International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I2P105

Volume 73 Issue 2, 31-47, February 2025
© 2025 Seventh Sense Research Group®

Review Article

A Survey of Compression Methods for Efficient Model Inferencing

Dhivya Nagasubramanian

Independent Researcher, Minneapolis, Minnesota, USA.

Corresponding Author : dhivya.nagasubramanian@usbank.com

Received: 27 December 2024 Revised: 20 January 2025 Accepted: 13 February 2025 Published: 27 February 2025

***Abstract** - The advent of Large Language Models (LLMs) has revolutionized the field of artificial intelligence, enabling a broad spectrum of applications across academic research and industrial domains. Central to this transformation is the rise of Transformer-based architectures, which have set new benchmarks in Natural Language Processing (NLP) tasks, including text generation, machine translation, and sentiment analysis. However, despite their remarkable performance, the computational demands of these models present significant challenges, particularly when it comes to deploying them in resource-constrained environments. Models like GPT-4, which boast upwards of 1.8 trillion parameters, require substantial processing power, memory, and storage, making them ill-suited for smaller devices such as those found on the Internet of Things (IoT) and embedded systems.*

This limitation raises a critical need for methods to make LLMs more efficient and deployable on edge devices, which often have strict constraints on computational resources. Several promising techniques have emerged to address this challenge, particularly those focused on model compression. These approaches, which involve reducing the precision of model weights and activations, offer potential avenues for shrinking model size and accelerating inference speed. This paper explores a range of model compression techniques, particularly emphasizing their applicability to LLMs. Our goal is to identify strategies that can enhance the efficiency of LLMs, enabling their deployment on devices with limited resources. Furthermore, the synergistic potential of combining multiple compression methods to optimize model performance is being investigated. The ultimate aim is to contribute to democratizing AI by making state-of-the-art models more accessible for real-world applications across diverse devices.

***Keywords** - Large Language Models, Neural Network Quantization, Model Compression, Quantization, Knowledge Distillation, Pruning.*

1. Introduction

Machine learning has experienced rapid growth since 2000, particularly in the past decade. Among its various branches, deep learning stands out, requiring significantly more data and computational power than traditional machine learning methods. A language model is a subset of machine learning focused on understanding and generating natural language. It works by interpreting the context of input prompts and producing the corresponding coherent and contextually relevant text.

Language models are generally categorized into four main types: Neural Language Models (NLM), Statistical Language Models (SLM) [1-5], Pre-trained Language Models (PLM)[6-10], and Large Language Models (LLM). Each type employs a unique approach to natural language processing [14], utilizing different techniques and capabilities for managing linguistic data.

Large language models, especially those involving complex neural networks with transformer architecture, demand high-performance Graphical Processing Units (GPUs) to process vast amounts of data and perform intricate calculations. However, the ideal scenario of unlimited access to GPU resources for training these models remains out of reach due to their prohibitively high costs. Though the endless possibilities of generative AI look promising and fascinating, it remains out of reach for many due to its requirement for high GPU resources.

To achieve efficient, real-time inferencing through large language models [11][12][16] without sacrificing accuracy, it's essential to rethink the entire process of designing, training [13,14,15] and deploying these models. A substantial body of research has been dedicated to tackling these challenges by optimizing neural networks [21] for better performance in areas like memory usage, thereby improving latency and energy consumption while maintaining a strong balance between generalization and accuracy. These efforts have led to a variety of strategies, which can be grouped into the following key categories.

1.1. Integrated Design of Model Architecture and Hardware

A more recent line of research has focused on adapting and co-designing neural network (NN) [18][20] architectures specifically for target hardware platforms. This approach is critical because the performance overhead of a neural network [19] component—such as latency and energy consumption—can vary significantly depending on the hardware used. For instance, hardware equipped with a dedicated cache hierarchy can execute bandwidth-bound operations far more efficiently than systems without such a feature. Like NN architecture design, the initial efforts in architecture-hardware co-design were manual, with experts adjusting the NN architecture for specific hardware. However, more recently, automated techniques like AutoML and Neural Architecture Search (NAS) have been introduced to streamline this process.

1.2. Pruning

Pruning is a widely adopted method for reducing the memory requirement of Neural Networks (NNs) [24][25][26][27]. It involves selectively dropping neurons with minimal influence on the network's output or loss function, resulting in a more efficient and sparsely connected model. Two primary forms of pruning discussed in this paper are static and dynamic. Each approach offers different benefits and compromises regarding computational efficiency and model accuracy. The following sections will delve into these pruning strategies, examining their strengths, drawbacks, and effects on overall model performance.

1.3. Quantization

Quantization techniques offer an effective way to reduce the high costs of training Large Language Models (LLMs) by lowering their computational and resource requirements. By decreasing the number of bits used for each model weight, quantization significantly reduces the overall model size.

This results in LLMs that consume less memory require less storage, are more energy-efficient, and provide faster inference times. These benefits make it possible for LLMs to run on a wider range of devices, including embedded systems and single GPU setups. There are two common types of quantization techniques [34-37]: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). As the name implies, PTQ applies quantization to the neural network layers after the complete training process without requiring retraining or fine-tuning. In contrast, QAT applies quantization during the training itself. The approach used in this paper is QAT, where the student model is quantized layer by layer throughout the training process.

1.4. Knowledge Distillation

The fundamental concepts of knowledge distillation and transfer learning have been established for some time now. Earlier studies have [50][51] demonstrated that the information from an ensemble of models could be compressed into a single network. Later researchers expanded on this idea by exploring how shallow, wide, and fully connected networks could mimic the behavior of deeper neural networks [21].

Knowledge distillation (KD) [48][49] is a technique that trains smaller networks by leveraging the knowledge of larger, pre-trained models to enhance performance. In KD, the teacher network generates soft labels, which are then used to train the student network. This approach, which uses the knowledge embedded in previously trained models, has gained significant attention in various fields for tasks like model compression [22][23] and the development of learning algorithms. In this paper, KD has been explored in the context of training a quantized mixed-precision student model using reinforcement learning (QMPS-RL).

1.5. Lightweight Model Redesign

A key area of research has focused on optimizing neural network architectures [18][21] at both the micro and macro levels. At the micro-architecture level, this involves refining components like kernel types—such as depth-wise convolution or low-rank factorization—while at the macro-architecture level, it addresses module types like residual or inception networks. Traditionally, architectural innovations were driven by manual searches, a method that proved inefficient and unsustainable at scale. This led to the emergence of new approaches like Automated Machine Learning (AutoML) and Neural Architecture Search (NAS), which automate the discovery of optimal network architectures while adhering to constraints related to model size, depth, and width.

1.6. Low-Rank Decomposition

Low-rank transformations are techniques used to approximate large matrices by smaller matrices to make computations more efficient. In the context of large language models [11][12][16], like GPT (Generative Pre-trained Transformer), low-rank transformations can be particularly useful for optimizing memory and computational requirements. Applying a low-rank approximation [31][32] to the weight matrix is highly effective, resulting in a 3× compression of the fully connected layer [22][23].

The main contributions are as follows:

1. Provide a review of the following compression techniques
    a. Pruning [24][25]
    b. Quantization
    c. Knowledge Distillation

   d. Low-rank Decomposition
   e. A combination of compression (CoC)
2. Discuss two different types of pruning techniques: static and dynamic methods, and the way modes in which the model can be pruned – Offline vs Online.
3. Compare and contrast two quantization methods, QAT [38][40] and PTQ [42][43] and also discuss different techniques (weights, activation, key query value) with different precision schemes (Fixed, Mixed precision, etc).
4. Discussion of knowledge distillation techniques - response-based, feature-based, and relation-based.
5. Briefly highlight the method, advantages, and disadvantages of low-rank decomposition.
6. Finally, discuss the concept of combining multiple compression techniques [22].

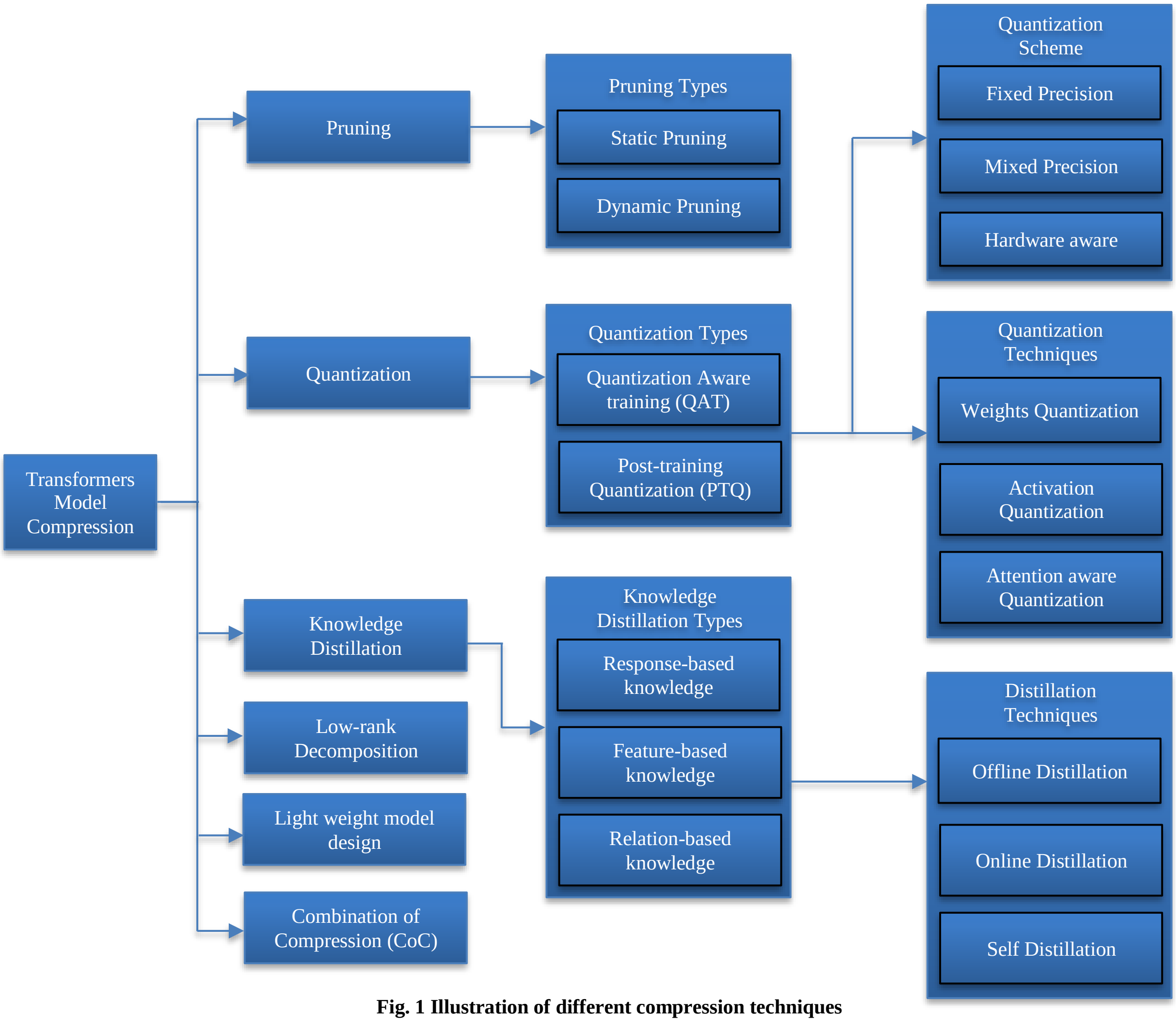

Fig. 1 Illustration of different compression techniques

2. Methods

2.1. Large Language Models

Language models are a crucial subfield of machine learning that focuses on understanding and generating natural language [14]. These models are designed to comprehend the context of input prompts and generate coherent, contextually appropriate responses by predicting the missing components of the text. Broadly, language models can be categorized into 4 broad types, with pre-trained language models (PLMs) [6-10] and large language models (LLMs) [11][12][16] garnering significant attention in recent years. Each of these categories represents distinct approaches to natural language processing (NLP) [14], utilizing different techniques and possessing varying capabilities for processing linguistic data. While statistical language models [1-5] and neural language models have been explored for decades, the recent rise in popularity of PLMs and LLMs has marked a paradigm shift, with applications spanning across both academia and industry, including areas such as recommendation systems.

Pre-trained language models [6-10] are first trained on vast amounts of text data before being fine-tuned for specific downstream tasks. These models, which leverage the Transformer architecture and self-attention mechanism [11]–[15], have significantly advanced the field of semantic processing in NLP. A notable example of a PLM is BERT (Bidirectional Encoder Representations from Transformers), which is pre-trained on a large corpus of text using unsupervised methods, including masked language modeling and next-sentence prediction tasks. BERT [6] is then fine-tuned for specific tasks such as question answering, sentiment analysis, and text classification. It has become the dominant framework in PLM research due to its efficiency, scalability, and exceptional performance across a wide range of tasks. In response to the growing demand for more efficient models, DistilBERT [17] emerged as a smaller version of BERT [6], maintaining similar levels of accuracy while being more lightweight and easier to deploy. This compact version is particularly advantageous for resource-constrained environments where faster inference times and reduced memory usage are critical. On the other hand, Large Language Models (LLMs) [11][12][16] build upon the foundational principles of PLMs but extend their capabilities by scaling both model size and the volume of training data. It has been observed that increasing the size of PLMs enhances their ability to perform effectively on a variety of downstream tasks. For instance, GPT-4, with its 1.8 trillion parameters, is a significantly larger model than traditional PLMs and shares similar pretraining objectives. However, GPT-4’s increased scale allows it to outperform standard PLMs, particularly in solving more complex and nuanced tasks. In creative domains such as poetry, song lyrics, and fiction writing, GPT-4 exhibits remarkable proficiency in generating content that adheres to specific styles or themes, a capability that models like BERT, with only 340 million parameters, do not possess. BERT, primarily designed for tasks like sentence completion or masked word prediction, lacks the generative and creative capabilities required for these types of creative tasks. Despite the promising advancements brought by LLMs, their inherent need for large-scale data processing and distributed parallel training makes them highly resource-intensive and costly to train. As a result, continuous experimentation with various training strategies and optimization techniques is required to push the boundaries of what LLMs can achieve, though this remains a significant challenge from both a computational and financial perspective. In summary, while PLMs such as BERT and smaller variants like DistilBERT have revolutionized NLP by offering efficient, scalable solutions for a wide range of tasks, LLMs like GPT-3 represent the next frontier, pushing the limits of what is possible in language generation and understanding.

3. Pruning

Network [24][27] is a key technique used to reduce both memory size and bandwidth requirements. Developed in the early 1990s, pruning methods aimed to compress a trained large neural network [21] into a smaller version without the need for retraining. This enabled the deployment of neural networks in resource-constrained environments such as embedded systems. The pruning process [25-26] involves removing redundant parameters or neurons that do not significantly impact the model’s accuracy. These redundant components often correspond to weights that are either zero, close to zero, or repeated. By eliminating these parts, pruning reduces the model’s computational complexity. When pruned networks are subsequently retrained, there is an opportunity to escape local minima and potentially improve accuracy.

Research on network pruning generally falls into two main categories: sensitivity calculation and penalty-term methods. Recent studies have continued to refine these approaches, often by combining elements from both categories. In addition, new pruning techniques have emerged. Modern network pruning strategies can be classified based on several criteria, such as 1) structured versus unstructured pruning, depending on whether the pruned network maintains symmetry; 2) neuron versus connection pruning, based on the type of element pruned; and 3) static versus dynamic pruning. Figure 8 illustrates the differences between static and dynamic pruning. In static pruning, all pruning steps are carried out offline before inference, whereas dynamic pruning occurs during runtime. Although there is some overlap between these categories, this paper will categorize network pruning methods as either static or dynamic. Figure 9 depicts the different levels of granularity at which pruning can occur.

3.1. Static Pruning

Static pruning refers to a pruning strategy where all pruning decisions are made before inference, typically during the training phase or as a post-processing step after the model has been trained. In this approach, the model is pruned once, and the removed parameters (e.g., weights, neurons, or filters) are no longer part of the model during deployment. These pruning decisions are based on predefined criteria such as the magnitude of the weights, their contribution to the loss function, or other importance scores derived from sensitivity analysis. The pruning process in static pruning is executed offline, and once the model has been pruned, it is then deployed in a fixed form, with no further pruning occurring during runtime. After pruning, the model typically undergoes a fine-tuning phase to adjust the remaining parameters and recover any accuracy loss that may result from the removal of certain weights or neurons. However, no additional pruning is performed during the inference stage. Static pruning can lead to significant improvements in efficiency, particularly in terms of memory usage and computation during inference, as the model’s architecture is fixed and optimized for deployment on resource-constrained platforms such as embedded systems, mobile devices, or edge devices.

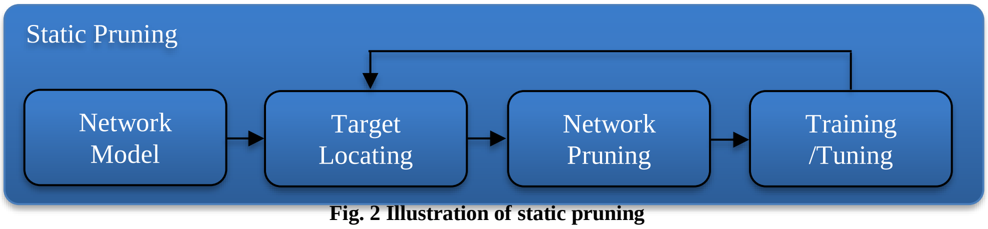

Fig. 2 Illustration of static pruning

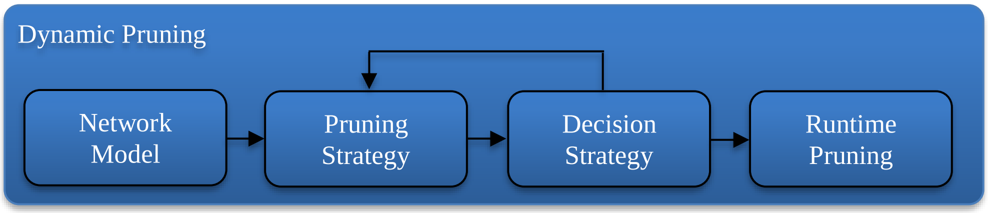

Fig. 3 Illustration of dynamic pruning

3.2. Dynamic Pruning

While static pruning offers significant benefits in terms of reducing the size and complexity of neural networks, it comes with notable limitations, particularly in its permanent alteration of the network structure. Except for recoverable pruning techniques, static pruning irreversibly eliminates weights or neurons, potentially leading to a degradation of model performance. Once pruned, the network cannot fully recover the lost information, even if the model is retrained. Although there have been efforts to develop methods to recover some of the pruned network capabilities, static pruning remains constrained in its ability to restore the original network functionality.

In contrast, dynamic pruning [29, 30] operates more flexibly by making pruning decisions at runtime. This enables the model to determine, in real-time, which neurons, channels, or layers should be excluded from the further computation. Notably, dynamic pruning can mitigate the limitations inherent in static pruning by dynamically adjusting the network based on changing input data, thereby reducing computational overhead, bandwidth usage, and power dissipation. One key advantage of dynamic pruning is its ability to adjust to varying input characteristics, allowing for optimized computation during inference without the need for extensive retraining or fine-tuning during runtime.

The effective implementation of dynamic pruning requires addressing several critical considerations. First, the decision-making system—responsible for determining which parts of the network to prune—plays a central role in the pruning process. Several key aspects of this decision system are as follows.

3.2.1. Side Decision Networks

A separate side network can be used to perform pruning, which tends to yield high performance but is often more challenging to train [153].

3.2.2. Additional Connections

These may be attached to the original network during either the inference or training phases, providing auxiliary information for pruning decisions.

3.2.3. Learnable Characteristics

Pruning decisions can also be influenced by connection characteristics that are learned through standard backpropagation algorithms.

3.2.4. Pruning Techniques

Channel-Wise Pruning: This approach involves removing entire channels from layers

Layer-Wise Pruning: Entire layers are pruned based on their importance or contribution to the network's overall performance.

Block-Wise Pruning: Smaller groups of layers or units are pruned together.

Network-Wise Pruning: Involves pruning large sections of the network or entire sub-networks. The selected pruning level influences the resulting model's architecture and is directly related to hardware considerations.

3.2.5. Input Data Consideration

One-Shot Information Feeding: In this approach, the entire input is fed to the decision system in a single pass, which allows for real-time pruning based on complete input data.

Layer-Wise Information Feeding: This method feeds data to the decision system in iterative windows, allowing for more granular control over the pruning process during the forward pass.

Decision Score Calculation

Norm-Based Scoring: One approach to pruning decisions involves using a norm-based score, which quantifies the importance of weights or neurons.

Alternative Scoring Methods: Other methods may utilize different criteria to assess the importance of individual elements in the network.

Automatic Thresholding or Dynamic Mechanisms: Alternatively, thresholds may be dynamically adjusted during runtime based on real-time performance metrics.

Stopping Criteria: Layer-Wise and Network-Wise Pruning: Some algorithms skip pruned layers or sub-networks during runtime, thereby simplifying computation without compromising the output.

Dynamic Data Path Selection: Certain dynamic pruning systems allow the data path to be selected dynamically, depending on which components of the network are active or pruned.

Termination and Output Generation: Some systems may terminate computation early, outputting predictions once sufficient processing has occurred. In such cases, the remaining layers are considered pruned.

Furthermore, dynamic pruning techniques can also be applied to networks that have already undergone static pruning, potentially further reducing the model's computational and bandwidth requirements.

Dynamic pruning, while offering significant advantages in terms of adaptability and computational efficiency, also presents several critical challenges that must be addressed for its successful implementation in Large Language Models (LLMs) [13]. These challenges include real-time pruning overhead, the trade-off between model accuracy and efficiency, the complexity of decision-making systems, and the interdependencies between model components.

Real-Time Pruning Overhead

One of the principal challenges associated with dynamic pruning in LLMs is the computational overhead introduced by the need to make pruning decisions during runtime. The process of evaluating and selecting which components (e.g., neurons, attention heads, or entire layers) to prune is computationally intensive and can introduce additional latency into the inference pipeline. This overhead may diminish the benefits of pruning by offsetting the computational savings gained from reducing the model's size or complexity.

For dynamic pruning to be effective in real-time applications, the decision-making process must be highly efficient. The algorithms used to evaluate which components to prune must be lightweight and executed with minimal latency to ensure that the pruning does not impair the overall system performance, especially in applications where response time is critical, such as real-time [14] processing tasks or interactive systems. Ensuring that pruning decisions are made swiftly and with minimal resource usage is crucial to maintaining the efficacy of dynamic pruning.

Accuracy vs. Efficiency Trade-Off

As with any pruning technique, dynamic pruning involves an inherent trade-off between accuracy and efficiency. While pruning aims to reduce the computational load by eliminating unnecessary components, it may also result in a loss of model performance if important parameters or structures are removed. This trade-off is particularly pronounced in tasks that require high precision, such as sentiment analysis, medical text processing, or legal document analysis, where even small degradations in accuracy can have significant consequences. Maintaining a balance between pruning efficiency and model accuracy is critical. This requires ongoing evaluation and fine-tuning of dynamic pruning strategies [29, 30] to ensure that the reduction in computation and memory usage does not come at the expense of the model's effectiveness in performing its intended tasks. Continuous monitoring of the model's performance and dynamic adjustment of pruning parameters are essential for optimizing this balance and minimizing accuracy degradation.

Complexity of Decision Systems

Designing and implementing decision-making systems for dynamic pruning in Large Language Models (LLMs) presents a significant challenge. These systems are tasked with determining which parts of the model should be pruned based on input data and runtime conditions, a process that becomes more intricate when advanced techniques, such as Reinforcement Learning (RL) or adaptive pruning strategies, are employed to guide these decisions. Training such decision systems is computationally demanding, particularly when utilizing auxiliary networks or RL algorithms that iteratively optimize pruning strategies. The incorporation of these decision-making components introduces additional complexity both during the training phase and at deployment. While they increase the computational load during training, they can also lead to higher model complexity during inference, potentially counteracting the intended goal of reducing resource consumption. Therefore, developing more streamlined and efficient decision systems—without compromising their effectiveness—remains an ongoing challenge.

Interdependence of Model Components

Another significant challenge in the dynamic pruning of LLMs is the interdependence between model components. In complex architectures like transformers, components such as attention heads, layers, and neurons are highly interconnected. Removing or pruning one component may have cascading effects on the performance of other interconnected components, making it difficult to isolate and prune individual units without inadvertently affecting the overall model behavior.

This interdependency complicates the identification of which components can be pruned without causing adverse effects on model accuracy. Dynamic pruning strategies must account for these interdependencies to avoid unintentional disruptions to the model's functionality.

For instance, pruning a particular attention head in one layer could impact the ability of other attention heads to capture relevant information, potentially leading to a decrease in model performance. Developing dynamic pruning techniques that carefully evaluate the relationships between components and preserve the model's integrity is a key challenge.

4. Quantization

Quantization techniques offer a promising solution to mitigate the high computational and resource costs associated with training and deploying large language models (LLMs). By reducing the bit-width representation of each model weight, quantization significantly decreases the overall size of the model.

This reduction not only lowers the memory and storage requirements but also enhances energy efficiency and accelerates inference times. Consequently, quantized LLMs are better suited for deployment across a wider range of devices, including those with limited computational resources, such as embedded systems and single-GPU environments.

The challenge of deploying AI models on resource-constrained platforms, such as SLAM (Simultaneous Localization and Mapping) robotics devices [19][20] or decentralized Web3 applications, is particularly pronounced. These systems face difficulties in running full-scale models due to their limited capacity for handling the intensive computations typically required by large models [21]–[24]. In such contexts, quantization becomes a critical enabling technology, as it reduces both the model's size and its computational burden, thus making it feasible to deploy LLMs in environments with strict resource limitations. Various methods for model quantization exist, each targeting different optimization goals, including faster computation, minimal accuracy degradation, and reduced model parameter size. This section explores several quantization techniques, with a focus on post-training quantization, which allows for efficient deployment without the need for extensive retraining, making it an attractive approach for many practical applications. There are two broad categories of quantization: quantization-aware training [41] and post-training quantization, respectively.

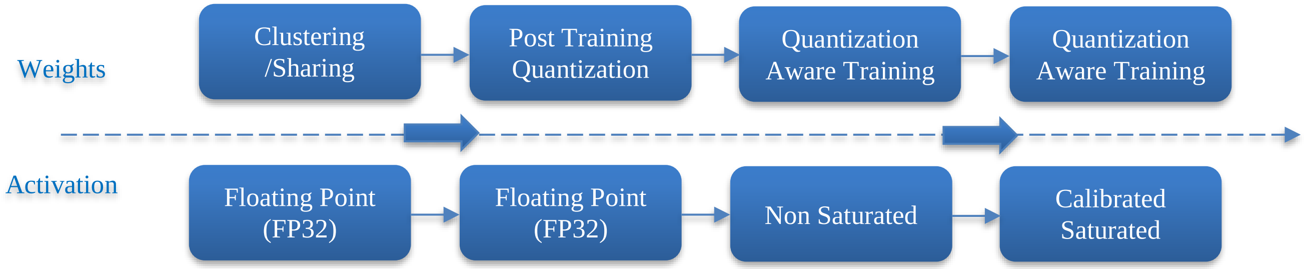

Fig. 4 Evolution of Quantization technique from left to right. The blue rectangle represents quantized data, while the white rectangle represents full precision (FP32) format

5. Post Training Quantization

Post-training quantization (PTQ) [42][43] is to apply quantization after the model has been fully trained. Two common approaches are used to achieve this:

5.1. Dynamic Quantization

In this method, quantization occurs at runtime after each activation. However, this approach introduces additional computational overhead, potentially slowing down performance due to the extra processing required for each activation.

5.2. Static Quantization

Here, the quantization parameters are pre-computed during the quantization process before runtime. This method ensures the quantization of weights while

6. Quantization Aware Training

The model accounts for the errors introduced by quantization by incorporating quantization operators at each activation during the training phase. These operators enable the model to recognize and adapt to quantization errors throughout the backpropagation process.

As a result, this approach typically leads to reduced performance degradation while also facilitating faster computation [39][40][41].

In general, how is quantization applied in LLM feedforward networks during the training process? Consider, for example, a dataset with N rows and a neural LLM model with L feedforward layers. Each layer, say, for example, layer l has weight parameters w_l, which are the model's full-precision weights.

Quantization helps reduce the precision of the weights and/or activations (e.g., from floating-point to fixed-point). For each layer, the weights w_l, and activations x_l, are quantized using a scale *s* and a zero-point *z*. The scale is calculated based on the maximum and minimum values of the weights, while the zero point is used to shift the values so they fit within the quantized range. The quantization of float 0 is z. This is an example of how weight quantization happens in QAT.

$$s = (w_{lmax} - w_{lmin}) / (w_{lqmax} - w_{lqmin})$$

$$z = (w_{lqmax}) - (w_{lmax} / s)$$

$$\textit{Quantized weight} = \quad w_{lq} = (w_l / s) + z$$

7. Granularity of Quantization

The granularity of quantization in LLMs [13] plays a crucial role in balancing model efficiency and performance. Quantization techniques can be tailored based on the specific requirements of deploying and training an LLM, with different levels of granularity offering trade-offs between computational savings and potential accuracy loss.

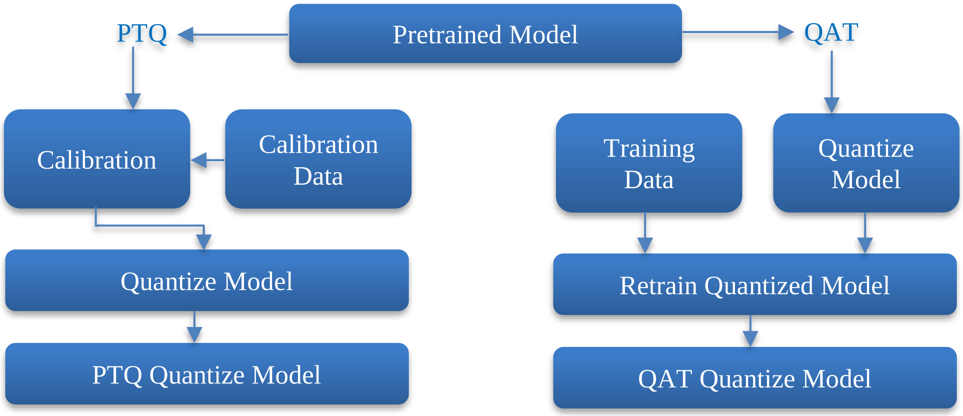

Fig. 5. Illustration of QAT vs PTQ Quantization technique

7.1. Per layer Quantization for LLMs

All the filters within the same layer of the LLM share identical quantization parameters. The quantization range is determined collectively for all filters within the layer, typically based on the range of values observed across all filters in that layer. This approach simplifies the quantization process by applying a uniform set of parameters to each filter, resulting in reduced computational complexity.

However, per-layer quantization often introduces higher quantization errors, particularly when individual filters have varying value distributions. These errors can accumulate across the layers of an LLM, potentially leading to a degradation in the model’s overall accuracy. In the context of LLMs, where precise language understanding and generation are critical, the trade-off between reduced model size and potential loss of performance becomes more pronounced. While per-layer quantization may be beneficial for scenarios with stringent resource constraints, its use in LLMs needs careful consideration to avoid significant performance degradation.

7.2. Per Channel Quantization for LLMS

To address the performance issues associated with per-layer quantization, per-channel quantization offers a more refined approach. Rather than applying a uniform quantization strategy at the layer level, quantization is performed at a finer granularity—specifically, at the filter level within each layer of the LLM. In this method, each filter is assigned its own set of quantization parameters tailored to the specific range of values observed within that filter.

Per-channel quantization introduces increased complexity compared to per-layer methods, requiring a more detailed analysis of the individual filters and their respective value distributions. However, this approach significantly reduces quantization errors for each filter, leading to improved model performance relative to lower-granularity techniques. For LLMs, where model accuracy is paramount for text generation, sentiment analysis, and question answering tasks, per-channel quantization strikes a better balance between computational efficiency and accuracy preservation.

While there are two broader categories of quantization, there are various methods within each that could be adapted based on the needs. They are as follows.

7.3. Weight Quantization

Instead of applying quantization to all parameters of a model, weight quantization selectively targets only specific components, typically focusing on the weight matrices while leaving activation values unquantized and in their original precision. This selective quantization approach effectively reduces the model’s memory and storage requirements, leading to more efficient deployments without sacrificing the integrity of activation values.

7.4. Activation Aware Quantization (AWQ)

Traditional weight quantization applies uniform quantization across all parameters within the weight matrices. However, alternative strategies seek to further optimize the process by reducing the number of parameters to be quantized, thus improving the speed and efficiency of quantization. One such strategy is Activation-Aware Weight Quantization (AWQ) [44], which introduces a more selective approach by recognizing that not all weights contribute equally to the model's performance. AWQ identifies the most critical weights based on the magnitudes of their corresponding activations, retaining these essential weights in full precision while quantizing less significant weights. This targeted approach minimizes accuracy degradation while reducing computational costs, effectively balancing efficiency and model performance.

7.5. Attention-Aware Weight Quantization

A more advanced technique, attention-aware weight quantization, takes advantage of the Hessian trace to assess the importance of weight matrices [32]. By leveraging the attention mechanism — a core component in LLMs — this method allocates higher-bit precision to weights deemed more important based on their attention scores. Weights with lower importance are quantized to lower-bit precision, adopting a mixed-precision quantization approach. This strategy leads to improved computational efficiency and performance, particularly in LLMs, without sacrificing model accuracy. Attention scores enable more precise quantization, ensuring that critical parameters are preserved while reducing the overall model size.

8. Determining Precision for Quantization

Fixed-Point Representation is a method commonly used in quantization, where numbers are expressed with a fixed number of digits after the decimal point. Unlike floating-point representation, which allows the position of the decimal point to shift dynamically, fixed-point precision maintains a constant number of digits after the decimal. This approach is especially effective in reducing the computational and memory demands of LLMs by constraining the bit-width of parameters, such as weights and activations [34][35][38], while still preserving acceptable levels of accuracy. Fixed-point and integer quantization offer several notable advantages.

Fixed-point or integer precision greatly minimizes the memory requirements of models by using a fixed and generally smaller number of bits per value (e.g., 8 bits instead of 32 or 64 bits for floating-point representation). This compact format allows large models to be stored more efficiently, particularly advantageous in resource-constrained environments such as mobile devices and edge-computing platforms.

Additionally, hardware designed for fixed-point computations can handle smaller bit-width values more efficiently than floating-point equivalents. Fixed-point arithmetic operations, including addition, multiplication, and comparison, require fewer computational resources, making this precision format well-suited for real-time or low-latency applications. Moreover, fixed-point operations are inherently less computationally demanding, leading to lower power consumption. This characteristic makes fixed-point quantization an ideal choice for battery-powered devices, where energy efficiency is critical.

While there are advantages to a methodology, there are also some disadvantages to this method.

8.1. Restricted Range

The primary limitation of fixed-point precision lies in its restricted dynamic range compared to floating-point representation. Fixed-point or integer formats cannot represent extremely small or large values with high fidelity, which introduces quantization errors. These errors are particularly pronounced in scenarios involving neural networks with very large or very small weights, potentially degrading model performance on tasks requiring precise computations [19][20].

8.2. Training Complexity

Converting between fixed-point and floating-point representations adds complexity to training and deployment pipelines. Quantization, especially during training through methods such as Quantization-Aware Training (QAT), demands meticulous handling. This not only complicates the workflow but can also lead to increased training times, thereby affecting efficiency.

8.3. Sensitivity

Furthermore, the fixed bit-width nature of fixed-point representation reduces the model's capacity to capture fine-grained variations in data, such as subtle weight adjustments during training. This constraint can be particularly detrimental for applications requiring high precision, such as scientific computing or domains dealing with sensitive and intricate data patterns.

8.4. Mixed Precision

A key limitation of earlier discoveries in quantization techniques is their requirement for all neural network (NN) parameters to be quantized using a uniform precision. While some approaches, as mentioned in Section 5, have attempted to apply different bit widths for weights, biases, and activations, these bit widths are still typically uniform within each type of parameter. This uniformity creates inefficiencies. Specifically, in the case of integer/fixed-point quantization, the precision of weights, for example, is often constrained by the most sensitive layer, which may require more bits for accuracy. This results in an unnecessarily high bit width for less sensitive layers that could have functioned with fewer bits. As a result, the overall compression rate is reduced, and the model could potentially be made smaller without sacrificing accuracy if fewer bits were allocated to the less sensitive layers.

Mixed Precision Quantization seeks to address these issues by combining the benefits of multiple quantization schemes. It extends the idea of using different bit widths for weights, biases, and activations to an even finer granularity: layer-level quantization. In this approach, each layer in the neural network is allocated a tailored bit width and precision based on its sensitivity to quantization. Layers more sensitive to precision loss (typically deeper or more complex layers) are assigned higher bit widths, while less sensitive layers can use lower bit widths. This strategy effectively balances the benefits of integer/fixed-point, binary, and ternary quantization, leading to improved accuracy at higher compression rates and more efficient inference.

However, the key challenge with mixed precision quantization [45][46] lies in determining the optimal bit width for each layer, as the search space for potential configurations grows exponentially with the number of layers.

8.5. Hessian-aware Quantization (HAWQ)

The approach presented in Hessian-Aware Quantization (HAWQ) [45] provides a systematic method during Quantization-Aware Training (QAT) to determine the precision required for each layer's weights and activations while either maintaining or improving the state-of-the-art quantization accuracy. This method utilizes second-order information, specifically the second-order partial derivatives, which are encapsulated in the Hessian matrix (the matrix of second derivatives), to assess the sensitivity of weights and activations to quantization. By leveraging this information, the minimum bit width necessary for each layer to preserve the network's overall accuracy is determined.

The central insight from this approach is that layers with a higher Hessian spectrum (i.e., larger eigenvalues) exhibit more volatile loss behaviors, meaning that these layers are more susceptible to fluctuations in loss when even small amounts of quantization noise, such as rounding errors, are introduced. Consequently, these layers are more sensitive to quantization and require a higher bit width to maintain accuracy. Conversely, layers with smaller eigenvalues tend to have a flatter loss landscape, exhibiting less sensitivity to quantization noise. As a result, these layers can tolerate lower bit widths without significantly affecting model performance.

This observation aligns with the understanding that flatter loss regions amplify quantization noise less than regions with sharper curvature. Based on this principle, HAWQ selects bit widths for each layer according to the Hessian information, manually assigning higher bit widths to layers with higher sensitivity to quantization and lower bit widths to less sensitive layers.

Another critical finding from HAWQ is the importance of the order in which layers are quantized, as this ordering can have a substantial impact on the accuracy of the final model. HAWQ prioritizes the quantization of layers with higher Hessian values and a larger number of parameters, as these layers are more sensitive to quantization noise. These layers are first quantized and retrained, while the less sensitive layers are quantized subsequently. The rationale behind this approach is that quantizing and retraining the less sensitive layers first is less effective. These layers are more resilient to quantization noise and adjust well to the initial introduction of noise, whereas it is more beneficial to "lock in" the quantized values of the sensitive layers first. This allows the less sensitive layers to recalibrate during retraining, enabling them to adjust their parameters to the newly quantized sensitive layers without significant degradation in performance. Even if this recalibration causes some parameters to drift further from their original floating-point values, their inherent robustness ensures that their quantization minimises the network's overall accuracy.

Hardware Aware Precision One of the primary objectives of quantization is to reduce inference latency. However, the speedup achieved from quantization varies across different hardware platforms. The effectiveness of quantization is hardware-dependent, with factors such as on-chip memory, bandwidth, and cache hierarchy influencing the extent of latency reduction. To fully capitalize on the advantages of quantization, it is crucial to account for these hardware-specific considerations, making hardware-aware quantization an essential approach for optimizing performance [46]. Notably, the study employs a reinforcement learning agent to determine optimal mixed-precision quantization settings, using a look-up table to correlate latency with different bit widths for various layers. However, this approach relies on simulated hardware latency. To overcome this limitation, recent research directly deploys quantized operations on actual hardware, measuring the real-world deployment latency for each layer across varying bit precisions.

9. Knowledge Distillation

Memory and processing power are key considerations, especially in real-world applications where resources are limited, such as on mobile devices or embedded systems. The concept of Knowledge Distillation (KD) [48-51] was introduced by Geoffrey Hinton and colleagues in 2015. The core idea is that a teacher model—typically a large and highly accurate deep neural network—can guide a smaller student model by providing "soft targets." These are probabilistic outputs that convey the model's confidence for each class, offering richer, more nuanced information compared to hard labels (which indicate only a single class, e.g., 0 or 1 for binary classification). Soft targets not only help the student model make the correct classification but also allow it to learn the underlying patterns and relationships captured by the teacher, ultimately enhancing the student's ability to generalize.

The choice of teacher-student architectures is vital to the success of knowledge distillation. Typically, the teacher is a larger, more complex, and more accurate model, while the student is smaller, faster, and less resource-intensive. The challenge lies in transferring the rich knowledge from the teacher to the student while maintaining the student's efficiency.

Knowledge distillation has proven effective across multiple machine learning domains, including Natural Language Processing (NLP), Computer Vision, and Speech Recognition. One major advantage of KD is its ability to compress large models, making them more suitable for deployment on resource-constrained devices without significant performance loss. Additionally, KD can improve the generalization capabilities of smaller models, especially when they are trained on limited data, by leveraging the teacher's knowledge.

In the context of Large Language Models (LLMs) and Transformer-based architectures, KD helps create compact yet high-performance models. These distilled models are ideal for deployment on devices with limited computational power and memory, making them suitable for edge computing applications. By transferring knowledge from a large pre-trained teacher model, the student benefits from the teacher's expertise.

Various architectural approaches, such as multi-branch networks or parallel training, have been explored to optimize this knowledge transfer process. Distillation leverages different types of knowledge, each contributing uniquely to the training of the student model. The main types of knowledge utilized in KD are as follows:

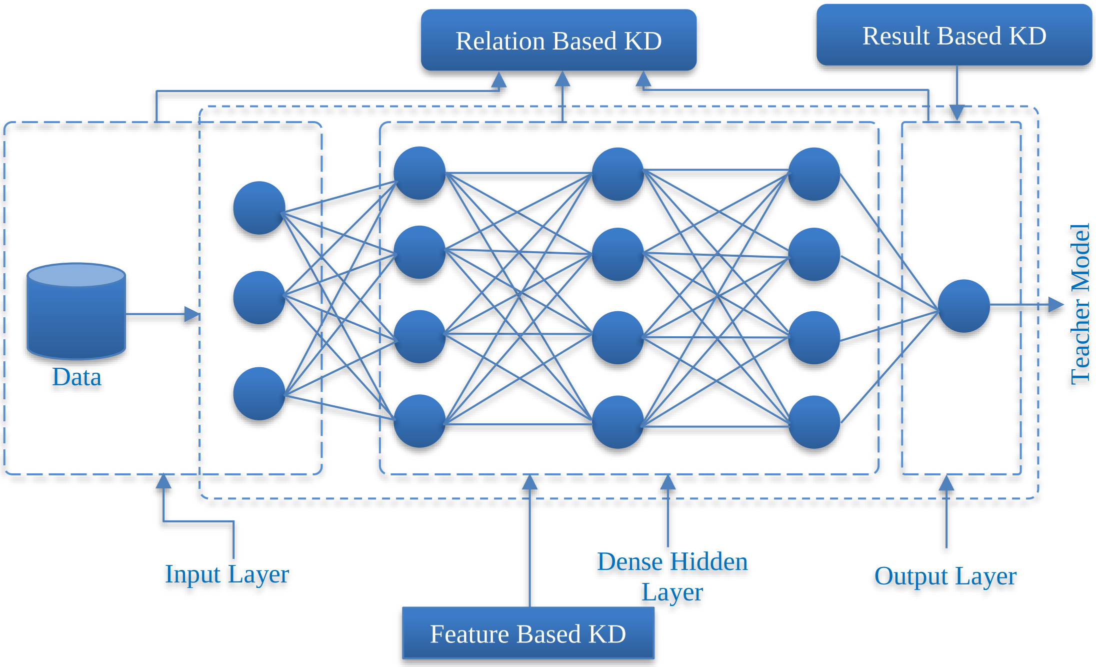

Fig. 6 Illustration of knowledge distillation technique

9.1. Response-Based Knowledge Distillation

Response-based knowledge [51-53] in knowledge distillation typically refers to the neural response of the final output layer of the teacher model. The core idea behind this approach is to directly mimic the final predictions of the teacher model, particularly its logits. These logits encapsulate not only the final classification decision but also the model's confidence in each possible class. By training the student model to replicate the teacher's output, response-based distillation enables the student to learn the generalization patterns captured by the teacher model, such as the relative likelihoods of various classes, without requiring access to the true hard labels. Response-based knowledge distillation offers several advantages, making it a popular method for transferring knowledge from a teacher model to a student model. Its simplicity stands out, as it only requires transferring the teacher's output logits or probabilities, avoiding the complexity of accessing and aligning intermediate layers. This method is computationally efficient and well-suited for scenarios where the teacher's internal feature representations are inaccessible. It is also model agnostic, enabling distillation between models with differing architectures, and provides a regularization effect by leveraging the teacher's soft targets, which encode uncertainty and relative class confidence, enhancing the student's generalization. Furthermore, response-based distillation scales effectively in distributed systems, as it involves transferring relatively small output data.

However, this approach has limitations. It only captures the knowledge present in the teacher's output, overlooking the rich information embedded in intermediate layers or feature representations, which can lead to suboptimal performance on complex tasks. Additionally, the method's success heavily depends on the quality of the teacher model, as poor teacher outputs can negatively affect the student's learning.

9.2. Feature-based Knowledge Distillation

In this form of knowledge distillation, the student model is trained to mimic the intermediate feature representations or activations of the teacher model [53]. This approach emphasizes transferring the teacher model's internal feature representations to the student.

The student model aligns its feature maps or activations with those of the teacher model's corresponding layers. This process can include transferring intermediate representations from the deep transformer layers of a large language model (LLM) to a smaller LLM, enabling the student to learn similar representations at each layer. Techniques such as attention matching, where the student replicates the teacher's attention maps, are often employed to enhance this learning process.

This approach offers several advantages. First, it enables deeper knowledge transfer by capturing the internal representations of the teacher model, which are often rich in semantic and structural information. This can significantly enhance the student model's performance, especially for complex tasks requiring nuanced understanding. Second, feature-based KD can promote layer-wise alignment, allowing the student to mimic the teacher's hierarchical learning process. Third, it is particularly effective for tasks like object detection and segmentation, where spatial and feature-level understanding is critical. Additionally, techniques like attention map alignment or activation matching can further improve the transfer efficiency.

However, feature-based distillation also has notable disadvantages. One key limitation is its computational overhead, as aligning intermediate features often requires substantial memory and processing power, especially for large models. This makes it less suitable for resource-constrained environments. Moreover, the method is architecture-sensitive, as effective feature alignment typically requires the teacher and student models to have similar layer structures or spatial dimensions, limiting flexibility. Another challenge is the increased complexity in implementation, as feature extraction and alignment demand careful design choices, such as selecting which layers to align and defining appropriate loss functions.

9.3. Relation-based Knowledge Distillation

This method focuses on transferring relational knowledge from the teacher to student models. Rather than directly aligning feature maps or output distributions, the student model learns the relative relationships between various components or features in the teacher model. These relationships may include interactions between token pairs, contextual dependencies, or attention patterns. In the case of relationship-based Knowledge Distillation (KD) for Large Language Models (LLMs), this typically involves transferring the attention or interaction patterns between tokens or layers in the teacher model to the student model.

For example, the student may learn how specific tokens interact within a self-attention mechanism by observing the teacher model's attention distribution over those tokens. This is the most complicated and highly resource-intensive method, and it is not a commonly used technique in LLM KD to train smaller models. Although complicated, it has several advantages: First, it captures more holistic and structural information, emphasizing the relationships between data samples rather than isolated feature representations. This is particularly useful for tasks where understanding relationships, such as clustering, ranking, or graph-based problems, is critical. Secondly, it provides additional regularization by encouraging the student model to replicate the teacher's understanding of data relationships, which can lead to improved generalization and robustness.

10. Distillation Schemes

This section provides an overview of the different distillation strategies (i.e., training approaches) for both teacher and student models. Knowledge distillation learning schemes can be broadly classified into three main categories—offline distillation, online distillation, and self-distillation—based on whether the teacher model is updated simultaneously with the student model. Offline Distillation Most traditional knowledge distillation methods operate in an offline manner [53][54][55]. In standard knowledge distillation, the knowledge is transferred from a pre-trained teacher model to a student model.

This process generally unfolds in two phases: (1) the teacher model is initially trained on a set of training samples before the distillation step, and (2) once trained, the teacher model's knowledge, in the form of logits or intermediate features, is used to guide the student model's learning during the distillation phase. The first phase, involving the training of the teacher model, is typically assumed to be predefined and is not usually discussed in detail within the context of knowledge distillation. Consequently, little attention is given to the teacher model's architecture or how it relates to the student model. Instead, offline distillation methods primarily focus on improving various aspects of the knowledge transfer process, such as the design of knowledge representations and the loss functions used to match features or distributions.

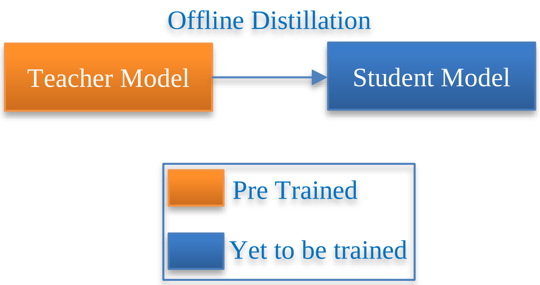

Fig. 7 Illustration of offline distillation

The main advantage of offline distillation lies in its simplicity and ease of implementation. For example, the teacher model may consist of a collection of models trained using different software frameworks or located on different machines and the extracted knowledge can be cached for later use. Offline distillation generally involves a one-way transfer of knowledge, with a two-phase training procedure where the teacher's knowledge is used to supervise the student model's learning.

However, a key limitation is that the training of the teacher model is resource-intensive and time-consuming, and it remains unavoidable even in offline distillation. On the other hand, the student model's training in offline distillation is typically more efficient, benefiting from the guidance provided by the teacher model. Additionally, a fundamental challenge remains: the substantial capacity gap between the large teacher model and the smaller student model. Consequently, the student often heavily depends on the teacher model's knowledge, making it difficult to achieve a fully autonomous student model.

Online Distillation While offline distillation methods are simple and effective, several limitations have sparked increasing attention from the research. To address these shortcomings, online distillation [52] has emerged as an alternative, particularly beneficial when a large-capacity, high-performance teacher model is unavailable. In online distillation, both the teacher and student models are updated simultaneously, enabling the entire knowledge distillation process to be trained end-to-end. This paradigm has gained significant interest in recent years, leading to the development of various online distillation approaches.

Online distillation [49][59] offers the advantage of a one-phase, end-to-end training process, often benefiting from efficient parallel computing. However, existing online approaches—such as mutual learning—still struggle with integrating high-capacity teacher models in online settings, which presents an intriguing avenue for further research. Understanding the relationship between teacher and student models in online environments is a critical and unresolved issue in the field.

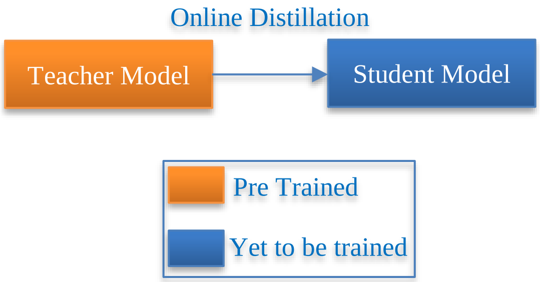

Fig. 8 Illustration of online distillation

Self-Distillation In self-distillation, the same network is used for both the teacher and student models. This approach can be viewed as a specific instance of online distillation. For instance, Zhang et al. (2019b) [33] introduced a novel self-distillation method where knowledge from the deeper layers of the network is transferred to the shallower layers. A similar approach, termed self-attention distillation, was proposed for lane detection, where the network leverages its attention

In addition to these approaches, several innovative self-distillation methods [52] have emerged recently. Yuan et al. (2020) [62] introduced a teacher-free knowledge distillation framework based on label smoothing regularization. Hahn and Choi (2019) [57] proposed a novel variant of self-knowledge distillation, where the "self-knowledge" consists of predicted probabilities rather than traditional soft probabilities. These predicted probabilities are derived from the feature representations of the training model, reflecting the data similarities in the feature embedding space.

Yun et al. (2020) introduced class-wise self-knowledge distillation, aiming to match the output distributions between intra-class samples and augmented samples from the same source within the same model. Moreover, Lee et al. (2019a) [58] applied self-distillation for data augmentation, distilling the augmentation-based self-knowledge into the model itself. Other approaches, such as those by Furlanello et al. (2018) [60] and Bagherinezhad et al. (2018) [61], employ self-distillation to optimize deep models by using a teacher-student optimization framework with identical architectures.

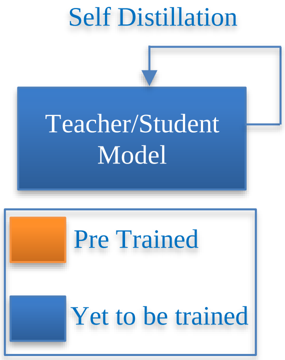

Fig. 9 Illustration of Self distillation

The concepts of offline, online, and self-distillation can also be intuitively compared to human teacher-student learning dynamics. Offline distillation involves a knowledgeable teacher imparting knowledge to a student; online distillation features both teacher and student learning collaboratively, while self-distillation corresponds to a student independently acquiring knowledge. Furthermore, these distillation strategies can complement each other by leveraging their respective strengths. For example, a multiple knowledge transfer framework has been proposed to integrate both self-distillation and online distillation effectively, combining the advantages of each approach to enhance model performance.

11. Low-Rank Decomposition

Low-rank decomposition is a technique used to approximate large, dense matrices by expressing them as the product of smaller, lower-rank matrices. This approach leverages the idea that many large matrices, particularly in neural networks, contain significant redundant information that can be captured with fewer parameters. In the context of LLMs, low-rank decomposition is employed to reduce the memory and computational overhead associated with the massive weight matrices, particularly in layers like attention mechanisms and feedforward networks in Transformer architectures. By approximating these large matrices, low-rank decomposition helps shrink the model size, leading to faster inference times and lower memory requirements, making LLMs more efficient and deployable in resource-constrained environments. However, while the technique offers notable advantages, such as reduced computational complexity and improved efficiency, it also comes with trade-offs. The primary disadvantage is a potential loss in model accuracy, as the low-rank approximation may not fully capture the complexity of the original model.

In the context of Large Language Models (LLMs), low-rank decomposition can be used to reduce the computational complexity and memory requirements of large, dense-weight matrices. Just as in convolutional neural networks (CNNs), where low-rank decomposition accelerates convolutional operations, applying this technique to LLMs can improve the efficiency of their fully connected layers, which often have enormous parameter counts.

12. Combination of Compression (CoC)

Throughout this paper, various compression techniques have been discussed. Combining multiple compression methods within the same model to create smaller models with minimal accuracy loss is an approach that has been used for some time. For instance, by combining pruning and quantization, the number of parameters is reduced, and further precision reduction leads to a significant reduction in model size. Similarly, applying quantization to the student model derived through knowledge distillation presents an intriguing area for future research.

13. Conclusion

This survey paper highlights the most widely used compression techniques today, including pruning, quantization, knowledge distillation, low-rank decomposition, and the combination of multiple compression methods (CoC). An in-depth analysis of each technique explores the various methods within each category. Different types of pruning were discussed—static and dynamic—and examine how structured and unstructured approaches can be used to prune layers effectively. Quantization techniques are divided it into two main types: quantization-aware training and post-training quantization. Explored how these techniques are applied to query-key pairs, activations, and weights, and also discussed the impact of fixed-precision versus mixed-precision quantization, among other considerations. Knowledge distillation, covers three types of teacher-student models: feature-based, relation-based, and result-based, each of which can be further adapted into online, offline, and self-distillation techniques.

Finally, explored how the low-rank decomposition technique can be applied to large language models (LLMs). This survey is the first comprehensive paper to summarize these key compression techniques, providing a valuable resource for ongoing research in the field.

References

[1] R. Rosenfeld, “Two Decades of Statistical Language Modeling: Where Do We Go From Here?,” *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1270-1278, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[2] Joshua T. Goodman, “*A Bit of Progress in Language Modeling: Extended Version*,” Microsoft Research Technical Report MSR-TR-2001-72, 2001. [Google Scholar] [Publisher Link]

[3] Frederick Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, pp. 1-283, 1997. [Google Scholar] [Publisher Link]

[4] S. Katz, “Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400-401, 1987. [CrossRef] [Google Scholar] [Publisher Link]

[5] Stanley F. Chen, and Joshua Goodman, “An Empirical Study of Smoothing Techniques for Language Modeling,” *Computer Speech & Language*, vol. 13, no. 4, pp. 359-394, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[6] Jacob Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv*, pp. 1-16, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[7] Alec Radford et al., “*Improving Language Understanding by Generative Pre-Training*,” OpenAI Technical Report, pp. 1-12, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[8] Yinhan Liu et al., “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *arXiv*, pp. 1-13, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Zhilin Yang et al., “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 5753-5763, 2019. [Google Scholar] [Publisher Link]

[10] Yu Sun et al., “ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation,” *arXiv*, pp. 1-22, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[11] Tom Brown et al., “Language Models are Few-Shot Learners,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020. [Google Scholar] [Publisher Link]

[12] Susan Zhang et al., “OPT: Open Pre-trained Transformer Language Models,” *arXiv*, pp. 1-30, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[13] Mohammad Shoeybi et al., “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism,” *arXiv*, pp. 1-15, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[14] Lewis Tunstall, Leandro von Werra, and Thomas Wolf, *Natural Language Processing with Transformers*, O'Reilly Media, pp. 1-408, 2022. [Google Scholar] [Publisher Link]

[15] Barret Zoph et al., “ST-MoE: Designing Stable and Transferable Sparse Expert Models,” *arXiv*, pp. 1-38, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[16] Wayne Xin Zhao et al., “A Survey of Large Language Models,” *arXiv*, pp. 1-140, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[17] Victor Sanh et al., “DistilBERT, A Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter,” *arXiv*, pp. 1-5, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[18] Y. LeCun et al., “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097-1105, 2012. [Google Scholar] [Publisher Link]

[20] David Silver et al., “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, pp. 484-489, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[21] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang, “Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers,” *Advances in Neural Information Processing Systems*, vol. 32, 2019. [Google Scholar] [Publisher Link]

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, “BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations,” *Advances in Neural Information Processing Systems*, vol. 28, 2015. [Google Scholar] [Publisher Link]

[23] Song Han, Huizi Mao, and William J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *arXiv*, pp. 1-14, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[24] Hao Li et al., “Pruning Filters for Efficient ConvNets,” *arXiv*, pp. 1-13, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[25] Suraj Srinivas, and R. Venkatesh Babu, “Data-Free Parameter Pruning for Deep Neural Networks,” *Proceedings of the British Machine Vision Conference*, 2015. [Google Scholar] [Publisher Link]

[26] Song Han et al., “Learning both Weights and Connections for Efficient Neural Networks,” *Advances in Neural Information Processing Systems*, vol. 28, 2015. [Google Scholar] [Publisher Link]

[27] Pavlo Molchanov et al., “Importance Estimation for Neural Network Pruning,” *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11264-11272, 2019. [Google Scholar] [Publisher Link]

[28] Aditya Kusupati et al., “Soft Threshold Weight Reparameterization for Learnable Sparsity,” *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, pp. 5544-5555, 2020. [Google Scholar] [Publisher Link]

[29] Junjie Liu et al., “Dynamic Sparse Training: Find Efficient Sparse Networks from Scratch with Trainable Masks,” *Advances in Neural Information Processing Systems*, pp. 1-14, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[30] Yiwen Guo, Anbang Yao, and Yurong Chen, “Dynamic Network Surgery for Efficient DNNs,” *Advances in Neural Information Processing Systems*, vol. 29, 2016. [Google Scholar] [Publisher Link]

[31] Xiangfei Hu et al., “LLaMA-Adapter: Efficient Fine-Tuning of Language Models with Zero-Init Attention,” *arXiv*, pp. 1-30, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[32] Brian Lester, Rami Al-Rfou, and Noah Constant, “The Power of Scale for Parameter-Efficient Prompt Tuning,” *arXiv*, pp. 1-15, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[33] Linfeng Zhang et al., “Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks Via Self Distillation,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713-3722, 2019. [Google Scholar] [Publisher Link]

[34] Ron Banner, Yury Nahshan, and Daniel Soudry, "Post Training 4-bit Quantization of Convolutional Networks for Rapid-Deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019. [Google Scholar] [Publisher Link]

[35] Ofir Zafrir et al., "Q8BERT: Quantized 8bit BERT," *Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition*, Vancouver, BC, Canada, pp. 36-39, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[36] Pierre Stock et al., "And the Bit Goes Down: Revisiting the Quantization of Neural Networks," *arXiv*, pp. 1-11, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[37] Tim Dettmers et al., "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv*, pp. 1-26, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[38] Hao Wu et al., "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," *arXiv*, pp. 1-20, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[39] Benoit Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704-2713, 2018. [Google Scholar] [Publisher Link]

[40] Steven K. Esser et al., "Learned Step Size Quantization," *arXiv*, pp. 1-12, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[41] Yunfan Shao et al., "Character-LLM: A Trainable Agent for Role-Playing," arXiv, pp. 1-35, 2023. [Google Scholar] [Publisher Link]

[42] Markus Nagel et al., "Up or Down? Adaptive Rounding for Post-Training Quantization," *arXiv*, pp. 1-12, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[43] Zhewei Yao et al., "ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers," *arXiv*, pp. 1-24, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[44] Vishnu Suresh Lokhande et al., "Generating Accurate Pseudo-Labels in Semi-Supervised Learning and Avoiding Overconfident Predictions via Hermite Polynomial Activations," *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seattle, WA, USA, pp. 11432-11440, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[45] Zhen Dong et al., "HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision," *arXiv*, pp. 1-12, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[46] Kuan Wang et al., "HAQ: Hardware-Aware Automated Quantization with Mixed Precision," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8612-8620, 2019. [Google Scholar] [Publisher Link]

[47] Gustavo Aguilar et al., "Knowledge Distillation from Internal Representations," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 5, pp. 7350-7357, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[48] Angeline Aguinaldo et al., "Compressing GANs Using Knowledge Distillation," *arXiv*, pp. 1-10, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[49] Rohan Anil et al., "Large Scale Distributed Neural Network Training through Online Distillation," *arXiv*, pp. 1-12, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[50] Yoshua Bengio, Aaron Courville, and Pascal Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798-1828, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[51] William Chan, Nan Rosemary Ke, and Ian Lane, "Transferring Knowledge from an RNN to a DNN," *arXiv*, pp. 1-5, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[52] Yoonho Boo et al., "Stochastic Precision Ensemble: Self-Knowledge Distillation for Quantized Deep Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 6794-6802, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[53] Defang Chen et al., "Online Knowledge Distillation with Diverse Peers," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 4, pp. 3430-3437, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[54] Defang Chen et al., "Cross-Layer Distillation with Semantic Calibration," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 7028-7036, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[55] Vasileios Belagiannis, Azade Farshad, and Fabio Galasso, "Adversarial Network Compression," *Proceedings of the European Conference on Computer Vision Workshops*, 2018. [Google Scholar] [Publisher Link]

[56] Haoli Bai et al., "Few Shot Network Compression Via Cross Distillation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 4, pp. 3203-3210, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[57] Sangchul Hahn, and Heeyoul Choi, "Self-Knowledge Distillation In Natural Language Processing," *arXiv*, pp. 1-8, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[58] Hankook Lee, Sung Ju Hwang, and Jinwoo Shin, "Rethinking Data Augmentation: Self-Supervision and Self-Distillation," *ICLR 2020 Conference*, pp. 1-11, 2019. [Google Scholar] [Publisher Link]

[59] Massimo Caccia et al., "Online Fast Adaptation and Knowledge Accumulation (OSAKA): A New Approach to Continual Learning," *Advances in Neural Information Processing Systems NeurIPS*, vol. 33, 2020. [Google Scholar] [Publisher Link]

[60] Tommaso Furlanello et al., "Born-Again Neural Networks," *Proceedings of the 35th International Conference on Machine Learning*, pp. 1607-1616, 2018. [Google Scholar] [Publisher Link]

[61] Hessam Bagherinezhad et al., “Label Refinery: Improving Imagenet Classification through Label Progression,” *arXiv*, pp. 1-16, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[62] Li Yuan et al., “Revisit Knowledge Distillation: A Teacher-Free Framework,” *ICLR 2020 Conference*, pp. 1-14, 2020. [Google Scholar] [Publisher Link]