Complex & Intelligent Systems (2025) 11:407
https://doi.org/10.1007/s40747-025-02019-z

SURVEY AND STATE OF THE ART

A review of state-of-the-art techniques for large language model compression

Pierre V. Dantas[1] · Lucas C. Cordeiro[1] · Waldir S. S. Junior[2]

Received: 14 November 2024 / Accepted: 1 July 2025 / Published online: 1 August 2025

Abstract
The rapid advancement of large language models (LLMs) has driven significant progress in natural language processing (NLP) and related domains. However, their deployment remains constrained by challenges related to computation, memory, and energy efficiency—particularly in real-world applications. This work presents a comprehensive review of state-of-the-art compression techniques, including pruning, quantization, knowledge distillation, and neural architecture search (NAS), which collectively aim to reduce model size, enhance inference speed, and lower energy consumption while maintaining performance. A robust evaluation framework is introduced, incorporating traditional metrics, such as accuracy and perplexity (PPL), alongside advanced criteria including latency-accuracy trade-offs, parameter efficiency, multi-objective Pareto optimization, and fairness considerations. This study further highlights trends and challenges, such as fairness-aware compression, robustness against adversarial attacks, and hardware-specific optimizations. Additionally, NAS-driven strategies are explored as a means to design task-aware, hardware-adaptive architectures that enhance LLM compression efficiency. Hybrid and adaptive methods are also examined to dynamically optimize computational efficiency across diverse deployment scenarios. This work not only synthesizes recent advancements and identifies open problems but also proposes a structured research roadmap to guide the development of efficient, scalable, and equitable LLMs. By bridging the gap between compression research and real-world deployment, this study offers actionable insights for optimizing LLMs across a range of environments, including mobile devices and large-scale cloud infrastructures.

Keywords Large language model compression · Knowledge distillation · Quantization · Pruning techniques · Neural architecture search · Resource-constrained environments · Scalable AI systems · Fairness in AI models · Robustness against adversarial attacks · Edge computing · Adaptive compression · Multi-objective optimization

Introduction

The development of large language models (LLMs), such as Bidirectional Encoder Representations from Transformers (BERT) [1], Generative Pre-trained Transformer (GPT) [2], and their variants, has revolutionized natural language processing (NLP), advancing tasks like sentiment analysis and question-answering. These deep neural networks (DNNs) with billions of parameters deliver high accuracy and generalization [3, 4]. However, their performance demands significant computational and memory resources, limiting use in environments like mobile devices and embedded systems [5–7]. This underscores the urgent need for model compression techniques that optimize both performance and efficiency.

Model compression reduces the computational and memory requirements of LLMs while maintaining their predictive accuracy [2, 4, 8]. Techniques like pruning, quantization, knowledge distillation and neural architecture search (NAS) are commonly used for this purpose [9–12]. Each method offers distinct optimization strategies but involves trade-offs between efficiency and accuracy. These challenges are partic-

✉ Pierre V. Dantas
pierre.dantas@postgrad.manchester.ac.uk ; pierre.dantas@gmail.com

Lucas C. Cordeiro
lucas.cordeiro@manchester.ac.uk

Waldir S. S. Junior
waldirjr@ufam.edu.br

1 Department of Computer Science, University of Manchester, Manchester, UK

2 Department of Electrical Engineering, Federal University of Amazonas (UFAM), Manaus, Brazil

ularly critical in real-time applications, where latency, energy use, and memory are key factors [13–15].

Historical context and real-world applications of LLM compression methods

The historical evolution of model compression techniques reveals a long-standing effort to optimize computational efficiency in machine learning (ML). Researchers have continuously sought innovative methods to reduce the size and complexity of models while maintaining or even enhancing their performance. From early approaches, such as pruning and quantization, to more recent advancements in knowledge distillation and NAS, the field has witnessed significant progress in balancing model efficiency with accuracy. These efforts stem from the growing need to deploy ML models on resource-constrained devices, such as mobile phones, embedded systems, and edge computing platforms.

Pruning, which involves removing less significant connections or neurons in DNNs, was first explored in the 1980s. Early works, such as "Optimal Brain Damage" [16] and "Optimal Brain Surgeon" [17], introduced second-order derivative-based methods to identify and remove redundant parameters. In the 2010s, modern magnitude-based pruning was developed, demonstrating that iterative weight pruning could significantly reduce model size while preserving accuracy [18]. More recently, dynamic pruning has been explored to adaptively adjust model sparsity based on input complexity, further improving efficiency in real-world applications [19].

Graph neural networks (GNNs) have gained significant attention in model compression research since the late 2010s, due to their ability to efficiently process non-Euclidean data structures [20, 21]. Recent studies in the early 2020s have explored pruning and quantization techniques specifically designed for GNNs, ensuring scalability while preserving essential structural information [22–24].

Quantization, a technique originating from signal processing, reduces the precision of DNN parameters to lower bit-width representations, reducing computational load and memory usage. Early works on this topic date back to the 1990s [25, 26]. In the 2010s, efforts shifted to hardware optimizations to accelerate processing speeds [27]. Subsequently, it was recognized that software optimizations and model compression techniques, including quantization, could reduce unnecessary parameters and computations while maintaining acceptable accuracy [28–30]. Quantization gained traction with binary and ternary quantization methods [31, 32], enabling drastic reductions in model size while preserving accuracy. More recently, in the 2020s, adaptive quantization strategies have been introduced to dynamically adjust precision levels based on hardware and application constraints, optimizing models for real-world deployments [33–35].

Knowledge distillation, introduced in 2015, presented a transformative approach to model compression by transferring knowledge from larger "teacher" models to smaller "student" models [36]. Initially designed to simplify DNNs for faster inference, it has become essential for deploying efficient models in applications requiring high accuracy with limited computational resources. For instance, knowledge distillation is widely used in edge computing, where smaller models enable real-time applications, such as real-time language translation and voice-activated virtual assistants [11, 37].

NAS, another major model compression advance, automated DNN design to optimize performance for specific constraints like low-latency and resource-constrained environments. Early ideas were inspired by evolutionary algorithms in 2002 [38]. However, differentiable architecture search (DARTS) in 2019, revolutionized the field by making architecture search more efficient [39]. These methods have enabled the discovery of efficient architectures, especially crucial for LLMs.

The *Lottery ticket hypothesis*, introduced in 2019, further advanced the field by showing that dense DNNs contain sparse, efficient subnetworks capable of achieving comparable performance [40]. This hypothesis challenges the traditional assumption that the size of DNNs is directly correlated with their learning capacity. The exploration of this hypothesis has opened possibilities for more efficient DNN designs, enabling the more recent development of models that are both efficient and high-performing, particularly beneficial for deployment in resource-constrained environments [9].

The 2020s have seen the rise of *hybrid approaches*, combining pruning, quantization, and distillation to further optimize LLMs. For instance, the effectiveness of parameter reduction in ALBERT, a lightweight variant of BERT, demonstrated that factorized embeddings and parameter sharing can achieve competitive results with fewer resources [1]. Additionally, the scalability of GPT-4 was achieved while incorporating efficiency considerations, enabling widespread adoption [2]. Hybrid compression strategies are now a key area of research, advancing low-power artificial intelligence (AI) for mobile, edge, and large-scale deployments [28, 34, 41, 42]. Figure 1 shows how key advancements in LLM compression techniques have evolved over time.

In summary, from early pruning to sophisticated LLM-tailored strategies, these advances have proven essential in bridging the gap between cutting-edge research and real-world deployment, powering transformative applications across domains.

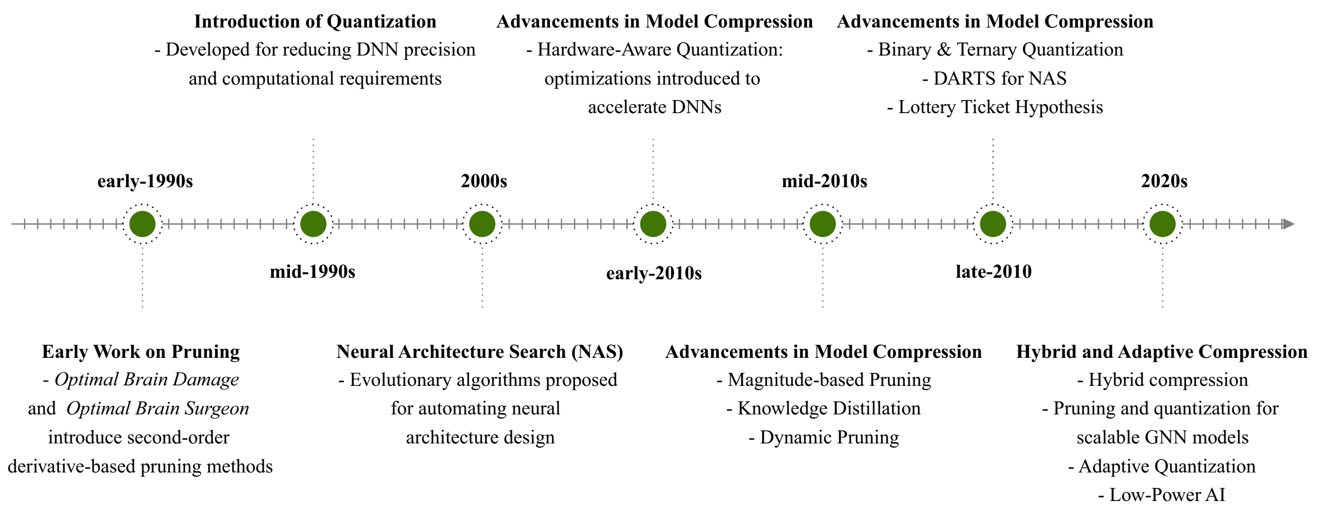
Introduction of Quantization
- Developed for reducing DNN precision and computational requirements
Advancements in Model Compression
- Hardware-Aware Quantization: optimizations introduced to accelerate DNNs
Advancements in Model Compression
- Binary & Ternary Quantization
- DARTS for NAS
- Lottery Ticket Hypothesis
early-1990s
mid-1990s
2000s
early-2010s
mid-2010s
late-2010
2020s
Early Work on Pruning
- Optimal Brain Damage and Optimal Brain Surgeon introduce second-order derivative-based pruning methods
Neural Architecture Search (NAS)
- Evolutionary algorithms proposed for automating neural architecture design
Advancements in Model Compression
- Magnitude-based Pruning
- Knowledge Distillation
- Dynamic Pruning
Hybrid and Adaptive Compression
- Hybrid compression
- Pruning and quantization for scalable GNN models
- Adaptive Quantization
- Low-Power AI

Fig. 1 Major milestones in LLM compression technology spanning four decades of innovation

Recent optimization achievements in LLMs

This section explores some of the most significant recent developments in LLM optimization. By integrating these cutting-edge optimizations, modern LLMs are becoming more efficient, adaptable, and suited to practical applications such as real-time conversational AI, code generation, edge computing, and privacy-preserving systems.

1. *Model sparsity*, a key optimization technique, has been employed in architectures like sparse mixture-of-experts, enabling the models, such as Switch Transformer and Generalist Language Model (GLaM), to achieve state-of-the-art performance while lowering computational overhead [43].
2. *Low-rank adaptation (LoRA)*, another major breakthrough, reduces the number of trainable parameters during fine-tuning, making the large-scale adaptation more efficient and cost-effective [15].
3. *Quantization techniques*, such as 4-bit and 8-bit quantization, have been instrumental in reducing memory consumption and improving inference speed without significant loss of accuracy [6, 44].
4. *Reinforcement learning (RL) with human feedback* has improved model alignment and mitigated harmful biases, as demonstrated in GPT-4 from OpenAI and the Claude models from Anthropic [45].
5. *Architectural efficiency improvements*, such as FlashAttention and transformer variants like Linformer and Performer, have facilitated the processing of longer sequences with reduced computational complexity [46].
6. *Retrieval-augmented generation (RAG)* further enhances model capabilities by integrating external memory, reducing the need for larger parameter counts while maintaining high factual accuracy [45].

These advancements are pivotal in shaping the next generation of AI applications, making LLMs more accessible and broadly applicable across diverse domains while mitigating the increasing computational demands associated with their training and deployment.

Recent advances in model compression have significantly influenced the development of modern LLMs including Claude by Anthropic, Large Language Model by Meta AI (LLaMA), and Pathways Language Model (PaLM) by Google. These models adopt distinct efficiency strategies—including pruning, quantization, distillation, and hybrid architectures—that balance performance with a reduced computational and memory requirements.

- *Claude by Anthropic* integrates RL with human feedback and a novel approach called Constitutional AI, which improves alignment and efficiency [47, 48]. Claude also employs hybrid symbolic-neural reasoning strategies that dynamically adjust computational depth depending on task complexity, enabling more efficient inference pipelines and lower average computation without compromising performance [49]. While specific pruning or quantization metrics have not been publicly disclosed, the system architecture suggests targeted efficiency enhancements for real-time and scalable applications.
- *LLaMA* has become a reference model for open-source LLM compression research. Studies have shown that structured pruning, such as depth removal of transformer layers, can enable substantial reductions in inference computational requirements while maintaining performance [50]. Further compression is achieved via quan-

tization methods like Quantized Low-Rank Adapter (QLoRA) [51] and Activation-Aware Weight Quantization (AWQ) [52], which enable 4-bit inference with minimal accuracy loss. Integrated pruning and distillation pipelines have demonstrated strong performance, with 4B-parameter models outperforming larger baseline models on standard benchmarks [53].

- *PaLM* emphasizes scaling efficiency through both architectural innovations and compression techniques. The PaLM 2 model demonstrates that smaller-scale models trained on larger datasets can outperform larger predecessors, offering higher computational efficiency [54]. Additionally, distillation strategies, such as "Distilling Step-by-Step", have transferred reasoning abilities from PaLM 540B to student models seven-hundred times smaller with comparable task performance [55]. These methods enable significant reductions in inference latency and memory requirements while preserving accuracy across logical reasoning benchmarks.

Benchmarks and evaluation platforms for LLM compression

For benchmarking compressed LLMs, various platforms and datasets offer specialized and general assessment suites, allowing the researchers to comprehensively evaluate model performance:

1. *Papers with Code (PwC)*: Provides standardized benchmarks for a range of language models, including their compressed variants, across a variety of NLP tasks. Each task has an associated leaderboard, and many compressed model entries to GitHub repositories for implementation [56].
2. *MLCommons*: Offers benchmark datasets and tasks for measuring the efficiency of compressed models, focusing on inference latency, accuracy, and memory efficiency across the NLP and other domains. The platform includes benchmarks that are designed for comparing LLMs after compression [57].
3. *Hugging Face Model Hub*: Provides a wide range of pre-trained models and their compressed variants, including evaluation metrics that cover tasks such as question answering, summarization, and text classification. The platform provides tools for comparing model sizes, inference latency, and accuracy [58].
4. *The Stanford Center for Research on Foundation Models (CRFM)*: Evaluates LLMs on multiple criteria, including fairness, robustness, and efficiency. The framework assesses a model's capabilities holistically, making it a critical resource for testing compressed models on diverse linguistic and reasoning tasks [59].
5. *Neural Magic's SparseZoo*: Offers pre-trained sparse models and benchmarks for pruning and quantization across various neural network architectures. The repository includes models optimized for inference on resource-constrained edge devices, making it ideal for evaluation of compressed LLMs [60].
6. *DAWNBench by Stanford*: Focuses on model training and inference latency, especially valuable for latency-sensitive applications. The benchmark includes both models and compression methods that are optimized through compression techniques [61].

Several platforms have been developed to benchmark LLMs, NLP tasks, and holistic evaluation methods such as the holistic evaluation of language models (HELM), developed by CRFM. Additionally, machine learning commons (MLCommons), the organization behind the Machine Learning Performance (MLPerf) benchmark suite, provides standardized tests to evaluate the performance, efficiency, and scalability of ML models. Table 1 provides the systematic comparison of specialized benchmarking platforms to evaluate compressed LLMs.

Contributions and novel aspects of this research

This work provides a systematic review and analysis of state-of-the-art compression techniques for LLMs, GNNs, and architectures optimized via NAS, focusing on their deployment in diverse hardware environments and resource-constrained devices. The key contributions and novel aspects of this review include:

1. *Comprehensive coverage of techniques*: We provide a detailed examination of fundamental compression techniques, including pruning, quantization, knowledge distillation, adaptive truncation, and hybrid compression strategies, demonstrating their efficacy in reducing memory usage, computational overhead, and energy consumption while maintaining the task-specific performance.
2. *Integration of NAS for model optimization*: We analyze the NAS as a method for co-designing architectures, enabling task-aware and hardware-adaptive models that achieve Pareto-optimal trade-offs between accuracy and efficiency across LLMs and GNNs.
3. *Evaluation framework*: We develop an integrated evaluation framework that moves beyond traditional metrics, such as task accuracy and perplexity (PPL), to include the Latency-Accuracy Trade-Off (LAT), parameter efficiency, multi-objective Pareto optimization, and fairness. This framework enables rigorous comparison of compression techniques.
4. *Trends and challenges*: We identified key developments, including fairness-aware compression, robustness against

PwC [56]	Provides benchmarks and leaderboards for various models, including compressed LLMs, across NLP tasks	State-of-the-art benchmarks, leaderboard comparisons, GitHub links for implementations	NLP tasks, model compression
MLCommons MLPerf [57]	Benchmark datasets and tasks for measuring efficiency of compressed models	Inference latency, accuracy, memory utilization for LLM	Cross-domain benchmarks, including NLP
Hugging Face Model Hub [58]	Provides pre-trained and compressed models with evaluation benchmarks	Model size, inference latency, accuracy, task performance comparisons	Multiple tasks including question answering and summarization
CRFM's HELM [59]	Evaluates language models on efficiency, robustness, fairness	Holistic performance across tasks, focus on fairness and robustness	Evaluation criteria for LLM assessment
Neural Magic's SparseZoo [60]	Offers pre-trained sparse models and benchmarks for pruning and quantization	Inference optimization on edge devices, sparsity-aware benchmarks	A focus on edge device compatibility
DAWNBench by Stanford [61]	Benchmarks training and inference latency, useful for latency-sensitive applications	Inference latency, training efficiency, especially relevant for compressed models	An emphasis on real-time applications

adversarial attacks, and hardware-specific optimizations, as well as persistent challenges such as bias amplification, preserving generalization, and ensuring scalability across heterogeneous hardware environments.

5. *Task and environment adaptability*: We emphasize adaptive compression methods capable of dynamically balance performance and resource efficiency, enabling the deployment of both LLMs and GNNs across multiple tasks and deployment contexts, including mobile devices and cloud infrastructure.
6. *Outline for future research*: We present a strategic outlook that synthesizes recent advancements and proposes future directions for developing efficient, scalable, and fairness-aware AI systems. The outline emphasizes interdisciplinary collaboration and the integration of methodologies driven by NAS with traditional compression techniques.

Through an in-depth review of techniques, the development of a robust evaluation framework, and an analysis of trends and practical challenges, this work presents a well-structured synthesis of the development of efficient, scalable, and robust AI systems. The inclusion of NAS-driven optimization underscores this review's contribution to addressing the emerging demands of AI applications.

Structure of this work

This work provides a structured review of LLM compression, covering its goals, methods, and practical implications. It outlines core challenges and evaluation criteria, analyzes state-of-the-art techniques, and includes case studies, a future research agenda, and a discussion of trade-offs and open questions. Figure 2 illustrates the paper's structure and section interconnections.

Compression techniques for LLMs

Foundational methods

The need for efficient deployment of LLMs has motivated the development of compression techniques to reduce model size, inference latency, and memory requirements without compromising task performance. Core approaches include pruning, quantization, knowledge distillation, and NAS, each offering distinct benefits and trade-offs. These techniques are frequently combined to maximize overall efficiency across tasks and platforms [62]. This section explores their mechanisms and applications in LLMs.

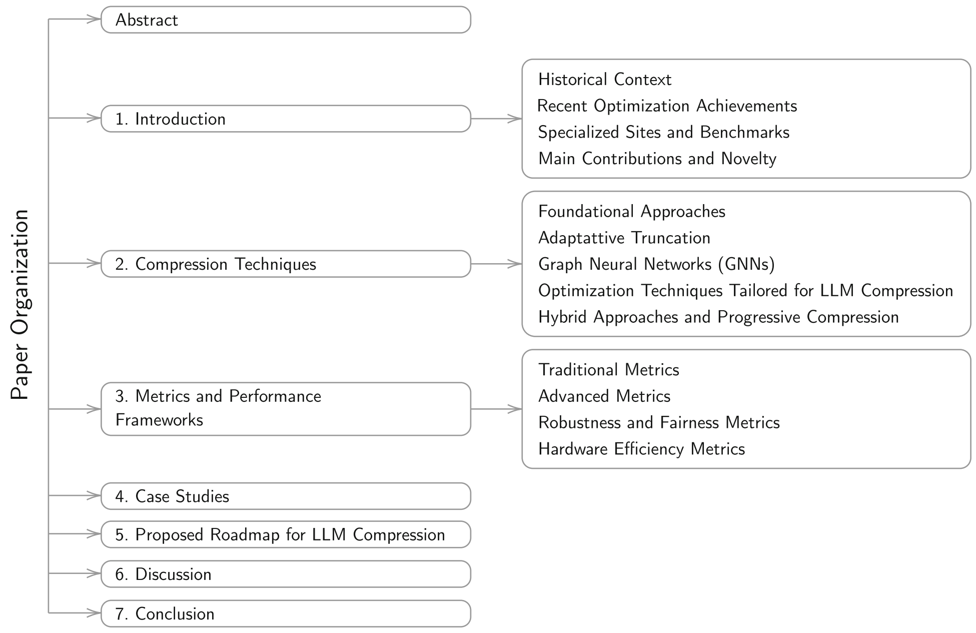
Paper Organization
Abstract
1. Introduction
Historical Context
Recent Optimization Achievements
Specialized Sites and Benchmarks
Main Contributions and Novelty
2. Compression Techniques
Foundational Approaches
Adaptattive Truncation
Graph Neural Networks (GNNs)
Optimization Techniques Tailored for LLM Compression
Hybrid Approaches and Progressive Compression
3. Metrics and Performance Frameworks
Traditional Metrics
Advanced Metrics
Robustness and Fairness Metrics
Hardware Efficiency Metrics
4. Case Studies
5. Proposed Roadmap for LLM Compression
6. Discussion
7. Conclusion

Fig. 2 Schematic overview of the paper structure, emphasizing the logical progression and thematic connections across sections

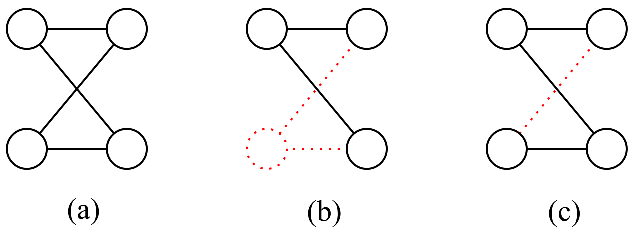

Fig. 3 Visualization of network pruning techniques. (**a**) Original network with full connectivity. (**b**) Node pruning, where a single node (indicated by a red dashed circle) and its associated connections are removed. (**c**) Edge pruning, in which specific connections (marked with red dashed lines) are selectively eliminated, resulting in a more efficient architecture

Pruning

Pruning reduces parameters or connections in a DNN to improve memory usage, inference latency, and computational efficiency while preserving task performance. By removing components deemed less critical—such as weights, neurons, or layers—based on criteria like magnitude or saliency, this technique enhances model deployment in resource-constrained environments [34, 62, 63]. Figure 3 illustrates how a dense network is transformed into a sparse one by removing both connections and nodes, resulting in a more efficient architecture tailored for practical deployment.

Pruning techniques are generally classified as structured or unstructured, each offering distinct trade-offs and suited to specific deployment scenarios.

- *Structured pruning*: This approach removes entire components, such as attention heads, neurons, or layers, based on their contribution to task performance. Structured pruning simplifies model architecture and enhances compatibility with hardware-accelerated operations such as matrix multiplication. For example, pruning underperforming attention heads in Transformers reduces memory requirements and inference latency. This makes it well suited for deployment on resource-constrained hardware [9].
- *Unstructured pruning*: Removes individual weights within layers, resulting in sparse weight matrices. Although it often achieves greater parameter reduction than structured pruning, the resulting irregular sparsity patterns present challenges for hardware acceleration. Recent advances in sparse matrix representations and specialized libraries help mitigate these limitations, enabling performance gains in resource-constrained tasks. Techniques like iterative magnitude pruning and the Lottery Ticket Hypothesis have shown significant parameter reductions while maintaining task performance [40, 62].

Equation (1) formulates pruning as an optimization problem, where the objective is to identify a pruned weight set W_p that minimizes the deviation in loss relative to the original model W. The function $\mathcal{L}(\cdot)$ denotes the loss function, and the norm captures the performance difference after pruning. If the loss is scalar, the absolute difference is used; otherwise, an appropriate norm (such as the L2 norm) quantifies the deviation over a batch or full dataset [64].

$$W_p^* = \arg\min_{W_p} \|\mathcal{L}(W) - \mathcal{L}(W_p)\|_2 \tag{1}$$

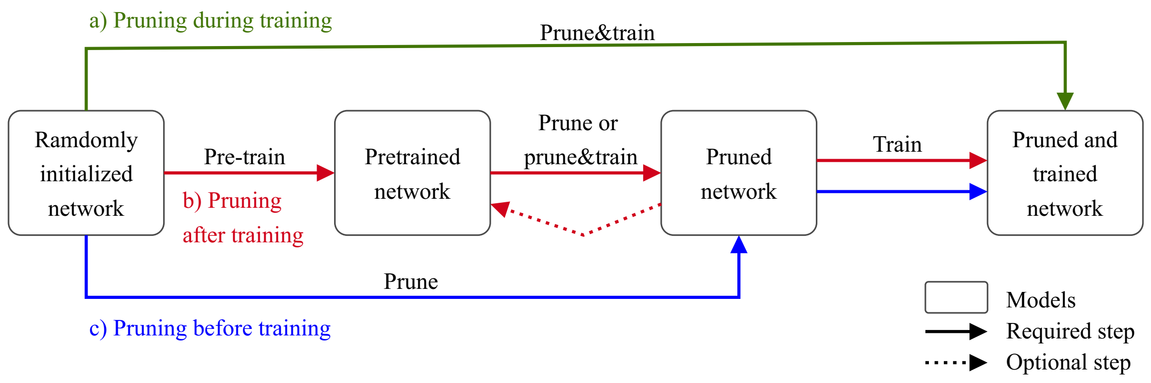
a) Pruning during training
Prune&train
Ramdomly initialized network
Pre-train
Pretrained network
Prune or prune&train
Pruned network
Train
Pruned and trained network
b) Pruning after training
Prune
c) Pruning before training
Models
Required step
Optional step

Fig. 4 Illustration of typical static pruning pipelines in DNNs. (**a**) Pruning during training (green): pruning and training are performed simultaneously. (**b**) Pruning after training (red): the network is pre-trained, then pruned, and optionally fine-tuned. (**c**) Pruning before training (blue): the network is pruned at initialization and then trained

Pruning techniques are increasingly employed in LLMs to reduce computational overhead while maintaining task performance. For example, structured pruning has been applied to attention heads in Transformer architectures, while unstructured approaches such as magnitude pruning have achieved high sparsity levels in compute-intensive settings [46].

Figure 4 shows three pruning pipelines: (a) pruning during training, where pruning and training are performed simultaneously, followed by fine-tuning, (b) pruning after training, where the network is pre-trained, pruned, and optionally fine-tuned, and c) pruning before training, where the network is pruned at initialization based on predefined criteria and then trained from scratch [63].

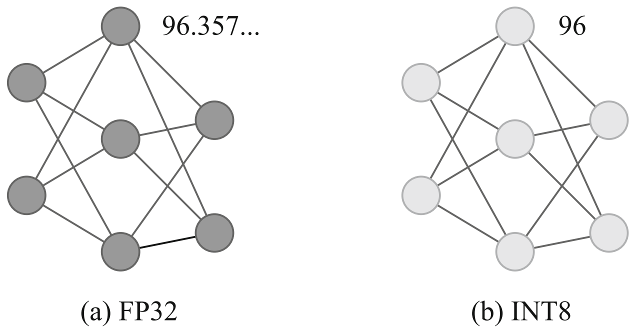

Fig. 5 Quantization of a LLM. (**a**) The original full-precision model (FP32), characterized by high memory requirements and computational overhead. (**b**) The quantized model (INT8), exhibiting reduced memory requirements, enhanced inference speed, and improved energy efficiency, while preserving comparable task performance

Quantization

Quantization reduces the precision of model parameters by representing weights and activations using reduced-precision formats such as 8-bit, 4-bit, or 1-bit (binary), in place of the standard 32-bit single-precision floating-point format [6]. This technique significantly reduces both memory requirements and computational overhead, making it particularly suitable for deployment on memory-constrained devices and specialized hardware accelerators such as graphics processing units (GPUs) and tensor processing units (TPUs) [10].

Figure 5 illustrates the transition from a full-precision DNN, depicted on the left as a 32-bit floating-point (FP32) model, to a quantized version, shown on the right as an 8-bit integer (INT8). The FP32 model requires more memory and computational resources, whereas the INT8 model is considerably more compact. Despite the reduction in precision, the quantized model is designed to preserve comparable accuracy, making it highly effective for deployment on resource-constrained hardware. This transformation illustrates the trade-off between model size, computational efficiency, and predictive performance in DNN quantization.

Quantization affects both the weights and activations of DNNs, influencing overall efficiency and performance. While weight quantization is well-studied, activation quantization introduces additional challenges due to the dynamic range and statistical distribution of activations [25, 44]. Accurate activation quantization is critical to reducing memory requirements while preserving task performance. Inaccurate activation quantization, particularly in early layers, can impair generalization and degrade downstream task performance [30].

Equation (2) defines the quantization error E_q as the discrepancy between the original full-precision weight matrix W and its quantized counterpart W_q. This formulation uses the matrix 2-norm to capture the maximum deviation introduced by quantization, corresponding to the largest singular value of the difference matrix $W - W_q$. The norm serves as a measure of how significantly the quantized representation diverges from the original in terms of spectral properties.

This error metric is particularly relevant when assessing the stability and robustness of compressed models, especially in contexts where spectral characteristics influence learning dynamics or downstream task performance.

$$E_q = \|W - W_q\|_2 \tag{2}$$

Quantization methods are commonly divided into three categories—uniform, non-uniform, and mixed-precision—each offering distinct strategies for balancing model efficiency and performance:

- *Uniform quantization*: Maps values to equally spaced levels, using fixed bit-widths across parameters or layers. This approach simplifies implementation and reduces computational overhead. It enables substantial memory savings with limited impact on task performance, especially in models like BERT and GPT-2. Post-quantization fine-tuning is commonly employed to adapt models to downstream tasks and recover potential accuracy loss [6, 10].
- *Non-uniform quantization*: Uses variable bit-widths for parameters and activations based on sensitivity to quantization. Higher-precision representations are preserved in critical layers, while less sensitive components may be quantized more aggressively. techniques like quantization-aware training (QAT) and post-training quantization (PTQ) enable selective quantization. QAT simulates quantization effects during training, allowing the model to adapt to low-precision representations and reducing performance degradation, while PTQ applies quantization after training using calibration datasets to minimize accuracy loss [10].
- *Mixed-precision quantization*: Extends non-uniform quantization by enabling weights and activations to be quantized at different bit-widths across layers within the same model based on their sensitivity. For instance, activations in high-variance layers may retain 8-bit precision, while less critical layers can be quantized to 4 bits or lower. This approach achieves a balance between memory efficiency and task performance, making it particularly effective for deployment in resource-constrained environments. [65].

Figure 6 illustrates the distinction between uniform and non-uniform quantization. (a) Uniform quantization, in which values are mapped evenly across the range, resulting in consistent step sizes. This approach is computationally efficient but can introduce higher quantization error for outlier values. (b) Non-uniform quantization, where step sizes vary to allocate more precision to frequently occurring values or critical ranges. This method balances accuracy and efficiency, making it particularly effective for datasets with non-uniform distributions.

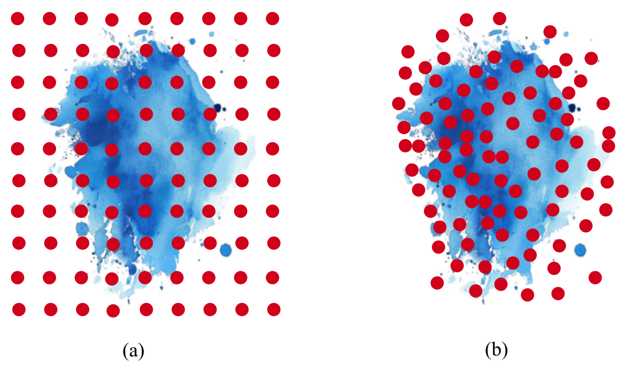

Fig. 6 Comparison of uniform and non-uniform quantization. (**a**) Uniform quantization maps values evenly across the representable range, offering implementation simplicity but potentially introducing higher quantization error for outlier values. (**b**) Non-uniform quantization employs variable step sizes to allocate greater precision to critical regions or frequently occurring values

Equation (3) illustrates the overall optimization problem for quantizing weights. This formulation seeks to minimize the degradation in task performance resulting from quantization by identifying the optimal quantized weight configuration W_q^*. The function $\mathcal{L}(\cdot)$ denotes the loss associated with the original model W and its quantized counterpart W_q, respectively. The norm captures the difference in performance, which can be interpreted as the deviation in task-specific loss introduced by quantization. Depending on the context, the norm may refer to an absolute value (for scalar loss) or an L_2 norm (for aggregated loss across data samples).

$$W_q^* = \arg\min_{W_q} \|\mathcal{L}(W) - \mathcal{L}(W_q)\| \tag{3}$$

Quantization is widely employed to optimize LLMs for deployment in resource-constrained environments. Quantized variants of models such as BERT and GPT-2 enable faster inference and lower energy consumption in cloud-based services like Google Translate and virtual assistants including Alexa and Siri. On mobile and edge devices, quantization enables efficient execution of LLMs on hardware with limited memory and computational capacity, including smartphones and internet of things (IoT) devices [7, 66]. Beyond consumer applications, quantization is used in recommendation systems and financial forecasting, where minimizing latency and operational cost without compromising accuracy is critical. In healthcare, it facilitates high-performance diagnostics on edge devices for real-time medical imaging and patient data analysis [67].

Knowledge distillation

Knowledge distillation compresses models by training a smaller "student" model to replicate the behavior of a larger "teacher" model. By transferring essential knowledge, the student achieves comparable performance with fewer parameters, making it effective for resource-constrained deployments [11, 68]. Rather than learning solely from ground-truth labels, the student model is trained to match the soft targets or output distributions of the teacher, promoting better generalization [69].

Figure 7 illustrates how knowledge distillation transfers learned information from a larger, complex teacher model to a smaller, more efficient student model by combining soft targets with supervised learning.

Knowledge distillation combines two complementary learning signals, as formalized in Eq. (4). The first component, $\mathcal{L}_{\text{CE}}$, is the standard cross-entropy loss, which guides the student model to learn from the ground-truth labels. The second component, $\mathcal{L}_{\text{KL}}$, corresponds to the Kullback–Leibler divergence, and encourages the student to mimic the softened output distribution of the teacher model. This softening is achieved through temperature scaling, where the logits from the teacher are divided by a temperature parameter τ, resulting in a smoother, less peaked distribution that is easier for the student to approximate. The hyperparameter λ controls the relative importance of the distillation term in the overall loss. The total knowledge distillation loss is expressed as:

$$\mathcal{L}_{\text{KD}} = \mathcal{L}_{\text{CE}} + \lambda\tau^2\mathcal{L}_{\text{KL}} \tag{4}$$

By integrating both hard labels and soft teacher predictions, the training objective enables the student model to benefit from direct supervision while capturing the generalization capacity of the larger teacher model. When combined with pruning or quantization, this approach yields resource-efficient architectures that closely approximate the performance of the original model while preserving task accuracy [34, 70].

Knowledge distillation is widely employed to facilitate the deployment of efficient models in resource-constrained environments, with notable applications including:

- *Speech recognition*: Facilitates real-time applications such as virtual assistants and automated transcription services [69].
- *Computer vision*: Enhances efficiency in tasks such as object detection and image classification on mobile and edge devices [37].
- *NLP*: Distilled models such as DistilBERT offer lightweight alternatives to larger architectures, enabling faster inference in applications like chatbots and question-answering systems [69].
- *Recommendation systems*: Reduces computational overhead in large-scale user-item interaction models while maintaining the accuracy of personalized content delivery, such as product suggestions and media recommendations [69, 71].

Neural architecture search (NAS)

NAS automates the design of neural network architectures by exploring configuration spaces to optimize performance under task-specific constraints [12, 72, 73]. In contrast to manual architecture design, which relies on expert-driven heuristics, NAS employs techniques such as RL [74], evolutionary algorithms [75], and gradient-based optimization [75] to efficiently identify high-performing architectures. NAS can be integrated with model compression techniques—such as as pruning, quantization, and knowledge distillation—to generate compact and efficient models. Additionally, weight-sharing strategies in One-Shot NAS enable structured pruning with minimal retraining overhead [76].

NAS operates by defining a search space that encompasses potential architectural components, such as layer types, connectivity patterns, and hyperparameters. A search strategy is then employed to explore this space, and candidate architectures are evaluated using a predefined objective function, such as task accuracy, inference latency, or energy efficiency. This iterative process enables NAS to progressively refine and converge toward optimal architectures. Recent advancements in differentiable NAS methods—such as DARTS—have significantly reduced the computational cost associated with traditional approaches, making the optimization process faster and more scalable [39].

Figure 8 illustrates the process of NAS, which automates the discovery of high-performing DNN architectures by replacing manual design with an iterative search-and-evaluation framework [12].

When compressing LLMs, the objective is to identify an optimal neural architecture that balances inference speed, task accuracy, and memory efficiency. This is commonly achieved through NAS, which operates as a two-stage optimization process:

- *Finding the best architecture* (α): The search process aims to identify the optimal architecture parameters α by minimizing the validation loss $\mathcal{L}$. However, because the performance of a given architecture depends on its associated model weights w, training is required at each evaluation step.
- *Training the model for each architecture*: Given a fixed architecture α, the corresponding model weights w are

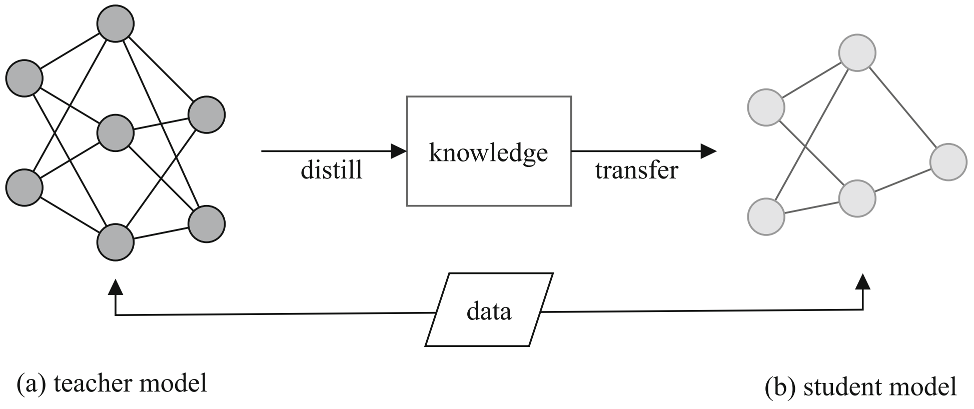

Fig. 7 Knowledge distillation framework. (**a**) The teacher model, a larger and more complex DNN, learns from the training data. (**b**) The student model, a smaller and more efficient network, is trained to replicate the teacher's behavior by receiving distilled knowledge through soft targets and supervision

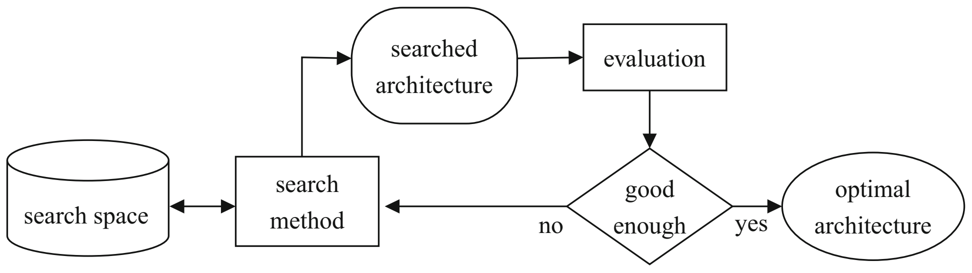

Fig. 8 Overview of the NAS process. The search space defines the set of candidate architectures, and the search algorithm iteratively samples and evaluates these candidates. The evaluation module assesses each architecture based on predefined performance metrics, and the process continues until an optimal architecture is identified

optimized by minimizing the training loss $\mathcal{L}$. This ensures that each candidate architecture is evaluated based on its fully trained performance.

The architecture search process is formulated as a bi-level optimization problem, as shown in Eq. (5). This formulation aims to identify the optimal architecture parameters α that minimize the validation loss, while accounting for the fact that each architecture must be evaluated using its corresponding trained weights $w^*(\alpha)$. The outer objective seeks the best architecture based on performance metrics, whereas the inner objective computes the optimal weights for a given architecture by minimizing the training loss. The function $\mathcal{L}(\cdot,\cdot)$ represents the loss used to evaluate performance, incorporating both architecture and weight dependencies. This hierarchical optimization captures the coupling between architecture design and model training in NAS.

$$\min_{\alpha} \mathcal{L}(w^*(\alpha), \alpha) \quad \text{subject to} \quad w^*(\alpha) = \arg\min_{w} \mathcal{L}(w, \alpha) \tag{5}$$

Evaluating architectures optimized through NAS requires a comprehensive perspective that goes beyond conventional metrics such as accuracy or PPL. Key considerations include latency-accuracy trade-offs, hardware adaptability, fairness, and energy efficiency. Furthermore, compared to manually designed architectures, NAS-generated models often require additional hyperparameter tuning to enhance robustness against adversarial perturbations and to satisfy deployment constraints.

Different NAS algorithms present distinct trade-offs in terms of search efficiency, model quality, and computational cost:

- *RL-based NAS:*: Methods such as NASNet explore the architecture search space using RL to maximize a reward function, typically incorporating task accuracy and constraints such as latency. While these approaches can produce high-quality architectures, they are computationally intensive and less practical for scaling to LLMs [74].
- *DARTS*: This approach relaxes discrete architectural choices into continuous variables, enabling gradient-based optimization and significantly reducing search time. However, it may overfit to the super-network used during search, leading to sub-optimal performance when the final architecture is retrained independently [39].
- *One-Shot NAS*: Methods such as ProxylessNAS train a super-network that encompasses all candidate sub-networks. These sub-networks are sampled and evaluated without retraining, enabling efficient and hardware-aware architecture optimization [67].
- *Evolutionary NAS*: Methods such as AmoebaNet apply iterative refinement to evolve candidate architectures over successive generations. These approaches are typically more robust to local optima but are generally slower and more computationally demanding than differentiable and One-Shot methods [38].

Table 2 Comparison of NAS methodologies, highlighting trade-offs in computational efficiency, scalability, and optimization capability

Algorithm	Advantages	Limitations
RL-based NAS [74]	Produces high-quality architectures; suitable for small-scale tasks	High computational requirements; limited scalability
Differentiable NAS [39]	Enables fast search; scalable to larger models	Prone to overfitting on the super-network
One-Shot NAS [67]	Low computational overhead; effective for hardware-aware optimization	Limited granularity in trade-offs control
Evolutionary NAS [38]	Enables robust exploration; delivers competitive performance	Slow search process; often impractical for LLMs

Table 2 compares NAS methodologies, highlighting their respective strengths and limitations across different application scenarios.

Moreover, NAS has been successfully applied to optimize LLMs across a range of real-world applications, demonstrating its ability to generate task-specific, hardware-efficient architectures while maintaining competitive performance:

- *Mobile and edge devices*: ProxylessNAS optimizes neural architectures for latency and memory efficiency, facilitating deployment on resource-constrained hardware platforms [5].
- *Cloud services*: NASNet and related approaches balance computational efficiency and model performance, making them suitable for large-scale cloud-based inference and service delivery [74].
- *Recommendation systems*: Architectures optimized through NAS improve throughput and reduce latency, enabling efficient real-time recommendation delivery [77].
- *Healthcare AI*: In medical imaging and diagnostics, NAS is used to design lightweight models that deliver high accuracy and fast inference [6].

Despite its advantages, NAS for LLM compression presents several notable challenges. A primary concern is scalability: evaluating thousands of candidate architectures for models with billions of parameters incurs substantial computational demand. Moreover, search strategies may introduce biases that favor specific datasets or tasks, limiting the generalizability of the resulting architectures. Addressing these limitations requires the development of more efficient search heuristics and fairness-aware optimization strategies.

Future research should prioritize the creation of hardware-aware NAS methodologies that tailor architectures for next-generation accelerators, such as TPUs, field-programmable gate array (FPGA), and low-power chips specialized for AI. Additionally, advancements in self-supervised NAS could help mitigate computational demands, enabling the design of more adaptive and efficient LLMs that dynamically adjust to deployment constraints.

Comparative evaluation of core compression strategies

To support the preceding explanation of foundational compression techniques, Table 3 (p. 15) presents a condensed overview comparing pruning, quantization, knowledge distillation, and NAS. It synthesizes key performance metrics, such as accuracy retention, compression ratios, inference speed, and energy efficiency, alongside implementation considerations including algorithmic complexity, hardware dependencies, and training requirements. The table also outlines the typical use cases and limitations of each method, offering a practical reference for selecting appropriate strategies based on specific application goals and deployment constraints.

Structure-aware adaptation

In addition to established compression strategies, structural adaptation techniques offer an alternative approach by modifying the internal architecture or operational topology of LLMs. Rather than focusing solely on parameter-level reductions or training-based optimizations, these methods reshape the model's computational structure to improve efficiency and adaptability across deployment contexts [4, 78]. This section explores two prominent approaches within this category: adaptive truncation, which reduces sequence length dynamically to minimize redundant computation, and GNNs, which provide a flexible, non-sequential representation for enhancing model expressivity and compression. Together, these methods reflect a shift toward structure-aware compression, expanding the design space for efficient LLMs.

Table 3 Comprehensive analysis of LLM compression techniques, integrating performance metrics, implementation considerations, and optimal use cases across scenarios

Aspect	Related subject	Pruning	Quantization	NAS	Knowledge distillation
Optimization technique	–	SparseGPT [12], Dynamic Sparsity (DynaBERT) [79], Embedding Layer Pruning [79]	QAT [79]	LORA [15], Memory-Efficient Fine-Tuning [6]	DistilBERT [6], Token Reduction via Adaptive Attention [34]
Key metrics	Accuracy retention	85–95% (S) 80–90% (U) [34, 63]	90–95% (8-bit) 80–90% (4-bit) [6]	95–98% [12]	85–95% depending on student size [80]
	Compression rate	2–5× (S) 5–10× (U) [63]	2–4× (8-bit) 4–8× (4-bit) [63]	1.5–3× [76]	3–6× [80]
	Inference speed	0.6–0.8× (S) 1.0–1.2× (U) [33]	0.3–0.5× [63]	Hardware-dependent	0.4–0.7× [24]
	Energy efficiency	1.3–1.8× (S) 1.1–1.4× (U) [81]	2.0–3.0× [35, 44]	1.2–1.5× [82]	1.5–2.0× [83]
Design aspects	Complexity	Medium	Low to medium	High	Medium to high
	Hardware dependency	Requires sparse tensor support	Minimal hardware dependency	Significant hardware dependency	Low hardware dependency
	Training requirements	Requires iterative fine-tuning	Can be applied during or after training	Requires extensive search phase	Requires a pretrained teacher model
	Hardware compatibility	Excellent for structured pruning. Requires sparse tensor support for unstructured pruning [12]	Excellent on modern accelerators with native framework support [84]	High on target-specific hardware [72]; requires significant computational resources [12]	Generally good compatibility; framework-independent deployment supported [11, 69]
Use cases	Best suited tasks	Feature extraction and models with high redundancy	Edge deployment, real-time inference, and battery-constrained environments	Hardware-specific optimization and performance-critical workloads	Task-specialized optimization, multi-task learning, and complex reasoning applications
	Challenges	Risk of performance degradation and intricate fine-tuning procedures	Accuracy loss on complex tasks and limitations due to hardware precision	High computational demands and extended search duration	Dependence on large teacher models and task-specific training requirements

(U) means unstructured pruning and (S) means structure pruning

Adaptive truncation

Adaptive truncation dynamically adjusts the sequence length or the number of active neurons within a layer during training and inference, thereby optimizing computational efficiency while preserving task performance. This technique is applicable to both fully connected and convolutional layers, selectively identifying and excluding less relevant neurons or input tokens based on task-specific relevance, thus reducing overall resource requirements [78].

For a given layer in a DNN, adaptive truncation selects a threshold k within the range $[N_{\min}, N]$, where N denotes the total number of neurons or tokens. Neurons with indices greater than k are excluded from computation, thereby reducing the effective size of the layer. The lower bound $N_{\min}$ ensures the preservation of baseline functionality, while neurons with indices approaching N are typically pruned due to their lower relevance [4].

This process introduces a controlled degree of stochasticity in the selection of neurons for exclusion—similar in spirit to dropout—but with a stronger emphasis on architectural optimization tailored to the target task. The resulting network dynamically adapts its size based on trade-offs between computational efficiency and predictive accuracy, making it particularly well-suited for deployment in resource-constrained environments, such as mobile platforms and edge devices.

In training scenarios, adaptive truncation can function as an effective regularization technique. By reducing the active set of neurons or sequence length, it introduces architectural variability during training, which helps mitigate overfitting. This approach is particularly beneficial in applications with highly variable input lengths, such as real-time translation and text summarization.

Equation (6) formalizes the adaptive truncation process as a constrained optimization problem. The objective is to identify the reduced sequence length L_r that minimizes computational cost $\mathcal{C}(L, L_r)$, subject to maintaining a minimum acceptable level of performance $\mathcal{P}_{\min}$. Here, L denotes the original input sequence length, while L_r is the optimized, truncated length. The cost function $\mathcal{C}(\cdot)$ captures computational resource usage—such as floating-point operations, latency, or memory—as a function of sequence length reduction. The performance function $\mathcal{P}(L_r)$ evaluates model accuracy (or another task-relevant metric) for a given L_r. The constraint $\mathcal{P}(L_r) \geq \mathcal{P}_{\min}$ ensures that truncation does not degrade task performance below an acceptable threshold, making this formulation particularly suitable for scenarios where resource-efficiency and task reliability must be jointly optimized.

$$\arg\min_{L_r}(\mathcal{C}(L, L_r)), \quad \text{subject to} \quad \mathcal{P}(L_r) \geq \mathcal{P}_{\min} \tag{6}$$

Figure 9 illustrates the adaptive truncation process, where input sequence lengths or active neurons are dynamically adjusted based on task requirements or input complexity. Each node represents a neuron in a dense layer. Red nodes denote inactive or removed neurons—either due to stochastic dropout or deterministic truncation—while gray nodes represent active neurons contributing to computation. Subfigure (a) illustrates dropout, which randomly deactivates neurons during training without permanently removing them. In contrast, subfigure (b) depicts truncation, where less relevant neurons are systematically excluded to improve computational efficiency while preserving the most informative ones.

While conceptually similar to dropout, adaptive truncation differs in several critical aspects:

- *Dropout*: Temporarily deactivates neurons at random during training, without altering the network's structure. This technique effectively averages over an ensemble of subnetworks to improve generalization.
- *Truncation*: Reduces the active size of the network either deterministically or probabilistically by applying thresholds to neuron or token activations. Unlike dropout, truncation directly modifies the computational graph, enabling explicit optimization of layer dimensions for deployment efficiency [72].

In fully connected DNNs, truncation is applied to hidden layers by progressively reducing the number of active neurons according to a probability distribution. Empirical studies have shown that this approach can effectively balance computational efficiency with predictive accuracy [81]. Despite its advantages, adaptive truncation presents several challenges when applied across multiple layers in deep architectures. One key challenge is threshold optimization, as the optimal truncation level is highly dependent on input variability—requiring fine-tuned, task-specific calibration. Another issue is the trade-off between efficiency and accuracy: overly aggressive truncation may result in the loss of critical contextual information, degrading performance in tasks that rely on long-range dependencies or full-sequence comprehension.

Graph-based approaches

GNNs have emerged as powerful tools for extending the capabilities of LLMs, particularly in areas such as knowledge integration, reasoning, and model compression. By leveraging graph-based representations, these networks effectively capture complex relational structures and long-range dependencies—key factors in enhancing the efficiency and interpretability of LLMs. Their applications in compression include graph-based knowledge distillation, structured

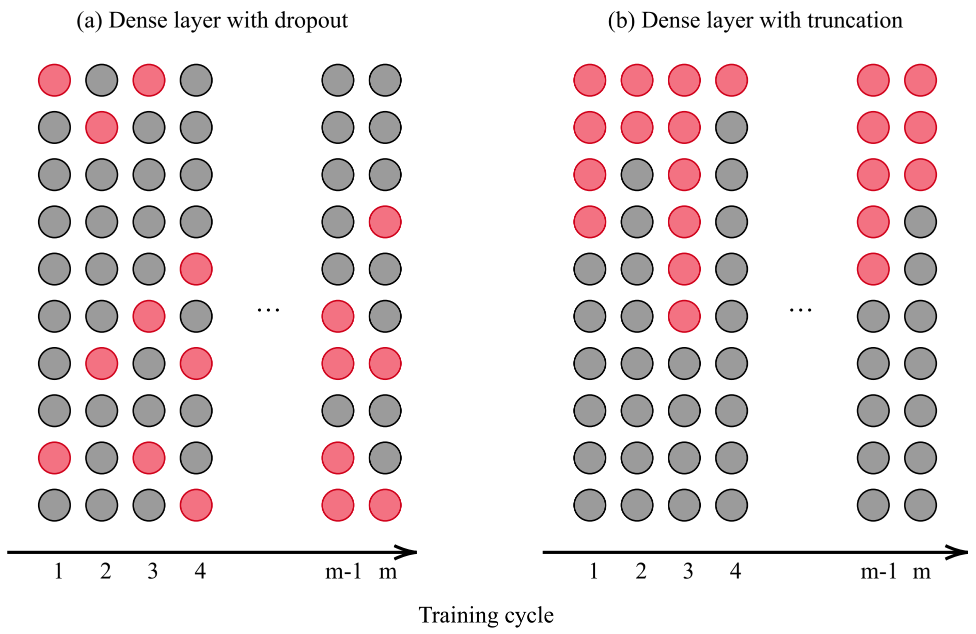

Fig. 9 Illustration of adaptive truncation applied to sequence processing. Each node represents a neuron in a dense layer; red nodes denote inactive or removed neurons, while gray nodes represent active neurons involved in computation. (**a**) Dropout randomly deactivates neurons during training without permanently removing them. (**b**) Truncation deterministically removes less relevant neurons, preserving only task-critical components. This selective reduction lowers memory requirements, computational requirements, and inference latency while maintaining task performance

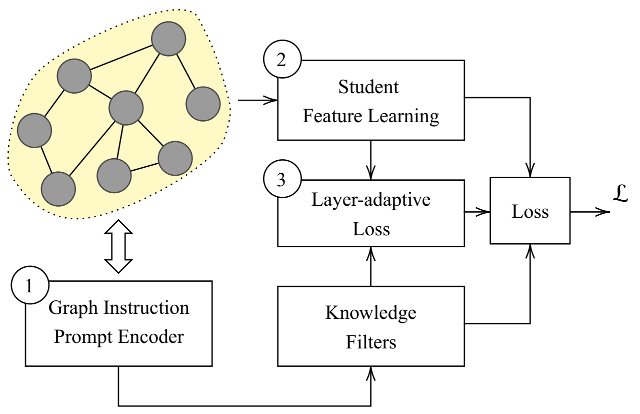

Fig. 10 GNN-based knowledge distillation framework for LLM compression. The architecture comprises three core components: (1) a graph instruction prompt encoder that fine-tunes the teacher model by incorporating structural and semantic information from the graph, (2) a GNN module that facilitates adaptive feature learning for the student model, enabling the acquisition of lightweight representations, and (3) a layer-adaptive loss mechanism that aligns teacher and student features through knowledge filtering, ensuring effective and high-fidelity knowledge transfer

pruning guided by topological insights, and optimization of quantization schemes through connectivity-aware analysis.

Figure 10 illustrates how GNNs offer a versatile and interpretable framework for compressing LLMs. Their ability to adapt to varying architectural and deployment constraints makes them especially valuable in real-world applications. Future research should advance these methods by integrating GNN-driven insights with traditional compression techniques, forming hybrid approaches that improve both scalability and performance [22].

GNNs facilitate knowledge distillation by encoding the teacher model's knowledge as a graph, where nodes represent semantic entities or architectural components, and edges capture their functional or hierarchical relationships [4]. A GNN learns from this structured representation to train a lightweight student model, effectively preserving complex interactions and enabling structured reasoning and task-specific generalization.

Beyond distillation, GNNs support compression through structured pruning. By modeling LLMs as computation graphs, they can identify redundant or low-salience components—such as attention heads or dense-layer weights—and inform pruning strategies that minimize accuracy degradation while improving efficiency. For example, graph-based analysis can guide the removal of unnecessary connections or reduce intermediate dimensions in Transformer architectures.

Additionally, GNNs assist in quantization by predicting layer-wise sensitivity to reduced precision. Representing model parameters as nodes in a computation graph enables the network to estimate which components can tolerate lower bit-widths without impairing overall performance. This approach supports fine-grained optimization of memory usage and computational overhead.

Targeted optimization techniques

The deployment of LLMs in resource-constrained environments is hindered by their substantial memory usage, computational demands, and limited compatibility with general-purpose hardware. These challenges necessitate targeted optimization techniques that go beyond generic compression strategies and instead address the architectural and operational complexity of transformer-based models [12, 34, 63].

Unlike conventional compression methods, which often struggle to balance model compactness and performance, targeted techniques are specifically designed to preserve critical functional components of LLMs while enhancing efficiency. Notable examples include QAT and PTQ to reduce bit-width representations [63], structured pruning applied to redundant attention heads and feed-forward layers [12], and hardware-aware neural architecture search (NAS) that generates lightweight, deployment-optimized models [34].

In addition to these, advanced approaches such as knowledge distillation and low-rank factorization have been applied to further reduce model size and computational burden without compromising accuracy [81, 83]. These methods are typically aligned with the structural characteristics of transformer models and optimized for compatibility with specialized hardware accelerators such as GPUs, TPUs, and edge inference devices.

By tailoring compression techniques to the unique demands of LLMs, researchers have achieved significant gains in deployment feasibility—enabling high-throughput, low-latency inference in environments with strict resource constraints. Such advances are particularly critical for integrating LLMs into mobile platforms, embedded systems, and real-time applications.

As model architectures continue to evolve in scale and complexity, the development and refinement of targeted optimization techniques remain essential. Their effectiveness depends not only on compression performance but also on their adaptability to emerging hardware and deployment contexts. Future research must continue to explore the interaction between model design, compression strategy, and system-level constraints to ensure sustainable and practical adoption of LLMs at scale.

Hybrid and progressive strategies

Hybrid approaches and progressive compression techniques integrate multiple compression strategies in a coordinated manner to maximize efficiency while preserving model performance. These methods leverage the complementary strengths of individual techniques and apply them either sequentially or adaptively throughout training or deployment, offering flexible solutions for compressing LLMs under varying resource constraints.

Integrated compression pipelines

Model compression techniques such as quantization, pruning, and knowledge distillation are often combined in hybrid compression approaches to address the substantial memory and computational demands of LLMs [4, 8]. These integrated methods aim to substantially reduce model size and inference latency while preserving task performance. However, optimizing a LLM using multiple compression strategies introduces significant challenges, including potential conflicts between techniques, tuning complexity, and degradation of model stability.

The conventional approach to model compression applies pruning, knowledge distillation, and quantization in a sequential manner [8]. Pruning eliminates redundant parameters, distillation transfers knowledge to a smaller model to maintain accuracy, and quantization leverages the distilled model's optimized representations to reduce precision. Alternative workflows have also been proposed, such as applying pruning or quantization prior to distillation, or using iterative cycles [81]. For example, quantization-robust pruning with knowledge distillation (QRPK) enhances quantization robustness by performing pruning before distillation, thereby ensuring efficiency across different bit-width configurations [34].

Combining pruning, distillation, and quantization enhances model compression while preserving performance [82, 85]. For instance, NVIDIA's Megatron-LM integrates mixed-precision training with pruning strategies to achieve significant reductions in memory requirements without compromising accuracy [43]. Similarly, Google's T5 framework employs sparsity-inducing pruning in conjunction with knowledge distillation to scale effectively, maintaining high performance across tasks such as translation and summarization [45]. These implementations demonstrate the growing adoption of hybrid methods to optimize large-scale models for real-world deployment.

AutoML-driven dynamic compression

Adaptive compression dynamically tailors compression strategies to task-specific requirements—for example, applying aggressive pruning in resource-constrained environments, while prioritizing accuracy preservation in latency-sensitive applications [66]. Automated ML approaches, such as once-for-all (OFA), optimize model architectures across multiple dimensions, including speed, size, and accuracy, thereby facilitating efficient deployment on mobile and edge devices. These hybrid approaches effectively balance parameter efficiency and model performance, demonstrating strong applicability across both computer vision and NLP tasks. Model compression is dynamically adjusted according to three core principles:

- *Task-specific requirements*: Accuracy is preserved for critical applications while trade-offs are acceptable in latency-sensitive scenarios.

- *Hardware constraints*: Aggressive compression is applied to edge devices and lighter compression to server-grade infrastructure.
- *Real-time responsiveness*: Systems switch between model variants to adapt to changing latency demands. This adaptive approach maximizes efficiency while maintaining acceptable performance across diverse deployment contexts [66].

This adaptive strategy enhances computational efficiency and enables cross-platform deployment while optimizing accuracy-performance trade-offs. However, its implementation demands careful design to overcome several challenges. The adaptation mechanisms must incur minimal overhead, maintain system stability, and avoid disruptive oscillations. Moreover, the dynamic switching logic and its associated metadata introduce deployment complexity that must be effectively managed. These considerations create a critical design trade-off between flexibility and reliability, which ultimately determines the feasibility of real-world deployment.

Hardware-aware co-design strategies

In practical scenarios, hardware-aware optimizations further enhance the efficiency of compressed models by aligning their execution with the capabilities of the underlying hardware architecture. For instance, NVIDIA's TensorRT framework incorporates hardware-specific techniques—such as layer fusion and mixed-precision inference—to accelerate model execution while preserving accuracy [67]. Similarly, Facebook's open-source PyTorch Mobile framework integrates hardware-aware strategies to enable efficient on-device inference in resource-constrained environments, including smartphones and IOT devices [12].

These optimizations underscore the importance of aligning model architecture with hardware capabilities to maximize performance and resource utilization. A more detailed discussion of hardware-aware optimization techniques—including their adaptation to diverse hardware platforms—is provided in Sect. 6.2.

Gradual compression with performance preservation guarantees

Progressive compression maintains model accuracy by gradually increasing compression intensity across multiple stages, with retraining after each step to adapt the network to structural changes. This staged approach systematically preserves critical parameters while eliminating redundant ones, allowing the model to recalibrate and recover performance after each compression phase. The iterative process enables task- and environment-specific optimization, offering a controlled trade-off between efficiency and accuracy.

Techniques such as light pruning and low-bit quantization, when interleaved with retraining, are applied to preserve model performance during compression. For instance, transformer-based models have achieved substantial reductions in memory usage with minimal accuracy loss in NLP tasks [9]. Likewise, convolutional neural networks (CNNs) have demonstrated parameter reductions of up to 50% without compromising accuracy in computer vision applications [63]. Recurrent neural networks (RNNs) have also sustained predictive accuracy in time-series forecasting despite undergoing aggressive compression [28].

Practical implementations of progressive compression include the Eyeriss system, which employs progressive pruning to enable energy-efficient edge AI without compromising accuracy [14], and Google's speech recognition models, which utilize progressive quantization to balance memory requirements and model performance [28]. By monitoring performance at each compression stage, these approaches effectively prevent degradation while achieving substantial reductions in computational and storage costs.

Trade-off management in hybrid compression systems

Hybrid model compression techniques offer substantial gains in model size and inference speed. However, they also introduce intricate trade-offs [86]. Each additional compression method increases the overall design complexity and necessitates careful balancing between accuracy and computational efficiency. Even selective pruning involves a trade-off between model size and performance, while compact architectures, such as MobileNetV3, are explicitly designed to optimize latency, accuracy, and computational cost simultaneously [7]. Consequently, hybrid compression approaches require the joint optimization of multiple objectives—including size, speed, and accuracy—rendering the search for optimal configurations a highly complex and non-trivial task.

Sequential application of compression techniques can introduce interference effects and compound errors. Recent theoretical studies demonstrate that pruning (which introduces sparsity) and quantization are not orthogonal processes; rather, their associated errors interact in nonlinear and often unpredictable ways [87]. The cumulative error resulting from quantization followed by pruning typically exceeds the sum of the individual errors and is highly sensitive to the order in which these techniques are applied [87].

For example, performing quantization first may distort the perceived importance of model weights, leading subsequent pruning stages to inadvertently remove critical components. Even when the order is carefully chosen, the resulting accuracy degradation can be substantial. These interference effects are frequently observed in practice: post-hoc compression pipelines often require additional fine-tuning to recover lost performance, indicating that errors from successive compression stages accumulate in a nonlinear manner. This complex interaction—commonly referred to as non-orthogonality—has been consistently reported across both computer vision and NLP models [87].

The sequence in which compression methods are applied has a substantial impact on final model performance. Treating pruning and quantization as independent, sequential stages often leads to suboptimal outcomes. As noted by previous studies, "the best network architecture for the full-precision model is not necessarily the optimal one after pruning and quantization" [88]. In practice, applying pruning prior to quantization—or vice versa—can result in significantly different behaviors. For instance, performing quantization first may alter the weight distribution and distort importance metrics, thereby compromising the effectiveness of subsequent pruning [87].

These findings highlight the importance of carefully selecting both the order and interaction strategy of compression operations. To mitigate performance loss, some methodologies advocate for joint or integrated optimization schemes rather than relying on naive, sequential pipelines [88]. In summary, the integration strategy and execution order represent critical considerations in the design of hybrid compression workflows.

Because compression methods interact, their associated errors accumulate in a nonlinear fashion. The combination of sparsity and quantization often results in greater accuracy degradation than the sum of their individual effects [87]. In practice, naively applying these techniques in sequence—such as pruning followed by quantization—can lead to unexpected accuracy losses that exceed intuitive expectations. This nonlinear compounding implies that the final performance degradation cannot be reliably estimated by analyzing each compression step in isolation. Consequently, many compression pipelines require additional corrective procedures, such as retraining or fine-tuning, to counteract the amplified errors introduced by successive operations.

To recover the accuracy lost through hybrid compression, researchers frequently employ fine-tuning or knowledge distillation, albeit at an additional computational requirements. Following pruning or quantization, it is standard practice to retrain the model on task-specific data; this fine-tuning step can significantly restore performance [89]. Similarly, knowledge distillation involves training a smaller model to replicate the behavior of a larger one, typically by minimizing the divergence between their output distributions.

Notably, auto-compression pipelines often report that each compression stage—whether pruning or quantization—may require dedicated fine-tuning to mitigate cumulative performance degradation. For instance, frameworks such as neural network distiller [90], AutoML for model compression [91], and MIT's MCUNet Compression Pipeline [92] integrate multiple stages of compression with adaptive search and tuning mechanisms. Although fine-tuning and distillation are effective in compensating for compression-induced errors, they introduce substantial training overhead that must be accounted for during model deployment.

A rigorous theoretical understanding of how hybrid compression errors interact remains limited. Recent studies have begun formally analyzing the combined effects of sparsity and quantization [87], but comprehensive error-theoretic models for pruning, quantization, and distillation remain an open challenge. Most existing methods rely on empirical tuning or proxy metrics rather than closed-form theoretical frameworks. As noted in recent surveys, improving explainability and understanding the changes and trade-offs in the compression process are critical future directions [93]. In summary, developing joint optimization strategies or analytical bounds that predict the compounded impact of multiple compression methods remains a key unresolved problem in the field.

Metrics and performance evaluation framework

Evaluating LLM compression techniques requires a rigorous and standardized performance framework. This chapter introduces the key metrics and evaluation criteria used to assess the trade-offs between model size, accuracy, efficiency, and deployment feasibility. By establishing a consistent foundation for performance analysis, this framework enables fair comparisons across compression methods and supports reproducible research in large-scale model optimization.

Core evaluation metrics

These are considered baseline metrics because they form the foundation for evaluating model performance in most NLP tasks. Accuracy measures correct predictions, while PPL assesses how well the model predicts language sequences. Though widely used, these metrics offer limited insight into compression-related trade-offs like efficiency, scalability, or robustness, which advanced metrics are designed to capture.

Perplexity (PPL)

PPL is a fundamental metric for evaluating the predictive capabilities of LLMs, as it quantifies the model's ability to capture and generalize language patterns and structures. It assesses the probability assigned to sequences, indicating how well the model approximates the underlying data distribution. Lower PPL values correspond to better performance in tasks such as text completion, question answering, and contextual inference. In the context of model compression, preserving low PPL while reducing model size serves as an indicator of the model's ability to retain essential linguistic knowledge.

The relationship between PPL and model effectiveness is well-established: lower PPL values correspond to stronger next-token prediction capabilities, making PPL a reliable proxy for overall model quality. Compression techniques affect PPL to varying degrees, thereby exposing the sensitivity of specific architectural components. Importantly, changes in PPL often correlate with variations in downstream task performance. As such, architectural choices significantly influence the trade-off between PPL and compression, underscoring the need for compression-aware design strategies.

Equation (7) shows the formal definition of PPL, computing the exponential of the negative average log-likelihood of the model's predictions over a given sequence. The expression $P(w_i \mid w_{<i})$ denotes the probability that the model assigns to the token w_i given its preceding context $w_{<i}$. By averaging the log probabilities across the total number of tokens N and applying the exponential function, the PPL score reflects the model's overall uncertainty: lower PPL indicates higher confidence and better language understanding. This metric is particularly useful for comparing models of different sizes, especially in compression scenarios where performance retention is critical.

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum \log P(w_i|w_{<i})\right) \tag{7}$$

Recent studies suggest that increases in PPL are closely correlated with performance degradation in compressed models. This relationship typically adheres to the following thresholds: a PPL increase of less than 15% generally results in minimal impact on downstream tasks, with negligible degradation; an increase between 15% and 30% indicates moderate degradation and often warrants task-specific performance evaluation; and an increase exceeding 30% poses a substantial risk of significant performance loss, particularly in knowledge-intensive tasks [94].

Accuracy

The evaluation of compressed LLMs must consider task-specific requirements and answer types. Different NLP tasks demand tailored evaluation strategies due to variations in output formats and success criteria. Table 4 summarizes key answer types alongside their associated evaluation metrics, including the F1-score, area under the curve of receiver operating characteristic (AUC-ROC), recall-oriented understudy for gisting evaluation (ROUGE), bilingual evaluation understudy (BLEU), exact match (EM), and the conference on natural language learning F1-score (CoNLL F1). Additional details regarding metric selection for each task category are also provided to support comprehensive model assessment.

Key considerations include the complexity of answer types, where binary or categorical tasks typically require simpler metrics compared to span-based or generative tasks. The use of partial credit metrics, such as the F1-score for token overlap, provides a more nuanced evaluation than strict EM. Moreover, multiple metrics are often necessary to capture diverse aspects of performance and to balance trade-offs between accuracy, coverage, and robustness. Finally, practical constraints—such as response time and resource utilization—should be incorporated to comprehensively assess the model's suitability for real-world deployment.

Efficiency-oriented metrics

These metrics go beyond baseline performance evaluation by quantifying trade-offs introduced during model compression, such as sparsity, latency versus accuracy, and parameter utilization. They are designed to assess how well a compressed model meets practical deployment constraints, making them essential for measuring real-world efficiency and effectiveness.

Sparsity regularization

Sparsity regularization is a fundamental concept in LLM compression, promoting representations with a reduced number of active parameters. Although not a compression technique in itself, it significantly enhances the effectiveness of methods such as pruning and quantization. By incorporating regularization terms—such as the L1 norm or structured sparsity penalties—into the training objective, sparsity regularization encourages weights or activations to converge toward zero, thereby inducing sparse structures within the model [63].

In the context of LLMs, sparsity facilitates compression by identifying and eliminating redundant or less critical components, such as neurons, weights, or even entire layers. This reduction not only lowers the model's memory requirements but also accelerates inference by decreasing the number of necessary computations. Furthermore, sparsity-aware techniques align well with modern hardware accelerators, which are increasingly capable of leveraging sparse data structures to optimize both storage and computational efficiency.

Equation (8) presents the formal definition of sparsity regularization. In this formulation, sparsity is encouraged by adding an L_1 regularization term to the original task-specific loss function. The goal is to reduce the number of nonzero weights in the model, thereby minimizing redundancy and promoting a more compact architecture. This approach is particularly valuable in LLMs, which often exhibit over-

Table 4 Unified classification of NLP tasks with evaluation metrics

Category	Type/task	Description	Balanced accuracy [95]	F1-score [96]	AUC-ROC [97]	Class-wise accuracy [98]	Token overlap [99]	CoNLL F1 [100]	ROUGE/BLEU [101]	Response time [102]
	Binary	Two-class decision outputs (e.g., sentiment analysis, topic classification)	✓	✓	✓					
	Multi-class	classification over a fixed set of categories (e.g., document classification, intent recognition)		✓		✓				
	Text classification	Assigns predefined categories to text samples		✓	✓	✓				
Extraction	Span-based	Extracts specific text segments from the input (e.g., question answering, entity recognition)					✓	✓		
	Question answering	Extracts or generates relevant answers from contextual input		✓			✓			✓
	Named entity recognition	Identifies and categorizes named entities (e.g., persons, locations, organizations) within the text		✓			✓	✓		

Table 4 continued

Category	Type/task	Description	Balanced accuracy [95]	F1-score [96]	AUC-ROC [97]	Class-wise accuracy [98]	Token overlap [99]	CoNLL F1 [100]	ROUGE/BLEU [101]	Response time [102]
Generation	Generated	Produces free-form text outputs (e.g., summarization, machine translation)							✓	
	Question answering (generative)	Generates answers beyond extractive capabilities, often requiring synthesis or reasoning							✓	✓

parameterization with many unnecessary parameters. By penalizing the absolute values of the weights, the regularization term $|W|_1$ drives less significant parameters toward zero. The regularization coefficient β controls the trade-off between model performance and sparsity. This technique is effective in enabling efficient compression through pruning and in accelerating inference on edge devices and other resource-constrained platforms.

$$\mathcal{L}_{\text{sparse}} = \mathcal{L}(W) + \beta \|W\|_1 \tag{8}$$

Latency-accuracy trade-off (LAT)

The LAT metric is a critical concept for balancing model performance and computational efficiency, particularly in resource-constrained environments. It addresses the inherent trade-off between processing speed and predictive quality, which is essential for deploying LLMs in practical scenarios. Unlike traditional evaluation methods that consider accuracy and latency as separate objectives, the LAT metric integrates both factors into a unified framework, enabling more informed deployment decisions [41].

This approach is particularly relevant in environments with limited computational resources—such as edge devices or real-time systems—where prioritizing one objective, such as accuracy, may increase inference latency, whereas emphasizing speed may reduce predictive quality. By introducing a deployment-specific weighting parameter α, the LAT metric enables customizable optimization tailored to the specific requirements of a given application.

The LAT metric is designed to capture the balance between prediction quality and computational efficiency in a single scalar value. It is particularly important when deploying LLMs in environments where both inference speed and accuracy are critical factors, such as real-time applications or edge devices.

Figure 9 illustrates the LAT metric, which applies a deployment-specific weighting parameter $\alpha \in [0, 1]$, allowing practitioners to prioritize either accuracy ($\alpha \rightarrow 1$) or latency ($\alpha \rightarrow 0$) based on the constraints of the target environment. It compares the compressed model's performance and latency against the original model's baseline values. The metric penalizes a decrease in accuracy and an increase in latency relative to the original model. Let acccomp and latencycomp denote the accuracy and latency of the compressed model, respectively, and acc and latency the corresponding values for the original model.

$$\text{LAT} = \alpha \cdot \left(1 - \frac{\text{acccomp}}{\text{acc}}\right) + (1 - \alpha) \cdot \left(\frac{\text{latencycomp}}{\text{latency}}\right) \tag{9}$$

This formulation quantifies degradation in both dimensions: the first term penalizes accuracy loss, while the second accounts for increased latency. Lower LAT values are preferable, indicating that the compressed model retains performance close to the original with minimal latency overhead.

Parameter efficiency

Parameter efficiency is a critical metric for evaluating how effectively a compressed model utilizes its reduced parameter space. As LLMs scale to hundreds of billions of parameters, balancing parameter reduction with performance retention becomes increasingly important. Traditional evaluation methods often focus solely on either compression ratios or task accuracy, overlooking the nuanced relationship between the number of retained parameters and their functional contribution to the model's overall performance.

By assessing both the quantity and functional utility of the retained parameters, parameter efficiency offers a more comprehensive measure of model optimization. This metric is particularly relevant for LLMs, where overparameterization is prevalent and identifying redundant components is essential for improving memory utilization and computational efficiency.

Equation (10) defines the parameter efficiency metric. It quantifies how effectively a compressed model retains predictive performance relative to the proportion of parameters it preserves. Let acc represent the accuracy of the original model, and acc_{comp} the accuracy after compression. Likewise, let params denote the total number of parameters in the original model, and $\text{params}_{\text{comp}}$ the number in the compressed version. The metric is calculated as the ratio between the relative accuracy and the relative parameter count.

$$\text{PE} = \left(\frac{\text{acc}_{\text{comp}}}{\text{acc}}\right) \Big/ \left(\frac{\text{params}_{\text{comp}}}{\text{params}}\right) \tag{10}$$

A value of $\text{PE} > 1$ suggests that the model achieves better-than-expected accuracy retention per parameter, indicating strong parameter utilization. Conversely, a value below 1 implies suboptimal efficiency, where too many parameters are required to maintain acceptable performance. State-of-the-art methods report parameter efficiency values in the range of 1.2–1.3 for standard compression techniques, 1.3–1.4 for advanced approaches incorporating knowledge distillation, and 1.4–1.5 for cutting-edge methods that integrate architectural optimization [28].

Deployment-oriented metrics

Fairness gap (FG), Robustness score (RS), and multi-objective pareto optimization are metrics designed to evaluate how compressed LLMs perform under real-world conditions where ethical considerations, reliability, and resource trade-offs are critical. By capturing dimensions like demographic equity, stability under perturbations, and performance-efficiency trade-offs, these metrics guide the responsible and effective deployment of models in high-stakes applications, including healthcare, finance, and public systems.

Multi-objective Pareto optimization

Multi-objective Pareto optimization provides an effective framework for balancing competing objectives in LLM compression, such as minimizing model size while preserving accuracy. In contrast to single-metric evaluations, it accounts for the trade-offs among multiple objectives. Rooted in the principle of Pareto efficiency, it identifies configurations in which improving one aspect (e.g., inference latency) inevitably leads to a deterioration in another (e.g., accuracy). The result is a Pareto frontier—a set of non-dominated solutions that represent the most balanced trade-offs across all considered objectives [82, 103].

In the context of LLMs, Pareto optimization is particularly relevant as it effectively captures the trade-offs among model size, computational efficiency, and task-specific performance. This approach is essential for ensuring that compressed models satisfy the diverse and often conflicting demands of real-world deployments, such as low latency, high accuracy, and limited resource availability.

The Pareto framework extends traditional Pareto analysis by incorporating domain-specific constraints and deployment considerations. It captures the interplay between model size, computational efficiency, and task performance, allowing practitioners to identify optimal compression configurations that balance multiple objectives, understand trade-offs between competing performance aspects, make decisions that align with specific deployment constraints, and assess the effectiveness of strategies relative to the Pareto frontier.

This methodology evaluates compression trade-offs across three key dimensions:

1. *Model size* (S), with theoretical bounds defined as $0.1 \leq S \leq 1.0$:

$$S = \frac{\text{params}_{\text{compr}}}{\text{params}} \tag{11}$$

2. *Inference latency* (L), with a practical range defined as $0.2 \leq L \leq 1.0$:

$$L = \frac{\text{inference}_{\text{compr}}}{\text{inference}} \tag{12}$$

3. *Accuracy loss* (A), with an acceptable range defined as $0 \leq A \leq 0.05$:

$$A = \text{acc} - \text{acc}_{\text{compr}} \tag{13}$$

The Pareto score integrates the three evaluation metrics—model size, inference latency, and accuracy loss—through Equations (11), (12), and (13), and is formally expressed in Equation (14), where w_i denotes the weighting factor assigned to each metric.

$$\text{Pareto score} = \frac{w_1 S + w_2 L + w_3 A}{w_1 + w_2 + w_3} \tag{14}$$

The weights w_i are selected according to deployment-specific priorities:

- A higher value of w_1 emphasizes reducing model size, which is critical in memory-constrained environments.
- A higher value of w_2 prioritizes minimizing inference latency, suitable for latency-sensitive applications.
- A higher value of w_3 ensures the preservation of model accuracy, essential for precision-critical tasks.

The Pareto Optimization framework offers a systematic methodology for evaluating model compression strategies by jointly considering model size, inference latency, and accuracy loss. This approach enables quantitative analysis of compression trade-offs, while the weighted Pareto score allows adaptation to a wide range of deployment scenarios—from memory- and compute-constrained edge devices to high-performance computing (HPC) environments.

Robustness score (RS)

Model robustness is a critical consideration in LLM compression, as compressed models often become more sensitive to input variations than their uncompressed counterparts. This increased sensitivity results from the potential reduction in representational capacity during compression, which may impair the model's ability to handle linguistic variability and maintain stable performance in dynamic, real-world settings. The RS offers a structured framework for evaluating a model's stability under various input perturbations, ensuring that compressed models remain reliable in the presence of noise, adversarial inputs, or distributional shifts.

In ML, a perturbation is defined as $\text{x}_{\text{perturbed}} = \text{x}+\delta$, where δ denotes the magnitude and type of perturbation applied to the input. Perturbations can manifest in various forms:

- *Random noise*: The addition of Gaussian or uniform noise to the input data.

- *Adversarial attacks*: Targeted modifications intentionally crafted to maximize prediction errors.
- *Distribution shifts*: Systematic changes in data distribution, such as those encountered in domain adaptation scenarios.
- *Semantic modifications*: Linguistic changes including synonym substitutions, paraphrasing, or restructuring of sentences.
- *Environmental variations*: External factors such as hardware conditions, computational constraints, or variations in input formatting.

Equation (15) expresses how the robustness of a compressed model is quantitatively assessed using the RS. This metric captures the model's ability to maintain predictive performance when subjected to various input perturbations. In this formulation, $\mathcal{P}$ denotes the performance metric (such as accuracy or F1-score) of the original, unperturbed model. The term $\mathcal{P}_{\text{pert}_i}$ represents the model's performance under the ith perturbation scenario. The absolute difference $|\mathcal{P} - \mathcal{P}_{\text{pert}_i}|$ quantifies the performance degradation induced by the perturbation. The expression inside the summation computes the relative performance retained in each perturbed case, and averaging over all N perturbations provides a comprehensive view of the model's stability. The final multiplication by 100% expresses the RS as a percentage.

$$\text{RS} = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{|\mathcal{P} - \mathcal{P}_{\text{pert}_i}|}{\mathcal{P}} \right) \times 100\% \tag{15}$$

This formulation is essential for evaluating the deployment reliability of compressed models, as it quantifies how well a model withstands noise, adversarial attacks, or distributional shifts—factors critical to real-world performance.

A concrete example of RS evaluation is provided in a study on compressed vision models [104]. The authors compared binarized neural networks (1-bit weight models) to a full-precision 32-bit baseline on the CIFAR-10 image classification task under various perturbation scenarios. For each model, accuracy was reported across multiple attacks, and the results were averaged to calculate a RS (i.e., mean accuracy retention) within each attack category.

Under a suite of five white-box adversarial attacks, the full-precision model achieved a RS of 32.16%, while the binarized model's score dropped to 16.53%. This indicates that the compressed model retained only about half of its clean-data accuracy on average, highlighting a substantial robustness degradation. In contrast, not all attack scenarios revealed such stark differences. For example, in one group of black-box attacks—which are typically less aggressive—the full-precision model achieved an RS of 27.92%, compared to 24.47% for the binarized model. This narrower gap suggests that the severity and nature of perturbation significantly influence robustness outcomes. While compressed models may handle mild or noisy distributional shifts comparably to uncompressed models, they are considerably more vulnerable under worst-case (adversarial) conditions.

Fairness gap (FG)

The FG metric addresses a critical challenge in LLM compression: ensuring that compression does not exacerbate disparities in performance across distinct data subgroups used to evaluate fairness. Specifically, the effect of compression on model performance should be consistent across these groups, avoiding disproportionate degradation in accuracy, prediction quality, or fairness-related metrics such as demographic parity, equalized odds, or equality of opportunity.

Although compression techniques are effective at reducing model size and computational overhead, they can disproportionately impact certain population segments—either amplifying existing biases or introducing new ones. This underscores the importance of the FG metric, particularly in high-stakes applications where equitable outcomes are imperative, such as healthcare, recruitment, and financial decision-making [105].

Equation (16) formally expresses FG, which quantifies the discrepancy in model fairness between two protected groups using a selected fairness metric. Specifically, Metric_1 and Metric_2 denote the values of the chosen fairness metric—such as demographic parity, equalized odds, or equality of opportunity—for each respective group. The absolute difference between these values reflects the degree to which compression may have introduced or amplified fairness disparities:

$$\text{FG} = |\text{Metric}_1 - \text{Metric}_2|. \tag{16}$$

A smaller FG indicates improved equity between groups, with $\text{FG} = 0$ denoting perfect parity according to the chosen fairness metric. However, realizing this ideal is often hindered by inherent tensions among fairness criteria. For example, optimizing for demographic parity may directly conflict with equalized odds, making it challenging to satisfy multiple objectives concurrently. As a result, model designers must evaluate and manage these trade-offs to ensure a balance that aligns with both ethical standards and deployment-specific constraints.

To aid interpretation, Table 5 presents standardized thresholds for FG values. These thresholds define three categorical levels of fairness, offering practical benchmarks for assessing and addressing disparities [63]. This framework supports practitioners in identifying fairness concerns and implementing corrective strategies, helping ensure that compressed

Table 5 Threshold-based categorization of FG values, providing quantitative ranges and corresponding qualitative interpretations to assess algorithmic bias and inform deployment decisions in ML applications

Range	Category	Interpretation
$FG < 0.05$	Excellent	Minimal demographic disparity; suitable for high-sensitivity or fairness-critical applications
$0.05 \leq FG < 0.10$	Acceptable	Moderate demographic disparity; acceptable for general-purpose applications
$FG \geq 0.10$	Concerning	Significant demographic disparity; mitigation strategies required prior to deployment

LLMs uphold ethical and equitable standards in high-stakes domains such as hiring, loan approvals, and healthcare.

Hardware-aware metrics

Hardware efficiency metrics are essential for assessing the practical deployability of compressed models, as they capture computational resource utilization and energy consumption. These metrics have direct implications for deployment cost, battery longevity in mobile devices, and overall system throughput.

Energy Efficiency

Energy efficiency is a critical metric for assessing computational performance relative to power consumption, especially in battery-powered and energy-constrained environments such as mobile devices and edge computing platforms [13, 35]. It captures the trade-off between inference accuracy and the energy required to sustain it, as formally defined in Eq. (17).

$$\text{Energy Efficiency} = \frac{\text{inference accuracy}}{\text{power consumption}} \tag{17}$$

Equation (18) introduces a normalized energy efficiency score, which facilitates meaningful cross-platform comparisons by quantifying the energy efficiency of a compressed model relative to its original counterpart.

$$\text{EE}_{\text{normalized}} = \frac{\text{EE}_{\text{compr}}}{\text{EE}} \tag{18}$$

High energy efficiency is critical for deploying LLMs in energy-constrained settings, where low power consumption is necessary to maintain device performance and thermal stability. Table 6 shows that mobile platforms typically require a normalized energy efficiency score above 2.0 to meet strict battery and thermal constraints.

Recent research underscores the importance of platform-specific energy efficiency targets, given the substantial variation in energy consumption across hardware platforms. For compressed LLMs, achieving optimal energy efficiency necessitates careful alignment of model architectures and compression techniques with underlying hardware capabilities. This alignment ensures that trade-offs among accuracy, power consumption, and deployment requirements are effectively managed, thereby enabling efficient operation across a range of application scenarios [13, 33].

Memory access efficiency (MAE)

Memory access efficiency (MAE) quantifies the proportion of useful memory operations—such as fetching weights and activations—relative to total memory accesses. It serves as a key metric for evaluating hardware efficiency in compressed models, particularly in scenarios constrained by memory bandwidth [106].

MAE is a critical factor in LLM compression, as it highlights hardware-level performance bottlenecks. It allows practitioners to adjust compression strategies to better align with hardware constraints, thereby facilitating efficient and portable model deployment. Consequently, it serves as a key metric in hardware-aware optimization.

Equation (19) formally defines MAE, which measures the proportion of memory operations that directly contribute to computation—such as fetching weights and activations—relative to the total number of memory operations executed. In this context, Effective memory accesses represent the subset of accesses that are essential for model execution, while Total memory accesses include all memory interactions, including those that are redundant or inefficient. These quantities are typically measured using profiling tools such as NVIDIA Nsight Compute, Intel VTune, or deep learning (DL) framework profilers like PyTorch Profiler.

$$\text{MAE} = \frac{\text{effective memory accesses}}{\text{total memory accesses}} \tag{19}$$

High MAE indicates effective utilization of memory bandwidth, contributing to low latency and energy-efficient inference. In contrast, low efficiency suggests overhead due to redundancy or suboptimal memory allocation. For

Table 6 Normalized energy efficiency thresholds for different deployment platforms, illustrating progressively higher efficiency requirements in mobile environments (greater than 2.0) compared to edge (greater than 1.5) and cloud (greater than 1.2) deployments

Platform	Target ($EE_{normalized}$)	Considerations
Mobile	> 2.0	Battery life is critical in energy-constrained environments, where stringent thermal constraints further limit model deployment options
Edge	> 1.5	Balances predictive performance with power efficiency, enabling deployment in resource-constrained environments
Cloud	> 1.2	Optimizes throughput and scalability, suitable for HPC environments

These thresholds reflect platform-specific operational constraints such as power availability, thermal limits, and usage patterns

compressed LLMs, maintaining high MAE is critical, as compression techniques such as pruning and quantization can disrupt memory access patterns and negatively impact performance.

Case studies and practical applications

This section presents real-world case studies of LLM compression, examining techniques such as pruning, quantization, knowledge distillation, and NAS applied to models like BERT and GPT. Each study evaluates impacts on performance, latency, energy efficiency, and task-specific accuracy, bridging theory and practice. By integrating empirical results and industrial use cases, the analysis demonstrates the value of compression in resource-constrained environments—ranging from mobile and edge devices to large-scale systems—while addressing trade-offs and limitations that inform future deployment strategies.

LLM deployment in IOT and edge devices

This section explores the deployment of LLM compression techniques in mobile environments, where limitations in computational capacity, energy availability, and latency are especially critical. By analyzing real-world applications, it highlights how methods such as pruning, quantization and knowledge distillation, are adapted to optimize LLMs for mobile devices.

Balancing efficiency and depth: lessons from MobileBERT

MobileBERT exemplifies the effective use of compression techniques, achieving a 4× reduction in model size and a 5.5× inference speedup while retaining 99% of BERT's original accuracy [6]. Empirical evaluations show MobileBERT outperforming alternatives such as DistilBERT and TinyBERT [69] in real-time NLP tasks, including sentiment analysis and question answering. For instance, its integration into mobile search applications reduces latency from 120 to 22 ms per query, enabling responsive user interactions on resource-constrained devices. This improvement is critical for maintaining a seamless user experience in latency-sensitive environments.

Despite its strengths, MobileBERT exhibits limitations in tasks requiring extensive contextual reasoning, such as document summarization and dialogue systems. These shortcomings underscore the need for hybrid compression strategies capable of dynamically allocating computational resources across layers, thereby balancing efficiency with the preservation of complex linguistic capabilities. Techniques such as NAS and adaptive sparsity offer promising avenues for further optimization, enabling compressed models to meet diverse task demands without incurring substantial performance degradation.

Conversational agents in mobile environments

TinyBERT, a compressed variant of BERT, demonstrates the efficacy of combining knowledge distillation with layer-wise compression to enhance performance in mobile environments [69]. Through task-specific distillation, TinyBERT achieves a 7.5× reduction in model size and a 9.4× decrease in inference latency, while retaining over 96% of the original BERT's accuracy across a range of NLP tasks. These characteristics render it especially well-suited for mobile applications, including conversational AI systems and voice-activated assistants.

For instance, when deployed in a virtual assistant on a mid-range mobile device, TinyBERT achieves a latency of just 15 ms per query, enabling real-time user interactions. Its reduced memory footprint ensures compatibility with devices equipped with limited random access memory (RAM), such as older smartphones and embedded systems. Furthermore, its energy efficiency contributes to extended battery life, making it particularly suitable for deployment in energy-constrained environments.

Despite its advantages, TinyBERT encounters limitations in tasks that demand deeper contextual understanding, such as multi-turn dialogue management and abstractive summarization. Overcoming these challenges may require the integration of advanced techniques, such as NAS or hybrid distillation, to enable dynamic resource allocation for handling more complex linguistic tasks.

The analysis of MobileBERT and TinyBERT underscores the transformative potential of LLM compression in mobile environments, where constraints such as computational resources, energy efficiency, and latency are paramount.

Energy-conscious compression for IOT workloads

The Eyeriss system illustrates the effective integration of iterative pruning and quantization to optimize energy efficiency without compromising model accuracy [14]. Empirical results indicate that Eyeriss achieves a twofold reduction in energy consumption compared to baseline implementations, making it particularly advantageous for energy-constrained IOT applications. For example, in motion detection systems, Eyeriss reduces power consumption from 1.2 to 0.6 W while maintaining a 98% accuracy rate in image recognition tasks. This efficiency is critical for prolonging battery life and minimizing operational costs in real-world IOT deployments.

The examination of the Eyeriss system bridges theoretical advancements in LLM compression with their practical deployment in IOT environments. The results underscore the transformative potential of compression techniques for optimizing energy efficiency, maintaining robust performance, and addressing the unique constraints inherent to IOT systems. Furthermore, the observed trade-offs offer critical insights for future research, highlighting the need for scalable and adaptive strategies capable of meeting the evolving demands of dynamic IOT applications.

Collaborative filtering enhancement at scale

In the rapidly evolving landscape of recommendation systems, data sparsity remains a persistent challenge, particularly in collaborative filtering approaches. This case study examines the implementation of the LLMRec framework, which utilizes LLMs to enhance collaborative filtering in a real-world deployment at Netflix, one of the world's leading streaming platforms [107].

Netflix, with its extensive catalog of movies and television shows, encountered challenges in accurately predicting user preferences due to the inherent sparsity of user-item interaction data. To address this, the LLMRec framework was employed to augment user-item interaction graphs and enhance recommendation accuracy. By integrating LLMs, LLMRec captures richer user preferences through natural language representations, enabling a more nuanced understanding of user interests.

The initial step at Netflix involved analyzing historical user interactions to identify latent patterns and preferences. LLMs were then employed to generate contextual embeddings for users and items, enriching the interaction graph with semantic information. This approach enabled the integration of content descriptions and user reviews, thereby enhancing the representation of user preferences and improving recommendation relevance.

To evaluate the effectiveness of the LLMRec framework, a series of experiments were conducted, benchmarking its performance against state-of-the-art collaborative filtering baselines. Standard evaluation metrics—including precision, recall, and F1-score—were employed to quantify the accuracy and relevance of the recommendations generated by the LLM-enhanced system.

The results of the case study demonstrated a substantial enhancement in recommendation accuracy with the adoption of the LLMRec framework at Netflix. By integrating LLMs, the system captured richer and more nuanced user preferences, enabling more personalized recommendations. Empirical evaluation revealed a 15% increase in precision and a 10% improvement in recall compared to conventional collaborative filtering approaches, underscoring the efficacy of LLM-based augmentation in real-world recommendation systems.

This case study highlights the potential of the LLMRec framework in enhancing collaborative filtering within recommendation systems, particularly at Netflix. By harnessing the power of LLMs, the framework effectively addresses data sparsity and significantly improves the quality of recommendations, offering a more personalized and accurate user experience.

Another case related to the application of LLM compression in recommendation systems is Alibaba's deployment of a hybrid strategy that integrates NAS and knowledge distillation [108]. This approach enables substantial performance gains, reducing inference latency by 40% while maintaining top-1 accuracy above 92%. For example, Alibaba's models handle billions of daily interactions, cutting average response time from 80 to 45 ms per query—a crucial improvement during high-traffic events like Singles' Day, where scalability is essential. Despite its success, the strategy depends heavily on computationally intensive NAS, with each iteration consuming up to 500 GPU-hours. This highlights a practical limitation that could be mitigated by adopting more efficient alternatives, such as ProxylessNAS [66].

Compression-aware hardware co-design

NVIDIA's TensorRT platform employs advanced optimization techniques, including PTQ and mixed-precision inference, to accelerate DL models in GPU environments [67]. By leveraging integer quantization (INT8), TensorRT significantly reduces computational overhead, enabling real-time performance even on resource-constrained hardware such as NVIDIA Jetson Orin devices. For example, object detection models optimized with TensorRT achieve a 3.7× reduction in latency on edge devices, with less than 1% accuracy loss compared to their original floating-point versions.

A notable application is You Only Look Once (YOLO)-v5, where TensorRT's dynamic tensor optimizations reduced inference latency by up to 4.35× on an NVIDIA RTX 3090 GPU, enhancing throughput while preserving detection accuracy. The platform's support for INT8 calibration further refines quantization by aligning activations with hardware-specific thresholds, minimizing quantization errors during execution [67]. These optimizations are critical in time-sensitive applications such as surgical instrument detection during minimally invasive procedures, where precision and low latency are essential.

Despite its strengths, TensorRT's dependence on NVIDIA hardware constrains its cross-platform adaptability. Addressing this limitation may involve developing hardware-agnostic frameworks that retain TensorRT's efficiency while broadening deployment compatibility. Such efforts would support more diverse AI scenarios across both edge and cloud environments.

Beyond TensorRT and NVIDIA GPUs, hardware-aware optimization techniques must be tailored to the specific constraints and capabilities of diverse platforms, including edge devices, TPUs, FPGAs, and custom AI accelerators. Each platform imposes distinct trade-offs among computational efficiency, memory bandwidth, and energy consumption, which must be addressed when designing and deploying compressed models.

Edge devices—such as mobile processors and embedded systems—typically operate under strict power and memory constraints. To facilitate efficient deployment, aggressive compression methods such as low-bit quantization, structured pruning, and knowledge distillation are frequently employed. These techniques substantially reduce model size and computational load, enabling real-time inference while conserving battery life. However, extreme compression can degrade accuracy, necessitating fine-tuned balancing between efficiency and model fidelity.

In contrast, GPUs and TPUs, optimized for parallel computation, benefit from structured sparsity, tensor decomposition (e.g., singular value and CP decomposition), and mixed-precision training. Modern GPUs support tensor core accelerations through reduced-precision arithmetic [e.g., 16-bit floating-point (FP16), INT8], enhancing inference speed while preserving numerical stability. TPUs, designed specifically for DL, exploit systolic array architectures, making matrix factorization-based compression especially effective. For these platforms, compression strategies must be adapted to maximize memory throughput and computational utilization, while avoiding irregular access patterns that reduce performance.

FPGAs and custom AI accelerators introduce additional complexity. Their reconfigurable architectures allow for workload-specific customization, supporting fine-grained parallelism and hardware-level optimizations. Techniques such as weight sharing, bit-width reduction, and model partitioning are particularly advantageous in these settings. In domains like real-time signal processing or industrial automation, FPGA-based inference engines can surpass conventional processors in power efficiency and throughput. However, optimal performance requires co-designing the compression strategy and hardware architecture, as suboptimal memory patterns can introduce critical performance bottlenecks.

Beyond selecting appropriate compression techniques for each hardware platform, it is essential to evaluate their deployment impact. Compressed models often exhibit varying trade-offs between accuracy and latency depending on the characteristics of the target hardware. For instance, highly quantized models [e.g., INT8 or 4-bit integer (INT4)] may execute efficiently on TPUs and dedicated mobile AI accelerators but often experience accuracy degradation on floating-point-oriented hardware, such as GPUs. Similarly, unstructured pruning can achieve high compression ratios but may result in inefficient memory access patterns on architectures that favor regular, vectorized operations. Knowledge distillation, while beneficial for improving model robustness in resource-constrained environments, introduces additional training overhead, which may hinder rapid deployment in certain scenarios.

To mitigate these challenges, hardware-aware automated tuning frameworks have been developed to dynamically adapt compression strategies to the target deployment platform. Approaches such as NAS and RL-based optimization can be employed to navigate the trade-offs between model compactness and hardware efficiency. Moreover, emerging platforms—including Google's Edge TPU, Apple's Neural Engine, and Qualcomm's Hexagon DSP—are increasingly integrating compiler-level optimizations that support adaptive compression, thereby reducing the need for manual fine-tuning.

Clinical NLP models optimized for electronic health record systems

Recent advancements in compressing clinical language models (CLMs) have demonstrated significant benefits in processing electronic health records for predictive tasks such as mortality risk estimation, length of stay prediction, diagnosis classification, and procedure planning. The OptimCLM framework integrates knowledge distillation, pruning, and quantization to enable efficient deployment in healthcare settings. Through ensemble learning, domain-specific fine-tuning, and post-training optimization, OptimCLM achieves high compression ratios and accelerated inference without sacrificing predictive accuracy [109].

In empirical evaluations, OptimCLM compresses models like BERT-PKD and TinyBERT, yielding compression ratios of up to 22.88× and inference speedups of 28.7× relative to the ensemble teacher model. Notably, the performance degradation remains below 5% for TinyBERT and under 2% for BERT-PKD across critical clinical prediction tasks.

This level of efficiency supports deployment in resource-constrained environments, including rural clinics, ambulances, and mobile health units, where latency, hardware limitations, and on-device inference are crucial. Moreover, the use of locally deployed models enhances patient privacy by minimizing data transmission and enabling secure inference on sensitive health records.

Collectively, these case studies underscore the transformative potential of LLM compression across domains. From enabling NLP on mobile devices to optimizing large-scale industrial systems and clinical applications, compressed models offer a compelling balance between performance and efficiency. By aligning model architectures with hardware constraints and application-specific demands, these techniques expand the practical applicability of LLMs in real-world deployments.

Proposed roadmap for scalable and efficient LLM compression

A central contribution of this work is the proposal of a structured roadmap for the development of efficient and high-performance compressed LLMs. This roadmap serves as a strategic framework to guide ongoing research and practical implementation in model compression. By addressing current limitations and integrating state-of-the-art methodologies, the proposed framework advances the field beyond ad hoc solutions, offering a cohesive vision for sustainable and equitable deployment.

The roadmap is anchored on four foundational pillars—adaptive compression, fairness, interpretability, and standardized evaluation—each of which targets a critical dimension of real-world applicability. Together, these pillars outline actionable directions to improve the scalability, robustness, and ethical viability of LLMs across diverse deployment scenarios. This holistic approach distinguishes the contribution by not only optimizing performance but also promoting responsible and context-aware model design.

Adaptive compression: Develop models that dynamically adjust size, architecture, and computational requirements based on deployment environments.

- Implement dynamic multi-objective optimization frameworks that balance latency, memory usage, and accuracy, enabling context-aware model adaptation for diverse applications.
- Design modular architectures that can be scaled down for edge devices or scaled up for high-throughput systems, ensuring structural flexibility without compromising core performance.
- Introduce task-specific compression techniques tailored to domain requirements (e.g., medical, legal, or financial tasks), optimizing efficiency while preserving model generalization and robustness.
- Integrate resource-aware decision logic that allows models to autonomously select compression modes (e.g., quantization levels or pruning depth) based on runtime hardware profiling.
- Outcome: Establish a class of versatile LLMs that maintain high utility across heterogeneous hardware settings, supporting scalable, real-world deployment.

Fairness in compression: Mitigate bias amplification during compression to ensure equitable performance across demographic and application-specific groups.

- Integrate fairness-aware compression strategies, such as bias-sensitive pruning and adversarial debiasing, to minimize disparities introduced during model size reduction.
- Develop and adopt fairness metrics specifically designed for compressed LLMs, supporting rigorous and consistent evaluation throughout the development lifecycle.
- Collaborate with domain experts to systematically identify context-dependent fairness risks and incorporate these insights into compression pipeline decisions.
- Outcome: Compressed LLMs that uphold ethical principles and deliver equitable predictive performance across diverse user groups and deployment contexts.

Enhancing interpretability: Preserve or improve interpretability to ensure transparency, accountability, and trust in high-stakes applications.

- Design compression-aware interpretability techniques, including attribution methods and structural visualizations, that account for changes induced by pruning, quantization, or architectural modifications.
- Establish post-compression interpretability assessment protocols to verify that model explanations remain coherent and meaningful after optimization.

- Customize interpretability approaches to meet the domain-specific demands of sectors such as healthcare, finance, and law, where explainability is legally or ethically mandated.
- Outcome: Transparent and accountable compressed LLMs capable of supporting decision-making in sensitive and regulated environments.

Standardized evaluation: Establish comprehensive and reproducible benchmarks to assess the multifaceted performance of compressed LLMs.

- Define unified evaluation criteria that integrate accuracy, inference latency, fairness, interpretability, energy efficiency, and hardware compatibility, capturing the full scope of real-world deployment needs.
- Develop and maintain open-source benchmarking suites to support rigorous comparative analysis, foster reproducibility, and accelerate methodological advancements.
- Perform in-depth case studies across diverse domains and deployment contexts to demonstrate benchmark applicability and reveal compression trade-offs in practical scenarios.
- Outcome: A robust and extensible evaluation framework that standardizes assessment practices and catalyzes progress in the field of model compression.

Figure 11 presents the proposed roadmap for the development of compressed LLMs. Each pillar is systematically connected to actionable research directions and corresponding practical implementations, culminating in outcomes that advance both technical efficiency and ethical robustness. The structure addresses current limitations in scalability, transparency, and equitable deployment of LLMs. Serving as both a conceptual framework and strategic reference, the figure helps researchers and practitioners navigate the complexities of designing compressed models that are efficient, fair, and interpretable for real-world deployment.

This roadmap serves as a practical guide for researchers and practitioners, providing clear, actionable steps to develop compressed LLMs that are efficient, fair, interpretable, and robust. By addressing the core pillars of adaptive compression, fairness, interpretability, and standardized evaluation, it supports the advancement of models capable of meeting the growing demand for high-performance solutions across diverse real-world applications.

Discussion and future directions

The field of LLM compression has made significant strides in reducing model size, improving inference speed, and enhancing energy efficiency. These advancements have facilitated broader deployment across diverse platforms and environments. However, key challenges remain—particularly in maintaining robustness, fairness, and interpretability under aggressive compression regimes. This section critically evaluates the current landscape, outlining core achievements, persisting limitations, and strategic directions for future research.

Persistent bottlenecks in LLM compression

Generalization preservation: Maintaining a model's ability to perform effectively on unseen or out-of-distribution data remains a critical challenge, particularly in real-world deployments. Compressed LLMs often underperform in such contexts, limiting their robustness in dynamic environments [4]. While techniques like pruning and quantization reduce model size, they frequently compromise performance on tasks requiring fine-grained reasoning or domain-specific understanding. Knowledge distillation offers a potential remedy, but its success hinges on the student model's capacity to faithfully replicate the teacher's generalization behavior.

Training time: Refers to the duration needed to train a model to reach satisfactory performance, which is a critical consideration in compression methods. It directly influences the feasibility of deploying compressed models in time-sensitive or resource-constrained settings. Knowledge distillation often involves a multi-step process in which a student model mimics a teacher model, followed by fine-tuning to recover potential performance losses, thereby extending training duration [69]. Similarly, pruning typically entails iterative removal and fine-tuning cycles, substantially increasing computational requirements [9].

Scalability: Refers to the ability of a method or system to maintain efficiency and performance as the size of the model or dataset increases—a critical challenge in the context of compressed LLMs. Techniques such as NAS, although effective for architecture optimization, introduce significant scalability issues due to their high computational demands and the need to explore vast parameter spaces [5]. Similarly, pruning and quantization face limitations when applied across heterogeneous hardware platforms, as sparsity-induced inefficiencies and precision adjustments often require extensive customization [63]. These constraints underscore the need for generalized and computationally efficient approaches to enable truly scalable solutions for LLM deployment.

Fairness, robustness, and bias amplification: Fairness refers to a model's ability to treat all demographic groups equitably, ensuring that its predictions do not disproportionately favor or disadvantage any specific group. Bias amplification, in contrast, occurs when a model exacerbates pre-existing biases in the training data, potentially resulting in skewed or discriminatory outcomes. Robustness is defined

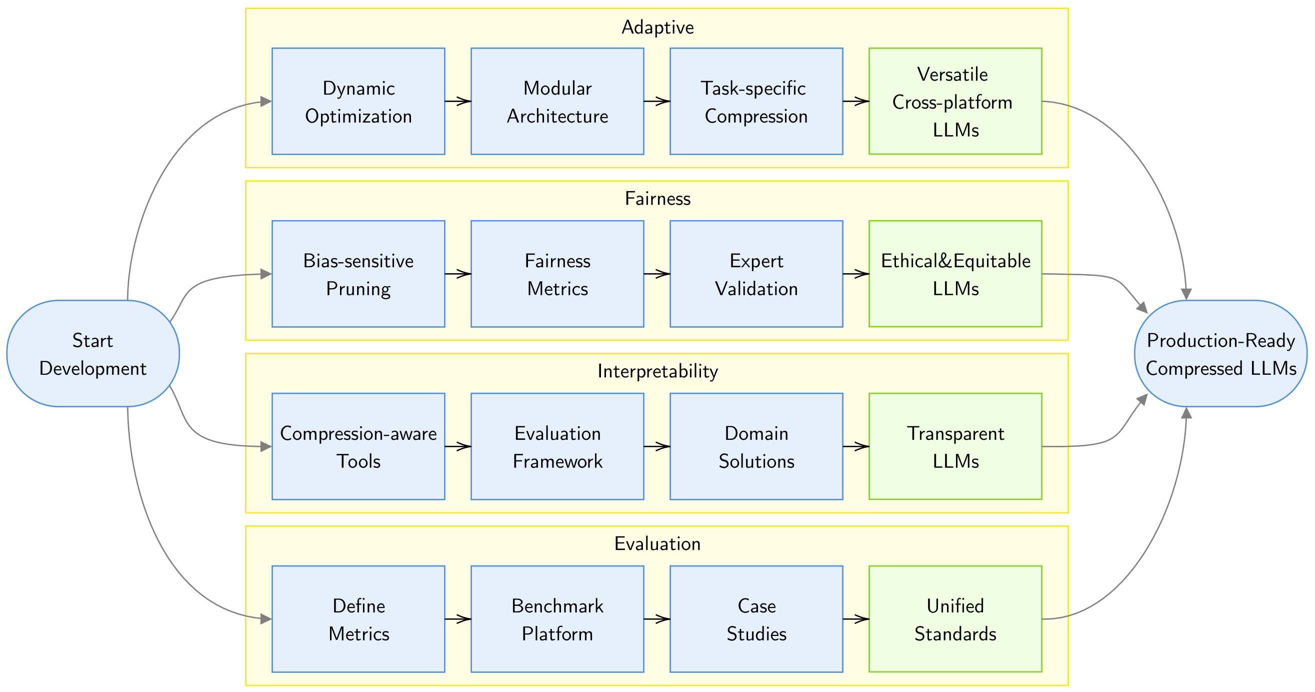

Fig. 11 This flowchart presents a systematic framework for advancing the efficiency, fairness, interpretability, and evaluation of compressed LLMs. Each pillar—adaptive compression, fairness, interpretability, and standardized evaluation—is linked to a set of actionable steps and leads to defined outcomes. Together, they address key challenges related to scalability, robustness, and practical applicability in diverse real-world deployment scenarios

as the model's capacity to maintain stable performance across a wide range of inputs, conditions, and scenarios, including adversarial perturbations and distributional shifts. A robust model should remain effective even in the presence of noise or unforeseen variations, making it essential for reliable deployment in real-world environments.

Ensuring fairness and robustness while mitigating bias amplification remains a critical challenge in the deployment of compressed LLMs. These concerns are especially pronounced with techniques such as pruning, quantization, and knowledge distillation (PQK), which can alter internal model representations in ways that magnify existing biases and reduce stability [86]. For instance, pruning may disproportionately remove parameters essential for accurate representation of minority groups, while quantization can distort decision boundaries, leading to uneven performance across demographic segments. Additionally, compressed models often exhibit increased sensitivity to adversarial inputs and distributional shifts, exacerbating fairness issues and reducing reliability in real-world settings. Addressing these limitations requires the development of compression strategies that preserve not only predictive accuracy but also fairness and robustness across deployment scenarios.

To mitigate these challenges, fairness- and robustness-aware strategies should be integrated throughout the entire compression pipeline. In the preprocessing stage, techniques such as dataset rebalancing and adversarial debiasing can address underlying societal biases and improve model stability prior to compression. During compression, incorporating fairness- and robustness-guided objectives—such as targeted pruning and precision-aware quantization—can help preserve critical features and decision pathways. These approaches reduce the likelihood of disproportionate degradation in performance for specific demographic groups and mitigate the model's susceptibility to adversarial perturbations [23].

Evaluation of fairness and robustness in compressed models is equally critical to their development. Quantitative metrics—such as demographic parity, disparate impact, equalized odds, and adversarial robustness—offer standardized means to assess bias and performance stability post-compression. Tools like Fairlearn and AIF360 provide comprehensive frameworks for operationalizing these assessments [110]. In parallel, fine-tuning strategies, including fairness-aware knowledge distillation and hybrid approaches integrating adversarial training, have shown efficacy in preserving equitable and robust behavior. For instance, fairness-guided pruning has reduced error rate disparities across demographic groups by more than 20% without significant accuracy degradation [23], while fairness-aware quantization techniques have effectively mitigated biased decision boundaries in gender-sensitive classification tasks [63].

Future research should focus on developing dynamic, fairness- and robustness-aware compression techniques that can adapt in real time based on evaluation feedback. Integrating bias-sensitive benchmarks and interpretability tools will be crucial to uncover how compression affects internal decision pathways, representation of marginalized groups, and vulnerability to perturbations. A comprehensive approach to fairness, robustness, and bias amplification will be essential to ensure that compressed LLMs are not only efficient but also reliable, ethical, and fit for deployment in high-stakes domains such as healthcare, finance, and legal systems.

Hardware-specific optimizations and compatibility: refer to the reliance of compression techniques on particular hardware architectures to achieve meaningful efficiency gains. Methods such as pruning and quantization often require platform-specific adaptations to leverage the strengths of accelerators like GPUs and TPUs, which support low-precision operations and optimized memory throughput [64]. However, these optimizations may underperform on general-purpose processors or hardware lacking support for such specialized features, thereby limiting model portability and scalability. Moreover, sparsity patterns introduced by pruning can result in irregular memory access, necessitating custom hardware or accelerators tailored to exploit structured sparsity [41]. These challenges underscore the importance of hardware-aware compression strategies that balance model efficiency with broad deployment compatibility.

Application-environment constraints: refer to the operational limitations posed by specific deployment contexts, including resource availability, latency requirements, and energy consumption. In edge environments and embedded systems, compressed LLMs must strike a balance between computational efficiency and predictive performance [30]. Real-time applications—such as interactive conversational agents or autonomous systems—require low-latency responses, which can be impeded by the computational overhead introduced by certain compression strategies. Additionally, battery-powered devices necessitate energy-efficient execution, calling for compression methods that minimize power usage without incurring significant accuracy degradation.

Adaptability to diverse tasks: denotes the capacity of a model to perform effectively across varied tasks without substantial retraining or fine-tuning. While many compression techniques are optimized for standard benchmarks, they often struggle with real-world challenges requiring domain sensitivity, contextual understanding, or multilingual robustness [2]. Compressed models may lose critical representational capacity—particularly through pruning or aggressive quantization—resulting in diminished performance in tasks such as domain-specific question answering or low-resource language modeling [34].

Hybrid compression approaches: integrate multiple strategies—such as PQK—to exploit their complementary strengths and achieve higher efficiency-performance trade-offs. For example, applying knowledge distillation to a pruned or quantized model can help recover lost accuracy, improving generalization in compressed architectures. However, these approaches introduce additional design and tuning complexity, requiring careful evaluation to ensure robustness across tasks and platforms. Simplifying these hybrid pipelines is a key step toward enabling widespread and practical adoption, particularly in non-specialist settings [63].

Taken together, these considerations underscore the importance of next-generation compression techniques that address fairness, robustness, task adaptability, and hardware compatibility—critical elements for scalable, ethical, and efficient deployment of LLMs in real-world environments.

Evaluation metrics and standardization: The lack of universally accepted evaluation benchmarks remains a significant bottleneck in advancing LLM compression. Existing frameworks predominantly emphasize metrics such as accuracy or PPL, which fail to capture the multidimensional trade-offs encountered in real-world deployments. Future research must prioritize the development of holistic benchmarking standards that incorporate latency-accuracy trade-offs, memory efficiency, robustness, fairness, and energy consumption. Establishing these standards will require close collaboration between academia and industry to enable reproducible comparisons and accelerate the adoption of effective compression strategies.

Software-hardware co-design: The next generation of efficient compression solutions will increasingly rely on the joint optimization of software algorithms and hardware architectures. Aligning model structures with the constraints and capabilities of specific hardware (e.g., GPUs, TPUs, or specialized AI accelerators) can yield substantial improvements in performance and energy efficiency. Techniques such as sparsity-aware scheduling, dynamic quantization, and tensor-core exploitation highlight the benefits of co-design. Future work should focus on creating adaptive compression schemes that generalize across heterogeneous platforms while maximizing hardware utilization.

Fairness-aware compression: Ensuring fairness in compressed LLMs is particularly critical when deployed in sensitive domains such as law, finance, and healthcare. Techniques like fairness-aware pruning, adversarial debiasing, and fairness-guided quantization offer promising avenues to mitigate bias amplification during compression. Embedding fairness objectives directly into training and evaluation pipelines can foster equitable model behavior, ensuring demographic parity, equalized odds, or other fairness metrics across groups and use cases [111].

Adaptive and multi-objective compression: Adaptive compression frameworks allow models to dynamically adjust computational costs and representational complexity based on operational constraints. Techniques such as runtime pruning and on-the-fly quantization make it possible to meet stringent latency or power budgets without re-training. Multi-objective optimization, particularly Pareto-based methods, provides a principled way to balance competing goals like accuracy, inference speed, and robustness. Research into scalable and low-overhead optimization algorithms will be essential to operationalize such adaptive strategies in real-world settings [73].

Cross-lingual and domain-specific compression: The current focus on general-purpose language models leaves significant gaps in performance and efficiency for multilingual and domain-specific applications. As models like BioBERT and XLM-R become essential in specialized fields, compression methods must account for linguistic diversity and domain nuances [112]. Future directions include developing compression-aware transfer learning methods, sparse adaptation layers, and domain-conditioned quantization strategies to address the needs of underrepresented languages and domains.

Integrating interpretability and explainability: Compression techniques often obscure internal model dynamics, raising concerns around trust, especially in high-stakes applications. Incorporating interpretability methods—such as saliency maps, attention visualization, or SHAP-based feature attribution—into the compression process can help retain transparency [113]. Further research should explore compression-aware interpretability metrics and explainable-by-design model architectures that support both human-understandable insights and performance efficiency.

Privacy-aware compression: With increasing emphasis on data privacy, particularly under regulatory frameworks like GDPR and HIPAA, compression strategies must be designed with privacy considerations in mind [114]. Techniques such as federated learning and secure multi-party computation can be integrated with compression to enable decentralized model deployment without compromising data integrity. This line of research should investigate compression-aware privacy guarantees, minimizing the attack surface in both training and inference.

Integrating data compression with model compression: The centralized inference architecture typical of LLM deployment—exemplified by applications such as ChatGPT—poses significant challenges for bandwidth and scalability. In contrast to computer vision or speech recognition, where on-device inference is common, LLMs often require user inputs to be transmitted to powerful servers. To mitigate network bottlenecks, integrating data compression with model compression presents a promising research frontier. Compressed sensing offers a compelling solution by leveraging the inherent sparsity of high-dimensional representations (e.g., embeddings or activations) [9, 82]. This technique allows for efficient signal reconstruction from fewer transmitted measurements, enabling bandwidth-efficient deployment without compromising model fidelity. Future research could extend these ideas to co-optimized pipelines that simultaneously reduce transmission overhead and inference costs.

In the context of LLM deployment, compressed sensing can be strategically applied at multiple stages to enhance efficiency and scalability:

- *Input data compression*: Prior to transmission, compressed sensing can encode user inputs—such as natural language queries—into compact, sparse representations. This substantially reduces data size without significant loss of semantic fidelity, thereby mitigating bandwidth constraints in client–server architectures.
- *Intermediate feature compression*: During inference, LLMs produce high-dimensional embeddings and feature maps. Compressed sensing techniques can compress these intermediate representations, reducing memory usage and computational load—particularly beneficial in split-computation scenarios where partial processing occurs on edge or mobile devices.
- *Joint optimization with model compression*: Compressed sensing can be synergistically integrated with model compression strategies such as pruning and quantization. For example, sparsity-aware pruning can align model structures with the compressed, sparse nature of input data, enabling efficient inference pipelines while minimizing accuracy degradation.

Sparse recovery methods have been shown to reduce transmission overhead by up to 50% while retaining over 95% task accuracy relative to full models [8, 46]. Furthermore, combining compressed sensing with quantization has yielded substantial memory savings in latency-sensitive applications [82]. Nonetheless, challenges remain. The effectiveness of compressed sensing hinges on the choice of sparsifying transforms and recovery algorithms, which may introduce computational overhead during both encoding and decoding. Moreover, harmonizing data compression with model compression requires the design of specialized architectures and co-training procedures that are both efficient and generalizable.

Future research should explore hybrid frameworks that embed compressed sensing throughout the LLM lifecycle—from input encoding to inference and deployment. Key directions include optimizing the trade-offs between compression ratio, recovery fidelity, and computational complexity, and establishing architectures capable of exploiting sparsity natively.

By bridging the traditionally distinct domains of data and model compression, compressed sensing offers a promising pathway toward scalable and energy-efficient LLM deployments. This approach has the potential to alleviate the dependence on centralized infrastructure and unlock practical deployment on resource-constrained edge devices.

Towards hardware-aligned and sustainable model optimization

Modern LLM compression must align with the constraints of target hardware platforms [33, 115]. Deployment settings—including edge devices, GPUs and TPUs—shape the trade-offs between efficiency and accuracy, influencing how pruning, quantization, and distillation are applied. A hardware-aware approach optimizes these techniques for platform-specific performance (e.g., speed, memory, energy) while maintaining acceptable accuracy [12, 41]. The discussion below outlines how each method adapts to various platforms and the resulting deployment trade-offs.

Edge devices: precision-aware quantization and distillation: Edge deployment demands overcoming hardware constraints like limited power, memory, and compute. Compression techniques—particularly low-precision quantization and knowledge distillation—are vital [81]. Quantization can reduce model size and latency by up to 4×; for example, 8-bit BERT retains 99% accuracy [79]. Precision-aware methods further align models with hardware operations, such as INT8 execution on mobile neural processing units (NPUs), minimizing performance degradation.

Uniform aggressive quantization (e.g., 4-bit, binary) often degrades accuracy. Mixed-precision approaches mitigate this by allocating lower bit-widths to less sensitive layers, preserving critical computations [31]. Knowledge distillation further aids compression by training compact student models to emulate larger teachers. DistilBERT, for instance, retains 97% of BERT's performance while being 40% smaller and 60% faster [69].

Recent studies show quantized transformers achieving real-time performance on microcontrollers with just 64 KB RAM [116], demonstrating the viability of deploying LLMs at the edge. These advances balance compression with accuracy, enabling responsive AI applications under strict resource constraints.

GPUs: structured pruning and mixed-precision acceleration: Effective compression for GPU deployment requires alignment with GPU execution patterns, particularly in handling sparsity and numeric precision. Fine-grained, unstructured pruning introduces irregular memory access, limiting speedups [117]. In contrast, structured pruning—removing entire neurons, filters, or heads—yields dense sub-tensors optimized for GPU kernels [118]. NVIDIA's 2:4 sparsity (two zeros per four values) exemplifies this, doubling throughput via specialized tensor cores [117].

Recent studies show that 2:4 sparsity in Transformers improves throughput by 30% and reduces latency by 20%, with minimal accuracy loss after fine-tuning [119]. While structured pruning may remove more critical weights, it guarantees practical acceleration by leveraging GPU-specific hardware. GPUs also benefit from native support for low-precision arithmetic (e.g., FP16, 16-bit brain floating point (BF16), INT8). Reducing precision cuts memory usage by 2-4× and increases operations per cycle. Mixed-precision execution—such as FP16 weights with FP32 accumulation—maintains accuracy while standardizing speedups [65].

Thus, combining structured sparsity with quantization provides a balanced strategy for efficient GPU deployment, enabling large models—e.g., 13B with 4-bit weights—to run on a single GPU, improving throughput per Watt and per dollar.

TPUs: architectural simplifications and parallelism: TPUs are optimized for dense matrix operations via systolic arrays, making them highly efficient for structured computations [115]. Irregular sparsity disrupts this regularity, causing compute underutilization. As a result, compression techniques that preserve structural coherence—such as block sparsity and low-rank decomposition—are preferred.

Low-rank factorization reduces computation by replacing large matrices with smaller, dense ones while maintaining TPU efficiency. Pruned weights must be repacked into dense blocks to avoid performance degradation from imbalanced workloads [115]. Model simplification through distillation or layer reduction can improve fit within on-chip memory, enhancing energy efficiency. However, excessive pruning risks underutilizing TPU cores, negating speed benefits [120].

TPUs support mixed-precision (e.g., BF16) natively, and recent versions allow INT8 inference, reducing memory usage and accelerating computations [115]. Effective TPU compression typically combines:

- Structured pruning and distillation to reduce inter-core communication.
- Low-precision quantization to save memory and improve throughput.
- Execution optimizations for dense, batched workloads.

Together, these strategies enhance energy efficiency and latency while ensuring parallelism and bandwidth utilization. Maintaining regular computation patterns is essential to fully exploit TPU architecture [15, 121].

Conclusion

This review has synthesized state-of-the-art techniques for compressing LLMs, underscoring their central role in improving computational efficiency, energy usage, and deployment scalability. Through a critical analysis of methods such as pruning, quantization, knowledge distillation, and NAS, it becomes clear that these strategies are essential for adapting LLMs to real-world constraints. Their successful application across mobile, edge, and clinical settings illustrates the growing maturity and practical value of compression techniques, marking a significant step toward broader, more sustainable use of AI.

The presented case studies underscore the growing need for adaptive and hybrid compression techniques that can dynamically balance performance and efficiency in accordance with deployment constraints. Emerging advances—such as fairness-aware compression, robustness-enhancing methods, and hardware-specific optimizations—further contribute to mitigating bias, strengthening generalization, and improving scalability. Additionally, the integration of NAS into the compression pipeline offers a compelling pathway to automate architecture selection and enable hardware-aware model design, reinforcing the potential for efficient and equitable LLM deployment across diverse environments.

Despite these advancements, key challenges persist. Future research should prioritize the development of standardized evaluation frameworks, lightweight NAS techniques, and cross-platform optimization strategies to extend the applicability of compressed LLMs. Addressing these challenges will require close collaboration between academia and industry to bridge the gap between theoretical progress and real-world deployment.

By synthesizing current research and proposing a structured roadmap for scalable and efficient LLM compression, this review offers actionable guidance for advancing scalable and efficient AI systems. As demand for accessible and sustainable AI continues to rise, these compression strategies will be essential for ensuring that the next generation of LLMs remains both powerful and practically deployable across diverse hardware environments.

Glossary

AI - Artificial intelligence is the field of computer science concerned with the design and development of systems capable of performing tasks that typically require human intelligence, such as reasoning, learning, perception, and decision-making.

AUC-ROC - Area Under the curve of receiver operating characteristic is a measure of the ability of a classifier to distinguish between classes.

AWQ - Activation-aware weight quantization is a quantization technique for large language models (LLMs) that leverages activation information to guide the weight quantization process, improving accuracy during low-bit inference.

BERT - Bidirectional encoder representations from transformers is designed for understanding the meaning of language by reading text bidirectionally.

BF16 - 16-bit brain floating point is a 16-bit floating-point format with the same exponent range as FP32 but fewer mantissa bits, allowing for faster computation and lower memory usage with minimal loss in accuracy.

BLEU - Bilingual evaluation understudy is a metric for evaluating machine-generated text against reference translations, primarily used in machine translation evaluation.

CLMs - Clinical language models are specialized machine learning (ML) models designed for processing and analyzing medical text, enabling tasks, such as diagnosis prediction, procedure planning, and clinical decision support.

CNN - Convolutional neural network is a class of deep neural networks (DNNs), most commonly applied to analyzing visual imagery.

CoNLL F1 - Conference on natural language learning F1-score is a metric used for evaluating named entity recognition and chunking tasks, calculated by comparing predicted entity spans with gold standard annotations. It considers both the entity type and boundary accuracy, providing a harmonious balance between precision and recall in entity recognition tasks.

CRFM - Stanford Center for Research on Foundation Models is a research center at Stanford focused on advancing foundation models.

DARTS - Differentiable architecture search is a neural architecture search (NAS) method that optimizes model architectures using gradient-based techniques instead of reinforcement learning (RL) or evolutionary algorithms.

DL - Deep learning is a subset of machine learning (ML) in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled.

DNN - Deep neural network is a neural network with a certain level of complexity, having multiple layers between the input and output layers.

EM - Exact match is a strict evaluation metric that measures whether a model's predicted answer matches the reference answer exactly, commonly used in question answering tasks where the score is 1 if the prediction matches the ground truth perfectly and 0 otherwise.

FG - Fairness gap is a metric that quantifies the difference in model performance or predictions across different demographic groups or protected attributes, measuring the level of algorithmic bias and disparate impact in machine learning (ML) systems.

FP16 - 16-bit floating-point is a numerical format using 16 bits to represent real numbers, offering reduced memory usage and faster computation at the cost of lower precision. Commonly used in mixed-precision training and inference.

FP32 - 32-bit floating-point is a numerical format that uses 32 bits to represent real numbers, typically following the IEEE 754 standard.

FPGA - Field-programmable gate array is an integrated circuit designed to be configured by the customer or designer after manufacturing.

GLaM - Generalist language model is a mixture-of-experts language model developed by Google, designed to improve efficiency by activating only a subset of model parameters during inference.

GNN - Graph neural network is a class of deep learning (DL) models designed to capture the structural and relational information in graph-structured data by operating on nodes, edges, and their features through message passing and neighborhood aggregation mechanisms.

GPT - Generative pre-trained transformer is a model that uses a transformer architecture to produce human-like text.

GPU - Graphic processing unit is a specialized processor used to accelerate graphics and computeintensive tasks.

HELM - Holistic evaluation of language models is a comprehensive benchmark for evaluating language models (LLMs) on multiple criteria.

HPC - High-performance computing is a infrastructure that combines multiple computing nodes and resources to deliver significantly higher processing power than general-purpose computers, enabling the execution of complex computational tasks and parallel processing of large-scale data.

INT4 - 4-bit integer is an low-precision numerical format using 4-bit signed integers, often applied in aggressive model quantization to reduce memory usage and accelerate inference with minimal accuracy degradation in well-optimized scenarios.

INT8 - 8-bit integer is a numerical format using 8-bit signed integers, primarily used in model quantization to reduce memory demands and accelerate inference while maintaining acceptable accuracy.

IoT - Internet of things is the network of physical objects-devices, vehicles, buildings, and other itemsembedded with electronics, software, sensors, and network connectivity that enables these objects to collect and exchange data.

LAT - Latency-accuracy trade-off is a metric that evaluates the relationship between model performance (accuracy) and computational efficiency (latency), helping to assess the balance between prediction quality and processing speed in machine learning (ML) systems.

LLaMA - Large language model by Meta AI (LLaMA) is a family of open-source large language models (LLMs) developed by Meta AI, designed to be more efficient and accessible than previous models while maintaining strong performance across various natural language processing (NLP) tasks.

LLM - Large language model is a type of artificial intelligence (AI) model trained on vast text data to understand and generate human language.

LoRA - Low-rank adaptation is a parameter-efficient fine-tuning method that adds pairs of rank decomposition matrices to existing weights, enabling model adaptation while maintaining most parameters frozen and significantly reducing memory requirements.

MAE - Memory access efficiency is a metric that quantifies how effectively a model utilizes memory resources by measuring the ratio of useful memory operations to total memory accesses, indicating the optimization level of memory usage patterns and cache utilization.

ML - Machine learning is a subset of artificial intelligence (AI) that focuses on building systems that learn from and make decisions based on data.

MLCommons - Machine learning commons is a non-profit organization that develops open engineering practices, public datasets, and benchmarks to improve the reproducibility and transparency of machine learning (ML) performance evaluations.

MLPerf - Machine learning performance is a benchmarking suite for evaluating the performance of machine learning (ML) models across different hardware and tasks.

NAS - Neural architecture search is a technique for automating the design of neural network architectures.

NLP - Natural language processing is the field of artificial intelligence (AI) focused on enabling machines to understand and process human language.

NPU - Neural processing unit is a specialized hardware accelerator designed to efficiently perform deep learning operations, particularly those involving matrix multiplications and neural network inference

OFA - Once-for-all is a neural architecture search (NAS) approach that trains a single neural network containing many architectural choices, allowing efficient deployment across different devices by selecting specific sub-networks without additional training.

PaLM - Pathways language model is a large language model (LLM) developed by Google AI that uses a pathways system architecture to process tasks more efficiently, demonstrating strong performance across hundreds of language tasks and reasoning capabilities.

PPL - Perplexity is a metric used to evaluate the performance of probabilistic models, particularly in natural language processing (NLP). It measures how well a model predicts a sample by assessing the uncertainty in its probability distribution, with lower perplexity indicating better predictive accuracy.

PQK - Pruning, quantization, and knowledge distillation is a hybrid model compression approach that sequentially applies pruning, quantization-aware training (QAT), and knowledge distillation to reduce model size and improve inference efficiency while retaining performance.

PTQ - Post-training quantization is a technique to reduce the precision of model weights after training to decrease model size and improve efficiency.

PwC - Papers with code is a platform for tracking the latest research and code implementations in machine learning (ML).

QAT - Quantization-aware training is a technique for training models with quantization in mind to maintain accuracy in low-precision formats.

QLoRA - Quantized low-rank adapter is a parameter-efficient fine-tuning method for large language models (LLMs) that combines quantization with low-rank adapters to reduce memory usage while maintaining performance.

QRPK - Quantization-robust pruning with knowledge distillation is a hybrid model compression technique that integrates pruning, quantization, and knowledge distillation to maintain robustness and accuracy in highly compressed deep learning (DL) models.

RAG - Retrieval-augmented generation is a neural architecture that combines a retrieval component with a generative model, enabling the system to condition outputs on relevant external documents retrieved at inference time.

RAM - Random access memory is a type of volatile memory used by computers and processors to temporarily store data that is actively being used or processed. In machine learning (ML), it is critical for holding intermediate computations and model parameters during training and inference.

RL - Reinforcement learning is a machine learning (ML) paradigm where an agent learns to make decisions by interacting with an environment, receiving rewards or penalties for its actions, and optimizing its behavior to maximize cumulative rewards over time.

RNN - Recurrent neural network is a type of deep neural network (DNN) designed to recognize patterns in sequences of data by maintaining an internal state (memory) that allows it to process sequential information by incorporating information from previous inputs.

ROUGE - Recall-oriented understudy for gisting evaluation is a set of metrics used to evaluate automatic summarization and machine translation by comparing generated text against reference texts, focusing on overlap of n-grams, word sequences, and word pairs.

RS - Robustness score is a metric that quantifies a model's ability to maintain consistent performance across different scenarios, perturbations, and input variations, providing a measure of model reliability and stability under various conditions.

TPU - Tensor processing unit is a processor developed by Google specifically for accelerating machine learning (ML) tasks.

YOLO - You only look once is a real-time object detection algorithm that processes images in a single pass, dividing the image into a grid and predicting bounding boxes and class probabilities simultaneously, offering a balance of speed and accuracy for diverse applications.

Acknowledgements The authors would like to thank the Center for Electronic and Information Technology (CETELI) and the Department of Electrical Engineering at the Federal University of Amazonas (UFAM), Brazil, for providing the infrastructure, technical support, and access to state-of-the-art resources that were fundamental to this research. The authors would like to express their gratitude to the Department of Computer Science at the University of Manchester (UoM) and the Systems and Software Security (S3) Research Group for their invaluable support, collaborative environment, and access to cutting-edge resources, which were instrumental in the success of this research. The authors acknowledge the Funda o de Amparo Pesquisa do Estado do Amazonas (FAPEAM), Brazil, for its financial support, essential for the development of this research. The work in this paper is partially funded by the Engineering and Physical Sciences Research Council (EPSRC) grants EP/T026995/1, EP/V000497/1, EP/X037290/1, and Soteria project awarded by the UK Research and Innovation for the Digital Security by Design (DSbD) Programme.

Author Contributions All authors contributed to the conception and design of the study. All authors performed material preparation, literature review, and analysis. The first draft of the manuscript was written by Pierre Dantas. Waldir Junior and Celso Carvalho commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Data availability statement No datasets were generated or analyzed during the current study. The experimental results presented, such as those in Table 3, were compiled from previously published studies. All data supporting these results are available in the cited literature.

Declarations

Conflict of interest All authors certify that they have no affiliations or involvement in any organization or entity with any financial or non-financial interest in the subject matter or materials discussed in this manuscript.

Ethical and informed consent for the data used Adherence to the ethical standards and principles of informed consent, we confirm that all data used in this manuscript were collected and used with due regard for ethical norms. The study was conducted transparently, and all authors made substantial contributions as detailed in the *Authors Contribution Statement* section. We affirm our unwavering commitment to ethical research practices and informed consent, assuring that this research aligns with established ethical guidelines.

References

1. Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R (2020) Albert: a lite BERT for self-supervised learning of language representations. In: International conference on learning representations. https://openreview.net/forum?id=H1eA7AEtvS
2. Lian S, Zhao K, Liu X, Lei X, Yang B, Zhang W, Wang K, Liu Z (2024) What is the best model? Application-driven evaluation for large language models. Springer Nature Singapore, pp 67–79. https://doi.org/10.1007/978-981-97-9437-9_6
3. Bahri Y, Dyer E, Kaplan J, Lee J, Sharma U (2024) Explaining neural scaling laws. Proc Natl Acad Sci 121(27):e2311878121. https://doi.org/10.1073/pnas.2311878121
4. Zhou Z, Ning X, Hong K et al (2024) A survey on efficient inference for large language models. https://doi.org/10.48550/arxiv.2404.14294
5. Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le Quoc V (2019) MnasNet: platform-aware neural architecture search for mobile. In 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR). IEEE, pp 2815–2823. https://doi.org/10.48550/arxiv.1807.11626
6. Lang J, Guo Z, Huang S (2024) A comprehensive study on quantization techniques for large language models. In: 2024 4th international conference on artificial intelligence, robotics, and communication (ICAIRC). IEEE, pp 224–231. https://doi.org/10.1109/icairc64177.2024.10899941
7. Howard A, Sandler M, Chen B, Wang W, Chen L-C, Tan M, Chu G, Vasudevan V, Zhu Y, Pang R, Adam H, Le Quoc V (2019) Searching for MobileNetV3. In 2019 IEEE/CVF international conference on computer vision (ICCV). IEEE, pp 1314–1324. https://doi.org/10.1109/iccv.2019.00140
8. Wan Z, Wang X, Liu C, Alam S et al (2023) Efficient large language models: a survey. https://doi.org/10.48550/arxiv.2312.03863
9. Ukil Arijit, Sahu Ishan, Biswas Mridul, Pal Arpan, Majumdar Angshul (April 2024) Structured Lottery Ticket Hypothesis for Effective Deep Neural Network Model Size Reduction. In 2024 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW), page 330–334. IEEE. https://doi.org/10.1109/icasspw62465.2024.10625908
10. Gholami A, Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K (2022) A survey of quantization methods for efficient neural network inference. Chapman and Hall/CRC, pp 291–326. https://doi.org/10.1201/9781003162810-13
11. Wu C, Wu F, Huang Y (2021) One teacher is enough? Pre-trained language model distillation from multiple teachers. In: Findings of the association for computational linguistics: ACL-IJCNLP 2021. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.findings-acl.387
12. Chitty-Venkata KT, Mittal S, Emani M, Vishwanath V, Somani AK (2023) A survey of techniques for optimizing transformer inference. J Syst Architect 144:102990. https://doi.org/10.1016/j.sysarc.2023.102990
13. Verdecchia R, Sallou J, Cruz L (2023) A systematic review of Green AI. WIREs Data Min Knowl Discov 13(4):e1507. https://doi.org/10.1002/widm.1507
14. Chen Y-H, Krishna T, Emer JS, Sze V (2017) Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J Solid-State Circuits 52(1):127–138. https://doi.org/10.1109/jssc.2016.2616357
15. Kim S-G, Noh K, Hahn H, Choi BK (2024) Instruction fine-tuning and LoRA combined approach for optimizing large language models. J Soc Korea Ind Syst Eng 47(2):134–146. https://doi.org/10.11627/jksie.2024.47.2.134
16. LeCun Y, Denker J, Solla S (1989) Optimal brain damage. In: Touretzky D (ed) Advances in neural information processing systems, vol 2. Morgan-Kaufmann. https://doi.org/10.5555/109230.109298
17. Hassibi B, Stork DG, Wolff GJ (1993) Optimal brain surgeon and general network pruning. In: IEEE international conference on neural networks. IEEE, p 293–299. https://doi.org/10.1109/icnn.1993.298572
18. Han S, Pool J, Tran J, Dally WJ (2015) Learning both weights and connections for efficient neural networks. In: Proceedings of the 29th international conference on neural information processing systems—volume 1, NIPS'15. MIT Press, Cambridge, pp 1135–1143. https://doi.org/10.5555/2969239.2969366
19. Lin J, Rao Y, Lu J, Zhou J (2017) Runtime neural pruning. In: Proceedings of the 31st international conference on neural information processing systems, NIPS'17. Curran Associates Inc., Red Hook, pp 2178–2188. https://doi.org/10.5555/3294771.3294979
20. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. IEEE Trans Neural Netw 20(1):61–80. https://doi.org/10.1109/tnn.2008.2005605
21. Liu J (2022) Distilling graph neural networks. Springer International Publishing, pp 131–151. https://doi.org/10.1007/978-3-031-16174-2_7
22. Guo Z, Zhang C, Fan Y, Tian Y, Zhang C, Chawla NV (2023) Boosting Graph Neural Networks via Adaptive Knowledge Distillation. In: Proceedings of the AAAI conference on artificial intelligence, vol 37, no 6, pp 7793–7801. https://doi.org/10.1609/aaai.v37i6.25944
23. Kumar P (2024) Adversarial attacks and defenses for large language models (LLMs): methods, frameworks & challenges. Int J Multimed Inf Retr 13(3):26. https://doi.org/10.1007/s13735-024-00334-8
24. Taveekitworachai P, Suntichaikul P, Nukoolkit C, Thawonmas R (2024) Speed up! Cost-effective large language model for ADAS via knowledge distillation. In: 2024 IEEE intelligent vehicles symposium (IV). IEEE, pp 1933–1938. https://doi.org/10.1109/iv55156.2024.10588799
25. Dundar G, Rose K (1995) The effects of quantization on multilayer neural networks. IEEE Trans Neural Netw 6(6):1446–1451. https://doi.org/10.1109/72.471364
26. Withagen H (1994) Reducing the effect of quantization by weight scaling. In: Proceedings of 1994 IEEE international conference on neural networks (ICNN-94), vol 4, ICNN-94. IEEE, pp 2128–2130. https://doi.org/10.1109/icnn.1994.374544
27. Chen T, Zidong D, Sun N, Wang J, Chengyong W, Chen Y, Temam O (2014) DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGARCH Comput Archit News 42(1):269–284. https://doi.org/10.1145/2654822.2541967
28. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning. Trained Quant Huffman Coding. https://doi.org/10.48550/arxiv.1510.00149
29. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) XNOR-Net: imageNet classification using binary convolutional neural networks. Springer International Publishing, pp 525–542. https://doi.org/10.1007/978-3-319-46493-0_32
30. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: 2018 IEEE/CVF conference on computer vision and pattern recognition. IEEE, pp 2704–2713. https://doi.org/10.1109/cvpr.2018.00286

31. Courbariaux M, Hubara I, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1. https://doi.org/10.48550/arxiv.1602.02830
32. Ding J, Wu J, Wu H (2017) Three-means ternary quantization. Springer International Publishing, pp 235–245. https://doi.org/10.1007/978-3-319-70096-0_25
33. Deng BL, Li G, Han S, Shi L, Xie Y (2020) Model compression and hardware acceleration for neural networks: a comprehensive survey. Proc IEEE 108(4):485–532. https://doi.org/10.1109/jproc.2020.2976475
34. Bibi U, Mazhar M, Sabir D, Butt M, Hassan A, Ghazanfar MA, Khan AA, Abdul W (2024) Advances in pruning and quantization for natural language processing. IEEE Access 12:139113–139128. https://doi.org/10.1109/access.2024.3465631
35. Reguero ÁD, Martínez-Fernández S, Verdecchia R (2025) Energy-efficient neural network training through runtime layer freezing, model quantization, and early stopping. Comput Stand Interfaces 92:103906. https://doi.org/10.1016/j.csi.2024.103906
36. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. https://doi.org/10.48550/arxiv.1503.02531
37. Xu K, Rui L, Li Y, Gu L (2020) Feature normalized knowledge distillation for image classification. Springer International Publishing, pp 664–680. https://doi.org/10.1007/978-3-030-58595-2_40
38. Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evol Comput 10(2):99–127. https://doi.org/10.1162/106365602320169811
39. Nakai K, Matsubara T, Uehara K (2020) Att-DARTS: differentiable neural architecture search for attention. In: 2020 international joint conference on neural networks (IJCNN). IEEE, pp 1–8. https://doi.org/10.1109/ijcnn48605.2020.9207447
40. Kim Y, Li Y, Park H, Venkatesha Y, Yin R, Panda P (2022) Exploring lottery ticket hypothesis in spiking neural networks. Springer Nature Switzerland, pp 102–120. https://doi.org/10.1007/978-3-031-19775-8_7
41. Wang Y, Qin Y, Liu L, Wei S, Yin S (2021) HPPU: an energy-efficient sparse DNN training processor with hybrid weight pruning. In: 2021 IEEE 3rd international conference on artificial intelligence circuits and systems (AICAS). IEEE, pp 1–4. https://doi.org/10.1109/aicas51828.2021.9458410
42. Gou J, Baosheng Yu, Maybank SJ, Tao D (2021) Knowledge distillation: a survey. Int J Comput Vis 129(6):1789–1819. https://doi.org/10.1007/s11263-021-01453-z
43. Fedus W, Zoph B, Shazeer N (2022) Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. J Mach Learn Res 23(120):1–39 (http://jmlr.org/papers/v23/21-0998.html)
44. Choukroun Y, Kravchik E, Yang F, Kisilev P (2019) Low-bit quantization of neural networks for efficient inference. In: 2019 IEEE/CVF international conference on computer vision workshop (ICCVW). IEEE. https://doi.org/10.1109/iccvw.2019.00363
45. Lewis P, Perez E, Piktus A, Petroni F et al (2020) Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds) Advances in neural information processing systems, vol 33. Curran Associates, Inc, pp 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf
46. Dao T, Fu D, Ermon S, Rudra A, Ré C (2022) FlashAttention: fast and memory-efficient exact attention with IO-awareness. In: Koyejo S, Mohamed S, Agarwal A, Belgrave D, Cho K, Oh A (eds) Advances in neural information processing systems, vol 35. Curran Associates, Inc, pp 16344–16359. https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf
47. Bai Y, Jones A et al (2022) Training a helpful and harmless assistant with reinforcement learning from human feedback. https://doi.org/10.48550/arxiv.2204.05862
48. Bai Y, Kadavath S, Kundu S, Askell A et al (2022) Constitutional AI: harmlessness from AI feedback. https://doi.org/10.48550/arxiv.2212.08073
49. Anthropic (2024) Claude 3 technical overview. https://www.anthropic.com/index/claude-3-family
50. Kim B-K, Kim G, Kim T-H, Castells T, Choi S, Shin J, Song H-K (2024) Shortened LLaMA: depth pruning for large language models with comparison of retraining methods. https://doi.org/10.48550/arxiv.2402.02834
51. Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L (2023) QLoRA: efficient finetuning of quantized LLMs. https://doi.org/10.48550/arxiv.2305.14314
52. Lin J, Tang J, Tang H, Yang S, Chen W-M, Wang W-C, Xiao G, Dang X, Gan C, Han S (2023) AWQ: activation-aware weight quantization for LLM compression and acceleration. https://doi.org/10.48550/arxiv.2306.00978
53. Muralidharan S, Sreenivas ST, Joshi R, Chochowski M, Patwary M, Shoeybi M, Catanzaro B, Kautz J, Molchanov P (2024) Compact language models via pruning and knowledge distillation. https://doi.org/10.48550/arxiv.2407.14679
54. Anil R, Dai Andrew M et al (2023) PaLM 2 technical report
55. Hsieh C-Y, Li C-L, Yeh C-K, Nakhost H, Fujii Y, Ratner A, Krishna R, Lee C-Y, Pfister T (2023) Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes. In: Findings of the association for computational linguistics: ACL 2023. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-acl.507
56. Papers with Code (2024) Papers with code: state-of-the-art leaderboards for machine learning. https://paperswithcode.com/
57. MLCommons (2024) MLPerf: benchmarking machine learning performance. https://mlcommons.org/
58. Face Hugging (2024) Hugging face model hub: pre-trained models and benchmarks for NLP. https://huggingface.co/models
59. Stanford Center for Research on Foundation Models (2024) HELM: holistic evaluation of language models. https://crfm.stanford.edu/helm/
60. Magic Neural (2024) SparseZoo: pre-trained sparse models and benchmarks for deep learning. https://sparsezoo.neuralmagic.com/
61. Stanford University (2024) DAWNBench: end-to-end deep learning benchmark and competition. https://dawn.cs.stanford.edu/benchmark/
62. Li Z, Song Z (2024) Structured pruning strategy based on interpretable machine learning. In: 2024 5th international conference on computer engineering and application (ICCEA). IEEE, pp 801–804. https://doi.org/10.1109/iccea62105.2024.10603526
63. Cheng H, Zhang M, Shi JQ (2024) A survey on deep neural network pruning: taxonomy, comparison, analysis, and recommendations. IEEE Trans Pattern Anal Mach Intell 46(12):10558–10578. https://doi.org/10.1109/tpami.2024.3447085
64. Han S, Liu X, Mao H, Jing P, Pedram A, Horowitz MA, Dally WJ (2016) EIE: efficient inference engine on compressed deep neural network. ACM SIGARCH Comput Archit News 44(3):243–254. https://doi.org/10.1145/3007787.3001163
65. Wang Y, Ma Z, Yang C (2023) A new mixed precision quantization algorithm for neural networks based on reinforcement learning. In: 2023 IEEE 6th international conference on pattern recognition and artificial intelligence (PRAI). IEEE, pp 1016–1020. https://doi.org/10.1109/prai59366.2023.10331945
66. Sze V, Chen Y-H, Yang T-J, Emer JS (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc

IEEE 105(12):2295–2329. https://doi.org/10.1109/jproc.2017.2761740

67. Cai H, Zhu L, Han S (2018) ProxylessNAS: direct neural architecture search on target task and hardware. https://doi.org/10.48550/arxiv.1812.00332
68. Zhu K, He Y-Y, Jianxin W (2023) Quantized feature distillation for network quantization. In: Proceedings of the AAAI conference on artificial intelligence, vol 37, no 9, pp 11452–11460. https://doi.org/10.1609/aaai.v37i9.26354
69. Jiao X, Yin Y, Shang L, Jiang X, Chen X, Li L, Wang F, Liu Q (2020) TinyBERT: distilling BERT for natural language understanding. In: Findings of the association for computational linguistics: EMNLP 2020. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.findings-emnlp.372
70. Liu Y, Cao J, Li B, Weiming H, Ding J, Li L, Maybank S (2024) Cross-architecture knowledge distillation. Int J Comput Vis 132(8):2798–2824. https://doi.org/10.1007/s11263-024-02002-0
71. Kang SK, Lee D, Kweon W, Hwanjo Yu (2022) Personalized knowledge distillation for recommender system. Knowl-Based Syst 239:107958. https://doi.org/10.1016/j.knosys.2021.107958
72. Yan W, Liu A, Huang Z, Zhang S, Van Gool L (2021) Neural architecture search as sparse Supernet. In: Proceedings of the AAAI conference on artificial intelligence, vol 35, no 12, pp 10379–10387. https://doi.org/10.1609/aaai.v35i12.17243
73. Yang Y, Shen Z, Li H, Lin Z (2023) Optimization-inspired manual architecture design and neural architecture search. Science China Inf Sci. https://doi.org/10.1007/s11432-021-3527-7
74. Cassimon A, Mercelis S, Mets K (2024) Scalable reinforcement learning-based neural architecture search. Neural Comput Appl 37(1):231–261. https://doi.org/10.1007/s00521-024-10445-2
75. Dong N, Xu M, Liang X, Jiang Y, Dai W, Xing E (2019) Neural architecture search for adversarial medical image segmentation. Springer International Publishing, pp 828–836. https://doi.org/10.1007/978-3-030-32226-7_92
76. Mousavi H, Loni M, Alibeigi M, Daneshtalab M (2023) DASS: differentiable architecture search for sparse neural networks. ACM Trans Embed Comput Syst 22(5s):1–21. https://doi.org/10.1145/3609385
77. Wei Z, Wang X, Zhu W (2021) AutoIAS: automatic integrated architecture searcher for click-trough rate prediction. In: Proceedings of the 30th ACM international conference on information & knowledge management, CIKM'21. ACM, pp 2101–2110. https://doi.org/10.1145/3459637.3482234
78. Nevzorov AA, Perchenko SV, Stankevich DA (2021) Truncation: a new approach to neural network reduction. Neural Process Lett 54(1):423–435. https://doi.org/10.1007/s11063-021-10638-z
79. Zafrir O, Boudoukh G, Izsak P, Wasserblat M (2019) Q8BERT: quantized 8Bit BERT. In: 2019 5th workshop on energy efficient machine learning and cognitive computing—NeurIPS edition (EMC2-NIPS). IEEE, pp 36–39. https://doi.org/10.1109/emc2-nips53020.2019.00016
80. Yang C, Zhu Y, Lu W, Wang Y, Chen Q, Gao C, Yan B, Chen Y (2024) Survey on knowledge distillation for large language models: methods, evaluation, and application. ACM Trans Intell Syst Technol. https://doi.org/10.1145/3699518
81. Widmann T, Merkle F, Nocker M, Schöttle P (2023) Pruning for power: optimizing energy efficiency in IoT with neural network pruning. Springer Nature Switzerland, pp 251–263. https://doi.org/10.1007/978-3-031-34204-2_22
82. Zhao Y, Guo T (2024) Carbon-efficient neural architecture search. ACM SIGEnergy Energy Inform Rev 4(3):3–9. https://doi.org/10.1145/3698365.3698367
83. Yuan Y, Shi J, Zhang Z, Chen K, Zhang J, Stoico V, Malavolta I (2024) The impact of knowledge distillation on the energy consumption and runtime efficiency of NLP models. In: Proceedings of the IEEE/ACM 3rd international conference on AI engineering—software engineering for AI, CAIN 2024. ACM, pp 129–133. https://doi.org/10.1145/3644815.3644966
84. TensorFlow (2024) TensorFlow model optimization toolkit (TF-MOT). https://www.tensorflow.org/model_optimization
85. Tao Z, Xia Q, Cheng S, Li Q (2023) An efficient and robust cloud-based deep learning with knowledge distillation. IEEE Trans Cloud Comput 11(2):1733–1745. https://doi.org/10.1109/tcc.2022.3160129
86. Kim J, Chang S, Kwak N (2021) PQK: model compression via pruning, quantization, and knowledge distillation. In: Interspeech 2021. ISCA. https://doi.org/10.21437/interspeech.2021-248
87. Harma SB, Chakraborty A et al (2024) Effective interplay between sparsity and quantization: from theory to practice. https://doi.org/10.48550/arxiv.2405.20935
88. Wang T, Wang K, Cai H, Lin J, Liu Z, Wang H, Lin Y, Han S (2020) APQ: joint search for network architecture, pruning and quantization policy. In: 2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR). IEEE, pp 2075–2084. https://doi.org/10.1109/cvpr42600.2020.00215
89. Dantas PV, Sabino W, da Silva L, Cordeiro C, Carvalho CB (2024) A comprehensive review of model compression techniques in machine learning. Appl Intell 54(22):11804–11844. https://doi.org/10.1007/s10489-024-05747-w
90. Zmora N, Jacob G, Zlotnik L, Elharar B, Novik G (2019) Neural network distiller: a python package for DNN compression research. https://doi.org/10.48550/arxiv.1910.12232
91. He Y, Lin J, Liu Z, Wang H, Li L-J, Han S (2018) AMC: AutoML for model compression and acceleration on mobile devices. Springer International Publishing, pp 815–832. https://doi.org/10.1007/978-3-030-01234-2_48
92. Lin J, Chen W-M, Lin Y, Cohn J, Gan C, Han S (2020) MCUNet: tiny deep learning on IoT devices. https://doi.org/10.48550/arxiv.2007.10319
93. Zhu X, Li J, Liu Y, Ma C, Wang W (2024) a survey on model compression for large language models. Trans Assoc Comput Linguist 12:1556–1577. https://doi.org/10.1162/tacl_a_00704
94. Huang J, Zhang J, Wang Q, Han W, Zhang Y (2024) Exploring advanced methodologies in security evaluation for large language models. Springer Nature Singapore, pp 135–150. https://doi.org/10.1007/978-981-97-4519-7_10
95. Brodersen KH, Ong CS, Stephan KE, Buhmann JM (2010) The balanced accuracy and its posterior distribution. In: 2010 20th international conference on pattern recognition. IEEE, pp 3121–3124. https://doi.org/10.1109/icpr.2010.764
96. Yacouby R, Axman D (2020) Probabilistic extension of precision, recall, and F1 score for more thorough evaluation of classification models. In: Proceedings of the first workshop on evaluation and comparison of NLP systems. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.eval4nlp-1.9
97. Liu Y, Li Y, Xie D (2023) Implications of imbalanced datasets for empirical ROC-AUC estimation in binary classification tasks. J Stat Comput Simul 94(1):183–203. https://doi.org/10.1080/00949655.2023.2238235
98. Deng S, Wu L, Shi G, Zhang H, Hu W, Dong R (2021) Emotion class-wise aware loss for image emotion classification. Springer International Publishing, pp 553–564. https://doi.org/10.1007/978-3-030-93046-2_47
99. Xue L, Barua A, Constant N, Al-Rfou R, Narang S, Kale M, Roberts A, Raffel C (2022) ByT5: towards a token-free future with pre-trained byte-to-byte models. Trans Assoc Comput Linguist 10:291–306. https://doi.org/10.1162/tacl_a_00461
100. Kumar S, Solanki A (2023) A natural language processing system using CWS pipeline for extraction of linguistic features. Proc Comput Sci 218:1768–1777. https://doi.org/10.1016/j.procs.2023.01.155

101. Evtikhiev M, Bogomolov E, Sokolov Y, Bryksin T (2023) Out of the BLEU: how should we assess quality of the code generation models? J Syst Softw 203:111741. https://doi.org/10.1016/j.jss.2023.111741
102. Kim JY, Jo SH, Sang-hyun H, Kim KH, Kang YJ, Jeong SC (2024) Comparison of AI model serving efficiency: response time and memory usage analysis. In: Human factors in design, engineering, and computing, AHFE Hawaii, vol 159. AHFE International. https://doi.org/10.54941/ahfe1005580
103. Choudhary T, Mishra V, Goswami A, Sarangapani J (2020) A comprehensive survey on model compression and acceleration. Artif Intell Rev 53(7):5113–5155. https://doi.org/10.1007/s10462-020-09816-7
104. Zhao P, Zhang J, Peng B, Wang L, Wei Y, Liu Y, Liu L (2023) ARBiBench: benchmarking adversarial robustness of binarized neural networks. https://doi.org/10.48550/arxiv.2312.13575
105. Makhlouf K, Zhioua S, Palamidessi C (2021) Machine learning fairness notions: bridging the gap with real-world applications. Inf Process Manag 58(5):102642. https://doi.org/10.1016/j.ipm.2021.102642
106. Na S, Jeong G, Ahn BH, Young J, Krishna T, Kim H (2024) Understanding performance implications of LLM inference on CPUs. In: 2024 IEEE international symposium on workload characterization (IISWC), pp 169–180. https://doi.org/10.1109/IISWC63097.2024.00024
107. Wei W, Ren X, Tang J, Wang Q, Su L, Cheng S, Wang J, Yin D, Huang C (2024) LLMRec: large language models with graph augmentation for recommendation. In: Proceedings of the 17th ACM international conference on web search and data mining, WSDM'24. ACM, pp 806–815. https://doi.org/10.1145/3616855.3635853
108. Zhao X, Liu H, Fan W, Liu H, Tang J, Wang C (2021) AutoLoss: automated loss function search in recommendations. In: Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining, KDD'21. ACM, pp 3959–3967. https://doi.org/10.1145/3447548.3467208
109. Hasan MJ, Rahman F, Mohammed N (2025) OptimCLM: optimizing clinical language models for predicting patient outcomes via knowledge distillation, pruning and quantization. Int J Med Inform 195:105764. https://doi.org/10.1016/j.ijmedinf.2024.105764
110. Bellamy RKE, Dey K, Hind M, Hoffman SC, Houde S, Kannan K, Lohia P, Martino J, Mehta S, Mojsilovic A, Nagar S, Natesan Ramamurthy K, Richards J, Saha D, Sattigeri P, Singh M, Varshney KR, Zhang Y (2019) AI fairness 360: an extensible toolkit for detecting and mitigating algorithmic bias. IBM J Res Dev 63(4/5):4:1–4:15. https://doi.org/10.1147/jrd.2019.2942287
111. Kamal M, Talbert D (2024) Beyond size and accuracy: the impact of model compression on fairness. In: The international FLAIRS conference proceedings, p 37. https://doi.org/10.32473/flairs.37.1.135617
112. Lee J, Yoon W, Kim S, Kim D, Kim S, So CH, Kang J (2019) BioBERT: a pre-trained biomedical language representation model for biomedical text mining. https://doi.org/10.48550/arxiv.1901.08746
113. Mosca E, Szigeti F, Tragianni S, Gallagher D, Groh G (2022) SHAP-based explanation methods: a review for NLP interpretability. In: Proceedings—international conference on computational linguistics COLING, vol 29, no 1, pp 4593–4603. https://aclanthology.org/2022.coling-1.406
114. Said A, Yahyaoui A, Abdellatif T (2024) HIPAA and GDPR compliance in IoT healthcare systems. Springer Nature Switzerland, pp 198–209. https://doi.org/10.1007/978-3-031-55729-3_16
115. He X, Pal S, Amarnath A, Feng S, Park D-H, Rovinski A, Ye H, Chen Y, Dreslinski R, Mudge T (2020) Sparse-TPU: adapting systolic arrays for sparse matrices. In: Proceedings of the 34th ACM international conference on supercomputing, ICS '20. ACM. https://doi.org/10.1145/3392717.3392751
116. Suwannaphong T, Jovan F, Craddock I, McConville R (2025) Optimising TinyML with quantization and distillation of transformer and mamba models for indoor localisation on edge devices. Sci Rep 15(1):10081. https://doi.org/10.1038/s41598-025-94205-9
117. Pool J (2020) Accelerating sparsity in the nvidia ampere architecture. https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf. Accessed 25 May 2025
118. Zhang H, XiaolongShi XS, Sun J, Sun G (2024) Structured pruning for large language models using coupled components elimination and minor fine-tuning. In: Findings of the association for computational linguistics: NAACL 2024. Association for Computational Linguistics, pp 1–12. https://doi.org/10.18653/v1/2024.findings-naacl.1
119. Kurtić E, Marques A, Kurtz M, Alistarh D, Pandit S (2025) 2:4 Sparse LLaMA: smaller models for efficient GPU inference. https://developers.redhat.com/articles/2025/02/28/24-sparse-llama-smaller-models-efficient-gpu-inference. Accessed 25 May 2025
120. Gondimalla A, Chesnut N, Thottethodi M, Vijaykumar TN (2019) SparTen: a sparse tensor accelerator for convolutional neural networks. In: Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture, MICRO '52. ACM, pp 151–165. https://doi.org/10.1145/3352460.3358291
121. Wang H, Ma S, Wang R, Wei F (2024) Q-sparse: all large language models can be fully sparsely-activated. https://doi.org/10.48550/arxiv.2407.10969