A Comprehensive Survey of Compression Algorithms for Language Models

SEUNGCHEOL PARK, Computer Science and Engineering, Seoul National University, Gwanak-gu, Korea (the Republic of)
JAEHYEON CHOI, Computer Science and Engineering, Seoul National University, Gwanak-gu, Korea (the Republic of)
SOJIN LEE, Computer Science and Engineering, Seoul National University, Gwanak-gu, Korea (the Republic of)
U KANG, Computer Science and Engineering, Seoul National University, Gwanak-gu, Korea (the Republic of)

How can we compress language models without sacrificing accuracy? The number of compression algorithms for language models is rapidly growing to leverage the remarkable advances of recent language models without side effects induced by their gigantic size, including increased carbon emissions, high latency, and restricted usage on resource-constrained mobile devices. While numerous compression algorithms have shown remarkable progress in compressing language models, it ironically becomes challenging to capture emerging trends and identify the fundamental concepts underlying them due to the excessive number of algorithms. In this article, we survey and summarize diverse compression algorithms including pruning, quantization, knowledge distillation, low-rank approximation, and dynamic inference. We not only summarize the overall trend of diverse compression algorithms but also select representative algorithms and provide in-depth analyses of them. Finally, we introduce and discuss promising future research directions.

CCS Concepts: • **Computing methodologies → Natural language processing;**

Additional Key Words and Phrases: Natural language processing, model compression, transformer, LLM

Seungcheol Park, Jaehyeon Choi and Sojin Lee contributed equally to this research.

This work was supported by Youlchon Foundation. This work was also supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) [No.2020-0-00894, Flexible and Efficient Model Compression Method for Various Applications and Environments], [No.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)], [No.RS-2025-25442338, AI star Fellowship Support Program(Seoul National Univ.)], and [No.RS-2024-00509257, Global AI Frontier Lab]. The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study.

Authors' Contact Information: Seungcheol Park, Computer Science and Engineering, Seoul National University, Gwanak-gu, Seoul, Korea (the Republic of); e-mail: ant6si@snu.ac.kr; Jaehyeon Choi, Computer Science and Engineering, Seoul National University, Gwanak-gu, Seoul, Korea (the Republic of); e-mail: jaehyeon_choi@snu.ac.kr; Sojin Lee, Computer Science and Engineering, Seoul National University, Gwanak-gu, Seoul, Korea (the Republic of); e-mail: lsjlsj5846@snu.ac.kr; U Kang (corresponding author), Computer Science and Engineering, Seoul National University, Gwanak-gu, Seoul, Korea (the Republic of); e-mail: ukang@snu.ac.kr.

ACM 0360-0300/2026/10-ART357
https://doi.org/10.1145/3819816

ACM Reference Format:
Seungcheol Park, Jaehyeon Choi, Sojin Lee, and U Kang. 2026. A Comprehensive Survey of Compression Algorithms for Language Models. *ACM Comput. Surv.* 58, 14, Article 357 (October 2026), 35 pages. https://doi.org/10.1145/3819816

1 Introduction

How can we compress language models without sacrificing accuracy? Language models [9, 118, 130, 131, 165] have achieved remarkable performance in various tasks by increasing their size up to hundreds of billions of parameters. The necessity of compressing language models has become significant to leverage advanced language models without side effects due to their gigantic size. For example, compression algorithms reduce carbon emissions for utilizing language models, accelerate the inference of language models, and facilitate the deployment of high-performance language models in resource-constrained environments, such as mobile devices. A large number of compression algorithms including pruning [30, 78, 96, 105, 148], quantization [24, 32, 61, 149], **knowledge distillation** (**KD**) [41, 84, 107, 126], **low-rank approximation** (**LRA**) [50, 101, 135, 164], and dynamic inference [94, 100, 119] have been proposed, and they significantly improve the efficiency of language models; however, an excessive number of research papers make it hard to figure out the overall trend and underlying fundamentals of compression algorithms. Although existing surveys [18, 23, 34, 35, 40, 46, 63] attempted to address this issue, they omit scalable compression algorithms to be applied to **large language models** (**LLMs**), which exhibit remarkable performance.

In this article, we conduct an extensive survey of various compression algorithms, focusing on how to improve the performance of scalable compression algorithms applicable to LLMs. Figure 1 shows the taxonomy of compression algorithms for language models covered in this article. We cover pruning, quantization, KD, LRA, and dynamic inference, the most effective compression algorithms. We group those algorithms into two main categories: high-cost and low-cost. High-cost compression algorithms require full-model training on large datasets, which is computationally expensive and time-consuming. High-cost algorithms are typically studied on million-scale language models, such as BERT, and they are too expensive to apply to billion-scale language models in general. On the other hand, low-cost compression algorithms require only a few samples and lightweight tuning, such as block-wise, layer-wise, or adaptor tuning, or even no tuning at all. Low-cost algorithms are designed to be applicable to billion-scale language models, but they exhibit reduced performance compared with the high-cost ones due to the restricted cost for tuning. In this survey, we encompass both high-cost and low-cost compression algorithms, focusing on revealing the essential properties to improve the performance of low-cost compression algorithms by carefully analyzing the success in high-cost compression algorithms. We compare the performance of compression algorithms and summarize their development trends. Furthermore, we select representative algorithms that significantly impact language model compression and conduct in-depth analyses of them. We propose five promising research directions to enhance the performance of language model compression algorithms and discuss them based on our survey results.

We summarize the main contribution of our article as follows.

- **Overview.** We summarize an overall trend of compression algorithms for language models. Our article covers diverse types of compression algorithms, including both high-cost and low-cost compression algorithms.
- **Analysis.** We select three representative algorithms each from pruning, quantization, and other compression algorithms, elaborating their details to deliver meaningful insights.

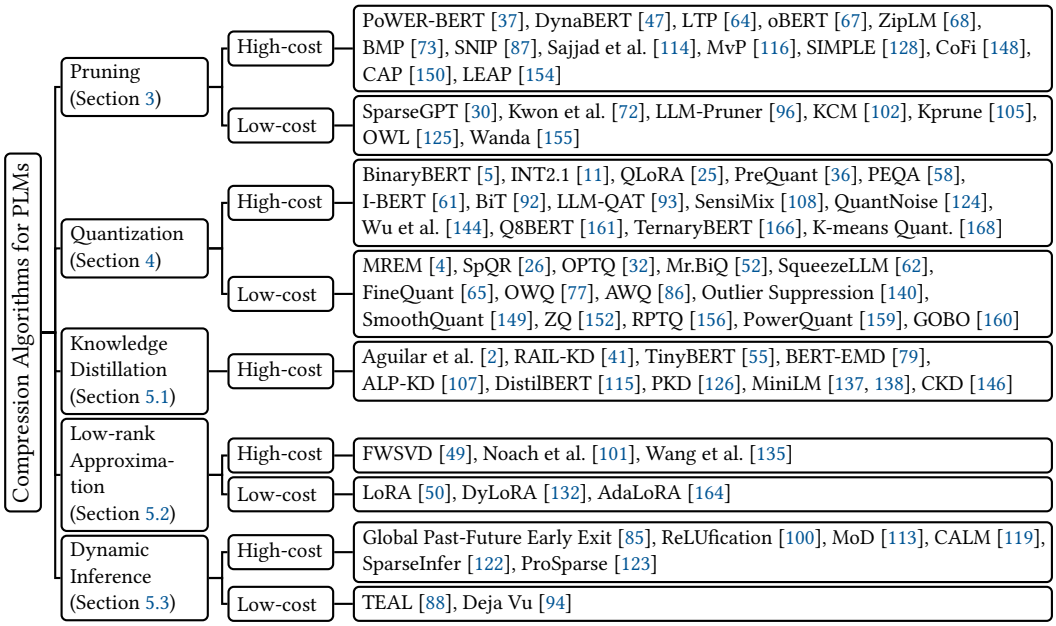


Fig. 1. Taxonomy of compression algorithms for language models.

— **Discussion.** We provide valuable discussion regarding compression algorithms, and introduce promising future research topics.

The rest of this article is organized as follows. In Section 2, we formally define a language model compression problem and describe related preliminaries. In Sections 3–5, we summarize existing compression algorithms and provide a detailed explanation of the representative algorithms regarding pruning, quantization, and other compression algorithms. We discuss promising future research areas found in our survey in Section 6 and conclude in Section 7.

2 Preliminaries

In this section, we formally define the language model compression problem in Section 2.1, introduce Transformer architecture which dominates the state-of-the-art language models in Section 2.2, give an overview of language model compression algorithms in Section 2.3, and describe the backgrounds to solve the language model compression problem in Section 2.4. Table 1 presents notations frequently used in this article.

2.1 Language Model Compression Problem

The language model compression problem is defined as follows. We have an accurate and large **pretrained language model** (**PLM**) $\mathcal{T}$ and dataset $\mathbb{D} = \{(d_i, y_i)\}$ where (d_i, y_i) is the tuple of the ith data point and its label. Our goal is to generate a tiny but still accurate compressed language model $\mathcal{S}$ that meets cost constraints, usually, in terms of FLOPs [72, 102], memory usage [74, 135], or latency [73, 148] upper bounds. In this article, we express the reduced memory usage ratio compared with the original PLM as the compression rate. If such information is not reported, we use the reduction in FLOPs as the compression rate and annotate with ♠. We assume that we have an accurate PLM, according to the practical settings in a dominant number of existing

Table 1. Symbols and Descriptions

Symbol	Description	Symbol	Description
$\mathcal{T}$	a pretrained language model (PLM)	$\mathbb{D}$, $\mathbb{S}$	target and calibration datasets
$\mathcal{S}$	a compressed language model	(d_i, y_i)	a tuple of the ith data and its label
$M(\cdot)$	a multi-head attention (MHA) function	X	an input matrix of a sublayer
$F(\cdot)$	a feedforward network (FFN) function	x_i	the ith token embedding in X
A_i	an output of the ith attention head	d	the dimension of token embeddings
Q_i, K_i, V_i	query, key, and value matrices	H	the number of attention heads
W^Q, W^K, W^V, W^O	weights in an MHA sublayer	$a(\cdot)$	a position-wise affine transformation
b^Q, b^K, b^V, b^O	biases in an MHA sublayer	$S(\cdot)$	a column-wise softmax function
U^I, U^O	weights in an FFN sublayer	$\mathcal{N}(\cdot)$	a layer normalization function
c^I, c^O	biases in an FFN sublayer	$\|\|$	vertical concatenation
$\mathcal{L}$	a task-specific objective function	g	a gradient of $\mathcal{L}$
H	a Hessian matrix of $\mathcal{L}$	$Q(\cdot)$	a quantization operator (quantizer)

works [24, 50, 135, 148, 149] based on the findings that language models are easily adapted to unexperienced tasks when they are pretrained on an extremely large dataset [9, 27]. The label y can be omitted for the unlabeled tasks such as language modeling [24, 96, 149]. For low-cost compression algorithms [72, 102, 152, 153], we use a small calibration dataset $\mathbb{S} \subset \mathbb{D}$ to minimize their cost.

2.2 Transformer Architecture

We elaborate on Transformer architecture which is dominantly used for PLMs [9, 130, 131, 165] in **natural language processing** (**NLP**). Transformer [133] is a language model that refines token embeddings reflecting the contextual information through a self-attention mechanism [17, 103, 133] to utilize them in downstream tasks. Transformer is composed of multiple layers consisting of **multi-head attention** (**MHA**) and **feed-forward network** (**FFN**) sublayers. MHA sublayers capture the contextual inter-token information via a self-attention mechanism and FFN sublayers polish intra-token information with a position-wise feedforward network. Original Transformer architecture [133] comprises Encoder and Decoder where Encoder encodes input token embeddings considering contextual information and Decoder generates output tokens by decoding the encoding results. However, we use encoder-only and decoder-only Transformers according to downstream tasks based on the previous works demonstrating the impressive performance of encoder-only [19, 27, 74, 90, 115] and decoder-only [9, 118, 130, 165] PLMs. We illustrate encoder-only and decoder-only Transformer architectures in Figure 2. Encoder-only Transformers (a) [19, 27, 74, 90, 115] generate useful embeddings that reflect contextual information within an input sequence via (bidirectional) self-attention. Encoder-only Transformers are used for **natural language understanding** (**NLU**) tasks including sentence similarity [10, 28], natural language inference [134, 141], and question answering [111, 112]. On the other hand, decoder-only Transformers (b) [9, 118, 130, 165] autoregressively predict output tokens via (unidirectional) masked self-attention which attends only current and previous tokens. We concatenate an output token to the end of an input sequence, and feed the augmented input sequence to the next iteration. Decoder-only models are used for **natural language generation** (**NLG**) tasks, such as question answering [57, 70], mathematical reasoning [20, 44], and code generation [14]. Note that decoder-only Transformers are broadly applicable since text-based language problems are convertible into a text-to-text format [110].

Figure 3 illustrates the three mechanisms in Transformer. Assume that an input $X = [x_1; x_2; ...; x_s] \in \mathbb{R}^{d \times s}$ consists of d-dimensional s token embeddings from the previous layer. The

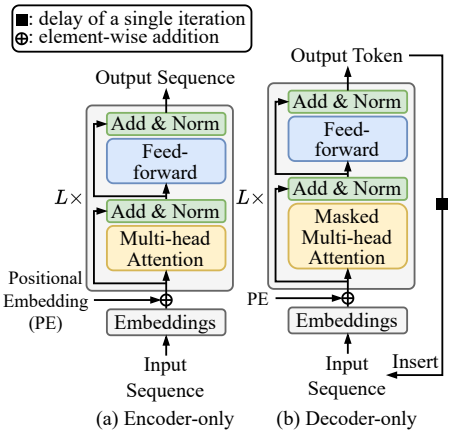


Fig. 2. Two variants of Transformer architecture.

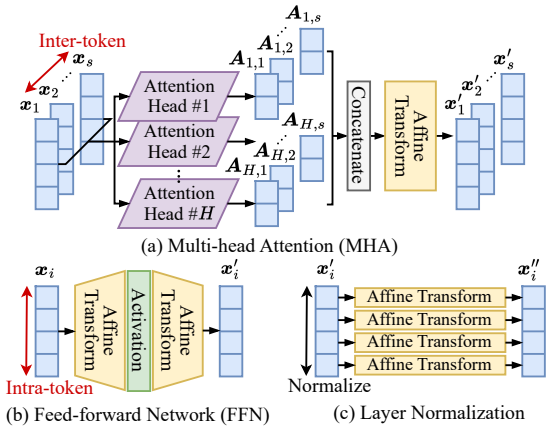


Fig. 3. Illustrations of multi-head attention (MHA), feed-forward network (FFN), and layer normalization.

output of each sublayer is computed as either $\mathcal{N}(X+M(X))$ for MHA sublayers or $\mathcal{N}(X+F(X))$ for FFN sublayers, where $M(\cdot)$ and $F(\cdot)$ are MHA and FFN functions, respectively. $\mathcal{N}(\cdot)$ represents layer normalization [3]. MHA extracts contextual information across tokens via self-attention. Multiple attention heads capture diverse patterns on attention maps; each head uses shrunk dimension $d_h = d/H$ to reduce the cost of using multiple heads, where H is the number of attention heads. We obtain an output $M(X)$ by applying a position-wise affine transformation on the concatenated output of attention heads, as in Equation (1). $\|$ represents vertical concatenation along embedding dimension, and $A_i \in \mathbb{R}^{d_h \times s}$ is an output of the ith attention head. $a(\cdot\,; W^O, b^O) : \mathbb{R}^{d \times s} \rightarrow \mathbb{R}^{d \times s}$ is a position-wise affine transformation parameterized by weight $W^O \in \mathbb{R}^{d \times d}$ and bias $b^O \in \mathbb{R}^d$. As in Equation (2), we compute the jth column of $M(X)$ as the affine transformation of the jth token embedding, and an output A_i of attention heads via scaled dot-product attention [133]. Query Q_i, key K_i and value V_i matrices are obtained via position-wise affine transformations of the input X with different weights $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_h \times d}$ and biases $b_i^Q, b_i^K, b_i^V \in \mathbb{R}^{d_h}$ in each attention head. Note that $Q_i, K_i, V_i \in \mathbb{R}^{d_h \times s}$, and $S(\cdot)$ is a column-wise softmax function across tokens for scoring.

$$M(X) = a\left(\|_{i=1}^{H} A_i; W^O, b^O\right) \quad where \quad A_i = V_i S\left(\left(K_i^T Q_i\right)/\sqrt{d_h}\right) \tag{1}$$

$$a\left(\|_{i=1}^{H} A_i; W^O, b^O\right)_{:,j} = W^O \left(\|_{i=1}^{H} A_i\right)_{:,j} + b^O \tag{2}$$

FFN separately and identically refines each token embedding via two position-wise affine transformations with weights $U^I \in \mathbb{R}^{d_f \times d}$, $U^O \in \mathbb{R}^{d \times d_f}$ and biases $c^I \in \mathbb{R}^{d_f}$, $c^O \in \mathbb{R}^d$. We transform each token embedding into the higher dimension of d_f to capture useful patterns within a token embedding; σ is a non-linear activation function, such as ReLU [1] or GELU [45].

$$F(X) = a\left(\sigma\left(a\left(X; U^I, c^I\right)\right); U^O, c^O\right) \tag{3}$$

We describe an example of layer normalization [3] in Figure 3 (c). Layer normalization is a modification of batch normalization [51] adaptable for NLP with flexible lengths of sequences. Layer normalization normalizes each token embedding and performs element-wise affine transformations. The normalized output of jth element of the ith input token x_i' is described in Equation (4) where

$\mu_i = \frac{1}{d}\sum_{j=1}^{d}\left(x_i'\right)_j$ and $\nu_i = \frac{1}{d}\sum_{j=1}^{d}\left(\left(x_i'\right)_j - \mu_i\right)^2$ are the mean and variance of the elements in x_i'. $\beta \in \mathbb{R}^d$ and $\gamma \in \mathbb{R}^d$ are learnable parameters for the element-wise affine transformations.

$$\left(x_i''\right)_j = \mathcal{N}(x_i')_j = \frac{\left(x_i'\right)_j - \mu_i}{\sqrt{\nu_i}}\gamma_j + \beta_j \tag{4}$$

2.3 Language Model Compression Algorithms

The goal of language model compression is to reduce the memory and computational costs of utilizing language models. Compression algorithms reduce the memory cost of representing weights and intermediate activations, and computational cost for computing intermediate activations. We categorize compression algorithms based on how they reduce these costs as follows.

- **Pruning (Section 3)** identifies and removes unnecessary weights of PLMs. The size of activation is also decreased when using a large group of parameters as a pruning granularity.
- **Quantization (Section 4)** reduces the bit-length to represent weights. We achieve acceleration in matrix multiplications by reducing the bit-length of the activations as well.
- **Knowledge distillation (Section 5.1)** improves the accuracy of compressed models by transferring useful knowledge from the PLM. It is typically combined with other compression algorithms to maximize the accuracy of the compressed models.
- **Low-rank approximation (Section 5.2)** substitutes the weights with the multiplication of low-dimensional matrices based on the low-rank assumption. The memory and computational costs are reduced since the number of weights is largely reduced.
- **Dynamic inference (Section 5.3)** dynamically selects activated PLM modules depending on inputs. It reduces inference cost by computing activations only in the selected modules.

We additionally classify PLM compression algorithms into low-cost and high-cost ones, considering the importance of compressing massive LLMs, independently of the aforementioned classification criterion. High-cost compression algorithms require extensive retraining processes on large datasets. They consequently generate more accurately compressed models; however, they are hardly affordable for gigantic LLMs. On the other hand, low-cost compression algorithms are applicable to compress LLMs; they do not require post-compression tuning, or perform only a simple weight-tuning process, such as block-wise tuning, layer-wise tuning, and adapter tuning, on small calibration data. Low-cost compression algorithms often suffer from severe accuracy degradation, and improving the accuracy is their main concern.

2.4 Backgrounds

In this section, we explain fundamental backgrounds to solve language model compression problem.

2.4.1 Taylor Expansion. Language model compression is a procedure that imposes perturbations δw on weights w in a PLM. Examples include setting weights to zeros [43, 75], low-precision representations [32, 61, 108], and low-rank representations [49, 135]. We leverage Taylor expansion to estimate the amount $\delta\mathcal{L}$ of the increment in an objective function $\mathcal{L}$ after applying the perturbation δw on weights as in Equation (5) [75].

$$\delta\mathcal{L} = \delta w^T g + \frac{1}{2}\delta w^T H \delta w + \mathcal{O}(\delta w^3) \approx \frac{1}{2}\delta w^T H \delta w, \tag{5}$$

where $g = \mathbb{E}\left[\frac{\partial\mathcal{L}(w^*)}{\partial w}\right]$ is the gradient and $H = \mathbb{E}\left[\frac{\partial^2\mathcal{L}(w^*)}{\partial w^2}\right]$ is the Hessian of $\mathcal{L}$ with respect to w. We approximate $g \approx 0$ since the PLM is pretrained at a local minimum whose gradient is zero. We neglect $\mathcal{O}(\delta w^3)$ since the size of the perturbation is sufficiently small. As a result, we have only a

quadratic term with the Hessian matrix H. Using Taylor expansion is efficient since we compute $\delta\mathcal{L}$ for any δw by computing the Hessian matrix once; we do not have to manually compute $\delta\mathcal{L}$ for each δw. We utilize the derivation in Equation (5) in various compression algorithms, for example, measuring sensitivity in quantization [62], measuring saliency score in pruning [43, 47, 72, 75], and selecting rank-1 components in low-rank approximation [49].

2.4.2 Lagrangian. Lagrangian function is used to solve a constrained optimization problem. Consider the problem of minimizing an objective function $f(w)$ with equality constraints $a^T w+b = 0$ on weights w. In language model compression problems, the objective function $f(w)$ is either cross-entropy loss or layer-wise least-square loss. a and b are vectors of constants to state constraints; e.g., $a_q = 1$ and $b_q = 0$ represent that the qth weight is pruned. The solution of the problem is computed by first forming the following Lagrangian function

$$L(w, \lambda) = f(w) + \lambda(a^T w + b) \tag{6}$$

and finding the stationary point of $L(w, \lambda)$ with respect to w; i.e., $\frac{\partial L(w,\lambda)}{\partial w} = 0$. For example, **optimal brain surgeon** (**OBS**) [43] finds the optimal perturbations (surgery) on weights through the Lagrangian function. The optimal surgery in OBS is crucial and widely used in state-of-the-art compression algorithms both in pruning [30, 68] and quantization [32, 86, 156]. We elaborate on more details about these algorithms in Sections 3.4 and 4.4.

2.4.3 Fisher Information Matrix. A Fisher information matrix is an information measure in statistics. However, in model compression, we use a Fisher information matrix to approximate a Hessian matrix since the Hessian matrix H of the cross-entropy loss $\mathcal{L}$ is able to be approximated by the Fisher information matrix $\mathcal{I}$ (see Lee et al. [76] for derivation). As in Equation (7), we approximate the Fisher information matrix $\mathcal{I}$ as an empirical Fisher information matrix $\hat{\mathcal{I}}$ estimated on the sample dataset $\mathbb{D}$ since computing the exact form is intractable. d is a data point in $\mathbb{D}$.

$$\mathbb{E}_d[H] = \mathbb{E}_d\left[\frac{\partial^2 \mathcal{L}(d; w)}{\partial w^2}\right] \approx \frac{1}{|\mathbb{D}|}\sum_{d\in\mathbb{D}}\left[\left(\frac{\partial \mathcal{L}(d; w)}{\partial w}\right)\left(\frac{\partial \mathcal{L}(d; w)}{\partial w}\right)^T\right] = \hat{\mathcal{I}} \tag{7}$$

We use the Fisher information matrix to approximate the Hessian in Equation (5) since computing an exact Hessian matrix of cross-entropy loss is computationally expensive. Note that Equation (7) holds only when the objective function is a cross-entropy loss; for other objective functions, we need to find alternative ways to compute the Hessian. For example, when our loss function is a layer-wise least-square loss [30, 32, 43], we directly compute the Hessian matrix.

2.4.4 Straight-Through Estimator (STE). STE [7] is a technique of passing upstream gradients directly to the next node in back propagation. We leverage STE to train a model that entails non-differentiable functions, such as a **round function** (**RF**). Weights cannot be updated in the backward pass since the gradient of RF is either 0 or not defined. We, therefore, substitute the derivative of RF with STE, which enables the gradient update by having gradients of 1 in all regions. Suppose, we quantize a weight matrix W using a RF $Q(\cdot)$, and compute the loss $\mathcal{L}$ in the forward pass. In the backward pass, the gradient of W regarding the loss $\mathcal{L}$ is approximated via STE as in Equation (8).

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial Q(W)} \cdot \frac{\partial Q(W)}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial Q(W)}. \tag{8}$$

STE is one of the core methodologies for quantization schemes that involve weight update with gradients, named quantization-aware training (QAT).

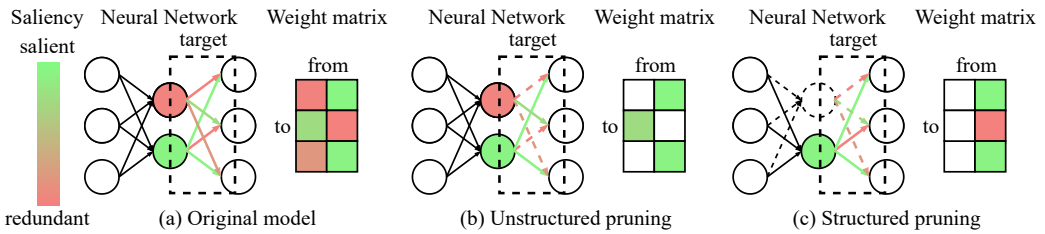


Fig. 4. An illustration of a two-layer neural network before and after pruning. (a) Shows the original network before pruning. (b) and (c) Depict the results of unstructured and structured pruning, respectively.

2.4.5 Singular Value Decomposition (SVD). SVD is a factorization of a matrix. SVD reduces the number of weights to represent weight matrices in PLMs. Given a weight matrix $W \in \mathbb{R}^{m \times n}$, we decompose the matrix with SVD as in Equation (9). $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$ are column orthogonal matrices where r is the rank of W. S is a diagonal matrix consisting of r singular values $\{s_i\}_{i=1}^{r}$.

$$W = USV^T = \sum_{i=1}^{r} s_i U_{:,i} (V_{:,i})^T \approx \sum_{i=1}^{k} s_i U_{:,i} (V_{:,i})^T = \overbrace{\left(U_{:,:k} (S_{:k})^{1/2}\right)}^{m \times k} \overbrace{\left((S_{:k})^{1/2} (V_{:,:k})^T\right)}^{k \times n} \quad k \ll r \tag{9}$$

We transform the equation into the sum of rank-1 components and approximate W as the sum of the k rank-1 matrices with the k largest singular values since a small number of singular values dominates other singular values in weight matrices in language models [21, 49]. We decompose S, and integrate it with the orthogonal matrices to remove redundant weights. As a result, the number of parameters for representing W is reduced from mn to $(m + n)k$, where $k \ll m, n$.

3 Pruning

In this section, we introduce pruning algorithms, which identify and prune redundant components in PLMs to generate compact and accurate language models. We give an overview of pruning algorithms in Section 3.1, elaborate on pruning granularities in Section 3.2, and explain pruning strategies in Section 3.3. In Section 3.4, we delve into SparseGPT [30] which is the state-of-the-art pruning algorithm for massive-scale LLMs up to 175B.

3.1 Overview

Pruning is a compression algorithm that eliminates the superfluous components in neural networks. Pruning algorithms [30, 68, 96, 105] have demonstrated the ability to compress PLMs to small models whose inference is much faster than PLMs on commodity hardware. For instance, ZipLM [68] accelerates the inference speed of BERT [27] up to 15 times with a small accuracy degradation.

In Figure 4 (a), we illustrate an example of pruning a two-layer neural network, specifically targeting the weights in the second layer. The weight matrix, which is the pruning target, is shown on the right side of the neural network. We first measure the saliency of each weight (arrow) and neuron (circle) to identify salient weights and neurons to be preserved. We color salient components in green and redundant components in red. After measuring the saliency, we have to choose the granularity of the pruning, which is either a weight or a neuron. Figure 4 (b) represents the result of an unstructured pruning algorithm whose pruning granularity is a weight. We get an accurate and compact model that has only salient green-colored weights. However, the pruned weight matrix is sparse and requires sparse matrix representation, such as **Compressed Sparse Row** (**CSR**) or Coordinate (COO) formats, which are inefficient to store and compute. Figure 4 (c) represents the result of a structured pruning algorithm that prunes neurons and generates a dense weight

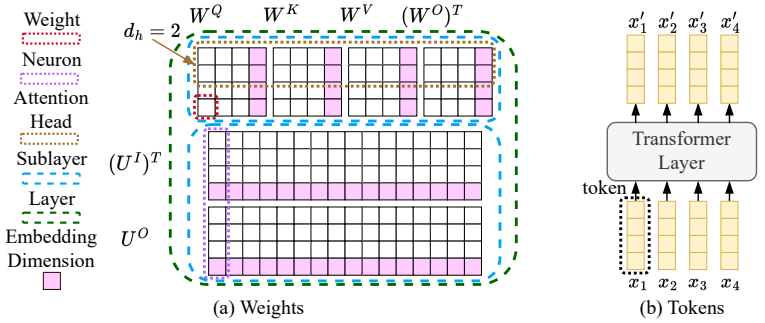


Fig. 5. Illustration of diverse pruning granularities regarding weights (a) and tokens (b). Dotted square boxes indicate each pruning granularity and we color the weights for the granularity of the embedding dimension in pink for simplicity. d_h represents the dimension of token embeddings in attention heads.

matrix which is efficient to store and compute. In general, the compressed models pruned with larger granularities (e.g., neurons) are more efficient than those with smaller granularities (e.g., weights); on the other hand, large granularities lead to inaccuracy. We describe more details of the accuracy-efficiency tradeoff regarding pruning granularity in Section 3.2.

To maximize the accuracy of pruned models, it is essential to carefully design a strategy to minimize pruning errors which refer to the distortion of the model's output induced by pruning components in PLMs. For example, we need to update the remaining weights (solid lines) in Figure 4 to compensate for the pruning errors induced by the pruning of redundant weights (dotted lines). It is crucial for pruning strategies to consider the following two problems to achieve high accuracy: (1) how to choose components to prune and (2) how to adequately compensate for pruning errors. We name the first issue of finding suitable pruning masks as a mask search problem, and the second issue of reducing pruning errors as an error compensation problem. We divide pruning strategies into high-cost and low-cost algorithms based on their computational costs. High-cost pruning algorithms jointly identify and remove unnecessary components by time-consuming retraining processes on a large dataset, compensating for pruning errors by tuning the weights in unpruned components. Although high-cost pruning algorithms generate accurate and compact models, they are too expensive to be used for LLMs which have more than billions of parameters. Low-cost pruning algorithms significantly reduce the computational cost of pruning by dividing pruning processes into efficient mask search and low-cost error compensation on a small calibration dataset. For instance, K-prune [105] prunes BERT within 10 minutes on the MNLI dataset while high-cost pruning algorithms [47] take up to 40 hours on the same dataset. However, low-cost algorithms suffer from severe accuracy degradation in high-compression regimes. We describe more details about pruning strategies and pruning costs in Section 3.3.

We summarize pruning algorithms for encoder-only Transformers in Table 2 and those for decoder-only Transformers in Table 3. We denote the granularity and the cost of each algorithm, then group them accordingly. We report the amount of accuracy loss and speedup after pruning.

3.2 Pruning Granularity: Unstructured vs. Structured

Pruning granularity refers to the size of units that we use for pruning and we provide an illustration of pruning granularities regarding weights and tokens in Figure 5. W^Q, W^K, W^V, and $W^O \in \mathbb{R}^{d \times d}$ are weights in an MHA sublayer. $U^I \in \mathbb{R}^{d_f \times d}$ and $U^O \in \mathbb{R}^{d \times d_f}$ are weights in an FFN sublayer. x_i and x'_i are ith token embeddings before and after transformation through a transformer layer, respectively.

Table 2. Comparison of Pruning Algorithms for Encoder-Only Transformers on MNLI, QQP, and $SQuAD_{1.1}$ (SQD) Benchmarks

Method	PLM♣	Pruning Granularity§	Pruning Cost	Comp. Rate	Accuracy/F1 Diff.✚ (%p) MNLI	QQP	SQD	Avg.	Speedup
MvP [116]	Bb	W	High	90.0%	-3.30	-1.20	-3.20	-2.57	-
LEAP [154]	Bb	W	High	90.0%	-3.30	-0.20	-4.80	-2.77	-
CAP [150]	Bb	W	High	90.0%	-2.50	-0.20	-2.80	-1.83	-
oBERT [67]	Bb	W	High	90.0%	-1.34	-0.17	-0.56	-0.69	-
oBERT [67]	Bb	W	High	97.0%	-3.54	-0.83	-3.89	-2.75	-
DynaBERT [47]	Bb	N, H	High	75.0%♠	-0.90	-0.20	-0.60	-0.57	-
DynaBERT [47]♦	Bb	N, H	High	75.0%♠	-4.30	-	-	-4.30	3.20×
SNIP [87]	Bb	N, H, S	High	75.0%	-6.40	-2.80	-	-4.60	-
BMP[73]	Bb	N, H	High	75.2%	-	-	-0.80	-0.80	2.25×
CoFi [148]	Bb	E, N, H, S	High	75.0%♠	-0.05	-	-0.41	-0.23	3.08×
CoFi [148]	Bb	E, N, H, S	High	90.0%♠	-2.19	-	-4.32	-3.26	6.68×
CAP [150]	Bb	N, H	High	90.0%	-3.50	-0.70	-8.10	-4.10	-
Sajjad et al. [114]	Bb	L	High	50.0%	-2.91	-0.72	-	-1.82	2.00×
ZipLM [68]	Bb	N, H, S	High	-	-1.50	-0.20	-2.50	-1.40	7.00×
ZipLM [68]	Bb	N, H, S	High	-	-3.90	-0.80	-7.20	-3.97	15.00×
PoWER-BERT [37]	Bb	T	High	-	-0.80	-1.00	-	-0.90	3.42×
LTP [64]	Rb	T	High	50.5%♠	-1.00	-0.70	-	-0.85	1.99×
Kwon et al. [72]†	Bb	N, H	Low	75.0%♠	-21.67	-19.42	-48.56	-29.88	1.75×‡†
KCM [102]	Bb	N	Low	20.0%♠	-7.29	-1.85	-7.19	-5.44	1.21×‡†
K-prune [105]	Bb	N, H	Low	75.0%♠	-5.18	-3.71	-9.26	-6.05	2.08×‡†

♣: Bb (Bert-base), Rb (RoBERTa-base), ‡: maximum speed up when accuracy drop < 3%p
§: W (weight), N (neuron), H (attention head), E (embedding dimension), S (sublayer), L (layer), T (token)
✚: (Compressed model's accuracy/F1 score) - (uncompressed model's accuracy/F1 score)
♠: FLOPs-based compression rate, ♦: reported results in CoFi [148], †: reported results in K-prune [105]

Colored dashed lines indicate the regions of weights included in each pruning granularity. Pink-colored weights are related to the same embedding dimension. The smallest granularity for pruning is an individual weight (W in Tables 2 and 3) for unstructured pruning [30, 67, 116, 154]. On a larger scale, we use a token (T), a neuron (N), an attention head (H), an embedding dimension (E), a sublayer (S), and a layer (L) for structured pruning [47, 68, 72, 73, 87, 96, 102, 105, 114, 148]. Larger pruning granularities lead to speedup in inference, but there is an accuracy degradation since it removes important weights along with unimportant weights. The number of weights does not change after pruning tokens and it reduces only the number of FLOPs.

Unstructured pruning algorithms [67, 116, 150, 154] have demonstrated high accuracy in compressing language models achieving up to 97% compression rate with a small accuracy degradation. However, their drawback lies in the acceleration difficulty due to the resulting sparse pruning patterns. Therefore, researchers have focused on extending the findings in unstructured pruning algorithms to structured pruning algorithms to achieve acceleration. For example, BMP [73] extends the findings of MvP [116] considering that the gradient magnitude of an objective function with respect to a weight is more important than the magnitude of the weight. Similarly, ZipLM [68] achieves a good performance on structured pruning of BERT by developing the findings in oBERT [67] that the optimal error compensation strategy of OBS [43] is useful for pruning BERT.

When extending unstructured pruning algorithms to structured pruning algorithms, it is crucial to determine the granularity for pruning, considering the characteristics of the model. Michel et al.

Table 3. Comparison of Pruning Algorithms for Decoder-Only Transformers on WikiText2 Datasets

Method	PLM	Pruning Granularity§	Pruning Cost	Comp. Rate	Perplexity PLM	Perplexity Comp.✠	Speedup
SIMPLE [128]	GPT2-124M*	E, N, H	High	48.5%*	14.49	17.14	-
ZipLM [68]	GPT2-85M	N, H, S	High	44.4%	24.10	30.30	1.50×
Wanda [155]♮	LLaMA-7B	W	Low	70.0%	5.68	85.77	-
SparseGPT [30]♮	LLaMA-7B	W	Low	70.0%	5.68	26.30	-
SparseGPT+OWL [125]♮	LLaMA-7B	W	Low	70.0%	5.68	19.49	-
SparseGPT [30]	OPT-175B	2:4	Low	50.0%	8.35	8.74	1.66×†
LLM-Pruner [96]	LLaMA-7B	N, H	Low	20.0%	12.62	17.37	1.18×
SlimLLM [39]	LLaMA-7B	N, H	Low	20.0%	12.62	15.55	-

§: W (weight), 2:4 (2:4 sparsity), N (neuron), H (attention head), S (sublayer)
♮: reported results in OWL [125], *: includes token embeddings, ✠: a compressed model
†: geometric mean of speedups in individual affine transformations
Lower Perplexities Represent Better Results.

[98] experimentally demonstrates that MHA sublayers are robust to the pruning of attention heads by showing that approximately 40% of attention heads are prunable with negligible accuracy degradation. Similarly, BMP [73] discovers the pattern that parameters in certain attention heads in MHA sublayers are entirely removed when we perform block-wise pruning with 16×16 weight blocks. For FFN sublayers, the weight blocks within specific neurons in intermediate layers are entirely removed, while weight blocks in other neurons are preserved. Based on these findings, subsequent structured pruning algorithms [47, 68, 72, 87, 96, 102, 105, 148, 150] employ attention heads and intermediate neurons as the pruning granularities for MHA and FFN sublayers, respectively. When we prune attention heads and neurons, we reformulate $M(X)$ and $F(X)$ in Equations (1) and (3) into $M(X;\zeta)$ and $F(X;\xi)$ in Equation (10) by imposing mask vectors $\zeta \in \mathbb{R}^H$ and $\xi \in \mathbb{R}^N$ that indicate the pruning status of H attention heads and N neurons, respectively; $\zeta_i = 0$ represents that the ith attention head is pruned. $W_i^O \in \mathbb{R}^{d \times d_h}$ is the subset of W^O corresponding to the ith attention head, i.e. from the $(d_h(i-1)+1)$th to the $(d_h i)$th columns of W^O. $U^I_{i,:} \in \mathbb{R}^{1\times d}$ and $U^O_{:,i} \in \mathbb{R}^{d\times 1}$ are weights corresponding to the ith neuron in FFN sublayers. We horizontally stack biases $b^O \in \mathbb{R}^d$, $c_i^I \in \mathbb{R}$, and $c^O \in \mathbb{R}^d$ as $B^O = [b^O, b^O, ..., b^O] \in \mathbb{R}^{d\times s}$, $C_i^I = [c_i^I, c_i^I, ..., c_i^I] \in \mathbb{R}^{1\times s}$, and $C^O = [c^O, c^O, ..., c^O] \in \mathbb{R}^{d\times s}$ to match the dimension in the equation where s is a sequence length. $c_i^I \in \mathbb{R}$ is a bias corresponding to the ith neuron.

$$M(X;\zeta) = \sum_{i=1}^{H} \left(\zeta_i W_i^O A_i\right) + B^O \text{ and } F(X;\xi) = \sum_{i=1}^{N} \left(\xi_i \left(U^O_{:,i}\sigma(U^I_{i,:}X + C_i^I)\right)\right) + C^O \tag{10}$$

As a result, the outputs of an MHA sublayer and an FFN sublayer are represented as the summation of outputs of independent attention heads and neurons. Therefore, the outputs do not change when we remove attention heads or neurons whose mask variable is zero.

However, we cannot fully exploit GPU parallelism by pruning attention heads and neurons since the number of matrix multiplications between input and weight matrices is rarely reduced; pruning attention heads or neurons reduces the size of weight matrices, but it rarely removes the weight matrices entirely. Therefore, we need to enlarge the pruning granularity to entire sublayers or even layers to maximize the inference speed of the compressed models. Sajjad et al. [114] compares the accuracy of the compressed models with diverse layer pruning patterns, and finds that removing top layers is effective for encoder-only models. CoFi [148] and ZipLM [68], which aim at prioritizing acceleration performance, additionally prune the entire sublayers and achieve a good speedup.

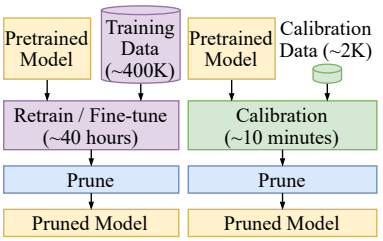
Another approach is token pruning, based on the observation that the cosine similarity between tokens increases significantly as we move toward the top layers of BERT [37]. PoWER-BERT [37] and LTP [64] achieve speedup more than two times with minimal accuracy degradation by using only a subset of tokens from the input sequence; token pruning algorithms do not reduce the number of weights since the number of weights depends on the dimension of token embeddings not on the number of tokens. While token pruning algorithms do not reduce the number of weights, their integration with other structured pruning algorithms employing different granularities such as CoFi [148] or ZipLM [68] holds the potential for achieving remarkable speedup gains.

We use the same pruning granularity for decoder-only models as for encoder-only models, e.g., a weight, an attention head, or a neuron. However, pruning decoder-only models is much more difficult than encoder-only models. For instance, structured pruning algorithms [68, 128] for GPT-2 show performance degradation at a low compression rate smaller than 50% while those [68, 73, 148] for BERT succeed in pruning up to 75% of weights with minimal accuracy degradation. Structured pruning approaches [96] for billion-scale LLMs show harsher accuracy degradation at a low compression rate of 20% since the excessive size of LLMs limits the use of fine-tuning processes. SparseGPT [30] demonstrates relatively better performance using an unstructured pruning pattern and expands its algorithm to cover 2:4 pruning granularity which supports acceleration on customized hardware; 2:4 pruning granularity represents a semi-structured pruning pattern that there are 2 zeros in 4 consecutive elements. Improving the accuracy of structured pruning with large granularities for decoder-only models is a promising future work.

3.3 Pruning Strategies: High-Cost vs. Low-Cost

After selecting the granularity for pruning, we need to determine the strategy for pruning, which includes methodologies for identifying unnecessary components and compensating for pruning errors induced by the removal of the identified components. We categorize pruning algorithms into two groups of high-cost [37, 47, 64, 67, 68, 73, 87, 114, 116, 128, 148, 150, 154] and low-cost [30, 72, 96, 102, 105, 125, 155] algorithms according to the cost of pruning. We give an overview of high-cost and low-cost pruning algorithms in Figure 6 where the size of datasets and the running time are based on the pruning algorithms [47, 105] for BERT on the MNLI dataset. As shown in (a), high-cost pruning algorithms generate an accurate compressed model through a computation-intensive retraining process on a large training dataset. However, high-cost pruning algorithms are intractable, requiring an excessive cost for pruning massive-scale LLMs, and low-cost algorithms in (b) address the issue by substituting an expensive retraining process with a lightweight calibration process on a small calibration dataset.

We categorize high-cost pruning algorithms into three groups: regularized training-based algorithms [73, 116, 139, 148, 154], OBS [43]-based algorithms [67, 68], and others [47, 87, 150]. Regularized training-based algorithms introduce sparsity regularizations which regularize the magnitude of weights and gradually remove unnecessary components during a retraining process. MvP [116] introduces a scoring matrix S for each weight matrix W and substitutes the weight matrix with $M \odot W$ where $M = (S > \tau)$ is a matrix of pruning masks that indicates whether the score of each weight exceeds the threshold τ or not. $\odot$ is a Hadamard product. MvP introduces a regularization term $R(S) = \lambda \sum_{i,j} \sigma_s(S_{i,j})$ that penalizes the scores of weights to promote sparsification where $\sigma_s(\cdot)$ is a sigmoid function. MvP manually controls the



(a) High-cost pruning (b) Low-cost pruning

Fig. 6. Comparison of high-cost and low-cost pruning algorithms. The size of datasets and the running times are based on the pruning algorithms [47, 105] for BERT on the MNLI dataset.

sparsity of the pruned model via tuning the coefficient λ. LEAP [154] addresses the inefficiency of manual selection of λ by proposing an algorithm that automatically controls the sparsity level to match the preset sparsity. BMP [73] extends MvP into structured pruning to accelerate inference by sharing scores of the weights in the same group defined by the structural pruning granularity.

Another approach for regularized training is to impose l_0 regularization. Let w be a vector containing all of the weights in a PLM, then its l_0 norm $||w||_0$ is the number of non-zero weights, which is the same as the size of the model. Reducing the l_0 norm is equal to the model sparsification, but we cannot directly impose l_0 regularization for training since it is not differentiable. Louizos et al. [95] addresses the problem by regularizing the expectation of the l_0 norm of weights multiplied by stochastic mask variables z, i.e., regularize $\mathbb{E}_z\left[||w \odot z||_0\right]$ rather than $||w||_0$. The modified regularization term is differentiable since the expectation of l_0 norm is computed as a closed form which is differentiable (see Louizos et al. [95] for details). FLOP [139] utilizes the stochastic mask variables for structured pruning of language models and substitutes the l_0 regularization with an augmented Lagrangian to ensure the compressed model has a preset sparsity. CoFi [148] achieves both high accuracy and fast inference speed via jointly pruning coarse and fine-grained granularities, from a neuron to a layer, using an augmented Lagrangian in FLOP [139].

Given a weight vector w and an input vector x, OBS [43] algorithm finds an optimal weight-update $\delta w_{(q)}$ after pruning qth weight w_q that minimizes the amount $||wx - \hat{w}x||_2^2$ of a pruning error where $\hat{w} = w + \delta w_{(q)}$ is a vector of weights after pruning. OBS is applied to language model compression by formulating the compression problem as a sublayer-wise output reconstruction problem after pruning [30, 68]. OBS is crucial for pruning PLMs since OBS-based algorithms demonstrate the state-of-the-art performance in both settings of high-cost [67, 68] and low-cost pruning algorithms [30]. oBERT [67] modifies the objective function of OBS into the cross-entropy and proposes an efficient algorithm to approximate the inverse of Hessian which is the computational bottleneck of OBS. ZipLM [68] extends the OBS algorithm to structured pruning by increasing the pruning granularity of OBS from a single weight to consecutive columns, and achieves remarkable acceleration up to 15× by finding runtime-aware configuration on the target hardware.

Besides regularized training-based and OBS-based pruning algorithms, there are other high-cost pruning algorithms that do not fall into these categories. DynaBERT [47] presumes a PLM as an overlap of inherent models with various configurations of attention heads, neurons, and sublayers. DynaBERT succeeds in training all of the inherent models in a single training process and obtains pruned models by sampling the inherent models that satisfy the desired compression rate. CAP [150] introduces a contrastive learning framework [16] to pruning, which aims to make the intermediate features of pruned models mimic both pretrained and fine-tuned model's intermediate features. SNIP [87] prunes attention heads and FFN sublayers whose maximum output is small, and proposes a spectral normalization that boosts the performance of pruned models. The approaches in Refs. [47, 87, 150] provide unique insights for a language model pruning problem, but they exhibit lower performance compared with regularized training-based [148] or OBS-based [68] pruning algorithms.

Low-cost pruning algorithms [72, 102, 105] significantly reduce the pruning cost by (1) separating mask search from error compensation, and (2) replacing costly retraining with an efficient weight update on a small calibration dataset. Kwon et al. [72] is the first work that proposes a low-cost pruning framework for language models. Kwon et al. efficiently estimates the saliency of attention heads and neurons via the empirical **Fisher information matrix** (**FIM**) that reflects the amount of influence on the objective function (cross-entropy) following the derivations in OBD [75]. Kwon et al. formulates an efficient sublayer-wise weight-tuning process that takes less than a minute using linear solvers. KCM [102] avoids the expensive FIM gradient computation of Kwon et al. by evaluating neuron importance via representative ranking, without computing gradients. However,

Kwon et al. and KCM show significant accuracy degradations at high compression rates due to their one-shot pruning process without iteration. K-prune [105] demonstrates the importance of an efficient iterative pruning process by achieving remarkable improvement up to 58% higher accuracy than previous works without losing the efficiency of low-cost pruning processes.

Decoder-only models have not been extensively researched regarding low-cost pruning algorithms, especially for LLMs [130, 131, 165]. SparseGPT [30] prunes LLMs up to 175B in three hours on a single A100 GPU by improving the efficiency of OBS [43]. We select SparseGPT as the representative pruning algorithm and detail it in Section 3.4. Wanda [155] shows comparable results with SparseGPT in a low compression rate of 50% using efficient unstructured pruning without weight update, but severely degrades in high-sparsity regimes [125]. OWL [125] improves the accuracy of both Wanda and SparseGPT by replacing their sublayer-wise uniform pruning rates with the non-uniform pruning rates reflecting the outlier ratio in each sublayer. For structured pruning of LLMs, LLM-pruner [96] prunes attention heads and neurons using a FIM, and uses LoRA [50] for error compensation. SlimLLM [39] improves the accuracy of pruned models by removing redundant attention heads whose Pearson correlation between the outputs before and after pruning is high. However, it still suffers from significant accuracy degradation even at a low compression rate of 20% on LLaMA-7B. In response, SPRINT [106] improves the accuracy–speedup tradeoff by pruning redundant MHA and MLP sublayers based on latency reduction and post-tuning sensitivity. Nevertheless, structured pruning algorithms for LLMs have much room for improvement.

3.4 Representative Algorithm: SparseGPT

In this section, we elaborate on the details of SparseGPT [30], which is the first algorithm that successfully prunes gigantic LLMs with more than 100B parameters. SparseGPT extends its unstructured pruning methodology to semi-structured pruning granularity, such as 2:4 sparsity, whose practical inference speedup is supported on A100 GPUs. SparseGPT focuses on exploiting the optimal weight update in OBS [43] to increase the accuracy of the pruned model, and reduce the number of computations of the Hessian inverse which is a computational bottleneck of the solution of OBS to maximize the efficiency.

Consider a binary pruning mask matrix $M \in \mathbb{R}^{d' \times d}$ for a weight matrix $W \in \mathbb{R}^{d' \times d}$ that projects d dimensional input token embeddings $X \in \mathbb{R}^{d \times s}$ to the dimension of d'. The authors formulate the error compensation problem for the weight W as a least-square problem to find an optimal weight $\widehat{W}$ that minimizes the change in outputs as in Equation (11). The authors divide the problem into d' independent row-wise problems on the right-hand side.

$$\arg\min_{\widehat{W}} \left\| WX - \left((M \odot \widehat{W}) X \right) \right\|_F^2 = \arg\min_{\widehat{W}} \sum_{r=1}^{d'} \left\| W_{r,:} X - \left((M_{r,:} \odot \widehat{W}_{r,:}) X \right) \right\|_2^2 \tag{11}$$

SparseGPT finds the optimal weight update $\delta w_{(r,m)} \in \mathbb{R}^d$ to compensate for the error induced by the pruning of the mth weight $W_{r,m}$ in the rth row, and the corresponding pruning error $\epsilon_{(r,m)} \in \mathbb{R}$ following OBS as in Equation (12). In other words, $\widehat{W}^*_{r,:} = W_{r,:} + \delta w^T_{(r,m)}$ is the optimal solution of the subproblem regarding the rth row when the mth element is pruned, and $\epsilon_{(r,m)} = \left\| W_{r,:} X - \widehat{W}^*_{r,:} X \right\|_2^2$.

$$\delta w_{(r,m)} = -\frac{W_{r,m}}{\left[H^{-1}_{(r)}\right]_{mm}} \left[H^{-1}_{(r)}\right]_{:,m} \text{ and } \epsilon_{(r,m)} = \frac{1}{2} \frac{W^2_{r,m}}{\left[H^{-1}_{(r)}\right]_{mm}} \text{ where } H_{(r)} = 2X_{(r)} X^T_{(r)} \tag{12}$$

$H_{(r)}$ is the Hessian of the objective function regarding the rth row, and $X_{(r)}$ is the subset of input features whose corresponding weights have not been pruned in the rth row. Computing $H^{-1}_{(r)}$ is

the major bottleneck in Equation (12); SparseGPT reduces the computational cost by adopting a column-wise pruning methodology. SparseGPT iteratively performs column-wise pruning; they identify unnecessary weights, and sequentially perform an optimal weight update process to prune the identified weights in each column. The column-wise pruning is efficient based on the following reasons. First, since the Hessian depends on a pruning mask, not weights, different rows have the same Hessian inverse. Therefore, they need to compute the Hessian inverse only once for each column when they prune in a column-wise manner. Second, there is an efficient technique that computes the Hessian inverse for the subsequent column using the Hessian inverse of the previous column [32]. As a result, SparseGPT achieves remarkable efficiency that prunes 175B LLMs in 3 hours on a single A100 GPU.

The authors demonstrate the compatibility of SparseGPT with the state-of-the-art quantization algorithm [32] and show that the combined algorithm shows better results than using only the quantization algorithm. However, there are drawbacks of SparseGPT that come from sublayer-wise pruning. The pruning mask found by SparseGPT has uniform pruning rates for all sublayers missing sublayer-wise sensitivity. OWL [125] addresses the drawback by adjusting the pruning rates of sublayers based on their outlier ratios and achieves notable accuracy improvements. The objective function of SparseGPT is a least-square in sublayer-wise outputs which is an indirect measure for the amount of errors in cross-entropy. Therefore, the optimal solution in Equation (12) might not be optimal for the direct objective function, and it would be improved when we directly reflect cross-entropy in the loss function.

4 Quantization

In this section, we introduce quantization algorithms that reduce the bit-width of parameters to compress large-scale models while retaining the accuracy. We first provide an overview in Section 4.1, and describe various quantization schemes featured by quantization steps and levels in Section 4.2, followed by quantization strategies in Section 4.3. We single out OPTQ [32] as the representative quantization algorithm for LLM compression, and provide detailed explanation of OPTQ and its improvement in Section 4.4.

4.1 Overview

Quantization is a core model compression technique that assigns continuous values distributed broadly to a small and discrete set. Since Bhandare et al. [8] first made an attempt to quantize Transformer-based models, various methodologies [32, 65, 153, 156] have been proposed to quantize LLMs consisting of over a hundred billions of parameters. Quantization minimizes memory footprint and inference latency of large-scale models by reducing the number of bits required to represent the parameters. Mapping continuous real values to their nearest integers is one of the most common approaches in neural network quantization. This approach called **round-to-nearest** (**RTN**) takes less effort for assigning values to discrete space in virtue of its simplicity; however, naive RTN often suffers from severe performance degradation due to its limited representation capacity.1 Other methodologies utilize non-uniform step sizes to quantize parameters for better representing the original ones. Outputs of a quantizer, also referred to as *quantization levels*, are also non-uniformly spaced under those methodologies in general. Precision and data type of quantization levels determine the compression rate and computational cost of quantized model, respectively. It is straightforward that a lower bit-precision representation has a higher compression rate, and integer values benefit from accelerated operations. We provide detailed explanation with regard to quantization step sizes and levels in Section 4.2.

Meanwhile, cost of a quantization algorithm depends on quantization strategy to minimize accuracy degradation. Quantization algorithms are computationally expensive when they entail the

Table 4. Comparison of Quantization Algorithms for Encoder-Only Transformers on MNLI, QQP, and $SQuAD_{1.1}$ (SQD) Benchmarks

Method	PLM♣	Uniform§	Cost	Comp. Rate	Accuracy/F1 Diff.✚ (%p) MNLI	QQP	SQD	Avg.	Speedup
Q8BERT [161]	Bb	UQ	High	75.0%	-	-	-0.72	-0.72	-
TernaryBERT [166]	Bb	UQ	High	93.8%	-1.20	-0.80	-1.30	-1.10	-
QuantNoise [124]	Rb	UQ	High	75.0%	-1.20	-	-	-1.20	-
BinaryBERT [5]	DyB	UQ	High	96.9%	-1.00	-0.20	-2.50	-1.23	-
I-BERT [61]	Rl	UQ	High	75.0%	-0.30	-0.20	-	-0.25	4.00×
SensiMix [108]	Bb	UQ	High	87.5%	-	-1.20	-	-1.20	4.82×
BiT [92]	Bb	UQ	High	96.9%	-5.40	-6.00	-14.80	-8.73	-
PreQuant [36]	Rl	UQ	High	87.5%	-0.80	-1.10	-	-0.95	-
K-means Quant. [168]	Bb	NUQ	High	90.6%	-0.60	-0.40	-	-0.50	-
MREM [4]	Bl	UQ	Low	93.8%	-3.10	-	-3.70	-3.40	-
Outlier Suppression [140]	Bb	UQ	Low	81.3%	-6.40	-6.25	-3.80	-5.48	-
GOBO [160]	Bl	NUQ	Low	89.8%	-0.69	-	-0.91	-0.80	-
Mr.BiQ [52]	Bb	NUQ	Low	93.6%	-0.86	-	-1.53	-1.20	-
PowerQuant [159]	Bb	NUQ	Low	87.5%	-1.52	-0.16	-	-0.84	-

♣: Bb/l (BERT-base/large), Rb/l (RoBERTa-base/large), DyB (DynaBERT)
§: UQ (Uniform quantization), NUQ (Non-uniform quantization)
✚: (Compressed model's accuracy/F1 score) - (uncompressed model's accuracy/F1 score)

retraining process of an entire model. Those high-cost algorithms, however, enable the resulting model to recover from the accuracy degradation caused by the low-precision representation. *Quantization-aware training (QAT)* is one of the representative quantization schemes that involve high-cost computation overhead to quantize models. The prior quantization schemes for encoder-only Transformers including BERT [27] are mostly based on QAT since the models have a relatively few parameters. BERT quantization algorithms have focused on extreme quantization (under 4-bit), assisted by full fine-tuning of the entire model on training data [5, 36, 56, 92, 108, 109, 121, 127, 166]. However, it is challenging to fully retrain the parameters of decoder-only Transformer, a mainstream architecture of LLMs. Note that recently proposed LLMs [9, 118, 165] consist of hundreds of billions of parameters, as explained in Section 2. This is why the low-cost quantization scheme including *post-training quantization (PTQ)* is the most frequently utilized for LLM compression. PTQ has drawbacks of severe accuracy degradation, but algorithms including [24, 32, 62, 149] succeed in preserving the performance under low-cost quantization scheme. We explain more details of quantization strategies related to their cost in Section 4.3.

Tables 4 and 5 summarize quantization methodologies for encoder-only and decoder-only Transformers, respectively. We categorize the algorithms based on the uniformity of quantization step size and the training cost, which we will describe in the following sections.

4.2 Quantization Steps and Levels

4.2.1 Uniformity. Quantization techniques are categorized into *uniform quantization* and *non-uniform quantization* according to the uniformity of quantization step sizes. Uniform quantization generally maps a real-valued input to the nearest integer, which results in the output values to be uniformly spaced. Non-uniform quantization, in contrast, leverages mapping functions that map input values to non-uniform space. While uniform quantization is easy to be implemented, the reconstruction error of quantized model is minimized under non-uniform quantization scheme. Specifically, UniQuan_F [104] takes advantage of both schemes by combining the optimizability

Table 5. Comparison of Quantization Algorithms for Decoder-Only Transformers on WikiText2 Dataset

Method	PLM	Uniform	Cost	Comp. Rate	Perplexity PLM	Perplexity Comp.✠	Speedup
ZeroQuant [152]♦	BLOOM-176B	UQ	Low	81.3%	8.11	8.35	-
FineQuant [65]	OPT-175B	UQ	Low	87.5%	9.08	9.84	0.84×
SmoothQuant [149]	LLaMA-65B	UQ	Low	75.0%	6.17	6.20	1.36×
OPTQ [32]♮	OPT-175B	UQ	Low	81.3%	8.34	8.68	3.20×
RPTQ [156]	OPT-175B	UQ	Low	81.3%	8.34	10.03	-
PEQA [58]	LLaMA-65B	UQ	Low	81.3%	3.53	4.27	-
LLM-QAT [93]	LLaMA-30B	UQ	Low	87.5%	7.00	7.70	-
INT2.1 [11]	LLaMA-7B	UQ	Low	93.8%	5.08	8.74	-
PTQ1.61 [167]	LLaMA-13B	UQ	Low	95.0%	5.09	9.67	-
AWQ [86]	LLaMA2-70B	UQ	Low	81.3%	3.32	3.74	3.90×
OWQ [77]	OPT-66B	UQ	Low	80.6%	9.34	9.31	-
SpQR [26]	LLaMA-65B	UQ	Low	75.6%	3.53	3.68	1.29×
SqueezeLLM [62]	LLaMA-13B	NUQ	Low	78.0%	5.09	5.60	2.40×
QLoRA [25]	LLaMA2-13B	NUQ	Low	87.5%	4.88	5.22†	-
UniQuan$_F$ [104]	LLaMA3-70B	NUQ	Low	90.6%	3.19	2.86	-

♦: reported results in ZeroQuant-V2 [153], †: reported results in LoftQ [82]
♮: identical to GPTQ [31], ✠: a compressed model
Lower Perplexities Represent Better Results.

of uniform quantization with the expressiveness of non-uniform quantization. We provide more detailed explanation about uniform and non-uniform quantization in the following.

Uniform quantization. The uniform quantizer $Q_U(\cdot)$ is defined as Equation (13), where $\lfloor \cdot \rfloor$ is a floor function, s is a scaling factor, and z is an integer offset. Given an input matrix W, we suppose b-bit quantization using the range of clipped inputs as $[\alpha, \beta]$ to calculate s and z.

$$Q_U(W) = \left\lfloor \frac{W}{s} + \frac{1}{2} \right\rfloor + z, \quad \text{where } s = \frac{\beta - \alpha}{2^b - 1} \text{ and } z = \left\lfloor -\frac{\beta \cdot 2^{b-1} + \alpha\left(2^{b-1} - 1\right)}{\beta - \alpha} + \frac{1}{2} \right\rfloor. \tag{13}$$

The clipping range $[\alpha, \beta]$ determines the scaling factor and the offset. Choosing α and β, also referred to as *calibration*, needs to be done adequately for decent representation of original real-valued inputs. The straightforward calibration uses an input matrix's minimum and maximum values. However, this approach is vulnerable to the input data with outliers resulting in a poor representation of the original inputs. One method to tackle the problem is using percentiles of inputs or finding the range that minimizes KL divergence between real-valued inputs and quantized outputs [143].

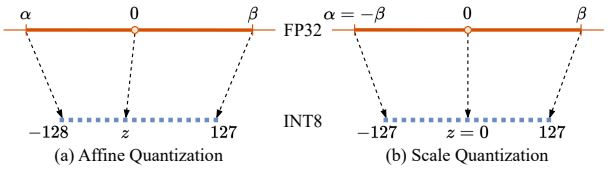


Fig. 7. Comparison of 8-bit affine and scale quantization.

Uniform quantization methods are sub-categorized into affine quantization and scale quantization based on the symmetry of clipping range. Figure 7 shows a brief comparison of those two different quantization schemes based on the calibration symmetry. The absolute values of α and β are different from each other in affine quantization; the integer offset z is consequently non-zero, as illustrated in Figure 7 (a). While affine quantization benefits from precise representation of original values,

non-zero z results in the computational overhead. Scale quantization bypasses this problem by using symmetric clipping range. Note that the range of quantized values accordingly needs to be symmetric (i.e., $[-2^{b-1}+1, 2^{b-1}-1]$). We redefine the quantizer for scale quantization as follows:

$$Q_U(W) = \left\lfloor \frac{W}{s} + \frac{1}{2} \right\rfloor, \quad \text{where } s = \frac{\beta}{2^{b-1}}.$$

Affine quantization is recommended when the target real-valued inputs have a skewed distribution, while scale quantization is preferred if reducing the computational cost is the most important factor.

Non-uniform quantization. While most PLM compression schemes rely on uniform quantization, non-uniform quantization [52, 62, 71, 158–160, 168] also has been spotlighted due to its improved representation of the original parameters over uniform quantization [91]. Uniformly quantizing language models leads to significant accuracy degradation since most of the parameters do not follow uniform distribution. Assigning different sizes of quantization steps to each interval enables the quantized model to adequately represent the original parameters. We categorize non-uniform quantization schemes into low-bit floating point, log-scaled, binary-coded, and codebook-based ones. Methods including [69, 99, 145] utilize low-bit floating points to represent outputs, as illustrated in Figure 8 (b). They leverage the quantizer $Q_F(\cdot)$ that maps the original value to its nearest low-precision floating point value. Log-scaled non-uniform quantization algorithms map input values to logarithmic-distributed space [54, 158, 159]. The quantizer $Q_L(\cdot)$ of log-scaled quantization algorithms is defined as Equation (14), where $Q_U(\cdot)$ is the uniform quantizer, $|W|$ denotes the matrix consisting of the absolute values of W, and the logarithmic base is 2.

$$Q_L(W) = Q_U(\log_2 |W|). \tag{14}$$

Note that $W \approx \text{Sign}(W) 2^{s \cdot Q_L(W)}$ where s is a scaling factor. Following the process described above, the original weight matrix in the leftmost part of Figure 8 is quantized as Figure 8 (c). Approaches leveraging binary-coding [52, 71] decompose the original parameters into full-precision scaling factors and binary matrices. As illustrated in Figure 8 (d), all elements of a single matrix share b scaling factors under b-bit binary-coded quantization, where b is 2 in the example. Binary-coded quantization algorithms focus on finding the appropriate scaling factor s_i and its corresponding binary matrix $Q_B(W)_i$ such that $W \approx \sum_{i=1}^{b} s_i \cdot Q_B(W)_i$. For example, the element at the first row and column of the quantized matrix in Figure 8 (d) is calculated as follows:

$$[Q_B(W)_1]_{11} \cdot s_1 + [Q_B(W)_2]_{11} \cdot s_2 = 6.375 + 6.325 = 12.7 = W_{11}.$$

The other technique, named codebook-based quantization [62, 124, 160, 168], utilizes look-up tables and indices. These methodologies are based on k-means clustering; the key point is to find 2^b centroids for b-bit quantization. The quantizer $Q_C(\cdot)$ of codebook-based quantization maps the original inputs to indices pointing to their corresponding centroids. Therefore, the original matrix is decomposed into the codebook consisting of full-precision centroids and the matrix with low-bit indices, as in Figure 8 (e).

4.2.2 Precision and Type. Precision of the resulting outputs is one of the most important features of quantizer, as well as datatype of outputs. Output precision determines the compression rate; for example, the compressed model is 4× smaller than the original one if the model with 32-bit floating points is quantized using an 8-bit quantization algorithm. It is common to quantize FP32 models with INT8 precision, while recently proposed algorithms focus more on *extreme quantization*. Extreme quantization refers to techniques that reduce the bit precision under 4-bit [4, 11, 12, 36, 86, 93, 129, 140, 144, 156, 166]. However, quantizing the model with extremely low-bit precision representation is vulnerable to accuracy degradation. One way to tackle the problem is

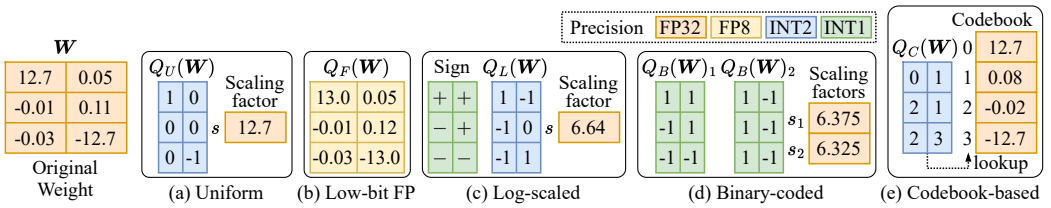


Fig. 8. An example of different quantization schemes under (a) naive round-to-nearest (RTN) uniform quantization, (b) non-uniform quantization using low-bit floating point, (c) non-uniform quantization using log-scaled values, (d) binary-coded non-uniform quantization, and (e) codebook-based non-uniform quantization.

using *mixed-precision quantization* algorithms. In Transformer-encoder model quantization, [108] leverages different bit-precision based on the sensitivity analysis. Similar approaches [26, 77, 167] are applied to Transformer-decoder models. In particular, PTQ1.61 [167] achieves an effective 1.61 bits per weight by combining 1-bit and 4-bit quantization.

Meanwhile, output datatype is related to the computation latency of the resulting model. Computing hardwares, such as CPUs or GPUs, generally support acceleration of integer arithmetic. *Dequantization* is the main reason why the quantized model is unable to be supported by integer arithmetic acceleration. Dequantizing the discrete values is the exact opposite of quantization; that is, dequantization projects quantized outputs back to continuous real values.

Approaches including [61, 161] devise novel LM quantization schemes that do not entail a dequantization process, so that the resulting model can perform faster inference. Some algorithms utilize different types of outputs; [58] adapts BFloat16 to store embedding and activation map, and [25] proposes a novel data type NormalFloat4 to store weights that have a normal distribution.

4.3 Quantization Strategies: High-Cost vs. Low-Cost

Quantization cost is decided based on whether the algorithms entail full retraining or fine-tuning of the entire model after quantization. High-cost methods require full training of quantized models to recover from the accuracy degradation. In contrast, low-cost methods require less or no training; thus, it is cheaper and faster to utilize low-cost algorithms than high-cost ones. However, the severe accuracy degradation is the most critical drawback of low-cost algorithms. We provide more detailed explanation of two main quantization strategies based on the training cost in the following.

4.3.1 High-Cost. Original full precision models pass through the entire retraining process under high-cost quantization schemes. A general approach of language model compression, called *quantization-aware training (QAT)*, (1) applies quantization to pretrained Transformer-based models, and (2) retrain or fine-tune the model to recover from the performance deterioration after quantization. However, it is impossible to compute the gradients during back propagation due to the existence of non-differentiable step functions. One of the most common methods to tackle this problem is utilizing **straight-through estimator** (**STE**), as described in Section 2. Although the functions for forward and backward pass are different from each other under STE, the algorithms still work well in practice. Since high-cost algorithms require the training or fine-tuning dataset as illustrated in Figure 9 (a), it is difficult to be applied when those data are inaccessible.

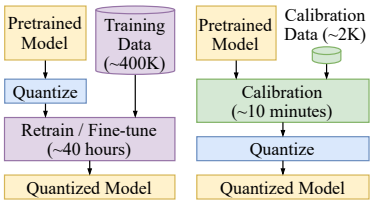


Fig. 9. Comparison of high-cost and low-cost quantization.

High-cost quantization algorithms for BERT compression [61, 108, 124, 161] retrain the quantized models utilizing task-specific fine-tuning data. Since BERT has plenty of redundancies [22] and a relatively small number of parameters, most of BERT quantization algorithms reduce the bit-length under 2-bit [5, 92, 121, 166], and retrain the quantized models to compensate for the quantization errors. However, retraining LLMs is highly time- and memory-consuming due to their prohibitive number of parameters. Unlike high-cost quantization algorithms, low-cost quantization algorithms reduce quantization cost by quantizing parameters without running full retraining processes. We explain the details of those low-cost quantization algorithms in the following section.

4.3.2 Low-Cost. Low-cost quantization algorithms do not require time- and memory-intensive model retraining or fine-tuning. Low-cost quantization is adequate to quantize models with a gigantic number of parameters; it consequently has become the mainstream approach to quantize LLMs, as shown in Table 5. Low-cost quantization algorithms mostly follow *post-training quantization (PTQ)*, a training-free quantization scheme. PTQ quantizes pretrained Transformer-based models without retraining or fine-tuning the model after quantization. As illustrated in Figure 9 (b), low-cost algorithms directly quantize the pretrained model; a tiny subset of the training data is needed only for calibration. Low-cost quantization algorithms are sub-categorized into *dynamic quantization* and *static quantization*.

Dynamic quantization [24, 152, 159] determines the clipping range of activations to calculate scaling factors and integer offsets during inference. Dynamic quantization algorithms generally show better performance than static algorithms since they utilize actual input data to compute scaling factors and integer offsets. However, dynamic quantization algorithms result in slower inference speed of the resulting model, due to the extra computational overhead during inference. Static quantization [149, 156], on the other hand, precomputes scaling factors and integer offsets of activations before inference. Static quantization algorithms require a small subset of training data; those partial data called *calibration data* decide the clipping range of activation. Although static methods for activation quantization are lightweight, they are not applied for most of LLM compression due to severe performance degradation. The degradation in performance results from various range of actual input data, caused by diverse lengths of sequence and embedding values.

Activation quantization aims to minimize the computational overhead of model inference; however, memory overhead dominates the entire inference time of LLMs, rather than the computational overhead [62]. In consequence, recent LLM quantization algorithms mainly focus on minimizing memory overhead of LLMs. Weight-only quantization, which is regarded as a static algorithm, is one representative approach to effectively reduce the memory usage of LLMs. Weight-only quantization algorithms [12, 26, 32, 62, 77, 86, 160] quantize only the weights of given models while keeping activations as full-precision values. It is obvious that weight-only quantization algorithms involve dequantization processes during the inference; the quantized models cannot be supported by integer arithmetic acceleration. Nevertheless, weight-only quantization optimizes inference speed by minimizing the latency associated with loading models into memory. Weight-only quantization algorithms normally do not require the calibration data; however, some of them [32, 86] need a small number of calibration data to quantize weights.

Meanwhile, recently proposed lightweight QAT algorithms minimize the cost of training, based on the fact that it is sufficient to fine-tune only the fraction of parameters for preserving the accuracy [11]. Lightweight QAT updates only some portion of parameters while keeping the others fixed. Scaling factors [58], parameters with high sensitivity [93], low-rank weight matrices [11, 25], and outliers [36] are the targets of lightweight QAT.

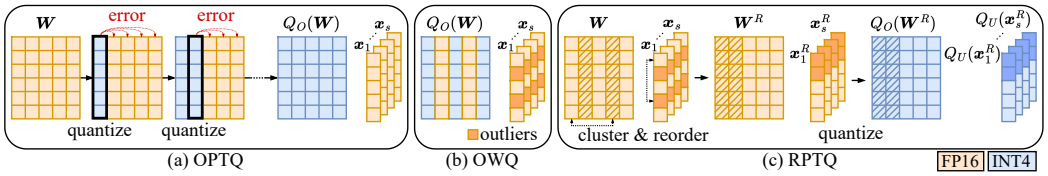


Fig. 10. Illustrations of quantizing FP16 weights W and token embeddings $\{x_i\}_{i=1}^{s}$ into INT4 values with OPTQ [32] and its variants [77, 156], where s is the sequence length. OWQ and RPTQ preserve the accuracy by carefully managing outliers represented as the dashed elements in $\{x_i\}_{i=1}^{s}$; dashed elements in W represent outlier-related weights. The superscript R denotes that a matrix or vector is reordered. $Q_O(\cdot)$ and $Q_U(\cdot)$ are the quantizers of OPTQ and round-to-nearest (RTN), respectively.

4.4 Representative Algorithm: OPTQ

OPTQ [32] resolves the accuracy degradation caused by naive RTN, and quantizes large-scale LLMs [118, 165] to under 3 bits with negligible performance degradation. We select OPTQ as the representative quantization algorithm, as it is the first one to successfully quantize LLMs to below 4 bits without significant performance degradation. Since then, subsequent works [6, 12, 77, 80, 156] have built upon OPTQ to further improve performance. As the basic mechanism of OPTQ is identical to that of SparseGPT [30] explained in Section 3.4, we focus more on how OPTQ is improved over time.

The major difference between SparseGPT and OPTQ lies in the resulting values; a target weight $W_{r,m}$ becomes the corresponding quantized value in OPTQ. The authors find the optimal weight update $\delta w_{(r,m)}$ to compensate for the error $\epsilon_{(r,m)}$ induced by quantizing the mth weight $W_{r,m}$ in the rth row as in Equation (15); the updated rth row $\widehat{W}_{r,:}$ is equal to $W_{r,:} + \delta w_{(r,m)}^T$.

$$\delta w_{(r,m)} = -\frac{W_{r,m} - Q_U(W_{r,m})}{\left[H_{(r)}^{-1}\right]_{mm}} \left[H_{(r)}^{-1}\right]_{:,m} \quad \textit{and}$$
$$\epsilon_{(r,m)} = \frac{1}{2}\frac{\left(Q_U(W_{r,m}) - W_{r,m}\right)^2}{\left[H_{(r)}^{-1}\right]_{mm}} \quad \textit{where } H_{(r)} = 2X_{(r)}X_{(r)}^T. \tag{15}$$

Similar to SparseGPT, OPTQ performs column-wise quantization to minimize the computations of $H_{(r)}^{-1}$. Since quantization does not require mask searching, OPTQ directly quantizes the columns followed by error compensation in a column-wise manner. We briefly depict the quantization process of OPTQ in Figure 10 (a). Note that all the algorithms improving OPTQ quantize weight matrices following this process.

Among various improvements over OPTQ, we introduce two algorithms: OWQ [77] and RPTQ [156]. Recent studies for LLM quantization [24, 26, 86] have shown that preserving activation outliers, which are significantly larger activation values than others, enhances the performance of the quantized models; quantizing activation outliers is the main cause of the accuracy degradation in the quantized LLMs. Based on OPTQ algorithm, both OWQ [77] and RPTQ [156] take the outliers into account, effectively reducing the accuracy degradation. OWQ is a weight-only quantization algorithm that enhances the performance by keeping the weights associated with the activation outliers in full precision, as depicted in Figure 10 (b). OWQ defines the sensitivity of each weight based on the activations; after finding the sensitive weights, OWQ quantizes the remaining insensitive weights with OPTQ algorithm. Unlike OPTQ which focuses on quantizing only the weights of a given model, RPTQ applies activation quantization on top of OPTQ, as illustrated in

Figure 10 (c). Note that RPTQ quantizes weight matrices with OPTQ algorithm, while utilizing RTN for quantizing activations. RPTQ leverages multiple scaling factors and integer offsets in order to represent both outliers and non-outliers more accurately. RPTQ groups the rows of activation matrix $X = [x_1; x_2; \cdots; x_s] \in \mathbb{R}^{d \times s}$ with similar value ranges, and individually quantizes each group with its optimal scaling factor and integer offset. To leverage hardware-efficient block-wise quantization, RPTQ reorders the activation matrices so that the consecutive rows within a block share the same scaling factor and integer offset; the weight matrices are reorganized accordingly.

The critical limitation of OPTQ and its variants is that they do not reflect the dependency between sublayers when they update weights. Furthermore, RPTQ suffers from an accuracy degradation problem when they quantize activation below 8 bits and OWQ suffers from slow inference speed because of the mixed precision weights. We need to address those issues to expand the practical usage of quantized language models.

5 Other Algorithms

In this section, we cover other compression methods for LLMs, including KD, LRA, and dynamic inference. We introduce a representative algorithm, LoRA [50], which fine-tunes a model by updating additional parameters in low-rank adaptors, rather than updating the original parameters. A brief description of each technique is as follows:

- **Knowledge distillation** is a technique that transfers useful knowledge from a pretrained teacher model to a small student model, making the student model mimic the teacher model's behavior.
- **Low-rank approximation** is a technique that reduces the size of a model by decomposing a high-dimensional weight matrix or tensor into lower-dimensional ones.
- **Dynamic Inference** is a technique that enhances the inference efficiency of a language model by dynamically computing only a subset of LLMs, depending on the input.

The rest of the section is organized as follows. We describe KD, LRA, and dynamic inference in Sections 5.1, 5.2, and 5.3, respectively. Finally, we provide detailed explanations of LoRA as a representative algorithm in Section 5.4.

5.1 Knowledge Distillation

KD is a technique that transfers useful knowledge from a large teacher model $\mathcal{T}$ to a small student model $\mathcal{S}$ to improve the accuracy of $\mathcal{S}$. This process encourages the student model to generate outputs that are similar to those of the teacher model to obtain the teacher model's generalized knowledge. KD matches the outputs of classifiers, embedding layers, and sublayers in $\mathcal{T}$ and $\mathcal{S}$ to distill diverse types of knowledge within different distillation sources. We explain diverse types of distillation sources used for KD in Section 5.1.1.

When we apply KD for language models, we need to consider how to match sublayers in $\mathcal{T}$ and $\mathcal{S}$ since they have different numbers of sublayers. We categorize the sublayer matching strategies into three groups: 1:1, many:1, and many:many strategies. We explain more details about each of the sublayer matching strategies in Section 5.1.2.

We summarize the performance of diverse KD algorithms for encoder-only Transformers in Table 6. We denote the distillation sources and the sublayer-matching strategies in columns 2 and 3, respectively. We report the amounts of accuracy/F1 score degradation and acceleration after compression on MNLI, QQP, and $\text{SQuAD}_{1.1}$ benchmarks in columns 5 to 8, where their compression rate and speedup are reported in columns 4 and 9, respectively. We focus on KD algorithms for encoder-only Transformers, as there is a lack of studies regarding KD algorithms for decoder-only

Table 6. Comparison of the Performance of Knowledge Distillation (KD) Algorithms for Encoder-Only Transformers on MNLI, QQP, and $SQuAD_{1.1}$ (SQD) Benchmarks

Method	Distillation Source§	Layer Matching	Comp. Rate	Accuracy/F1 Diff.✚ (%p) MNLI	QQP	SQD*	Avg.	Speedup
DistilBERT [115]	L	None	50.00%	-4.50	-1.10	-2.70	-2.77	-
MiniLM [138]	L, A	1:1	50.00%	-0.50	-0.30	-0.40†	-0.40	-
MiniLMv2 [137]	L, A	1:1	50.00%	-0.30	0.20	-0.50†	-0.20	2.00×
TinyBERT [55]‡	L, H, A, T	1:1	50.00%	0.70	0.50	-	0.60	2.00×
TinyBERT [55]‡	L, H, A, T	1:1	66.70%	-1.40	0.20	-	-0.60	9.40×
PKD [126]	L, H	1:1	50.00%	-2.20	-0.10	-	-1.15	-
Aguilar et al. [2]	L, H, A	1:1	50.00%	-	-0.07	-	-0.07	-
CKD [146]♦	L, H	many:1	50.00%	-1.48	-0.32	-	-0.90	-
ALP-KD [107]	L, H	many:1	50.00%	-1.53	-0.23	-	-0.88	-
BERT-EMD [79]	L, H, A, T	many:many	50.00%	0.30	0.40	-	0.35	1.90×
RAIL-KD [41]	L, H	many:many	50.00%	-1.20	0.40	-	-0.40	-

§: L (logits), H (hidden states), A (attention maps), T (token embeddings)
✚: (Compressed model's accuracy/F1 score) - (uncompressed model's accuracy/F1 score)
* $SQuAD_{1.1}$, †: experimental results of RoBERTa-base on $SQuAD_{2.0}$
‡: utilizes additional training data, ♦: reported results in ALP-KD [107]

We Summarize the Experimental Results of KD Algorithms Distilling Knowledge from a Pretrained BERT-Base to a Compressed one with Fewer Sublayers; a 50% Compression Rate Means the Student Model Contains the Half of the Number of Sublayers of the Teacher Model (BERT-Base).

Transformers, especially for LLMs; most KD algorithms are high-cost algorithms requiring prohibitively expensive costs to be applied to LLMs with more than 100 billion parameters [66, 89].

5.1.1 Distillation Source. A distillation source is a set of information in $\mathcal{T}$ and $\mathcal{S}$ that is used for distilling knowledge. KD uses logits (L), hidden states (H), attention maps (A), and token embeddings (T) as distillation sources. Logits contain task-specific knowledge which directly affects the performance of language models, and all KD algorithms [2, 38, 41, 55, 107, 115, 126, 137] in Table 6 include logits in their list of distillation sources. KD distills inherent knowledge in logits by reducing the KL divergence between logits $z^{\mathcal{T}}$ and $z^{\mathcal{S}}$ as in Equation (16). Superscripts $\mathcal{T}$ and $\mathcal{S}$ represent that the item is related to the teacher and student models, respectively. $\mathrm{D}_{\mathrm{KL}}(\cdot)$ is a KL divergence function and $s_t(\cdot)$ is a softmax function with temperature t. A softmax function with a higher temperature produces a softer probability distribution than that with a lower temperature.

$$\mathcal{L}_{\text{pred}} = \mathrm{D}_{KL}(s_t(z^{\mathcal{T}})||s_t(z^{\mathcal{S}})). \tag{16}$$

KD matches sublayer representations of $\mathcal{T}$ and $\mathcal{S}$ to distill generalized knowledge. This matching is vital since it gradually aligns the outputs of $\mathcal{T}$ and $\mathcal{S}$ from the lowest sublayer, finally aiding in the synchronization of their predictions. There are two distillation sources in Transformer sublayers: hidden states and attention maps. PKD [126] distills the knowledge of hidden states by reducing the gap between hidden states $H_n^{\mathcal{S}}$ and $H_m^{\mathcal{T}}$ as in Equation (17). n and m are indices of matched sublayers in $\mathcal{S}$ and $\mathcal{T}$, respectively. We explain how we match those sublayers in Section 5.1.2. PKD introduces a weight W_h to match the dimensions of $H_{n^{\mathcal{S}}}$ and $H_{m^{\mathcal{T}}}$.

$$\mathcal{L}_{\text{hid}} = \mathrm{MSE}(H_m^{\mathcal{T}}, W_h H_n^{\mathcal{S}}) \tag{17}$$

A self-attention mechanism is the core of Transformer-based language models and an attention map contains abundant knowledge of how to capture contextual information. The attention map P_i

of the ith attention head is computed as $S(K_i^T Q_i / d_h)$ in Equation (1). The jth column $(P_i)_{:,j}$ indicates how the jth token is affected by each token. Aguilar et al. [2] reduces the gap between attention maps of $\mathcal{T}$ and $\mathcal{S}$ to transfer the knowledge within the attention map as in Equation (18). In this equation, m and n represent the indices of the sublayers of $\mathcal{T}$ and $\mathcal{S}$, respectively.

$$\mathcal{L}_{\text{att}} = \sum_{i=1}^{H} \sum_{j=1}^{s} \mathrm{D}_{KL}\left((P^{\mathcal{T}}_{m,i})_{:,j} \| (P^{\mathcal{S}}_{n,i})_{:,j}\right). \tag{18}$$

TinyBERT [55] distills the knowledge in token embeddings by reducing the gap between embedding matrices $E^{\mathcal{S}}$ and $E^{\mathcal{T}}$. TinyBERT introduces a weight W_e to match the dimensions of $E^{\mathcal{S}}$ and $E^{\mathcal{T}}$.

$$\mathcal{L}_{\text{emb}} = \mathrm{MSE}(E^{\mathcal{T}}, W_e E^{\mathcal{S}}). \tag{19}$$

5.1.2 Layer Matching. Layer matching strategies represent how to match sublayers of $\mathcal{T}$ and $\mathcal{S}$ to distill knowledge even though they have different numbers of sublayers. We categorize sublayer matching strategies into 1:1, many:1, and many:many based on the number of sublayers used for knowledge distillation.

1:1 sublayer matching strategy distills the knowledge from individual sublayers of $\mathcal{T}$ to a corresponding sublayer in $\mathcal{S}$. Skip and Last mapping strategies are widely used to match the sublayers of $\mathcal{T}$ and $\mathcal{S}$ [2, 33, 53, 55, 59, 83, 126, 142]. The skip mapping strategy presumes that the useful knowledge of $\mathcal{T}$ is widely spread across its sublayers; thus, it matches each sublayer of $\mathcal{S}$ with a sublayer of $\mathcal{T}$ at uniform intervals. In contrast, the Last mapping strategy presumes that the essential knowledge of $\mathcal{T}$ is concentrated in the latter parts of $\mathcal{T}$, and accordingly they sequentially match the uppermost sublayers of $\mathcal{T}$ to those of $\mathcal{S}$. However, 1:1 sublayer matching strategies have the drawback that they lose knowledge within the unselected sublayers in $\mathcal{T}$ since the number of sublayers of $\mathcal{T}$ is larger than that of $\mathcal{S}$ in most cases.

Many:1 sublayer matching strategy allows a sublayer of $\mathcal{S}$ to receive knowledge from multiple sublayers of $\mathcal{T}$. CKD [146] groups the sublayers of $\mathcal{T}$ into multiple buckets and distills the knowledge within a bucket to a corresponding sublayer in $\mathcal{S}$. CKD resolves the knowledge loss problem of 1:1 matching strategies by letting all sublayers of $\mathcal{T}$ participate in KD. However, they employ inaccurate heuristics for defining buckets and matching them to sublayers in $\mathcal{S}$ [107]. ALP-KD [107] and Universal-KD [147] employ an attention mechanism to learn the amount of knowledge that each sublayer in $\mathcal{S}$ receives from each sublayer in $\mathcal{T}$. They do not suffer from inaccurate heuristic mappings of sublayers in $\mathcal{S}$ and $\mathcal{T}$ since they automatically learn an accurate mapping using the attention mechanism. However, they require a long training time since they distill knowledge between all pairs of sublayers in $\mathcal{S}$ and $\mathcal{T}$.

Many:many sublayer matching strategy distills the knowledge from multiple sublayers of $\mathcal{T}$ to those of $\mathcal{S}$. In many:many matching strategy, it is important to determine the matching pattern between sublayers of $\mathcal{T}$ and those of $\mathcal{S}$ since the matching pattern effectively enhances the performance of $\mathcal{S}$. BERT-EMD [79] finds a matching pattern between sublayers of $\mathcal{T}$ and $\mathcal{S}$ by employing **earth mover's distance** (**EMD**). Although the resulting model of BERT-EMD shows improved performance, BERT-EMD is time-intensive due to the computational overhead of searching the pattern. RAIL-KD [41] adopts a lightweight approach compared with BERT-EMD; it (1) randomly selects sublayers of $\mathcal{T}$ at each epoch, and (2) matches the selected sublayers with those of $\mathcal{S}$. RAIL-KD sets the number of the selected sublayers of $\mathcal{T}$ the same as that of $\mathcal{S}$. RAIL-KD shows that the randomized selection enables all sublayers of $\mathcal{T}$ to provide their knowledge to those of $\mathcal{S}$, minimizing the computational overhead of finding the optimal pattern for sublayer matching.

Table 7. Comparison of Low-Rank Approximation (LRA) for Encoder-Only Transformers.

Method	Target§	PLM♣	Cost	Comp. Rate	Accuracy/F1 Diff.✚ (%p) MNLI	QQP	SQD*	Avg.
Noach et al. [101]	P	Bb	High	40.73%	-1.70	0.90	-	-0.40
SVD [49]	P	Bb	High	50.00%	-1.90	-0.50	-	-1.20
FWSVD [49]	P	Bb	High	50.00%	-1.70	-0.20	-	-0.95
Wang et al. [135]	P	Bb	High	85.70%	0.00	0.50	-	0.25
Wang et al. [135]	P	Bb	High	97.91%	-3.80	-0.40	-	-2.10
LoRA [50]	A	Rb	Low	0.24%¶	-0.10	-0.90	-	-0.50
AdaLoRA [164]	A	Db	Low	0.17%¶	0.76	-0.64	0.90	0.34

§: P (original parameter), A (adaptor), ¶: (the size of adapter) / (the size of model)
♣: Bb (BERT-base), Rb (RoBERTa-base), Db (DeBERTaV3-base), *: $\text{SQuAD}_{1.1}$
✚: (Compressed model's accuracy/F1 score) - (uncompressed model's accuracy/F1 score)
We Compare the Performance of PLMs on MNLI, QQP, and $\text{SQuAD}_{1.1}$ After Being Compressed with Each LRA Algorithm.

5.2 Low-Rank Approximation

LRA is a technique that decomposes a high-dimensional matrix or tensor into a multiplication of lower-dimensional matrices or tensors that approximates the original one. LRA is effective for language models as their parameters typically exhibit low-rank characteristics [15, 21, 135, 136]; the parameters are located in the lower-dimensional subspace. We summarize the performance of LRA algorithms for encoder-only Transformers in Table 7 and divide them into two groups based on their target matrices for matrix (tensor) decomposition.

The first group of algorithms directly decomposes original parameters (P) in language models to reduce the memory and computational cost of language models [49, 101, 135]. These approaches are inherently high-cost algorithms, as they require an expensive optimization process on the decomposed matrices (tensors) to reduce the gap between the outputs of models before and after compression. Noach et al. [101] applies **singular value decomposition** (**SVD**) for each parameter matrix, and FWSVD [49] proposes to select singular vectors in SVD considering Fisher information which is directly related to the cross-entropy loss. Wang et al. [135] achieves a remarkable compression rate over 97.91% without significant accuracy degradation by exploiting tensor decomposition. They generate a three-dimensional tensor by (1) concatenating all parameter matrices in each sublayer, and (2) vertically stacking the concatenated matrices. SharVeT [157] extends this direction to LLMs by sharing SVD-based low-rank bases across similarity-grouped modules and refining sharing-induced discrepancies with lightweight vector-based tuning.

The second group of algorithms, called **parameter-efficient fine-tuning** (**PEFT**) [50, 132, 164], applies LRA to adaptors (A) to reduce the memory footprint of fine-tuning processes. PEFT algorithms enable low-cost training by fixing the original parameters in the language model and updating only low-rank adaptors using gradient descent. They merge low-rank adaptors into the original parameters after fine-tuning, thus the size of a language model does not change. The value of PEFT algorithms lies in the significant reduction of the memory footprint during fine-tuning processes because of the reduced amount of gradients to save, and thus, they are useful for fine-tuning billion-scale LLMs on limited computational resources. LoRA [50] is the first algorithm that introduces low-rank adaptors for fine-tuning language models. However, finding the proper rank of each adaptor reflecting its importance is challenging. AdaLoRA [164] proposes an algorithm to find the proper rank for each adaptor by formulating the rank selection problem as a pruning

problem: measuring the importance of each rank-one component and gradually pruning the least important ones. PEFT algorithms contribute to improving the accuracy of pruning [96, 163] and quantization [25, 82] for LLMs, which require memory-efficient fine-tuning processes to increase accuracy. We explain more about PEFT algorithms with LoRA in Section 5.3.

5.3 Dynamic Inference

Dynamic inference is a model compression technique that enhances inference efficiency by selectively activating only a small part of a model according to the given input. For example, **Mixture-of-Depth** (**MoD**) dynamically skips certain layers during inference when they are deemed unnecessary for processing the input. We categorize these algorithms into high-cost [85, 100, 113, 119, 122, 123] and low-cost [88, 94], depending on their cost for compression. High-cost algorithms typically require updating all the parameters in LLMs, resulting in substantial training costs. These costs arise from the need to recover performance losses caused by architectural modifications such as skipping layers or pruning activations. In contrast, low-cost algorithms either train only lightweight auxiliary modules (e.g., routers) or require no additional training at all, making them significantly more practical than high-cost ones for large-scale models. Another criterion for categorizing dynamic inference algorithms is the dimension they control, either width or depth.

We categorize high-cost algorithms into two groups based on their target dimension: width [100, 122, 123] and depth [85, 113, 119]. Width-adaptive methods improve the inference efficiency of language models by promoting activation sparsity within each layer in the models. ReLUfication [100] replaces smooth activation functions, such as GeLU [45], with ReLU to promote activation sparsity and fine-tunes all the parameters in the model to recover performance degradation. ProSparse [123] mitigates the performance degradation of ReLUfication by imposing progressive sparsity regularization during fine-tuning and further introduces hardware-friendly structured activation sparsity. SparseInfer [122] accelerates Sparse LLMs by leveraging sign-bit-based XOR gating and custom CUDA kernels, enabling efficient inference without retraining. We categorize SparseInfer as a high-cost compression algorithm since it requires a sparsified model after an expensive fine-tuning, and has the ability to control sparsity using the hyperparameter for gating. Depth-adaptive methods accelerate the inference of language models by adjusting the number of layers to compute. Early exit approaches [85, 119] reduce unnecessary computation in later layers by fine-tuning the model to make confident early predictions. Router-based approaches [113] typically require auxiliary classifiers called routers at each layer to enable exit decisions. Global Past-Future Early Exit [85] improves the reliability of exit decisions by incorporating both past and future contexts when estimating token-level confidence. CALM [119] aims to prevent degradation in overall sequence quality caused by premature exit decisions. To achieve this, it estimates the global impact of each exit by measuring the change in sequence-level loss when a token exits early, and calibrates confidence scores accordingly during inference. MoD [113] enables token-dependent depth by learning routing policies that determine whether each token should be skipped or executed at each layer, without requiring a sequential execution from lower to higher layers.

Low-cost algorithms [88, 94] reduce inference cost by increasing activation sparsity without fine-tuning the entire model. Most low-cost methods focus on width-adaptive approaches, selectively skipping subsets of neurons or attention heads within each layer, since depth-adaptive approaches typically demand expensive fine-tuning, causing greater performance degradation compared with width-adaptive ones. Deja Vu [94] selects neurons and attention heads to activate using an auxiliary predictor, which requires lightweight training. TEAL [88] selectively computes activations based on their magnitudes without requiring any additional training.

5.4 Representative Algorithm: LoRA

Low-rank adaptation (LoRA) [50] is an algorithm that fine-tunes pretrained language models for downstream tasks by introducing a small number of low-rank adaptors. LoRA keeps the original parameters in the pretrained language model unchanged and updates only the low-rank adaptors to increase the efficiency of the adaptation processes. LoRA is an essential algorithm for compressing LLMs since it significantly reduces the memory footprint during the fine-tuning process by reducing the number of gradients to memorize. LoRA demonstrates its effectiveness by reducing the memory footprint from 1.2TB to 350GB for fine-tuning a GPT-3 [9] 175B model [50].

Consider a forward pass $h = Wx$ that generates a d_o dimensional output $h \in \mathbb{R}^{d_o}$ using a fine-tuned weight matrix $W \in \mathbb{R}^{d_o \times d_i}$ and a d_i dimensional input $x \in \mathbb{R}^{d_i}$. LoRA separates W into the sum of a pretrained weight matrix $W_0 \in \mathbb{R}^{d_o \times d_i}$ and an incremental weight matrix $\Delta W \in \mathbb{R}^{d_o \times d_i}$. After that, LoRA approximates ΔW with the multiplication of two low-rank matrices $B \in \mathbb{R}^{d_o \times r}$ and $A \in \mathbb{R}^{r \times d_i}$ where r is the rank of the low-rank matrices. The modified forward pass for computing h is represented in Equation (20).

$$h = Wx = W_0 x + \Delta W x \approx W_0 x + BAx \tag{20}$$

After fine-tuning, B and A are merged into the original weight matrix W_0 so that the number of parameters of the language model is not changed. Efficiently finding a proper rank r that maximizes the accuracy of a fine-tuned model is crucial for LoRA, since it requires intractable time to manually examine the accuracy of fine-tuned models across all possible ranks. DyLoRA [132] fine-tunes LMs across a range of ranks instead of focusing on a single rank. This method selects the rank that shows the best performance within the range of ranks. LoRA and DyLoRA overlook the importance of individual weight matrices, and utilize the same rank for all weight matrices. In contrast, AdaLoRA [164] tackles this limitation by dynamically assigning ranks based on the importance of each weight matrix.

The potential of LoRA stems from its compatibility and ease of integration with various compression algorithms, such as pruning [96, 164] and quantization [11, 25], as well as the **parameter-efficient fine-tuning (PEFT)** algorithms [13, 48, 81, 97, 162]. These algorithms utilize a fine-tuning process with LoRA to compensate for errors induced by compression to improve the performance of the compressed models.

6 Discussion

In previous sections, we summarize recent trends in language model compression algorithms, including useful background knowledge and in-depth analysis of representative compression algorithms. In this section, we propose promising research directions discovered through our survey, especially focusing on how to enhance the performance of low-cost compression algorithms for LLMs. We summarize the discovered directions as follows.

- **Low-cost iterative algorithms.** How can we efficiently apply iterative compression algorithms to LLMs to improve the accuracy of compressed models?
- **Direct task-objective optimization algorithms.** How can we directly reduce the target objective function rather than proxy objective functions such as sublayer-wise reconstruction errors in LLMs?
- **Application of high-cost algorithms to LLMs using PEFT.** How can we leverage PEFT algorithms to replace expensive retraining processes of accurate high-cost algorithms?
- **Unification of diverse compression algorithms.** How can we unify diverse compression algorithms to maximize compression rates?

— **System support for compressed models.** How can we build an effective system to support compressed models?

6.1 Low-Cost Iterative Algorithms

Compression algorithms consist of two stages: (1) modifying the representation of weights to make them computationally or memory efficient, and (2) compensating for compression errors induced by the modification. In most language model compression scenarios, there is no access to a large corpus used for pretraining, but only relatively small task-specific datasets or even smaller calibration datasets are available. Therefore, language model compression algorithms should preserve useful features learned from the large corpus since they cannot be reconstructed using a small dataset.

High-cost compression algorithms successfully preserve the useful features since they gradually transform weights and compensate for compression errors. However, low-cost compression algorithms tend to transform all weights at once and lose the useful features of the pretrained models. Though they attempt to recover the useful features of the pretrained models via weight tuning, they fail to recover the lost features. For example, LLM-pruner [96] fails to restore the PLM's performance under a low compression rate of 20% even though it fine-tunes the compressed model with LoRA [50]. One notable approach is K-prune [105]. K-prune addresses the accuracy degradation problem of low-cost compression algorithms by employing an iterative compression process consisting of mild pruning and error compensation; it prunes a small part of a pretrained model at each iteration and recovers performance through an efficient weight-tuning process, preventing the pretrained model from losing its core features. As a result, K-prune achieves significant performance improvement compared with other low-cost compression algorithms [72, 102] while maintaining efficiency. The benefit of iterative compression algorithms is also reported in numerous works [29, 42, 117] for other types of neural networks, therefore it is a promising approach to design an accurate and iterative low-cost algorithm to achieve high accuracy without losing efficiency.

6.2 Direct Task-Objective Optimization Algorithms

In high-cost compression scenarios, it is natural to perform gradient descent on task-specific objective function. However, many low-cost algorithms, especially those designed for LLMs, reduce their cost for compression by minimizing inexpensive proxy objective functions, such as sublayer-wise or weight-wise reconstruction errors, using least-squares. Nevertheless, since the goal of compression is to minimize the task-specific objective function, such alternatives lead to suboptimal performance. Prominent examples are FWSVD [49] and K-prune [105]. Although SVD theoretically guarantees minimal reconstruction error for matrices, FWSVD shows that incorporating SVD with gradient information regarding the task-specific objective function yields a significant performance improvement. In K-prune, the authors show that additionally preserving task-specific objective knowledge on top of sublayer-wise reconstruction yields significant accuracy improvements. Therefore, replacing proxy objective functions, such as sublayer-wise reconstruction, with task-specific ones is a promising direction for algorithms [30, 32, 155, 156].

6.3 Application of High-Cost Algorithms to LLMs Using PEFT

High-cost algorithms typically outperform low-cost ones by minimizing compression errors through a long fine-tuning process. However, mimicking those fine-tuning processes in low-cost compression scenarios requires prohibitively expensive fine-tuning processes. PEFT algorithms [50, 164] contribute to improving the efficiency of the expensive fine-tuning process in LLMs. PEFT algorithms enable more accurate fine-tuning results than local weight reconstruction in current low-cost compression scenarios since they directly reduce the objective function, such as cross-entropy. We expect that various useful techniques previously utilized in high-cost compression scenarios such

as KD and LRA can be applied to low-cost compression scenarios by improving PEFT algorithms to further reduce the cost of fine-tuning.

6.4 Unification of Diverse Compression Algorithms

In this survey, we cover diverse types of compression algorithms, including pruning, quantization, KD, LRA, and dynamic inference. It is a promising research direction to unify different types of compression algorithms to achieve higher compression rates, since they address different kinds of inefficiencies in language models. For example, we might explicitly remove meaningless weights using pruning algorithms and retain other weights in low bits using quantization algorithms. Extensive studies [5, 30, 135, 148] have demonstrated that combining different compression algorithms results in high compression rates by mitigating diverse inefficiencies within the model. Moreover, Kim et al. [60] highlight that compression order is also important, suggesting that weaker perturbations should precede stronger ones in joint compression pipelines. However, there is a lack of research that unifies three or more compression algorithms at once, especially for LLMs. Therefore, designing an algorithm to integrate diverse compression algorithms considering their characteristics is a promising approach to achieve extremely high compression rates without sacrificing accuracy.

6.5 System-Level Support for Compressed Models

In this survey, we focus on compression algorithms designed for language models, and compressed models often require additional system-level support for practical usage. For example, quantized models require specialized kernels to achieve expected acceleration, and lower-precision models may be slower than higher-precision models without optimized kernels [120]. SparseInfer [122] is a notable example that provides powerful system support for accelerating pruned models; it accelerates a pruned Llama-2 13B model up to 1.65×. However, there is a lack of system-level support for compressed models generated with unstructured pruning [30, 67, 116], mixed-precision quantization [151], or other algorithms. Therefore, it is promising to design an effective system to maximally leverage compressed models.

7 Conclusion

In this article, we perform an extensive survey of language model compression algorithms including pruning, quantization, KD, LRA, and dynamic inference algorithms. We cover compression algorithms for both encoder-only and decoder-only Transformers. We elaborate on the details of background knowledge and three representative algorithms to promote a better understanding of readers. Finally, we propose and discuss promising research directions focusing on cost-effective compression algorithms for LLMs, which are the most important language models.

References

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). *arXiv:1803.08375* (2018).
[2] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. 2020. Knowledge Distillation from Internal Representations. In *AAAI*.
[3] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv:1607.06450* (2016).
[4] Haoli Bai, Lu Hou, Lifeng Shang, Xin Jiang, Irwin King, and Michael R. Lyu. 2022. Towards efficient post-training quantization of pre-trained language models. In *NeurIPS*.
[5] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael R. Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of bert quantization. In *IJCNLP*.
[6] Kayhan Behdin, Ayan Acharya, Aman Gupta, Sathiya Keerthi Selvaraj, and Rahul Mazumder. 2023. QuantEase: Optimization-based quantization for language models - An efficient and intuitive algorithm. *arXiv:2309.01885* (2023).
[7] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432* (2013).

[8] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram A. Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation Model. *arXiv:1906.00532* (2019).

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell et al. 2020. Language models are few-shot learners. In *NeurIPS*.

[10] Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. *arXiv:1708.00055* (2017).

[11] Yuji Chai, John Gkountouras, Glenn G. Ko, David Brooks, and Gu-Yeon Wei. 2023. INT2.1: Towards fine-tunable quantized large language models with error correction through low-rank adaptation. *arXiv:2306.08162* (2023).

[12] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2023. QuIP: 2-bit quantization of large language models with guarantees. In *NeurIPS*.

[13] Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. In *ICLR*.

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman et al. 2021. Evaluating large language models trained on code. *arXiv:2107.03374* (2021).

[15] Patrick H. Chen, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. 2021. DRONE: Data-aware low-rank compression for large nlp models. In *NeurIPS*.

[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*.

[17] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *EMNLP*.

[18] Tejalal Choudhary, Vipul Kumar Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *AIR* 53, 7 (2020), 5113–5155.

[19] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

[20] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv:2110.14168* (2021).

[21] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 2020. Multi-head attention: Collaborate instead of concatenate. *arXiv:2006.16362* (2020).

[22] Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy in pretrained transformer models. In *EMNLP*.

[23] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *IEEE* 108, 4 (2020), 485–532.

[24] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *NeurIPS*.

[25] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized llms. In *NeurIPS*.

[26] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. SpQR: A sparse-quantized representation for near-lossless llm weight compression. *arXiv:2306.03078* (2023).

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

[28] William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *IWP*.

[29] Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*.

[30] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *ICML*.

[31] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv:2210.17323*.

[32] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate quantization for generative pre-trained transformers. In *ICLR*.

[33] Hao Fu, Shaojun Zhou, Qihong Yang, Junjie Tang, Guiquan Liu, Kaikui Liu, and Xiaolong Li. 2021. LRC-BERT: Latent-representation contrastive knowledge distillation for natural language understanding. In *AAAI*.

[34] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Hassan Sajjad, Preslav Nakov, Deming Chen, and Marianne Winslett. 2021. Compressing large-scale transformer-based models: A case study on bert. *TACL* 9 (2021), 1061–1080.

[35] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A survey of quantization methods for efficient neural network inference. *arXiv:2103.13630*.
[36] Zhuocheng Gong, Jiahao Liu, Qifan Wang, Yang Yang, Jingang Wang, Wei Wu, Yunsen Xian, Dongyan Zhao, and Rui Yan. 2023. PreQuant: A task-agnostic quantization approach for pre-trained language models. In *ACL 2023*.
[37] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. PoWER-BERT: Accelerating bert inference via progressive word-vector elimination. In *ICML (PMLR, Vol. 119)*.
[38] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *ICLR*.
[39] Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. 2025. SlimLLM: Accurate structured pruning for large language models. In *ICML*.
[40] Manish Gupta and Puneet Agrawal. 2022. Compression of deep learning models for text: A survey. *ACM TKDD* 16, 4 (2022), 1–55.
[41] Md. Akmal Haidar, Nithin Anchuri, Mehdi Rezagholizadeh, Abbas Ghaddar, Philippe Langlais, and Pascal Poupart. 2022. RAIL-KD: RAndom intermediate layer mapping for knowledge distillation. In *NAACL*.
[42] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In *NIPS*.
[43] Babak Hassibi and David G. Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*.
[44] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *NeurIPS Datasets and Benchmarks*.
[45] Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv:1606.08415* (2016).
[46] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *JMLR* 22, 241 (2021), 1–124.
[47] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. DynaBERT: Dynamic bert with adaptive width and depth. In *NeurIPS*.
[48] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *ICML*.
[49] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. In *ICLR*.
[50] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*.
[51] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*.
[52] Yongkweon Jeon, Chungman Lee, Eulrang Cho, and Yeonju Ro. 2022. Mr.BiQ: Post-training non-uniform quantization based on minimizing the reconstruction error. In *CVPR*.
[53] Ananya Harsh Jha, Dirk Groeneveld, Emma Strubell, and Iz Beltagy. 2023. How to train your (compressed) large language model. *arXiv:2305.14864v2* (2023).
[54] Tianchu Ji, Shraddhan Jain, Michael Ferdman, Peter A. Milder, H. Andrew Schwartz, and Niranjan Balasubramanian. 2021. On the distribution, sparsity, and inference-time quantization of attention values in transformers. In *IJCNLP*.
[55] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling bert for natural language understanding. In *EMNLP (ACL)*.
[56] Jing Jin, Cai Liang, Tiancheng Wu, Liqin Zou, and Zhiliang Gan. 2021. KDLSQ-BERT: A quantized bert combining knowledge distillation with learned step size quantization. *arXiv:2101.05938* (2021).
[57] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.
[58] Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. 2023. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. In *NeurIPS*.
[59] Junho Kim, Jun-Hyung Park, Mingyu Lee, Wing-Lam Mok, Joon-Young Choi, and SangKeun Lee. 2022. Tutoring helps students learn better: Improving knowledge distillation for bert with tutor network. In *EMNLP*.
[60] Minjun Kim, Jaehyeon Choi, Hyunwoo Yang, Jongjin Kim, Jinho Song, and U Kang. 2026. Prune-then-quantize or quantize-then-prune? Understanding the impact of compression order in joint model compression. In *ICLR*.
[61] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only bert quantization. In *ICML (PMLR, Vol. 139)*.

[62] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. 2024. SqueezeLLM: Dense-and-sparse quantization. In *ICML*.
[63] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W. Mahoney, Yakun Sophia Shao, and Amir Gholami. 2023. Full stack optimization of transformer inference: A survey. *arXiv:2302.14017*.
[64] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *KDD*.
[65] Young Jin Kim, Rawn Henry, Raffy Fahim, and Hany Hassan Awadalla. 2023. FineQuant: Unlocking efficiency with fine-grained weight-only quantization for llms. *arXiv:2308.09723*.
[66] Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. 2024. DistiLLM: Towards streamlined distillation for large language models. In *ICML*.
[67] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. In *EMNLP*.
[68] Eldar Kurtic, Elias Frantar, and Dan Alistarh. 2023. ZipLM: Hardware-aware structured pruning of language models. In *NeurIPS*.
[69] Andrey Kuzmin, Mart van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. 2022. FP8 quantization: The power of the exponent. In *NeurIPS*.
[70] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee et al. 2019. Natural questions: A benchmark for question answering research. *TACL* 7 (2019), 453–466.
[71] Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. 2022. AlphaTuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. In *EMNLP*.
[72] Woosuk Kwon, Sehoon Kim, Michael W. Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. In *NeurIPS*.
[73] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M. Rush. 2021. Block pruning for faster transformers. In *EMNLP*.
[74] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite bert for self-supervised learning of language representations. In *ICLR*.
[75] Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal brain damage. In *NeurIPS*.
[76] Byung-Kwan Lee, Junho Kim, and Yong Man Ro. 2022. Masking adversarial damage: Finding adversarial saliency for robust and sparse network. In *CVPR*.
[77] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. OWQ: Lessons learned from activation outliers for weight quantization in large language models. In *AAAI*.
[78] Hyun Dong Lee, Seongmin Lee, and U Kang. 2021. AUBER: Automated bert regularization. *PLOS ONE* 16, 6 (2021), 1–16.
[79] Jianquan Li, Xiaokang Liu, Honghong Zhao, Ruifeng Xu, Min Yang, and Yaohong Jin. 2020. BERT-EMD: Many-to-many layer mapping for bert compression with earth mover's distance. In *EMNLP*.
[80] Liang Li, Qingyuan Li, Bo Zhang, and Xiangxiang Chu. 2024. Norm tweaking: High-performance low-bit quantization of large language models. In *AAAI*.
[81] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL/IJCNLP*.
[82] Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. 2023. LoftQ: LoRA-fine-tuning-aware quantization for large language models. *arXiv:2310.08659*.
[83] Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Less is more: Task-aware layer-wise distillation for language model compression. In *ICML*.
[84] Kevin J. Liang, Weituo Hao, Dinghan Shen, Yufan Zhou, Weizhu Chen, Changyou Chen, and Lawrence Carin. 2021. MixKD: Towards efficient distillation of large-scale language models. In *ICLR*.
[85] Kaiyuan Liao, Yi Zhang, Xuancheng Ren, Qi Su, Xu Sun, and Bin He. 2021. A global past-future early exit method for accelerating inference of pre-trained language models. In *NAACL-HLT*.
[86] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2024. AWQ: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*.
[87] Zi Lin, Jeremiah Z. Liu, Zi Yang, Nan Hua, and Dan Roth. 2020. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *EMNLP*.
[88] James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. 2025. Training-free activation sparsity in large language models. In *ICLR*.

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

[90] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv:1907.11692*.

[91] Zechun Liu, Kwang-Ting Cheng, Dong Huang, Eric P. Xing, and Zhiqiang Shen. 2022. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *CVPR*.

[92] Zechun Liu, Barlas Oguz, Aasish Pappu, Lin Xiao, Scott Yih, Meng Li, Raghuraman Krishnamoorthi, and Yashar Mehdad. 2022. BiT: Robustly binarized multi-distilled transformer. In *NeurIPS*.

[93] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024. LLM-QAT: Data-free quantization aware training for large language models. In *Findings of ACL*.

[94] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. 2023. Deja Vu: Contextual sparsity for efficient llms at inference time. In *ICML*.

[95] Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through l_0 regularization. In *ICLR*.

[96] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-pruner: On the structural pruning of large language models. In *NeurIPS*.

[97] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 2022. UniPELT: A unified framework for parameter-efficient language model tuning. In *ACL*.

[98] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one?. In *NeurIPS*.

[99] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu et al. 2022. FP8 formats for deep learning. *arXiv:2209.05433*.

[100] Seyed-Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C. del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. ReLU strikes back: Exploiting activation sparsity in large language models. In *ICLR*.

[101] Matan Ben Noach and Yoav Goldberg. 2020. Compressing pre-trained language models by matrix decomposition. In *IJCNLP*.

[102] Azade Nova, Hanjun Dai, and Dale Schuurmans. 2023. Gradient-free structured pruning with unlabeled data. In *ICML (PMLR, Vol. 202)*.

[103] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *EMNLP*.

[104] Seungcheol Park, Jeongin Bae, Beomseok Kwon, Minjun Kim, Byeongwook Kim, Se Jung Kwon, U Kang, and Dongsoo Lee. 2025. Unifying uniform and binary-coding quantization for accurate compression of large language models. In *ACL*.

[105] Seungcheol Park, Hojun Choi, and U Kang. 2024. Accurate retraining-free pruning for pretrained encoder-based language models. In *ICLR*.

[106] Seungcheol Park, Sojin Lee, Jongjin Kim, Jinsik Lee, Hyunjik Jo, and U Kang. 2025. Accurate sublayer pruning for large language models by exploiting latency and tunability information. In *IJCAI*.

[107] Peyman Passban, Yimeng Wu, Mehdi Rezagholizadeh, and Qun Liu. 2021. ALP-KD: Attention-based layer projection for knowledge distillation. In *AAAI*.

[108] Tairen Piao, Ikhyun Cho, and U Kang. 2022. SensiMix: Sensitivity-aware 8-bit index & 1-bit value mixed precision quantization for bert compression. *PLOS ONE* 17, 4 (2022), 1–22.

[109] Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. 2022. BiBERT: Accurate fully binarized bert. In *ICLR*.

[110] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR* 21, 140 (2020), 1–67.

[111] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *ACL*.

[112] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

[113] David Raposo, Samuel Ritter, Blake A. Richards, Timothy P. Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv:2404.02258*.

[114] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *CSL* 77 (2023), 101429.

[115] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv:1910.01108*.
[116] Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*.
[117] Pedro Savarese, Hugo Silva, and Michael Maire. 2020. Winning the lottery with continuous sparsification. In *NeurIPS*.
[118] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé et al. 2022. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv:2211.05100*.
[119] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In *NeurIPS*.
[120] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *ICLR*.
[121] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI*.
[122] Jiho Shin, Hoeseok Yang, and Youngmin Yi. 2025. SparseInfer: Training-free prediction of activation sparsity for fast llm inference. In *DATE*.
[123] Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. 2025. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *COLING*.
[124] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2021. Training with quantization noise for extreme model compression. In *ICLR*.
[125] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *ICLR*.
[126] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. In *IJCNLP*.
[127] Hanlin Tang, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. 2022. MKQ-BERT: Quantized bert with 4-bits weights and activations. *arXiv:2203.13483*.
[128] Chaofan Tao, Lu Hou, Haoli Bai, Jiansheng Wei, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. 2023. Structured pruning for efficient generative pre-trained language models. In *ACL*.
[129] Chaofan Tao, Lu Hou, Wei Zhang, Lifeng Shang, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. 2022. Compression of generative pre-trained language models via quantization. In *ACL*.
[130] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv:2302.13971*.
[131] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.
[132] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *EACL*.
[133] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.
[134] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.
[135] Benyou Wang, Yuxin Ren, Lifeng Shang, Xin Jiang, and Qun Liu. 2022. Exploring extreme parameter compression for pre-trained language models. In *ICLR*.
[136] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*.
[137] Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *IJCNLP (ACL)*.
[138] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *NeurIPS*.
[139] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020. Structured pruning of large language models. In *EMNLP*.
[140] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. 2022. Outlier suppression: Pushing the limit of low-bit transformer language models. In *NeurIPS*.
[141] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*.
[142] Chuhan Wu, Fangzhao Wu, and Yongfeng Huang. 2021. One teacher is enough? pre-trained language model distillation from multiple teachers. In *IJCNLP (ACL)*.

[143] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv:2004.09602*.
[144] Xiaoxia Wu, Cheng Li, Reza Yazdani Aminabadi, Zhewei Yao, and Yuxiong He. 2023. Understanding int4 quantization for transformer models: Latency speedup, composability, and failure cases. In *ICML (PMLR)*.
[145] Xiaoxia Wu, Zhewei Yao, and Yuxiong He. 2023. ZeroQuant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats. *arXiv:2307.09782*.
[146] Yimeng Wu, Peyman Passban, Mehdi Rezagholizadeh, and Qun Liu. 2020. Why skip if you can combine: A simple knowledge distillation technique for intermediate layers. In *EMNLP*.
[147] Yimeng Wu, Mehdi Rezagholizadeh, Abbas Ghaddar, Md. Akmal Haidar, and Ali Ghodsi. 2021. Universal-kd: Attention-based output-grounded intermediate layer knowledge distillation. In *EMNLP*.
[148] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *ACL*.
[149] Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *ICML (PMLR, Vol. 202)*.
[150] Runxin Xu, Fuli Luo, Chengyu Wang, Baobao Chang, Jun Huang, Songfang Huang, and Fei Huang. 2022. From dense to sparse: Contrastive pruning for better pre-trained language model compression. In *AAAI*.
[151] June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024. No token left behind: reliable kv cache compression via importance-aware mixed precision quantization. *arXiv:2402.18096*.
[152] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. In *NeurIPS*.
[153] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. 2023. ZeroQuant-v2: Exploring post-training quantization in llms from comprehensive study to low rank compensation. *arXiv:2303.08302*.
[154] Zhewei Yao, Xiaoxia Wu, Linjian Ma, Sheng Shen, Kurt Keutzer, Michael W. Mahoney, and Yuxiong He. 2022. LEAP: Learnable pruning for transformer-based models. *arXiv:2105.14636*.
[155] Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Mykola Pechenizkiy, Yi Liang, Zhangyang Wang, and Shiwei Liu. 2024. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. In *ICML*.
[156] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. RPTQ: Reorder-based post-training quantization for large language models. *arXiv:2304.01089*.
[157] Jeongin Yun, Jaeri Lee, Jongjin Kim, Minjun Kim, Jinho Song, and U Kang. 2026. SharVeT: Similarity-aware parameter sharing with vector-based tuning for efficient llm compression. In *ACL*.
[158] Edouard Yvinec, Arnaud Dapogny, and Kevin Bailly. 2025. NUPES : Non-uniform post-training quantization via power exponent search. *TPAMI* 47, 11 (2025), 10012–10021.
[159] Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. 2023. PowerQuant: Automorphism search for non-uniform quantization. In *ICLR*.
[160] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. GOBO: Quantizing attention-based nlp models for low latency and energy efficient inference. In *MICRO*.
[161] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit bert. In *NeurIPS*.
[162] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *ACL*.
[163] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. 2024. LoRAPrune: Structured Pruning meets low-rank parameter-efficient fine-tuning. In *Findings of ACL*.
[164] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.
[165] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin et al. 2022. OPT: Open pre-trained transformer language models. *arXiv:2205.01068*.
[166] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit bert. In *EMNLP*.
[167] Jiaqi Zhao, Miao Zhang, Ming Wang, Yuzhang Shang, Kaihao Zhang, Weili Guan, Yaowei Wang, and Min Zhang. 2025. PTQ1.61: Push the real limit of extremely low-bit post-training quantization methods for large language models. In *ACL*.
[168] Zihan Zhao, Yuncong Liu, Lu Chen, Qi Liu, Rao Ma, and Kai Yu. 2020. An investigation on different underlying quantization schemes for pre-trained language models. In *NLPCC*.

Received 27 January 2024; revised 10 May 2026; accepted 15 May 2026