efficiently.

Runtime frameworks such as vLLM, which integrates PagedAttention [22], demonstrate that algorithmic optimizations can translate directly into practical deployment. Together, these approaches show that inference-time memory optimization depends on the combination of quantization, cache compression, and adaptive runtime scheduling.

### 4.3.4 System-Level and Hardware-Aware Co-Design

Algorithmic compression reduces local memory demand, but sustained efficiency requires coordination at the system and hardware level. Even with quantization and checkpointing, model execution often remains limited by bandwidth and memory hierarchy. Distributed sharding frameworks such as ZeRO [289], ZeRO-Offload [290], and FSDP [291] mitigate these constraints by distributing parameters and optimizer states across heterogeneous devices.

**Memory scheduling frameworks** dynamically manage data placement among high-bandwidth memory (HBM), dynamic random-access memory (DRAM), and non-volatile memory express (NVMe) [288], [287]. Hardware innovations, including the Intelligence Processing Unit (IPU) [297], and efficient kernels such as FlashAttention [20] and PagedAttention [22], exemplify how algorithm–hardware synergy enhances throughput and reduces latency. Emerging HBM3E capacities, Unified Memory strategies, and NVLink/NVSwitch topologies further shape

memory placement and sharding efficiency, requiring co-design with kernels and runtimes.

Modern deployment frameworks such as ONNX Runtime and NVIDIA TensorRT integrate quantization, kernel fusion, and hierarchical memory management into unified inference pipelines. These developments signal a convergence of compiler optimization, runtime scheduling, and hardware specialization, forming a foundation for scalable memory-efficient LLM deployment.

### 4.3.5 Open Problems and Outlook

Despite significant progress, memory optimization for LLMs remains a persistent challenge. Existing solutions typically target individual phases—training, inference, or system design—without achieving unified optimization across layers. Each method involves trade-offs: checkpointing increases computation, offloading depends on interconnect bandwidth, and quantization may affect long-context stability, particularly when KV caches are quantized or shared across layers [281], [282].

Future research will likely emphasize **(i) integrated memory–computation co-design**, **(ii) compiler-guided scheduling**, and **(iii) multi-tier runtime orchestration** across GPUs, NPUs, and edge accelerators. Additionally, rigorous benchmarking and formal analysis of quantization-induced errors are required to ensure stability as sequence lengths approach millions of tokens. Emerging kernel-level methods such as FlashAttention and serving frameworks like vLLM exemplify the convergence of algorithmic and system-level optimization. These directions align with the hardware-support strategies discussed in Section 4.4, pointing toward a unified, memory-aware AI ecosystem.

## 4.4 Hardware Support

### 4.4.1 Background

The performance and efficiency of large language model (LLM) inference depend critically on hardware–software co-design. While model compression reduces parameter counts and computation, real-world deployment remains constrained by power consumption, memory bandwidth, and latency [298], [299]. Different deployment contexts prioritize distinct objectives: mobile devices emphasize efficiency and low power, servers aim for throughput, and embedded controllers must meet stringent memory and energy budgets.

The evolution of accelerator design provides a valuable historical perspective. Early convolutional accelerators such as Eyeriss and SCNN pioneered dataflow-centric architectures with on-chip buffering [300], [301]. These principles later influenced transformer-specific hardware, particularly in sparsity exploitation and memory hierarchy optimization. Today, the hardware landscape spans three main categories: **(i) general-purpose CPUs and GPUs**, **(ii) specialized accelerators** such as NPUs, DSPs, VPUs, and MCUs, and **(iii) custom hardware** including FPGAs, ASICs, and processing-in-memory (PIM) architectures. Each class targets a different balance between flexibility and efficiency.

The following subsections (Sec. 4.4.2–Sec. 4.4.5) examine these categories in detail, outlining how architecture, kernel design, and runtime systems collectively shape the deployment efficiency of modern LLMs.

### 4.4.2 General-Purpose CPUs and GPUs

General-purpose processors remain the most ubiquitous inference platform due to their wide availability and flexible programming models. Central processing units (CPUs) enable broad portability across desktops, edge nodes, and embedded systems. Vector instruction extensions such as Arm NEON, x86 AVX2 and AVX-512, and RISC-V RVV facilitate fine-grained parallelism and can approximate neural processing unit (NPU)-level efficiency for small-batch workloads [298]. The AccLLM framework [302] demonstrates how co-designed kernels improve single-instruction multiple-data (SIMD) utilization. The TinyMLBench benchmark [303] evaluates TinyML workloads on MCU and NPU platforms, reaffirming the CPU's role as a scheduling hub. Practical implementations such as llama.cpp [304] show that INT4 and INT8 GGUF variants can achieve real-time throughput by mitigating memory bottlenecks. Earlier efforts such as the EIE accelerator [305] explored similar directions through weight sparsity and local computation. However, CPUs remain limited by cache hierarchy and bandwidth, especially for large scale workloads.

Graphics processing units (GPUs) dominate edge servers and Jetson-class systems. Recent edge-performance studies [306] show that throughput declines as context length grows, even under INT8 or FP8 quantization, largely because of orchestration overhead between CPU and GPU subsystems [302]. This challenge has motivated kernel-level innovation. FlashAttention [20] minimizes memory traffic via IO-aware tiling, FlashAttention-3 [253] introduces asynchronous execution, and PagedAttention [22] integrates KV cache paging directly with TensorRT-LLM. The LLM Hardware Survey [298] emphasizes that kernel fusion and tiling are essential for efficiency. Overall, CPUs offer portability for short contexts, while GPUs deliver high throughput given proper memory scheduling.

### 4.4.3 Specialized Edge Accelerators: NPUs, DSPs, VPUs, and MCUs

Specialized accelerators balance domain specificity with efficiency. Neural processing units (NPUs) and digital signal processors (DSPs) dominate mobile and IoT platforms by offering strong performance

within limited power envelopes. The OnDeviceNPUstudy [307] shows that NPUs can outperform GPUs in both latency and energy efficiency for small-to-medium LLMs, though CPU fallback remains necessary for unsupported operators. The TinyMLBench benchmark [303] corroborates this, showing that NPUs achieve lower latency but rely on CPU orchestration for high-level task management.

Vision processing units (VPUs) and deep-learning accelerators (DLAs) were initially tailored for vision tasks. The OpenVINO Device Guide [308] indicates that legacy VPU/HDDL support was deprecated after 2023. The LLM Hardware Survey [298] observes that VPUs perform well on CNN-centric workloads but struggle with multimodal transformers without auxiliary accelerators. At the extreme edge, microcontrollers (MCUs) coupled with micro-NPUs ($\mu$NPUs) or TinyML compilers enable ultra-low-power operation. The TinyML-Mamba project [309] demonstrates that quantization and distillation allow compact Transformers and Mamba variants to fit within 64 KB of memory. The HybridPIM design [310] achieves 1-bit operation in this domain. TinyMLBench [303] offers detailed energy profiling, while the MicroFlow runtime [311] demonstrates Rust-based orchestration for MCU-class devices. Together, NPUs and DSPs enable mobile inference, VPUs maintain relevance for visual tasks, and MCUs set the lower bound for energy budgets.

#### 4.4.4 Customized and Reconfigurable Hardware: FPGA, ASIC, and PIM Architectures

As model sizes increase, conventional CPUs and GPUs cannot satisfy efficiency requirements. Reconfigurable and custom hardware provide solutions through domain-specific optimization. Foundational accelerators such as Eyeriss and SCNN [300], [301] continue to influence transformer hardware mapping. Field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) now play a central role in customized LLM deployment. The EdgeLLM framework [312] introduces a CPU–FPGA hybrid for edge inference, FlightLLM [313] exploits INT4 quantization and sparsity, and TerEffic [314] proposes a ternary-quantized accelerator. Additional FPGA workflows such as HLSTransform [315] and LightMamba [316] enhance data reuse and bandwidth efficiency. Processing-in-memory (PIM) and near-data processing (NDP) architectures address memory-bandwidth limits by executing computation near storage. The CXL-PIM design [299] exemplifies this approach, while Samsung's HBM-PIM prototype demonstrates feasibility for large-scale adoption. The HybridPIM system [310] achieves high energy efficiency for long-context inference. Overall, FPGA and ASIC designs provide short-term, workload-specific solutions, whereas PIM/NDP architectures represent promising long-term directions for bandwidth-bound computing.

#### 4.4.5 Compiler and Runtime Integration

Effective use of heterogeneous hardware depends on compiler and runtime integration. Modern runtimes evolve together with kernel design. FlashAttention [20], FlashAttention-3 [253], and PagedAttention [22] illustrate the co-optimization between algorithmic and memory scheduling layers. At the compiler level, TVM and MetaSchedule [249] employ probabilistic search for tensor optimization, while Relax supports higher-level operator fusion and partitioning. Benchmarks such as MLPerf Tiny [317], CMSIS-NN [318], and LLM Inference Bench [319] standardize evaluation metrics. The MultiInstanceEdge framework [320] explores scheduling across multiple instances for throughput improvement. Future compiler–runtime research focuses on three directions: **(i) heterogeneous co-execution** across CPUs, GPUs, and NPUs with minimal transfer overhead, **(ii) memory-aware scheduling** that unifies kernel and runtime optimization, and **(iii) automation and standardization** to reduce manual tuning. Recent hardware surveys [321], [322] highlight ecosystem fragmentation and the importance of aligning compiler design with green AI objectives such as energy efficiency and carbon-aware scheduling.

#### 4.4.6 Open Problems and Outlook

Although the hardware ecosystem for LLMs is rapidly evolving, significant challenges persist. Hardware–software integration remains fragmented, and optimization strategies seldom generalize across device classes. Co-design methodologies vary widely among vendors, hindering portability and standardization. Compiler maturity also lags behind hardware innovation, limiting realized efficiency.

Future progress will depend on three key directions: **(i) unified hardware abstraction layers** connecting CPUs, GPUs, NPUs, and custom accelerators; **(ii) energy-aware co-design** integrating PIM/NDP with quantized and sparse models; and **(iii) standardized benchmarks** evaluating end-to-end efficiency rather than isolated kernels. In parallel, sustainability and carbon-aware deployment will become equally important metrics alongside latency and throughput. By combining architectural innovation with compiler intelligence, future hardware ecosystems can enable scalable, efficient, and environmentally responsible LLM inference across all deployment tiers.

### 4.5 Edge–Cloud Collaboration

#### 4.5.1 Background

The difficulty of running LLMs on edge devices has led to a new paradigm: Edge Cloud Collaborative Computing. This approach, which we will refer to as ECCC, is designed to overcome the limitations of running inference solely on the edge or in the cloud. The core idea is to perform computation as close to the data source as possible, which reduces latency, saves energy, and enhances user privacy [323], [324]. ECCC

operates on an edge-first, cloud-fallback principle, handling tasks locally whenever possible but offloading more demanding computations to the cloud. As LLMs grow more resource-intensive, this flexible architecture is crucial for balancing responsiveness, accuracy, and system cost.

To meet these intertwined algorithmic and system-level demands, recent studies have explored several complementary directions that together shape the evolution of hybrid edge–cloud inference. First, **collaborative inference and speculative decoding** (Sec. 4.5.2) dynamically adjust computation depth and token generation according to input complexity, striking a balance between speed and accuracy. Building on this adaptivity, **distributed experts and orchestration** (Sec. 4.5.3) extend coordination to larger heterogeneous infrastructures, effectively managing MoE workloads for improved scalability and throughput. In parallel, **collaborative training and personalization** (Sec. 4.5.4) use federated and split learning to tailor models locally while preserving overall consistency across devices. Furthermore, **caching and system co-design** (Sec. 4.5.5) mitigate memory and I/O bottlenecks through integrated software–hardware optimizations, and **privacy, security, and domain-specific protection** continue to drive design choices that ensure trustworthiness and regulatory compliance. Collectively, these approaches represent a shift from isolated computation toward cohesive algorithm–system–hardware cooperation, enabling scalable, efficient, and privacy-preserving edge–cloud LLM inference.

### 4.5.2 Collaborative inference and speculative decoding

A key challenge in edge–cloud inference is sustaining high accuracy under limited latency and bandwidth. Full cloud execution guarantees quality but induces high communication overhead, while on-device inference is restricted by computing capacity. To balance this trade-off, hierarchical partitioning distributes computation adaptively across device, edge, and cloud. Approaches such as Early Exit [325] and Speculative Decoding [326] embody this principle by aligning computation depth with input complexity. In Early Exit systems like DIMEE [327], confidence-based layers determine whether to proceed locally or offload, cutting energy and communication by 43% with minimal accuracy loss. RecServe [328] extends this mechanism to three stages, maintaining cloud-level quality while halving communication, confirming the effectiveness of confidence-driven partitioning for efficiency–fidelity balance.

At the token level, speculative decoding forges a tighter bond between edge and cloud collaboration. A lightweight edge model drafts candidate tokens, which are verified by a stronger cloud LLM [329]. CE LSLM [326] further improves this process by aligning KV caches and pruning redundant attention to achieve up to six-fold speedup, while Fast SECD [330] reduces latency by 35% and halves API cost. In summary, Early Exit adapts computation across model layers, and speculative decoding accelerates generation at the token level; together they offer complementary pathways that integrate hierarchical partitioning with parallel verification for efficient and adaptive LLM inference across the edge and cloud.

### 4.5.3 Distributed experts and orchestration

Mixture-of-Experts (MoE) architectures scale LLMs effectively, but edge–cloud deployment faces resource imbalance and communication overhead. Cloud execution ensures accuracy yet incurs latency, while edge-only inference is limited by capacity. To mitigate this, recent work emphasizes workload-aware expert placement. DanceMoE [331] improves locality by assigning frequently activated experts to nearby edge nodes, reducing latency and communication by about 30% without accuracy loss. EC2MoE [332] complements this with hardware-aware gating, routing light tasks to edges and complex ones to the cloud, achieving 1.6–2.3× throughput gains with minimal quality degradation. Together, these studies show that expert co-location and resource-guided scheduling effectively balance latency, accuracy, and cost in hierarchical MoE inference.

Beyond static placement, adaptive orchestration introduces elasticity for dynamic workloads. Frameworks such as those in [331], [333] monitor expert load and trigger migration only when beneficial, maintaining throughput while minimizing transfer overhead. Hardware-aware optimization, including ternary quantization with operator fusion on FPGAs [314], further compresses models and reduces power consumption with little accuracy loss. By combining placement, adaptive migration, and hardware-level compression, MoE orchestration evolves from fixed allocation to a self-optimizing pipeline, enabling scalable and efficient inference across heterogeneous edge–cloud systems.

### 4.5.4 Collaborative training and personalization

Edge–cloud training integrates Federated Learning (FL) and Split Learning (SL) to trade off privacy, efficiency, and device capacity. FL aggregates updates securely, while SL partitions models to reduce on-device workload. To refine this hybrid setup, SplitBud [334] and Auto-Split [335] automate layer partitioning and use quantized transfer to cut communication cost. The combined FL–SL framework [336] executes partial forward propagation on devices with FL-style aggregation, speeding convergence with lower overhead. Based on this foundation, Ravan [337] applies multi-head low-rank adaptation, updating only selected subheads for efficiency and robustness under non-IID data. Together, these approaches tighten the coupling of partitioning, aggregation, and adaptation to scale edge–cloud training.

Beyond efficiency, adaptability and privacy have become central goals in collaborative learning. MIDDLE [338] enhances generalization through cross-edge knowledge transfer, achieving 1.5–6.8× faster convergence by mitigating data bias. HyFedRAG [339] integrates edge-side anonymization and compression with cloud-based retrieval-augmented generation, reducing inference latency by about 80% while maintaining data confidentiality [339]. Together, these studies highlight a shift in FL–SL co-design from mere aggregation toward adaptive and privacy-preserving collaboration, enabling personalized and communication-efficient learning across diverse edge–cloud environments.

#### 4.5.5 Caching and system co-design

In edge–cloud inference, frequent KV cache operations are a dominant source of communication and memory overhead. To address this, CE-LSLM [326] improves efficiency by aligning attention layers and sharing KV entries across devices, while Semantic Caching [340] replaces raw states with contextual summaries, reaching an 84% hit rate and about 50% lower latency. Together, they advance caching from structural KV reuse to semantic-level reuse, improving data locality and reducing redundant transfers between edge and cloud.

Profiling further reveals I/O and storage as major performance bottlenecks. EdgeProfiler [341] identifies weight loading and KV read/write as key delays, motivating deeper system optimization. AccLLM [302] addresses these issues with a lambda-shaped attention design and 4-bit KV compression on FPGA, boosting throughput nearly threefold. Building on this, MEADOW [302] and memory-aware layouts [342] reorganize data access to minimize movement across computing units. Collectively, these studies illustrate a coherent optimization hierarchy: semantic caching mitigates redundancy at the algorithmic level, while hardware–software co-design reduces I/O cost and improves throughput at the system level, enabling efficient long-context inference across edge–cloud deployments.

#### 4.5.6 Open Problems and Outlook

Despite progress in efficiency and collaboration, privacy and security remain key challenges in edge–cloud inference. Transmitting intermediate activations or embeddings can expose sensitive information, as they may be inverted to reconstruct original inputs [343]. System-level defenses such as Trusted Execution Environments and adversarial perturbation improve confidentiality with minor accuracy loss but introduce hardware cost and scalability limits, leaving a gap between protection strength and practical deployment.

To close this gap, recent studies propose domain-aware and edge-centric privacy mechanisms. AgentStealth [344] uses a locally deployed reinforcement-guided small model to anonymize text before transmission, mitigating identifier leakage in domains like healthcare and finance. Principle-Guided Verilog Optimization [345] adopts a similar strategy in chip design, uploading abstract design rules instead of raw code to prevent IP leakage while enabling cloud-side optimization. Together, these works mark a progression from generic system protection to application-specific, edge-strengthened privacy solutions that better balance security, accuracy, and scalability in future edge–cloud intelligence.

## 5 Evaluation

### 5.1 Experimental Setup

To understand the trade-offs inherent to different model compression techniques, we evaluated a set of representative models following the A-L-E-M protocol (Sec.2.4). Our selection spans five key approaches, namely quantization, pruning, knowledge distillation, low rank adaptation, and hybrid methods. To this end, we tested these models at two common scales of roughly 1 and 4 billion parameters to reveal their distinct impacts on Accuracy, Latency, Energy, and Memory[346].

Our evaluation protocol is divided into two tracks. The **multiple-choice track** assesses general knowledge and reasoning using ARC-c (25-shot), ARC-e (25-shot), Winogrande (0-shot), and HellaSwag (10-shot). The **generation track** measures mathematical reasoning with GSM8K (8-shot), scored by exact match. For each model, we report an unweighted average over the multiple-choice tasks (MC-Avg). All evaluations use deterministic decoding, with specific generation parameters for GSM8K (`temperature=0.2`, `top_p=0.95`) to ensure reproducibility [346].

Performance metrics are measured under a standardized edge-like scenario. **Latency** is reported as Time-to-First-Token (TTFT) for responsiveness and steady-state tokens per second (tok/s) for throughput, reflecting both prefill and decoding phases [347], [22]. **Memory** is the peak allocated GPU VRAM in megabytes, recorded via `torch.cuda.max_memory_allocated`. **Energy** is the total energy consumed per generation in Joules, measured by integrating GPU power samples from NVML. All performance metrics are averaged over 50 runs with a batch size of one, using an input/output length of 128 tokens.

### 5.2 Results and Analysis

Our empirical findings are presented in two parts to facilitate a clear analysis. **Table 6** details the accuracy evaluation, focusing on task-specific and averaged scores. Subsequently, **Table 7** presents the A-L-E-M performance metrics, focusing on latency, memory, and energy efficiency. We analyze these results to derive method-level insights.

**Table 6**: Accuracy evaluation of representative models.

| Method | Model (size) | MC-Avg[a] | ARC-c | ARC-e | Wino. | Hella. | GSM8K |
|---|---|---|---|---|---|---|---|
| Quant. | Qwen2.5-1.5B INT4 | 0.630 | 0.485 | 0.782 | 0.654 | 0.472 | 0.122 |
| | Qwen2.5-3B INT4 | 0.655 | 0.491 | 0.821 | 0.700 | 0.506 | 0.130 |
| Pruning | Sheared-1.3B-Pruned | 0.374 | 0.192 | 0.441 | 0.504 | 0.360 | 0.010 |
| | Llama3-4B-MLP-Pruned | 0.322 | 0.195 | 0.269 | 0.510 | 0.313 | 0.112 |
| Distill. | TinyLLM-1.11B | 0.517 | 0.352 | 0.687 | 0.586 | 0.445 | 0.015 |
| | Nemotron-Mini-4B | 0.624 | 0.486 | 0.793 | 0.692 | 0.525 | 0.120 |
| LoRA | TinyLLaMA-1.1B-merged | 0.471 | 0.276 | 0.609 | 0.588 | 0.413 | 0.010 |
| | sos-qwen3-4b LoRA | 0.631 | 0.578 | 0.840 | 0.608 | 0.497 | 0.625 |
| Hybrid | LlamaGuard-1B-INT4 | 0.596 | 0.452 | 0.795 | 0.668 | 0.438 | 0.241 |
| | Minitron-4B-Base | 0.622 | 0.485 | 0.796 | 0.695 | 0.522 | 0.380 |

[a] We report scores on individual tasks and the multiple-choice average (MC-Avg).

**Table 7**: Performance evaluation: Latency, Energy, and Memory.

| Method | Model (size) | TTFT (ms)↓[a] | tok/s↑ | Memory (MB)↓ | Energy (J)↓ |
|---|---|---|---|---|---|
| Quant. | Qwen2.5-1.5B INT4 | 58.4 | 117.12 | 1183.9 | 1826.72 |
| | Qwen2.5-3B INT4 | 112.2 | 82.91 | 3199.1 | 1829.91 |
| Pruning | Sheared-1.3B-Pruned | 552.6 | 115.87 | 2592.3 | 263.96 |
| | Llama3-4B-MLP-Pruned | 823.6 | 78.03 | 22 930.8 | 421.20 |
| Distill. | TinyLLM-1.11B | 207.4 | 196.09 | 2108.6 | 20.35 |
| | Nemotron-Mini-4B | 213.3 | 152.61 | 8011.2 | 144.65 |
| LoRA | TinyLLaMA-1.1B-merged | 560.8 | 115.68 | 4208.4 | 244.25 |
| | sos-qwen3-4b LoRA | 1010.5 | 63.35 | 15 366.9 | 522.50 |
| Hybrid | LlamaGuard-1B-INT4 | 103.4 | 285.64 | 2120.0 | 168.21 |
| | Minitron-4B-Base | 210.3 | 150.42 | 8200.2 | 1456.56 |

[a] The arrows (↓↑) indicate whether lower or higher values are better for the corresponding metric.

### 5.2.1 Analysis of Model Accuracy

As shown in Table 6, methods that preserve the full model architecture during compression tend to achieve the highest accuracy. In the 4B parameter class, the quantized Qwen2.5 model (0.655 MC-Avg) and the LoRA-finetuned sos-qwen3 model (0.631 MC-Avg) demonstrate the most effective knowledge retention. Notably, the LoRA-tuned model excels at mathematical reasoning, achieving the highest GSM8K score (0.625) by a large margin, underscoring the power of task-specific fine-tuning. In contrast, the pruned models exhibit a significant drop in accuracy across all tasks. This highlights the inherent difficulty of removing structural components via one-shot methods without impacting the model's complex reasoning capabilities, suggesting that more sophisticated recovery mechanisms or light retraining may be necessary.

### 5.2.2 Analysis of Performance Metrics

The performance data in Table 7 reveals the distinct trade-offs inherent to each technique. **Latency:** Quantized models are the undisputed leaders in responsiveness, with the Qwen2.5-1.5B achieving the lowest TTFT (58.4 ms). This is a direct result of the reduced model size and efficient low-bit integer kernels. Conversely, distilled and hybrid models demonstrate superior sustained throughput, with the Llama-Guard-3-1B leading at 285.64 tok/s, suggesting their compact architectures are highly optimized for the memory-bandwidth-bound decoding phase. **Energy:** A key insight is that energy consumption is not solely dependent on size or speed. The distilled TinyLLM-1.11B is the most energy-efficient model by a large margin (20.35 J). This indicates its architectural efficiency leads to a very low power draw per generated token. In contrast, the quantized models, while fast, are the most power-hungry, revealing a critical trade-off between speed and energy. **Memory:** Peak VRAM usage clearly reflects the nature of each method. Quantization is the most memory-efficient, directly reducing

the bit-width of weights. Distilled and hybrid models also show small footprints due to their compact designs. LoRA models require substantial memory to hold both the base model and adapters, while the high memory usage of the pruned 4B model suggests potential overhead from managing sparse structures in the framework.

#### 5.2.3 Synthesis and Method-Level Takeaways

Synthesizing the findings from both tables, we can outline a practical guide for choosing a compression strategy based on specific on-device constraints:

- **Quantization** is the optimal choice when **ultra-low TTFT and minimal memory footprint** are the primary requirements, making it ideal for applications demanding instant responses.
- **Knowledge Distillation** excels in scenarios where **high throughput and maximum energy efficiency** are paramount, such as continuous text generation on battery-powered devices.
- **Low-Rank Adaptation** is best suited for applications that require **specialized, high-quality reasoning**, provided that the higher latency and memory overhead are acceptable. Merging adapters before deployment is crucial to mitigate these costs.
- **Pruning** presents a complex trade-off. In our one-shot setting, it struggled to maintain competitive accuracy, and its performance benefits depend heavily on co-design with sparse-aware kernels.
- **Hybrid methods**, such as the Llama-Guard-3 model, offer a **balanced profile**, effectively combining fast TTFT, high throughput, and moderate memory, making them excellent all-rounders for general-purpose on-device assistants.

Ultimately, our evaluation underscores that there is no single best compression method. The optimal choice is contingent upon the specific A-L-E-M priorities of the target application and deployment environment.

## 6 Applications

On-device and edge deployment is attractive for its low latency, privacy guarantees, offline robustness, and cost efficiency. Hardware and business constraints vary across scenarios: mobile devices prioritize responsiveness and energy; industrial and in-vehicle settings emphasize reliability and safety; healthcare requires compliance and interpretability; education and office value usability and productivity. We summarize representative tasks and implementation advice below. Figure 8 outlines the key on-device/edge application verticals and the end-to-end optimization pipeline across device–edge–cloud.

### 6.1 Mobile Intelligent Assistants

**Representative tasks.** Call and voice-note summarization, screenshot understanding, keyboard writing aids, and lightweight vision–language use under weak connectivity. **Implementation.** Models under one billion parameters with 4–8 bit inference, KV cache optimization, and device runtimes or graph compilers such as ExecuTorch, NNAPI, and Core ML. System measurements across devices in LLM-Mobile [348] and MELT [349] show that mobile LLMs are primarily bounded by memory and bandwidth; quantization and efficient kernels are therefore decisive for usability. For these small models, structural choices such as deeper–narrower stacks, GQA, and embedding sharing, as shown by MobileLLM [8], provide a stable blueprint for typical mobile tasks. **Deployment tips.** Prefer W4A8 or W4A4; use device-native kernels for hotspots; combine KV paging and compression with short-context priors; prioritize lightweight speech and image encoders; keep an offline path with edge and cloud fallback as recommended by LLM-Mobile [348] and MobileLLM [8].

### 6.2 In-Vehicle Intelligence

**Representative tasks.** Cabin assistants, context-aware prompts, and driving-behavior explanation. **Evidence.** The human-factors–constrained "Rewrite + ReAct + Reflect" workflow from In-Vehicle LLM [350] improves success and acceptance for in-vehicle assistants. DriveGPT-4V2 [351] explores incorporating LLM capabilities into closed-loop driving for joint reasoning and planning. In practice, a clear boundary between the language interaction loop and the control loop should be enforced: the former fits on-device, the latter must obey functional-safety and real-time constraints. **Deployment tips.** Small models with wake word and scenario triggering; constrained tool use (navigation, HVAC, media); firewall and fallback between language and control loops, following In-Vehicle LLM [350].

### 6.3 Industrial IoT for Production Lines, Campuses, and Gateways

**Representative tasks.** Multi-sensor semantic alarms, maintenance QA, anomaly explanation, and edge–cloud visual inspection. **Evidence.** EdgeFM [352] demonstrates feasible designs for leveraging foundation models on resource-limited edge nodes to handle open-set recognition and anomaly detection.MIMIC-Instr [353] and RAG-Fin [354] show that low-latency service abstraction and dynamic scheduling can be migrated to edge clusters to meet SLAs. **Deployment tips.** Adopt a three-tier plan across device, edge, and cloud: device-side lightweight preprocessing and local alarms; edge clusters for compression, retrieval, and batched inference; cloud for offline training and distillation. Prefer INT8 or INT4 inference and fail-safe runtimes on gateways, per EdgeFM [352].

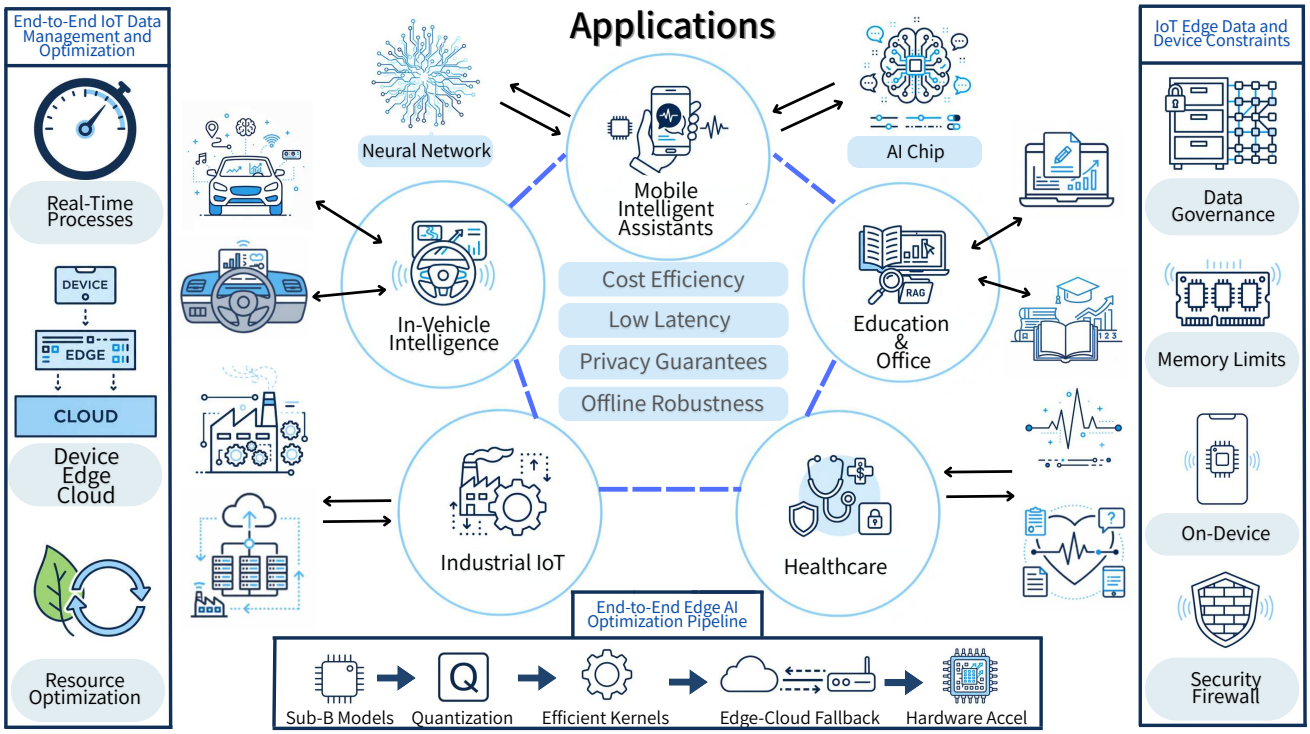


**Fig. 8**: Overview of on-device/edge applications and the end-to-end optimization pipeline across device–edge–cloud.

### 6.4 Healthcare

**Representative tasks.** In-hospital note summarization and QA, interactive triage (question and follow-up), and EHR structuring under strict privacy and compliance. **Evidence.** MediQ [355] drives interaction-centric evaluation closer to real clinical workflows. MIMIC-Instr [353] and EHRNoteQA [356] provide instruction-tuning corpora and benchmarks that enable practical hospital-intranet deployments. **Deployment tips.** Private networks and sandboxes; distillation with quantization (W8A8 or W4A8); template-based explainability with high-risk human-in-the-loop overrides, as supported by MediQ [355] and EHRNoteQA [356].

### 6.5 Education and Office

**Representative tasks.** Writing and marking assistance, meeting-note generation and retrieval QA, and enterprise knowledge assistants using retrieval-augmented generation. **Evidence.** User studies in EdgeFM [352] report productivity and confidence gains with writing assistants, while highlighting dependence and quality-control issues.RAG-Fin [354] proposes auditable two-stage retrieval–generation pipelines for domain RAG. **Deployment tips.** On-device preprocessing with local summarization and controlled retrieval; enforce atomic query decomposition and source traceability to keep hallucinations from contaminating enterprise corpora, following RAG-Fin [354].

### 6.6 Selected Industry Examples

*Android, Gemini Nano.* On-device model capabilities for call summaries, screenshot understanding, and offline text. *Apple, LLM in a Flash.* Layered inference over flash to relax DRAM limits on consumer devices. *Samsung, Galaxy AI.* Continued rollout of on-device summarization and translation features. *Meta, ExecuTorch.* An on-device runtime and toolchain for mobile and edge models.

## 7 Challenges and Future Directions

### 7.1 Current Core Challenges

**Consistency for low bit weights, activations, and KV.** In the past two years, low bit compression has moved toward an end to end pipeline. Weight only post training quantization is comparatively mature with GPTQ [49] and AWQ [37], but once activations and the KV cache are brought into the picture, one faces a chain of trade offs around outliers, rotation or flattening transforms, calibration data, and extra inference time operators. The real challenge is not the accuracy of any single component, but the joint consistency of all three. For instance, rotation or flattening via QuaRot [357] or FlatQuant [14] can stabilize W4A4 zero shot performance, yet if these pre transforms are not fused into kernels, the added operator overhead tends to surface as longer decode latency in edge and embedded settings. Meanwhile, two to four bit asymmetric KV quantization with KIVI [45]

or KVQuant [54] alleviates capacity and bandwidth pressure but propagates quantization error through attention outputs, which in turn perturbs sampling and scheduling; the calibration objective and the runtime integration must therefore be designed together, rather than treating the KV cache as an add on. In practice, these consistency issues are the first to show up when serving 7B or 8B models under W4A4 with KV4.

**Pruning with little or no retraining**. One shot, training free pruning with Wanda [97] offers an easy starting point for large models, but recovery at high sparsity remains limited; structured pruning over width, depth, heads, and channels is more deployment friendly, yet it couples sub structure importance estimation, distillation based recovery, and interfaces to quantization and low rank modules. A pragmatic route is structural re parameterization with distillation recovery: Sheared LLaMA [85] shows that structured reductions can produce competitive small models; more recent works such as SlimLLM [89] and DLP [101] model layer and channel importance and dynamic budgets as search variables, improving stability and reproducibility at high sparsity. What is still missing is cross task evaluation discipline and ready to use edge kernels for the chosen structures.

**Distillation and low rank adaptation**. For LLMs, the difficulty of knowledge distillation lies less in a smaller student per se and more in the choice of supervision objectives and data regimes. MiniLLM [140] illustrates how a unified teacher and student target can drift in the autoregressive setting, leading to well aligned yet weak reasoning or uneven knowledge coverage. Low rank adapters minimize trainable parameters but have long raised concerns about expressivity and sensitivity to initialization and optimization; DoRA [194] and PiSSA [196] mitigate this via weight decomposition and singular vector aware initialization, yet when combined with four bit finetuning as in QLoRA [187], error amplification between quantization and the low rank search space can reappear. In short, KD with low rank is the most training efficient line, but to converge on inference quality and transferability, we still lack a structured distillation protocol that generalizes across tasks.

### 7.2 Future Research Directions

**Unified low bit pipeline**. Treat rotation and flattening with QuaRot [357] and FlatQuant [14] as learnable pre transform layers; during calibration, jointly optimize error targets for weights, activations, and the KV cache; at deployment, force kernel fusion to eliminate runtime overhead, establishing a train time add and inference time merge steady state, especially on edge or SoC devices. The pipeline should explicitly target W4A4 with KV4 by default and ship open scripts and cross task metrics such as perplexity, complex reasoning, MMLU, and GSM8K, with AWQ [37] as a stable activation baseline and KIVI [45] or KVQuant [54] as representative KV paths, so researchers can land on the same quality, latency, and memory Pareto front. This unified approach moves the field further than incremental one off tweaks.

**Joint search for structured pruning and distillation**. Search pruning axes over width, depth, heads, channels, and blocks together with distillation axes over logits, intermediate states, and self generated data. Initialize with a safe one shot or structured baseline using Wanda [97] or Sheared LLaMA [85], refine with layer wise adaptive methods such as SlimLLM [89] or DLP [101], and then condition the distillation targets on a task profile such as context length and the mix of instruction versus reasoning. Evidence around NeurIPS 2024 indicates that structured plus KD tends to dominate pure sparsity or pure KD alone; the next step is to package this into a reproducible auto recipe with a defined search space and early stopping rules, reducing artisanal tuning and yielding deployable students in the three to seven billion range.

**Train and serve unification for hybrid compression**. Practice has shifted from concatenating tricks at inference to co regularizing during training. SpQR [78] obtains near lossless compression with a sparse and quantized format; SLTrain [235] bakes low rank with sparsity into the pre training parameterization, converging faster under the same compute budget. The next step is to plug these parameterizations into QLoRA [187], DoRA [194], and PiSSA [196] style low rank finetuning to form a mergeable train and serve format, that is, train time components that collapse into inference weights with zero extra operators, while keeping edge operators unchanged. This line should further lower the gap from usable to truly good small models.

## 8 Conclusion

This survey systematizes the on-device and edge LLM stack from model miniaturization through deployment. We reviewed quantization, pruning, knowledge distillation, low rank factorization, and hybrid approaches, and connected them to compiler and runtime optimization, memory and KV management, hardware support, and edge and cloud orchestration under the unified ALEM lens. A clear message emerges: accuracy alone is not enough. Device level performance is governed by memory capacity and bandwidth and by the long context KV path. Therefore compression must be carried in the IR, compiled, and scheduled with data locality for input and output in mind. In practice, quantize first pipelines with structured pruning and mergeable low rank compensation, kernel friendly formats such as INT8, INT4, and FP8 with group wise protection, and paged, quantized, and evicted KV cache translate theoretical savings into lower time to first token, higher tokens per second, and better energy use. Looking ahead, bandwidth aware cosearch over bit width, sparsity, rank, and KV policy, multi version compilation with online specialization for dynamic shapes, and elastic memory hierarchies for KV residency offer

the most leverage, together with privacy preserving on-device personalization. Effective edge deployment is a codesign problem: model side compression creates the opportunity, while compilers, runtimes, and KV policies determine the realized speedups and reliability in real workloads.

# Declarations

## Author contributions

Wanyi Chen led the conception and organization of this survey, and was the primary contributor to the sections on model compression and optimization (Section 3), including the comparative analysis and experimental validation of quantization, pruning, low-rank factorization, and hybrid methods. She also drafted the initial manuscript and integrated feedback across all sections. Yanbiao Ma supervised the research process, provided the overall direction and technical guidance, and refined the theoretical framework and manuscript structure. Junhao Wang, Yiwei Zhang, and Yufan Shi contributed to literature collection and assisted in reviewing the sections on system-level optimization and runtime deployment. Tianyi Jiang, Shengxian Zhou, Chenxu Wu, and Andi Zhang supported experimental verification and figure preparation. Chenyue Zhou, Minxuan Wang, Xinyu Liu, Xiaoshuai Hao, Yinan Wu, Yichen Li, Yuwei Hu, Zhao Cao, Yang Lu, and Mengke Li assisted in proofreading, consistency checking, and polishing of the manuscript. Zhiwu Lu and Jungong Han provided academic supervision, critical feedback, and final manuscript revisions. All authors discussed, reviewed, and approved the final version of the paper.

## Funding

This research received assistance and support from Andi Zhang.

# References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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

[3] OpenAI. Gpt-4 technical report, 2023.

[4] An Yang and et al Anfeng Li. Qwen3 technical report, 2025.

[5] Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, and Mohammad Rastegari. Openelm: An efficient language model family with open training and inference framework, 2024.

[6] Marah Abdin and et al Jyoti Aneja. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

[7] Gemma Team and et al Morgane Riviere. Gemma 2: Improving open language models at a practical size, 2024.

[8] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. Mobilellm: optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[9] et al. DeepSeek-AI, Daya Guo. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[10] Laha Ale, Ning Zhang, Scott A. King, and Dajiang Chen. Empowering generative ai through mobile edge computing. *Nature Reviews Electrical Engineering*, 1:478–486, 2024.

[11] Ean Heng Lim, Tong Yuen Chai, Manoranjitham A-P Muniandy, Tien Fui Yong, Boon Yaik Ooi, and Jim-Min Lin. Edge computing and ai for iot: Opportunities and challenges. In *2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*, pages 357–358, 2023.

[12] T. Meuser, L. Lovén, M. Bhuyan, S. G. Patil, S. Dustdar, A. Aral, et al. Revisiting edge ai: Opportunities and challenges. *IEEE Internet Computing*, 28(4):49–59, 2024.

[13] J. Ning, C. Zheng, and T. Yang. Dssd: Efficient edge-device llm deployment and collaborative inference via distributed split speculative decoding. In *International Conference on Machine Learning (ICML)*, 2025.

[14] Y. Sun et al. Flatquant: Flatness matters for llm quantization. In *International Conference on Machine Learning (ICML)*, 2025.

[15] T. Kim et al. Guidedquant: Llm quantization via guided optimization, 2025.

[16] S. Kim et al. Resq: Mixed-precision quantization of llms. In *International Conference on Machine Learning (ICML)*, 2025.

[17] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[18] Q. Le et al. Probe pruning: Dynamic pruning via model-probing. In *International Conference on Learning Representations (ICLR)*, 2025. Poster.

[19] H. Chen et al. Rotpruner: Llm pruning in rotated space. In *International Conference on Learning Representations (ICLR)*, 2025.

[20] Tri Dao, Daniel Fu, Stefano Ermon, Atri Rudra, and Christopher Re. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[21] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus, 2024.

[22] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[23] J. Ansel, M. Kirisame, I. Vozna, T. Hai, J. Roesch, S. Heller, M. Scherer, C. Smith, J. Zou, A. Sulsky, et al. Pytorch 2.0: Torchdynamo/inductor. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[24] Yaoyao Ding, Cody Hao Yu, Bojian Zheng, Yizhi Liu, Yida Wang, and Gennady Pekhimenko. Hidet: Task-mapping programming paradigm for deep learning tensor programs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, page 370–384. ACM, January 2023.

[25] Thomas Joshi, Herman Saini, Neil Dhillon, Antoni Viros i Martin, and Kaoutar El Maghraoui. Paged attention meets flexattention: Unlocking long-context efficiency in deployed inference, 2025.

[26] H. Liu et al. Rap: Runtime-adaptive pruning for llm inference, 2025.

[27] Y. Li et al. Lemix: Unified scheduling for multi-gpu training & inference, 2025.

[28] Evangelos Georganas and et al. Pushing the envelope of llm inference on ai-pc (1–2 bit kernels), 2025.

[29] European Union. Gdpr (regulation eu 2016/679). Technical report, Official Journal of the European Union, 2016.

[30] U.S. Department of Health and Human Services (HHS). Summary of the hipaa privacy rule. Technical report, HHS.gov, 2025.

[31] International Telecommunication Union (ITU-T). G.114: One-way transmission time. Technical report, ITU, 2003.

[32] Android Developers. Gemini nano (aicore on-device). Technical report, Google LLC, 2025.

[33] Vijay Janapa Reddi, David Kanter, Peter Mattson, Jared Duke, Thai Nguyen, Ramesh Chukka, Ken Shiring, Koan-Sin Tan, Mark Charlebois, William Chou, Mostafa El-Khamy, Jungwook Hong, Tom St. John, Cindy Trinh, Michael Buch, Mark Mazumder, Relia Markovic, Thomas Atta, Fatih Cakir, Masoud Charkhabi, Xiaodong Chen, Cheng-Ming Chiang, Dave Dexter, Terry Heo, Gunther Schmuelling, Maryam Shabani, and Dylan Zika. Mlperf mobile inference benchmark, 2022.

[34] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[35] M. Satyanarayanan. The emergence of edge computing. *IEEE Computer*, 50(1):30–39, 2017.

[36] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024.

[37] Ji Lin, Ji Tang, Haotian Tang, Shizhen Yang, Wei-Ming Chen, Wen-Chih Wang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *Proceedings of the Machine Learning and Systems Conference (MLSys)*, 2024.

[38] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[39] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and

Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[40] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.

[41] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[42] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization, 2020.

[43] Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization, 2020.

[44] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[45] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[46] Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more, 2024.

[47] Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization, 2024.

[48] Shiyao Li, Xuefei Ning, Luning Wang, Tengxuan Liu, Xiangsheng Shi, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Evaluating quantized large language models, 2024.

[49] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[50] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models, 2024.

[51] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers, 2022.

[52] Boris van Breugel, Yelysei Bondarenko, Paul Whatmough, and Markus Nagel. Fptquant: Function-preserving transforms for llm quantization, 2025.

[53] Hao Zhang, Aining Jia, Weifeng Bu, Yushu Cai, Kai Sheng, Hao Chen, and Xin He. Flexq: Efficient post-training int6 quantization for llm serving via algorithm-system co-design, 2025.

[54] C. Hooper et al. Kvquant: Towards 10m-token context. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[55] Amir Zandieh, Majid Daliri, and Insu Han. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead, 2024.

[56] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks, 2018.

[57] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. Hawqv3: Dyadic neural network quantization, 2021.

[58] Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for llms, 2024.

[59] Ke Xu, Xiangyang Shao, Ye Tian, Shangshang Yang, and Xingyi Zhang. Autompq: Automatic mixed-precision neural network search via few-shot quantization adapter. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pages 1–13, 2024.

[60] DaYou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. BitDistiller: Unleashing the potential of sub-4-bit LLMs via self-distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 102–116, Bangkok,

Thailand, August 2024. Association for Computational Linguistics.

[61] Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models, 2025.

[62] Wenqiang Zhou, Zhendong Yu, Xinyu Liu, Jiaming Yang, Rong Xiao, Tao Wang, Chenwei Tang, and Jiancheng Lv. Precision neural network quantization via learnable adaptive modules, 2025.

[63] Junbiao Pang and Tianyang Cai. Stabilizing quantization-aware training by implicit-regularization on hessian matrix, 2025.

[64] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization, 2024.

[65] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization, 2024.

[66] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024.

[67] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits, 2024.

[68] Tianyi Zhang and Anshumali Shrivastava. Leanquant: Accurate and scalable large language model quantization with loss-error-aware grid, 2024.

[69] Hao Yu, Yang Zhou, Bohua Chen, Zelan Yang, Shen Li, Yong Li, and Jianxin Wu. Treasures in discarded weights for llm quantization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(21):22218–22226, Apr. 2025.

[70] Seungcheol Park, Jeongin Bae, Beomseok Kwon, Minjun Kim, Byeongwook Kim, Se Jung Kwon, U Kang, and Dongsoo Lee. Unifying uniform and binary-coding quantization for accurate compression of large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28468–28488, Vienna, Austria, July 2025. Association for Computational Linguistics.

[71] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models, 2023.

[72] Yury Nahshan, Brian Chmiel, Chaim Baskin, Evgenii Zheltonozhskii, Ron Banner, Alex M. Bronstein, and Avi Mendelson. Loss aware post-training quantization, 2020.

[73] Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards accurate post-training network quantization via bit-split and stitching. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9847–9856. PMLR, 13–18 Jul 2020.

[74] Xiuying Wei, Ruihao Gong, Yuhang Li, Xianglong Liu, and Fengwei Yu. Qdrop: Randomly dropping quantization for extremely low-bit post-training quantization, 2023.

[75] Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling, 2023.

[76] Janghwan Lee, Minsoo Kim, Seungcheol Baek, Seok Joong Hwang, Wonyong Sung, and Jungwook Choi. Enhancing computation efficiency in large language models through weight and activation quantization, 2024.

[77] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization, 2021.

[78] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression, 2023.

[79] Chao Zhang, Li Wang, Samson Lasaulce, and Merouane Debbah. Baq: Efficient bit allocation quantization for large language models, 2025.

[80] Xuan Shen, Peiyan Dong, Lei Lu, Zhenglun Kong, Zhengang Li, Ming Lin, Chao Wu, and Yanzhi Wang. Agile-quant: Activation-guided quantization for faster inference of llms on the edge, 2025.

[81] Xing Hu, Zhixuan Chen, Dawei Yang, Zukang Xu, Chen Xu, Zhihang Yuan, Sifan Zhou, and

Jiangyong Yu. Moequant: Enhancing quantization for mixture-of-experts large language models via expert-balanced sampling and affinity guidance, 2025.

[82] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21702–21720. Curran Associates, Inc., 2023.

[83] Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024.

[84] Gui Ling, Ziyang Wang, Yuliang Yan, and Qingwen Liu. Slimgpt: Layer-wise structured pruning for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 107112–107137. Curran Associates, Inc., 2024.

[85] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Representation Learning*, volume 2024, pages 5385–5409, 2024.

[86] Bowen Zhao, Hannaneh Hajishirzi, and Qingqing Cao. APT: Adaptive pruning and tuning pretrained language models for efficient training and inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 60812–60831. PMLR, 21–27 Jul 2024.

[87] Yifei Yang, Zouying Cao, and Hai Zhao. Laco: Large language model pruning via layer collapse. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6401–6417, Miami, Florida, USA, Nov 2024. Association for Computational Linguistics.

[88] Shangqian Gao, Chi-Heng Lin, Ting Hua, Tang Zheng, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. Disp-llm: Dimension-independent structural pruning for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 72219–72244. Curran Associates, Inc., 2024.

[89] Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. SlimLLM: Accurate structured pruning for large language models. In *Forty-second International Conference on Machine Learning*, 2025. ICML 2025 poster.

[90] Jiujun He and Huazhen Lin. Olica: Efficient structured pruning of large language models without retraining. In *Forty-second International Conference on Machine Learning*, 2025.

[91] Guinan Su, Li Shen, Lu Yin, Shiwei Liu, Yanwu Yang, and Jonas Geiping. Gptailor: Large language model pruning through layer cutting and stitching, 2025.

[92] Mingzhe Yang, Sihao Lin, Changlin Li, and Xiaojun Chang. Let LLM tell what to prune and how much to prune. In *Forty-second International Conference on Machine Learning*, 2025.

[93] Bairu Hou, Qibin Chen, Jianyu Wang, Guoli Yin, Chong Wang, Nan Du, Ruoming Pang, Shiyu Chang, and Tao Lei. Instruction-following pruning for large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[94] Qi Le, Enmao Diao, Ziyan Wang, Xinran Wang, Jie Ding, Li Yang, and Ali Anwar. Probe pruning: Accelerating LLMs through dynamic pruning via model-probing. In *The Thirteenth International Conference on Learning Representations*, 2025.

[95] Huanrong Liu, Chunlin Tian, Xuyang Wei, Qingbiao Li, and Li Li. Runtime adaptive pruning for llm inference, 2025.

[96] Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR, 23–29 Jul 2023.

[97] Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on*

*Representation Learning*, volume 2024, pages 4942–4964, 2024.

[98] Yuxin Zhang, Lirui Zhao, Mingbao Lin, Sun Yunyun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse llms. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Representation Learning*, volume 2024, pages 249–264, 2024.

[99] Hang Shao, Bei Liu, and Yanmin Qian. One-shot sensitivity-aware mixed sparsity pruning for large language models. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11296–11300, 2024.

[100] Guangji Bai, Yijiang Li, Chen Ling, Kibaek Kim, and Liang Zhao. Sparsellm: Towards global pruning of pre-trained language models. In Douwe Kiela, Faisal Ladhak, Denis Paperno, Anna Rogers, and Avirup Sil, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 52213–52234. Curran Associates, Inc., 2024.

[101] Yuli Chen, Bo Cheng, Jiale Han, Yingying Zhang, Yingting Li, and Shuhao Zhang. DLP: Dynamic layerwise pruning in large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[102] Samiul Basir Bhuiyan, Md. Sazzad Hossain Adib, Mohammed Aman Bhuiyan, Muhammad Rafsan Kabir, Moshiur Farazi, Shafin Rahman, and Nabeel Mohammed. Z-pruner: Post-training pruning of large language models for efficiency without retraining, 2025.

[103] Wei Jiang, Anying Fu, and Youling Zhang. Improved methods for model pruning and knowledge distillation, 2025.

[104] Wei Huang, Anda Cheng, and Yinggui Wang. Mitigating catastrophic forgetting in large language models with forgetting-aware pruning, 2025.

[105] Ameen Ali Ali, Shahar Katz, Lior Wolf, and Ivan Titov. Detecting and pruning prominent but detrimental neurons in large language models. In *Second Conference on Language Modeling*, 2025.

[106] Xin Luo, Xueming Lu, Zihang Jiang, and S. Kevin Zhou. Icp: Immediate compensation pruning for mid-to-high sparsity. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9487–9496, 2025.

[107] Hanyu Hu and Xiaoming Yuan. Spap: Structured pruning via alternating optimization and penalty methods, 2025.

[108] Yiheng Liu, Junhao Ning, Sichen Xia, Xiaohui Gao, Ning Qiang, Bao Ge, Junwei Han, and Xintao Hu. Pruning large language models by identifying and preserving functional networks, 2025.

[109] Jiayu Qin, Jianchao Tan, Kefeng Zhang, Xunliang Cai, and Wei Wang. Maskprune: Mask-based llm pruning for layer-wise uniform structures, 2025.

[110] Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. Lorashear: Efficient large language model structured pruning and knowledge recovery, 2023.

[111] Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. Compresso: Structured pruning with collaborative prompting learns compact large language models, 2023.

[112] Yupeng Ji, Yibo Cao, and Jiucai Liu. Pruning large language models via accuracy predictor, 2023.

[113] Shangyu Wu, Hongchao Du, Ying Xiong, Shuai Chen, Tei-Wei Kuo, Nan Guan, and Chun Jason Xue. Evop: Robust llm inference via evolutionary pruning, 2025.

[114] Rongguang Ye and Ming Tang. One-for-all pruning: A universal model for customized compression of large language models, 2025.

[115] Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. Model compression and efficient inference for large language models: A survey, 2024.

[116] Arnav Chavan, Raghav Magazine, Shubham Kushwaha, Mérouane Debbah, and Deepak Gupta. Faster and lighter llms: A survey on current challenges and way forward, 2024.

[117] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[118] Minchong Li, Feng Zhou, and Xiaohui Song. Bild: Bi-directional logits difference loss for large language model distillation, 2025.

[119] Yixing Li, Yuxian Gu, Li Dong, Dequan Wang, Yu Cheng, and Furu Wei. Direct preference knowledge distillation for large language models, 2025.

[120] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.

[121] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2020.

[122] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes, 2023.

[123] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023.

[124] Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. Orca 2: Teaching small language models how to reason, 2023.

[125] Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step, 2024.

[126] Hongzhan Chen, Siyue Wu, Xiaojun Quan, Rui Wang, Ming Yan, and Ji Zhang. Mcc-kd: Multi-cot consistent knowledge distillation, 2023.

[127] Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. Scott: Self-consistent chain-of-thought distillation, 2023.

[128] Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models, 2023.

[129] Chenglin Li, Qianglong Chen, Liangyue Li, Caiyu Wang, Yicheng Li, Zulong Chen, and Yin Zhang. Mixed distillation helps smaller language model better reasoning, 2024.

[130] Kaituo Feng, Changsheng Li, Xiaolu Zhang, Jun Zhou, Ye Yuan, and Guoren Wang. Keypoint-based progressive chain-of-thought distillation for llms, 2024.

[131] Xin Chen, Hanxian Huang, Yanjun Gao, Yi Wang, Jishen Zhao, and Ke Ding. Learning to maximize mutual information for chain-of-thought distillation, 2024.

[132] Zhanming Shen, Zeyu Qin, Zenan Huang, Hao Chen, Jiaqi Hu, Yihong Zhuang, Guoshan Lu, Gang Chen, and Junbo Zhao. Merge-of-thought distillation, 2025.

[133] Hongyan Xie, Yitong Yao, Yikun Ban, Zixuan Huang, Deqing Wang, Zhenhe Wu, Haoxiang Su, Chao Wang, and Shuangyong Song. Mitigating spurious correlations between question and answer via chain-of-thought correctness perception distillation, 2025.

[134] Huanxuan Liao, Shizhu He, Yao Xu, Yuanzhe Zhang, Kang Liu, and Jun Zhao. Neural-symbolic collaborative distillation: Advancing small language models for complex reasoning tasks, 2025.

[135] Suhas Kamasetty Ramesh, Ayan Sengupta, and Tanmoy Chakraborty. On the generalization vs fidelity paradox in knowledge distillation, 2025.

[136] Nathaniel Weir, Bhavana Dalvi Mishra, Orion Weller, Oyvind Tafjord, Sam Hornstein, Alexander Sabol, Peter Jansen, Benjamin Van Durme, and Peter Clark. From models to microtheories: Distilling a model's topical knowledge for grounded question answering, 2024.

[137] Yuqiao Wen, Zichao Li, Wenyu Du, and Lili Mou. f-divergence minimization for sequence-level knowledge distillation, 2023.

[138] Jiahui Gao, Renjie Pi, Yong Lin, Hang Xu, Jiacheng Ye, Zhiyong Wu, Weizhong Zhang, Xiaodan Liang, Zhenguo Li, and Lingpeng Kong. Self-guided noise-free data generation for efficient zero-shot learning, 2023.

[139] Zexue He, Marco Tulio Ribeiro, and Fereshte Khani. Targeted data generation: Finding and fixing model weaknesses, 2023.

[140] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models, 2024.

[141] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes, 2024.

[142] Abdul Waheed, Karima Kadaoui, and Muhammad Abdul-Mageed. To distill or not to distill? on the robustness of robust knowledge distillation, 2024.

[143] Yuhang Zhou and Wei Ai. Teaching-assistant-in-the-loop: Improving knowledge distillation from imperfect teacher models in low-budget scenarios, 2024.

[144] Luyang Fang, Yongkai Chen, Wenxuan Zhong, and Ping Ma. Bayesian knowledge distillation: A Bayesian perspective of distillation with uncertainty quantification. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12935–12956. PMLR, 21–27 Jul 2024.

[145] Seongryong Jung, Suwan Yoon, DongGeon Kim, and Hwanhee Lee. Todi: Token-wise distillation via fine-grained divergence control, 2025.

[146] Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. Want to reduce labeling cost? gpt-3 can help, 2021.

[147] Bosheng Ding, Chengwei Qin, Linlin Liu, Yew Ken Chia, Shafiq Joty, Boyang Li, and Lidong Bing. Is gpt-3 a good data annotator?, 2023.

[148] Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. Knowledge fusion of large language models, 2024.

[149] Chuanguang Yang, Xinqiang Yu, Han Yang, Zhulin An, Chengqing Yu, Libo Huang, and Yongjun Xu. Multi-teacher knowledge distillation with reinforcement learning for visual recognition, 2025.

[150] Simon Lupart, Mohammad Aliannejadi, and Evangelos Kanoulas. Disco: Llm knowledge distillation for efficient sparse retrieval in conversational search, 2025.

[151] Yue Wang, Dingyi Zhang, Haoyu Wenren, Yue Wang, and Yingming Li. Ekd4rec: Ensemble knowledge distillation from llm-based models to traditional sequential recommenders. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, New York, NY, USA, 2025. Association for Computing Machinery.

[152] Jie Song, Ying Chen, Jingwen Ye, and Mingli Song. Spot-adaptive knowledge distillation. *IEEE Transactions on Image Processing*, 31:3359–3370, 2022.

[153] Sayantan Dasgupta, Trevor Cohn, and Timothy Baldwin. Cost-effective distillation of large language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7346–7354, Toronto, Canada, July 2023. Association for Computational Linguistics.

[154] Xiao Yu, Qingyang Wu, Yu Li, and Zhou Yu. Lions: An empirically optimized approach to align language models, 2024.

[155] Zheng Li, Xiang Li, Xinyi Fu, Xin Zhang, Weiqiang Wang, Shuo Chen, and Jian Yang. Promptkd: Unsupervised prompt distillation for vision-language models, 2024.

[156] Songming Zhang, Xue Zhang, Zengkui Sun, Yufeng Chen, and Jinan Xu. Dual-space knowledge distillation for large language models, 2024.

[157] Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. Distillm: Towards streamlined distillation for large language models, 2024.

[158] Chen Jia. Adversarial moment-matching distillation of large language models, 2024.

[159] Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Zhiqi Bai, Jie Liu, Ge Zhang, Jiakai Wang, Yanan Wu, Congnan Liu, Wenbo Su, Jiamang Wang, Lin Qu, and Bo Zheng. Ddk: Distilling domain knowledge for efficient large language models, 2024.

[160] En-hui Yang and Linfeng Ye. Markov knowledge distillation: Make nasty teachers trained by self-undermining knowledge distillation fully distillable. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 154–171, Cham, 2025. Springer Nature Switzerland.

[161] Lingyuan Liu and Mengxiang Zhang. Being strong progressively! enhancing knowledge distillation of large language models through a curriculum learning framework, 2025.

[162] Taiqiang Wu, Chaofan Tao, Jiahao Wang, Runming Yang, Zhe Zhao, and Ngai Wong. Rethinking kullback-leibler divergence in knowledge distillation for large language models, 2024.

[163] Fangxun Shu, Yue Liao, Le Zhuo, Chenning Xu, Lei Zhang, Guanghao Zhang, Haonan Shi, Long Chen, Tao Zhong, Wanggui He, Siming Fu, Haoyuan Li, Bolin Li, Zhelun Yu, Si Liu, Hongsheng Li, and Hao Jiang. Llava-mod: Making llava tiny via moe knowledge distillation, 2024.

[164] Jialiang Tang, Shuo Chen, and Chen Gong. Hybrid data-free knowledge distillation, 2024.

[165] Guoqiang Gong, Jiaxing Wang, Jin Xu, Deping Xiang, Zicheng Zhang, Leqi Shen, Yifeng Zhang, JunhuaShu JunhuaShu, ZhaolongXing ZhaolongXing, Zhen Chen, Pengzhang Liu, and Ke Zhang. Beyond logits: Aligning feature

dynamics for effective knowledge distillation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23067–23077, Vienna, Austria, July 2025. Association for Computational Linguistics.

[166] Hao Peng, Xin Lv, Yushi Bai, Zijun Yao, Jiajie Zhang, Lei Hou, and Juanzi Li. Pre-training distillation for large language models: A design space exploration, 2024.

[167] Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning, 2023.

[168] Hailin Chen, Amrita Saha, Steven Hoi, and Shafiq Joty. Personalised distillation: Empowering open-sourced llms with adaptive learning for code generation, 2024.

[169] Zhiwei Hao, Jianyuan Guo, Kai Han, Han Hu, Chang Xu, and Yunhe Wang. Vanillakd: Revisit the power of vanilla knowledge distillation from small scale to large scale, 2023.

[170] Yile Wang, Peng Li, Maosong Sun, and Yang Liu. Self-knowledge guided retrieval augmentation for large language models, 2023.

[171] Mingke Yang, Yuqi Chen, Yi Liu, and Ling Shi. Distillseq: A framework for safety alignment testing in large language models using knowledge distillation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '24, page 578–589. ACM, September 2024.

[172] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2025.

[173] Flavio Di Palo, Prateek Singhi, and Bilal Fadlallah. Performance-guided llm knowledge distillation for efficient text classification at scale, 2024.

[174] Shreyansh Padarha. Enhancing reasoning capabilities in slms with reward guided dataset distillation, 2025.

[175] Anshumann, Mohd Abbas Zaidi, Akhil Kedia, Jinwoo Ahn, Taehwak Kwon, Kangwook Lee, Haejun Lee, and Joohyung Lee. Sparse logit sampling: Accelerating knowledge distillation in llms, 2025.

[176] Ziyang Ma, Qingyue Yuan, Linhai Zhang, and Deyu Zhou. Slow tuning and low-entropy masking for safe chain-of-thought distillation, 2025.

[177] Zhiheng Ma, Anjia Cao, Funing Yang, Yihong Gong, and Xing Wei. Curriculum dataset distillation, 2025.

[178] Jinyin Chen, Xiaoming Zhao, Haibin Zheng, Xiao Li, Sheng Xiang, and Haifeng Guo. Robust knowledge distillation based on feature variance against backdoored teacher model, 2024.

[179] Shuoxi Zhang, Zijian Song, and Kun He. Neural collapse inspired knowledge distillation, 2024.

[180] Xiao Cui, Mo Zhu, Yulei Qin, Liang Xie, Wengang Zhou, and Houqiang Li. Multi-level optimal transport for universal cross-tokenizer knowledge distillation on language models, 2025.

[181] Chen Zhang, Qiuchi Li, Dawei Song, Zheyu Ye, Yan Gao, and Yan Hu. Towards the law of capacity gap in distilling language models, 2025.

[182] Xinfeng Wang, Jin Cui, Yoshimi Suzuki, and Fumiyo Fukumoto. Rdrec: Rationale distillation for llm-based recommendation, 2025.

[183] Qianhan Feng, Wenshuo Li, Tong Lin, and Xinghao Chen. Align-kd: Distilling cross-modal alignment knowledge for mobile vision-language model, 2024.

[184] Yudi Shi, Shangzhe Di, Qirui Chen, and Weidi Xie. Enhancing video-llm reasoning via agent-of-thoughts distillation, 2025.

[185] Junjie Zhou, Ke Zhu, and Jianxin Wu. All you need in knowledge distillation is a tailored coordinate system, 2025.

[186] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[187] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[188] Qiang Zhang et al. Adalora: Adaptive low-rank adaptation, 2023.

[189] Dongho Kim et al. rslora: A rank stabilization scaling factor for lora, 2023.

[190] Yao Xu et al. Relora: High-rank training through low-rank updates. In *ICLR*, 2024.

[191] J. Chen et al. Longlora: Efficient fine-tuning of long-context llms, 2024.

[192] Xiang Liu et al. Bayesian lora: Bayesian low-rank adaptation, 2024.

[193] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: efficient low rank adaptation of large models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[194] Sheng-Yu Liu et al. Dora: Weight-decomposed low-rank adaptation, 2024.

[195] Rui Zhang et al. Autolora: Automatic rank search for peft, 2024.

[196] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[197] Kerim Büyükakyüz. Olora: Orthonormal low-rank adaptation of large language models, 2024.

[198] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. Delta-loRA: Fine-tuning high-rank parameters with the delta of low-rank matrices, 2024.

[199] Han Luo et al. Krona: Kronecker-factorized adapters for efficient tuning. In *CVPR*, 2024.

[200] Emanuele Zangrando, Francesco Rinaldi, Francesco Tudisco, et al. debora: Efficient bilevel optimization-based low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*. International Conference on Learning Representations, 2025.

[201] Yiping Ji, Hemanth Saratchandran, Cameron Gordon, Zeyu Zhang, and Simon Lucey. Efficient learning with sine-activated low-rank matrices. *arXiv preprint arXiv:2403.19243*, 2024.

[202] Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-pro: Are low-rank adapters properly optimized? In *The Thirteenth International Conference on Learning Representations*, 2025.

[203] Yibo Zhong, Jinman Zhao, and Yao Zhou. Low-rank interconnected adaptation across layers. *arXiv preprint arXiv:2407.09946*, 2024.

[204] Lin Mu, Xiaoyu Wang, Li Ni, Yang Li, Zhize Wu, Peiquan Jin, and Yiwen Zhang. Denselora: Dense low-rank adaptation of large language models, 2025.

[205] K. Zhang et al. Lplr: Matrix compression via randomized low rank and low precision factorization. In *NeurIPS (Poster)*, 2023.

[206] Yixiao Li, Yifan Yu, Chen Liang, Nikos Karampatziakis, Pengcheng He, Weizhu Chen, and Tuo Zhao. Loftq: LoRA-fine-tuning-aware quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[207] R. Saha et al. Caldera: Compressing llms using low rank and low precision decomposition, 2024.

[208] J. Park et al. Qdylora: Quantized dynamic low-rank adaptation, 2024.

[209] S. Sundrani et al. Low-rank compression via differentiable rank selection (llrc), 2025. ICLR 2025 under review.

[210] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression, 2025.

[211] Zhenzhong Lan et al. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*, 2020.

[212] Sinong Wang et al. Linformer: Self-attention with linear complexity. In *ICML*, 2020.

[213] Krzysztof Choromanski et al. Performer: Rethinking attention with performers. In *NeurIPS*, 2020.

[214] Yunyang Xiong et al. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *AAAI*, 2021.

[215] Tri Dao et al. Monarch: Expressive structured matrices for efficient transformers. In *ICML*, 2022.

[216] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2024.

[217] Samuel Horváth, Stefanos Laskaridis, Shashank Rajput, and Hongyi Wang. Maestro: uncovering low-rank structures via trainable decomposition. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[218] Seijin Kobayashi, Yassir Akram, and Johannes von Oswald. Weight decay induces low-rank attention layers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[219] Qihang Fan, Huaibo Huang, and Ran He. Breaking the low-rank dilemma of linear attention. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 25271–25280, 2025.

[220] Jingcheng Hu, Houyi Li, Yinmin Zhang, Zili Wang, Shuigeng Zhou, Xiangyu Zhang, Heung-Yeung Shum, and Daxin Jiang. Multi-matrix factorization attention. *arXiv preprint arXiv:2412.19255*, 2024.

[221] Amir Gholami et al. A survey of quantization methods for efficient Neural Network inference. *ACM Computing Surveys*, 2022.

[222] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[223] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019.

[224] Yu-Liang Zhan, Zhong-Yi Lu, Hao Sun, and Ze-Feng Gao. Over-parameterized student model via tensor decomposition boosted knowledge distillation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[225] Jinyang Guo, Jianyu Wu, Zining Wang, Jiaheng Liu, Ge Yang, Yifu Ding, Ruihao Gong, Haotong Qin, and Xianglong Liu. Compressing large language models by joint sparsification and quantization. In *Forty-first International Conference on Machine Learning*, 2024.

[226] Xiaoyi Qu, David Aponte, Colby Banbury, Daniel P Robinson, Tianyu Ding, Kazuhito Koishida, Ilya Zharkov, and Tianyi Chen. Automatic joint structured pruning and quantization for efficient neural network training and compression. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 15234–15244, 2025.

[227] Han Guo, Philip Greengard, Eric P Xing, and Yoon Kim. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. *arXiv preprint arXiv:2311.12023*, 2023.

[228] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

[229] Cheng Zhang, Jianyi Cheng, George A Constantinides, and Yiren Zhao. Lqer: Low-rank quantization error reconstruction for llms. *arXiv preprint arXiv:2402.02446*, 2024.

[230] Wenjin Ke, Zhe Li, Dong Li, Lu Tian, and Emad Barsoum. Dl-qat: Weight-decomposed low-rank quantization-aware training for large language models. *arXiv preprint arXiv:2504.09223*, 2025.

[231] Yoonjun Cho, Soeun Kim, Dongjae Jeon, Kyelim Lee, Beomsoo Lee, and Albert No. Assigning distinct roles to quantized and low-rank matrices toward optimal weight decomposition. *arXiv preprint arXiv:2506.02077*, 2025.

[232] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024.

[233] M. Zhang et al. Loraprune: Structured pruning meets low-rank. In *ACL Findings*, 2024.

[234] Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. Moe-i2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. *arXiv preprint arXiv:2411.01016*, 2024.

[235] Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Jawanpuria, and Bamdev Mishra. Sltrain: a sparse plus low-rank approach for parameter and memory efficient pretraining. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc.

[236] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models, 2023.

[237] Yanxi Li and Chengbin Du. Optimizing quantized diffusion models via distillation with cross-timestep error correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 18530–18538, 2025.

[238] TW Wang, T Wang, et al. Efficientvlm: Fast and accurate vision-language models via knowledge distillation and modal-adaptive pruning (2022). *arXiv preprint arXiv:2210.07795*, 4, 2022.

[239] Dong Chen, Ning Liu, Yichen Zhu, Zhengping Che, Rui Ma, Fachao Zhang, Xiaofeng Mou, Yi Chang, and Jian Tang. Epsd: Early pruning

with self-distillation for efficient model compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11258–11266, 2024.

[240] Maolin Wang, Yao Zhao, Jiajia Liu, Jingdong Chen, Chenyi Zhuang, Jinjie Gu, Ruocheng Guo, and Xiangyu Zhao. Large multimodal model compression via iterative efficient pruning and distillation. In *Companion Proceedings of the ACM Web Conference 2024*, pages 235–244, 2024.

[241] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation, 2024.

[242] James Pan and Guoliang Li. A survey of llm inference systems, 2025.

[243] Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. A survey on inference engines for large language models: Perspectives on optimization and efficiency, 2025.

[244] Tianqi Chen, Thierry Moreau, Ziheng Jiang, and et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, 2018.

[245] Jared Roesch, Steven Lyubomirsky, Marisa Kirisame, Logan Weber, Josh Pollock, Luis Vega, Ziheng Jiang, Tianqi Chen, Thierry Moreau, and Zachary Tatlock. Relay: A high-level compiler for deep learning, 2019.

[246] Chris Lattner, Jacques Pienaar, Mehdi Amini, and et al. Mlir: A compiler infrastructure for the end of moore's law. In *PLDI*, 2020.

[247] Tianqi Chen, Lianmin Zheng, Eddie Yan, and et al. Learning to optimize tensor programs. In *NeurIPS*, 2018.

[248] Lianmin Zheng, Eddie Yan, Yuwei Hu, and et al. Ansor: Generating high-performance tensor programs for deep learning. In *MLSys*, 2021.

[249] Junru Shao, Xiyou Zhou, Siyuan Feng, Bohan Hou, Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro Masuda, Cody Hao Yu, and Tianqi Chen. Tensor program optimization with probabilistic programs, 2022.

[250] Zhihao Jia and et al. Optimizing dnn computation with tvm auto-scheduling. *Communications of the ACM*, 2020.

[251] Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery.

[252] Tri Dao, Daniel Fu, Tianjun Zhao, et al. Flashattention-2: Faster attention with better parallelism and work partitioning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[253] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[254] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025.

[255] Deepak Narayanan and et al. Memory-efficient pipeline-parallel llm inference, 2021.

[256] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[257] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, and Man Wang. Glow: Graph lowering compiler techniques for neural networks, 2019.

[258] Jiarong Xing, Leyuan Wang, Shang Zhang, Jack Chen, Ang Chen, and Yibo Zhu. Bolt: Bridging the gap between auto-tuners and hardware-native performance, 2021.

[259] Zihua Wang, Ruibo Li, Haozhe Du, Joey Tianyi Zhou, Yu Zhang, and Xu Yang. Flash: Latent-aware semi-autoregressive speculative decoding for multimodal tasks, 2025.

[260] Yang Li and et al. Lemix: Unified scheduling for llm training and inference on multi-gpu systems, 2025.

[261] eLLM Authors. Elastic memory management framework for efficient llm serving, 2025.

[262] Siyuan Feng et al. TensorIR: An abstraction for automatic tensorized program optimization. In *ASPLOS*, 2023.

[263] Adrián Castelló et al. Experience-guided, mixed-precision matrix multiplication with Apache TVM for ARM processors. *The Journal of Supercomputing*, 2025.

[264] Guillermo Alaejos et al. Automatic generators for a family of matrix multiplication routines with Apache TVM, 2023.

[265] Yixin Song et al. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *OSDI*, 2024.

[266] Ryan Metcalfe et al. Enhancing local LLM performance through heterogeneous multi-device computing. In *IEEE MIPR*, 2024.

[267] Hitoaki Uetani and Yasuhiko Nakashima. Implementation and evaluation of LLM on a CGLA. In *CANDAR*, 2024.

[268] Keivan Alizadeh et al. LLM in a flash: Efficient LLM inference with limited memory. In *MLSys*, 2024.

[269] Yılmaz Küçük and Hakan Çevikalp. Classification of data corruption in microcontroller-based serial-optical communication with TensorFlow-Lite. In *SIU*, 2024.

[270] Jose Nunez-Yanez. Fused architecture for dense and sparse matrix processing in TensorFlow Lite. *IEEE Micro*, 2022.

[271] ONNX Community. ONNX format specification, 2025.

[272] Shih-Yang Liu et al. LLM-FP4: 4-bit floating-point quantized transformers, 2023.

[273] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018.

[274] Jerry Chee et al. QuIP: 2-bit quantization of large language models with guarantees. In *NeurIPS*, 2023.

[275] Josse Van Delm et al. HTVM: Efficient neural network deployment on heterogeneous TinyML platforms. In *DAC*, 2023.

[276] Xiangxiang Chu et al. MobileVLM: A fast, strong and open vision language assistant for mobile devices, 2023.

[277] Bin Lin et al. MoE-LLaVA: Mixture of experts for large vision-language models, 2024.

[278] Wei Dai and Daniel Berleant. Benchmarking contemporary Deep Learning hardware and frameworks: A survey of qualitative metrics. In *IEEE CogMI*, 2019.

[279] Hanrui Wang et al. Hardware-aware transformers for efficient NLP. In *ACL*, 2020.

[280] Jinrui Zhang et al. Heterogeneous parallel acceleration for edge intelligence systems: Challenges and solutions. *IEEE Consumer Electronics Magazine*, 2025.

[281] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12):1–37, March 2023.

[282] Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. KV cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4623–4648, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[283] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[284] Xiaoxin He, Fuzhao Xue, Xiaozhe Ren, and Yang You. Large-scale deep learning optimizations: A comprehensive survey, 2021.

[285] Haocheng Xi, Han Cai, Ligeng Zhu, Yao Lu, Kurt Keutzer, Jianfei Chen, and Song Han. Coat: Compressing optimizer states and activation for memory-efficient fp8 training, 2025.

[286] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016.

[287] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization, 2021.

[288] Benoit Steiner, Mostafa Elhoushi, Jacob Kahn, and James Hegarty. MODeL: Memory optimizations for deep learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32618–32632. PMLR, 23–29 Jul 2023.

[289] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2020.

[290] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. Zero-offload: Democratizing billion-scale model training, 2021.

[291] Jiangtao Wang, Jan Ebert, Oleg Filatov, and Stefan Kesselheim. Memory and bandwidth are all you need for fully sharded data parallel, 2025.

[292] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2025.

[293] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024.

[294] William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. Reducing transformer key-value cache size with cross-layer attention, 2024.

[295] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023.

[296] Guangxuan Xiao et al. StreamingLLM: Efficient streaming language models with attention sinks. In *ICLR*, 2024.

[297] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking, 2019.

[298] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, Yu Wang, and Guohao Dai. Large language model inference acceleration: A comprehensive hardware perspective, 2025.

[299] Yufeng Gu, Alireza Khadem, Sumanth Umesh, Ning Liang, Xavier Servot, Onur Mutlu, Ravi Iyer, and Reetuparna Das. Pim is all you need: A cxl-enabled gpu-free system for large language model inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, page 862–881. ACM, March 2025.

[300] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[301] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *SIGARCH Comput. Archit. News*, 45(2):27–40, June 2017.

[302] Yanbiao Liang, Huihong Shi, Haikuo Shao, and Zhongfeng Wang. Accllm: Accelerating long-context llm inference via algorithm-hardware co-design, 2025.

[303] Pietro Bartoli, Christian Veronesi, Andrea Giudici, David Siorpaes, Diana Trojaniello, and Franco Zappa. Benchmarking energy and latency in tinyml: A novel method for resource-constrained ai, 2025.

[304] Georgi Gerganov and Contributors. llama.cpp: Port of facebook's llama model in c/c++. https://github.com/ggerganov/llama.cpp, 2023. Accessed: 2025-10-03.

[305] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254, 2016.

[306] Mayank Arya and Yogesh Simmhan. Understanding the performance and power of llm inferencing on edge accelerators, 2025.

[307] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Fast on-device llm inference with npus, 2024.

[308] Intel. Supported devices — openvino documentation (2024), 2024. Accessed: 2025-10-03.

[309] Thanaphon Suwannaphong, Ferdian Jovan, Ian Craddock, and Ryan McConville. Optimising tinyml with quantization and distillation of transformer and mamba models for indoor localisation on edge devices, 2024.

[310] Jinendra Malekar, Peyton Chandarana, Md Hasibul Amin, Mohammed E. Elbtity, and Ramtin Zand. Pim-llm: A high-throughput hybrid pim architecture for 1-bit llms, 2025.

[311] Matteo Carnelos, Francesco Pasti, and Nicola Bellotto. Microflow: An efficient rust-based inference engine for tinyml, 2025.

[312] Mingqiang Huang, Ao Shen, Kai Li, Haoxiang Peng, Boyu Li, Yupeng Su, and Hao Yu. Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models, 2025.

[313] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. Flightllm: Efficient large language model inference with a complete mapping flow on fpgas, 2024.

[314] Chenyang Yin, Zhenyu Bai, Pranav Venkatram, Shivam Aggarwal, Zhaoying Li, and Tulika Mitra. Tereffic: Highly efficient ternary llm inference on fpga, 2025.

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

[316] Renjie Wei, Songqiang Xu, Linfeng Zhong, Zebin Yang, Qingyu Guo, Yuan Wang, Runsheng Wang, and Meng Li. Lightmamba: Efficient mamba acceleration on fpga with quantization and hardware co-design, 2025.

[317] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. Mlperf tiny benchmark, 2021.

[318] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus, 2018.

[319] Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators, 2024.

[320] Mumuksh Tayal and Yogesh Simmhan. Evaluating multi-instance dnn inferencing on multiple accelerators of an edge device, 2025.

[321] Christoforos Kachris. A survey on hardware accelerators for large language models. *Applied Sciences*, 15(2):586, January 2025.

[322] Nikoletta Koilia and Christoforos Kachris. Hardware acceleration of llms: A comprehensive survey and comparison, 2024.

[323] SiYoung Jang and Roberto Morabito. Edge-first language model inference: Models, metrics, and tradeoffs, 2025.

[324] Roberto Morabito and SiYoung Jang. Smaller, smarter, closer: The edge of collaborative generative ai, 2025.

[325] Zixu Hao, Huiqiang Jiang, Shiqi Jiang, Ju Ren, and Ting Cao. Hybrid slm and llm for edge-cloud collaborative inference. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*, EdgeFM '24, page 36–41, New York, NY, USA, 2024. Association for Computing Machinery.

[326] Pengyan Zhu and Tingting Yang. Ce-lslm: Efficient large-small language model inference and communication via cloud-edge collaboration, 2025.

[327] Divya Jyoti Bajpai and Manjesh Kumar Hanawal. Distributed inference on mobile edge and cloud: An early exit based clustering approach, 2024.

[328] Zhiyuan Wu, Sheng Sun, Yuwei Wang, Min Liu, Bo Gao, Jinda Lu, Zheming Yang, and Tian Wen. Recursive offloading for llm serving in multi-tier networks, 2025.

[329] Jiaming Xu, Jiayi Pan, Yongkang Zhou, Siming Chen, Jinhao Li, Yaoxiu Lian, Junyi Wu, and Guohao Dai. Specee: Accelerating large language model inference with speculative early exiting, 2025.

[330] Yeshwanth Venkatesha, Souvik Kundu, and Priyadarshini Panda. Fast and cost-effective

speculative edge-cloud decoding with early exits, 2025.

[331] Tian Wu, Liming Wang, Zijian Wen, Xiaoxi Zhang, Jingpu Duan, Xianwei Zhang, and Jinhang Zuo. Accelerating edge inference for distributed moe models with latency-optimized expert placement, 2025.

[332] Zheming Yang, Yunqing Hu, Sheng Sun, and Wen Ji. Ec2moe: Adaptive end-cloud pipeline collaboration enabling scalable mixture-of-experts inference, 2025.

[333] Akash Dhasade, Anne-Marie Kermarrec, Erick Lavoie, Johan Pouwelse, Rishi Sharma, and Martijn de Vos. Practical federated learning without a server. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, EuroMLSys '25, page 1–11. ACM, March 2025.

[334] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai, 2021.

[335] Boris Radovič, Marco Canini, Samuel Horváth, Veljko Pejović, and Praneeth Vepakomma. Towards a unified framework for split learning. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, EuroMLSys '25, page 183–191, New York, NY, USA, 2025. Association for Computing Machinery.

[336] Qiang Duan, Shijing Hu, Ruijun Deng, and Zhihui Lu. Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions. *Sensors*, 22(16):5983, August 2022.

[337] Arian Raje, Baris Askin, Divyansh Jhunjhunwala, and Gauri Joshi. Ravan: Multi-head low-rank adaptation for federated fine-tuning, 2025.

[338] Chenyu Jiang, Ye Tian, Zhen Jia, Shuai Zheng, Chuan Wu, and Yida Wang. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping, 2024.

[339] Cheng Qian, Hainan Zhang, Yongxin Tong, Hong-Wei Zheng, and Zhiming Zheng. Hyfedrag: A federated retrieval-augmented generation framework for heterogeneous and privacy-sensitive data, 2025.

[340] Camille Couturier, Spyros Mastorakis, Haiying Shen, Saravan Rajmohan, and Victor Rühle. Semantic caching of contextual summaries for efficient question-answering with language models, 2025.

[341] Alyssa Pinnock, Shakya Jayakody, Kawsher A Roxy, and Md Rubel Ahmed. Edgeprofiler: A fast profiling framework for lightweight llms on edge using analytical model, 2025.

[342] Rui Xie, Asad Ul Haq, Linsen Ma, Yunhua Fang, Zirak Burzin Engineer, Liu Liu, and Tong Zhang. Reimagining memory access for llm inference: Compression-aware memory controller design, 2025.

[343] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 8(12):9706–9716, 2021.

[344] Chenyang Shao, Tianxing Li, Chenhao Pu, Fengli Xu, and Yong Li. Agentstealth: Reinforcing large language model for anonymizing user-generated text, 2025.

[345] Jing Wang, Zheng Li, Lei Li, Fan He, Liyu Lin, Yao Lai, Yan Li, Xiaoyang Zeng, and Yufeng Guo. Principle-guided verilog optimization: Ip-safe knowledge transfer via local-cloud collaboration, 2025.

[346] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2024.

[347] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA, 2024. USENIX Association.

[348] Xiang Li, Zhenyan Lu, Dongqi Cai, Xiao Ma, and Mengwei Xu. Large language models on mobile devices: Measurements, analysis, and insights. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*, EdgeFM '24, page 1–6, New York, NY, USA, 2024. Association for Computing Machinery.

[349] Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. Melting point: Mobile evaluation of language transformers. In

*Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, page 890–907, New York, NY, USA, 2024. Association for Computing Machinery.

[350] Huifang Du, Xuejing Feng, Jun Ma, Meng Wang, Shiyu Tao, Yijie Zhong, Yuan-Fang Li, and Haofen Wang. Towards proactive interactions for in-vehicle conversational assistants utilizing large language models. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024.

[351] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee. K. Wong, Zhenguo Li, and Hengshuang Zhao. Drivegpt4-v2: Harnessing llm capabilities for enhanced closed-loop autonomous driving. In *Proceedings of CVPR*, 2025.

[352] Bufang Yang, Lixing He, Neiwen Ling, Zhenyu Yan, Guoliang Xing, Xian Shuai, Xiaozhe Ren, and Xin Jiang. Edgefm: Leveraging foundation model for open-set learning on the edge, 2023.

[353] Zhenbang Wu, Anant Dadu, Michael Nalls, Faraz Faghri, and Jimeng Sun. Instruction tuning large language models to understand electronic health records. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[354] Yiyun Zhao, Prateek Singh, Hanoz Bhathena, Bernardo Ramos, Aviral Joshi, Swaroop Gadiyaram, and Saket Sharma. Optimizing LLM based retrieval augmented generation pipelines in the financial domain. In Yi Yang, Aida Davani, Avi Sil, and Anoop Kumar, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 279–294, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[355] Shuyue Stella Li, Vidhisha Balachandran, Shangbin Feng, Jonathan S. Ilgen, Emma Pierson, Pang Wei Koh, and Yulia Tsvetkov. Mediq: Question-asking LLMs and a benchmark for reliable interactive clinical reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[356] Sunjun Kweon, Jiyoun Kim, Heeyoung Kwak, Dongchul Cha, Hangyul Yoon, Kwanghyun Kim, Jeewon Yang, Seunghyun Won, and Edward Choi. Ehrnoteqa: An llm benchmark for real-world clinical practice using discharge summaries, 2024.

[357] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.