A Survey on Model Compression for Large Language Models

Authors: Xunyu Zhu, Jian Li (Corresponding author), Yong Liu, Can Ma, Weiping Wang

Affiliations:

- Institute of Information Engineering, Chinese Academy of Sciences
- School of Cyber Security, University of Chinese Academy of Sciences
- Gaoling School of Artificial Intelligence, Renmin University of China

Contact: zhuxunyu@iie.ac.cn, lijian9026@iie.ac.cn, macan@iie.ac.cn, wangweiping@iie.ac.cn, liuyonggsai@ruc.edu.cn

Abstract

Large Language Models (LLMs) have transformed natural language processing tasks successfully. Yet, their large size and high computational needs pose challenges for practical use, especially in resource-limited settings. Model compression has emerged as a key research area to address these challenges. This paper presents a survey of model compression techniques for LLMs. We cover methods like quantization, pruning, and knowledge distillation, highlighting recent advancements. We also discuss benchmarking strategies and evaluation metrics crucial for assessing compressed LLMs. This survey offers valuable insights for researchers and practitioners, aiming to enhance efficiency and real-world applicability of LLMs while laying a foundation for future advancements.

Introduction

Large Language Models (LLMs) refer to Transformer language models that contain billions (or more) of parameters, which are trained on massive text data. LLMs consistently exhibit remarkable performance across various tasks, but their exceptional capabilities come with significant challenges stemming from their extensive size and computational requirements. For instance, the GPT-175B model, with an impressive 175 billion parameters, demands a minimum of 350GB of memory in half-precision (FP16) format. Furthermore, deploying this model for inference necessitates at least five A100 GPUs, each featuring 80GB of memory, to efficiently manage operations. To tackle these issues, a prevalent approach known as model compression offers a solution. Model compression involves transforming a large, resource-intensive model into a compact version suitable for deployment on resource-constrained devices. Additionally, model compression can enhance LLM inference speed and optimizes resource efficiency.

In our paper, our primary objective is to illuminate the recent strides made in the domain of model compression techniques tailored specifically for LLMs. Our work conducts an exhaustive survey of methodologies, metrics, and benchmarks of model compression for LLMs. The taxonomy of model compression methods for LLMs

includes quantization, pruning, knowledge distillation, and low-rank factorization. Furthermore, our study sheds light on prevailing challenges and offers a glimpse into potential future research trajectories in this evolving field. We advocate for collaborative efforts within the community to pave the way for an ecologically conscious, all-encompassing, and sustainable future for LLMs. While there were previous surveys on neural networks model compression and it has been lightly discussed in prior surveys on LMs and LLMs, our work is the inaugural survey dedicated solely to model compression for LLMs.

Metrics and Benchmarks

Metrics

Model compression of LLMs can be measured using various metrics, which capture different aspects of performance. These metrics are commonly presented alongside accuracy and zero-shot ability to comprehensively evaluate the LLM.

Model Size in a LLM typically is measured by the number of total parameters of the LLM. In general, LLMs with more parameters often requires more computational resources and memory for both training and inference.

Floating Point Operations (FLOPs) is an indicator that measures the computational efficiency of LLMs, representing the number of floating-point operations required for the LLM to perform an instance. In model compression, reducing FLOPs helps to make the LLM run faster and more efficiently.

Mean FLOPS Utilization (MFU) quantifies the practical efficiency of computational resource utilization by LLMs during tasks. MFU measures the ratio of actual FLOPS utilized by the LLM to the maximum theoretical FLOPS of a device. Unlike FLOPs, which estimates the maximum operations an LLM might perform, MFU assesses the actual effectiveness of resource use in operation. Essentially, while FLOPs measures a LLM's theoretical compute needs, MFU shows how effectively these computations are utilized in practice.

Inference time (i.e., latency) measures the time taken by the LLM to process and generate responses for input data during inference. Inference time is particularly crucial for real-world applications where the LLM needs to respond for user queries or process large amounts of data in real-time.

Speedup Ratio measures how much faster a compressed LLM performs tasks compared to the original LLM. Specifically, it measures the ratio of the inference time of the uncompressed model over the inference time of the compressed model. Higher ratios mean greater efficiency and reduced computation time, highlighting effective compression.

Compression Ratio measures how much a LLM's size is reduced through compression, calculated as the original size divided by the compressed size. Higher ratios mean greater size reduction, showing the compression's effectiveness in saving storage and memory.

Benchmarks and Datasets

The main goal of these benchmarks and datasets is to measure the efficiency and performance of compressed LLMs in comparison to their uncompressed counterparts. These benchmarks and datasets typically consist of diverse tasks and datasets that cover a range of natural language processing challenges.

Common Benchmarks and Datasets

The majority of research evaluates compressed LLMs on well-established NLP benchmarks and datasets. For instance, WikiText-2, C4, and PTB are designed for evaluating the perplexity performance of language models. LAMBADA, PIQA, and OpenBookQA are designed to evaluate the zero-shot ability of language models. GSM8K, CommonsenseQA and StrategyQA are designed to evaluate the reasoning ability of language models.

BIG-Bench

BIG-Bench (BBH) is a benchmark suite designed for LLMs, covering over 200 NLP tasks, e.g., Text Comprehension Tasks, Inference Tasks, Mathematical Reasoning Tasks. The aim of BBH is to evaluate the performance of LLMs across these various complex tasks. The compressed LLMs use BBH to measure their capability across a multidimensional spectrum of tasks.

Unseen Instructions Datasets

Unseen instructions datasets are used to evaluate the performance of LLMs on unseen tasks. For instance, the Vicuna-Instructions dataset created by GPT-4 includes 80 complex questions across nine different categories like generic, knowledge-based, and writing tasks. Another dataset, User-Oriented-Instructions, consists of 252 carefully selected instructions inspired by various user-focused applications such as Grammarly, StackOverflow, and Overleaf. These datasets evaluate how well compact LLMs can handle and carry out new tasks by presenting them with unfamiliar instructions.

EleutherAI LM Harness

The EleutherAI LM Harness is an advanced framework for evaluating LLMs, providing a unified testing platform that supports over 60 standard academic benchmarks along with hundreds of subtasks and variants. The standardized evaluation tasks provided by the harness ensure the reproducibility and comparability of evaluation, which is essential for implementing fair and reproducible evaluations for the compressed LLMs.

Quantization

Quantization refers to the process of reducing the number of bits (i.e., precision) in the parameters of the model with minimal loss in inference performance. Quantization can be categorized into two main approaches: Quantization-Aware Training (QAT), and Post-Training Quantization (PTQ). The primary distinction between

the two approaches lies in whether retraining is needed during quantization. PTQ enables direct use of quantized models in inference, while QAT requires retraining to rectify errors introduced by quantization.

Quantization-Aware Training

QAT involves retraining a quantized model to counteract performance degradation caused by quantization. For instance, LLM-QAT implements the standard QAT framework directly onto LLMs. LLM-QAT distills knowledge by generating data from the LLM itself, and train the quantized LLM to align with the output distribution of the original LLM based on the generated data. BitDistiller merges QAT with self-distillation, enhancing LLM performance at sub-4-bit precisions. It employs tailored asymmetric quantization, clipping, and a Confidence-Aware Kullback-Leibler Divergence objective for faster convergence and superior results. OneBit introduces a novel 1-bit parameter representation method and an effective parameter initialization method to implement 1-bit quantization for LLM weight matrices, paving the way for the extremely low bit-width deployment of LLMs.

Remark: While QAT can mitigate quantization's accuracy degradation, retraining demands a lot of effort due to tens or hundreds of billions of parameters in LLMs. A practical solution is to incorporate Parameter-Efficient Fine-Tuning (PEFT) into the retraining process of QAT. Currently, methods like QLORA, PEQA and LoftQ combine quantization with PEFT for model fine-tuning efficiency. However, these methods are typically task-dependent. L4Q makes a preliminary attempt to enhance generality by leveraging LoRA-wise learned quantization step size for LLMs. We think that introducing PEFT to enhance QAT efficiency is not only feasible but also holds significant promise, warranting thorough exploration.

Post-Training Quantization

PTQ efficiently converts a full-precision LLM to low-precision without retraining, saving memory and computational costs. We categorize PTQ for LLMs into three groups: Weight-Only Quantization, Weight-Activation Quantization, and KV Cache Quantization. The disparity between these groups lies in their quantization objectives. Weight-only quantization focuses solely on quantizing weights, whereas weight-activation quantization extends its objective to both weights and activations. Prior research indicates that activation quantization is typically more sensitive to weight quantization, allowing weight-only quantization to achieve lower bit-width. However, since quantized weights necessitate dequantization before multiplication with activations, weight-only quantization inevitably introduces additional computational overhead during inference and cannot enjoy the accelerated low-bit operation supported by specific hardware. Furthermore, kv cache quantization targets the KV cache, which stores keys and values of attention layers. The KV cache often consumes lots of memory, acting as a bottleneck for input streams containing lengthy tokens. By implementing kv cache quantization, it is possible to increase throughput and accommodate inputs with longer tokens more efficiently.

Weight-Only Quantization

Weight-only quantization is the most conventional and widespread method. For example, LUT-GEMM uses binary-coding quantization (BCQ) format, which factorizes the parameters of LLMs into binary parameters and a set of scaling factors, to accelerate quantized matrix multiplications in weight-only quantization. GPTQ proposes a layer-wise quantization method based on Optimal Brain Quantization (OBQ), which updates weights with inverse Hessian information, and quantizes LLMs into 3/4-bit. QuIP optimally adjusts weights by utilizing the LDL decomposition of the Hessian matrix derived from vectors drawn uniformly at random from a calibration set, and multiplies weight and Hessian matrices with a Kronecker product of random orthogonal matrices to ensure incoherence between weight and Hessian matrices. Combining these two steps, QuIP successfully quantizes LLMs into 2-bits with minimal performance loss.

To further minimize quantization errors in the weight-only quantization of LLMs, lots of works identify sensitive weights, which have an important effect on LLMs' quantization performance, and store these sensitive weights in high precision. For example, AWQ stores the top 1% of weights that have the most significant impact on LLM performance in high-precision, and integrates a per-channel scaling method to identify optimal scaling factors. Here, "channel" denotes individual dimensions or feature maps within the model. Similar with AWQ, OWQ store weights sensitive to activation outliers in high-precision, and quantizes other non-sensitive weights. Different from OWQ, SpQR employs the L2 error between the original and quantized predictions as a weight sensitivity metric. Furthermore, SqueezeLLM introduces a weights clusters algorithm based on sensitivity, using k-means centroids as quantized weight values, to identify sensitive weights. The sensitivity is approximated by the Hessian matrix of weights. Then, SqueezeLLM stores sensitive weights in an efficient sparse format, and quantize other weights. SqueezeLLM quantizes LLMs in 3-bit, and achieves a more than 2x speedup compared to the FP16 baseline.

Weight-Activation Quantization

Alongside works centered on weight-only quantization in LLMs, there is a plethora of research focusing primarily on weight-activation quantization in LLMs. For example, ZeroQuant is the first work to implement weight-activation quantization for LLMs, which uses group-wise quantization for weight and token-wise quantization for activations, and reduces the precision for weights and activations of LLMs to INT8.

LLMs have outliers in activations, and the performance of LLMs declines a lot, if these activations with outliers are directly quantized. Recent works try to treat these outliers specially to reduce quantization errors in weight-activation quantization. For example, LLM.int8() stores these outlier feature dimensions into high-precision, and uses vector-wise quantization, which assigns separate normalization constants to each inner product within matrix multiplication, to quantize other features. LLM.int8() quantizes weights and activations of LLMs into 8-bit without any performance degradation. SmoothQuant designs a per-channel scaling transformation to smooths the activation outliers based on the discovery that different tokens have similar variations across channels of activations. RPTQ finds that the range of values varies greatly between different channels, and integrates a channel reordering method, which clusters and reorders the channels in the activation and uses the

same quantization parameters to quantize the values in each cluster, into layer normalization and linear layer weights to efficiently reduce the effect of numerical range differences between channels. OliVe thinks that outliers are more important than the normal values, and uses an outlier-victim pair (OVP) quantization to handle outlier values locally with low hardware overheads and significant performance benefits. OS+ further finds that outliers are concentrated in specific and asymmetric channels. Based on the findings, OS+ incorporates channel-wise shifting to eliminate the impact of asymmetry and channel-wise scaling to balance the distribution of outliers. LLM-FP4 uses floating-point formats (specifically FP8 and FP4) to address the limitations of traditional integer quantization (such as INT8 and INT4) to deal with outliers. Furthermore, LLM-FP4 points out that exponent bits and clipping range are important factors that effect the performance of FP quantization, and introduces a search-based framework for determining the optimal exponent bias and maximal quantization value. OmniQuant handles the activation outliers by equivalently shifting the challenge of quantization from activations to weights, and optimizes the clipping threshold to adjust the extreme values of the weights.

KV Cache Quantization

With the increasing number of input tokens supported by LLMs, the memory usage of the KV cache also increases. Recent efforts begin to focus on kv cache quantization to reduce the memory footprint of LLMs and accelerate their inference. For example, KVQuant proposes several KV Cache Quantization methods, such as Per-Channel Key Quantization, PreRoPE Key Quantization, and Non-Uniform kv cache quantization, to implement 10 million context length LLM inference. Through an in-depth analysis of the element distribution within the KV cache, KIVI finds that key caches should be quantized per-channel, while value caches should be quantized per-token. Finally, KIVI succeeds in quantizing the KV cache to 2 bits without fine-tuning. WKVQuant presents an innovative approach for quantizing large language models (LLMs) by integrating past-only quantization to refine attention computations, employing a two-dimensional quantization strategy to manage the distribution of key/value (KV) caches effectively, and utilizing cross-block reconstruction regularization for optimizing parameters. This method enables the quantization of both weights and KV caches, resulting in memory savings that rival those of weight-activation quantization, while nearly matching the performance levels of weight-only quantization.

Pruning

Pruning is a powerful technique to reduce the size or complexity of a model by removing redundant components. Pruning can be divided into Unstructured Pruning, Semi-Structured Pruning, and Structured Pruning. Structured pruning removes entire components like neurons, attention heads, or layers based on specific rules while preserving the overall network structure. On the other hand, unstructured pruning prunes individual parameters, resulting in an irregular sparse structure. Semi-structured pruning is a method that lies between structured pruning and unstructured pruning, capable of achieving fine-grained pruning and structural regularization simultaneously. It prunes partial parameters based on specific patterns rather than entire channels, filters, or neurons, making it a fine-grained form of structured pruning.

Unstructured Pruning

Unstructured pruning preserves the pruned model's performance, hence, works related to unstructured pruning of LLMs often dispense with retraining to restore performance. Nevertheless, unstructured pruning renders the pruned model irregular, necessitating specialized handling or software optimizations for inference acceleration. An innovative approach in this domain is SparseGPT, which introduces a one-shot pruning strategy without retraining. SparseGPT frames pruning as an extensive sparse regression problem and solves it using an approximate sparse regression solver. SparseGPT achieves significant unstructured sparsity, even up to over 50% on the largest GPT models like OPT-175B and BLOOM-176B, with minimal increase in perplexity. To reduce the cost about the weight update process required by SparseGPT, Wanda achieves model sparsity by pruning weights with the smallest magnitudes multiplied by the norm of the corresponding input activations, without the need for retraining or weight updates. To further minimize pruning-induced errors while upholding the desired overall sparsity level, SAMSP utilizes the Hessian matrix as a metric for weight matrix sensitivity evaluation, and dynamically adjusts sparsity allocation based on sensitivity. Furthermore, DSnoT minimizes the reconstruction error between dense and sparse models through iterative weight pruning-and-growing on top of sparse LLMs to enhance LLM performance across various sparsity rates, especially at high sparsity levels. To provide hardware support for handling unstructured pruning on the GPU Tensor Core hardware, Flash-LLM introduces an unstructured sparse matrix multiplication method, which loads weight matrices in a sparse format from global memory and reconstructs them in a dense format within high-speed on-chip buffers for computation using tensor cores.

Structured Pruning

Compared to unstructured pruning, structured pruning offers the advantage of being hardware-agnostic, allowing for accelerated inference on traditional hardware post-pruning. However, the removal of larger and potentially more critical components in structured pruning may result in performance degradation, typically requiring efficient parameter fine-tuning for recovery. We divide LLMs structured pruning works into several groups based on pruning metrics: Loss-based Pruning, Magnitude-based Pruning, Regularization-based Pruning.

Loss-based Pruning assesses the significance of a pruning unit by measuring its impact on loss or gradient information (e.g., first-order or second-order derivatives of loss). For example, LLM-Pruner introduces a one-shot structured pruning on LLMs based on gradient information. Specifically, LLM-Pruner identifies dependent structures via a dependency detection algorithm and selects optimal pruning groups using gradient information, rather than solely relying on loss changes, in a task-agnostic manner. Different from LLM-Pruner, which focuses on narrowing LLMs' width, Shortened LLaMA introduces a one-shot depth pruning on LLMs. Shortened LLaMA chooses the Transformer block as the prunable unit, and prunes these unimportant Transformer blocks, where the importance of Transformer blocks is evaluated by loss and its second-order

derivative. After Pruning, both LLM-Pruner and Shortened LLaMA utilize LoRA to rapidly recover the performance of the pruned model.

Magnitude-based Pruning involves devising a heuristic metric based on the magnitudes of pruning units, and use the metric to assess the importance of pruning units, subsequently pruning those units whose scores fall below a predefined threshold. For example, FLAP utilizes a structured fluctuation metric to assess and identify columns in the weight matrix suitable for pruning, measuring the variation of each input feature relative to a baseline value to estimate the impact of removing a column of weights. Additionally, FLAP uses an adaptive structure search to optimize global model compression, and restores the model's performance post-pruning through a baseline bias compensation mechanism, avoiding the need for fine-tuning. To further maintain the pruned model's performance, SliceGPT leverages the computational invariance of transformer networks and optimizes the pruning process through Principal Component Analysis (PCA). Specifically, SliceGPT employs PCA as the pruning metric, applying it at each layer of the transformer network to project the signal matrix onto its principal components and eliminate insignificant columns or rows from the transformed weight matrices, ultimately aiming to compress the model effectively.

Regularization-based Pruning typically adds a regularization term (e.g., L0, L1, and L2 regularization) into the loss function to induce sparsity for LLMs. For example, Sheared LLaMA uses a pair of Lagrange multipliers based on pruning masks to impose constraints on the pruned model shape directly, thereby formulating pruning as a constrained optimization problem. Through solving this optimization problem, Sheared LLaMA derives optimal pruning masks. Additionally, Sheared LLaMA introduces dynamic batch loading, a strategy that adapts training data loading based on each domain's loss reduction rate, enhancing the efficiency of data utilization during training.

Remark: Structured pruning typically reduces model size by removing redundant parameters, but it may degrade model performance. A novel approach is to combine knowledge distillation with structured pruning. Knowledge distillation allows knowledge extracted from a LLM to be transferred to a smaller model, helping the smaller model maintain its performance while reducing its size.

Semi-Structured Pruning

Apart from unstructured pruning and structured pruning, there are many works which use semi-structured pruning to prune partial weights of LLMs based on specific patterns. N:M sparsity, where every M contiguous elements leave N non-zero elements, is an example of semi-structured pruning. For example, E-Sparse implements N:M sparsity by introducing information entropy as a metric for evaluating parameter importance to enhances the significance of parameter weights and input feature norms. E-Sparse incorporates global naive shuffle and local block shuffle to efficiently optimize information distribution and mitigate the impact of N:M sparsity on LLM accuracy. Furthermore, many pruning works can also be generalized to semi-structured patterns. For example, SparseGPT and Wanda also explore N:M sparsity of LLMs. SparseGPT employs block-wise weight partitioning, with each block containing M weights. It identifies and prunes N weights with the

lowest reconstruction error(based on Hessian information), ensuring a sparsity ratio of N:M. This process iteratively prunes and updates model weights, addressing one block at a time until the desired sparsity level is achieved across the entire model. Wanda achieves structured N:M pruning by dividing the weight matrix into groups of M consecutive weights and computing an importance score for each weight. The score is determined by the product of the weight's magnitude and the norm of the corresponding input activations. Within each weight group, the N weights with the highest scores are retained, while the rest are set to zero, thereby implementing structured N:M pruning. Furthermore, choosing the optimal pruning strategy is crucial for compatibility with the target hardware. For instance, the Ampere Tensor Core GPU architecture (e.g., A100 GPUs) proposes 2:4 fine-grained semi-structured sparsity to accelerate Sparse Neural Networks on this hardware. However, the current implementation of the Ampere architecture supports only the 2:4 ratio, leaving other ratios without acceleration.

Remark: LLMs often perform well on multiple tasks, which means they contain a multitude of parameters for various tasks. Dynamic pruning methods can dynamically prune different parts of the model based on the current task's requirements to provide better performance on specific tasks. This helps strike a balance between performance and efficiency.

Remark: For PTQ and pruning, preparing a high-quality calibration dataset to assist in improving the performance of compressed LLMs is crucial. Specifically, extensive empirical studies on the effect of calibration data upon model compression methods find that the performance of downstream tasks can vary significantly depending on the calibration data selected. High-quality calibration data can improve the performance and accuracy of the compressed model, so careful selection and preparation of calibration data are necessary.

Knowledge Distillation

Knowledge Distillation (KD) is a technique aimed at transferring knowledge from a large and complex model (i.e., teacher model) to a smaller and simpler model (i.e., student model). We classify these methods into two clear categories: Black-box KD, where only the teacher's outputs are accessible, typically from closed-source LLMs, and White-box KD, where the teacher's parameters or output distribution are available, usually from open-source LLMs.

Black-box KD

Black-box KD usually prompts the teacher LLM to generate a distillation dataset for fine-tune the student LM, thereby transfering capabilities from teacher LLM to the student LM. In Black-box KD, teacher LLMs such as ChatGPT (gpt-3.5-turbo) and GPT4 are typically employed, while smaller LMs (SLMs), such as GPT-2, T5, FlanT5, and CodeT5, are commonly utilized as student LMs. On the other hand, researchers find that LLMs have emergent abilities, which refers to a significant improvement in performance when the model reaches a certain scale, showcasing surprising capabilities. Lots of Black-box KD methods try to distill emergent abilities

from LLMs to student LMs, and we introduce three commonly used emergent ability distillation methods: Chain-of-Thought (CoT) Distillation, In-Context Learning (ICL) Distillation, and Instruction Following (IF) Distillation.

Chain-of-Thought Distillation

CoT prompts LLMs to generate intermediate reasoning steps, enabling them to tackle complex reasoning tasks step by step. Various works employ LLMs to prompt the generation of explanations and leverage a multi-task learning framework to bolster the reasoning capabilities of smaller models while enhancing their capacity for generating explanations. Research shows that LLMs' reasoning capability can be transferred to SLMs via knowledge distillation, but there's a trade-off between model and dataset size in reasoning ability. Works use zero-shot CoT techniques to prompt LLMs to generate diverse rationales to enrich the distillation dataset for the student models. Some distill two student models: a problem decomposer and a subproblem solver, which the problem decomposer decomposes complex problems into a sequence of subproblems, and the subproblem solver solves these subproblems step by step. Others incorporate contrastive decoding during rationale generation for teacher models and address shortcut issues by introducing a counterfactual reasoning objective during student model training. Research demonstrates that increasing task-specific capabilities through distillation may inadvertently lead to reduced performance in solving generalized problems, and focus on improving mathematical capability of student LMs via distillation. PaD prompts LLMs to generate Program-of-Thought (PoT) rationales instead of Chain-of-Thought (CoT) rationales to construct distillation dataset, and fine-tunes SLMs with the distillation dataset. Work establishes a multi-round interactive learning paradigm that enables student LMs to provide feedback to teacher LLMs during the distillation process, thereby obtaining tailored training data. Additionally, DRA introduces a self-reflection learning mechanism, allowing the student LMs to learn from their mistakes and enhance their reasoning abilities. Research finds that negative data generated from teacher LMs also has reasoning knowledge, and guides student LMs to learn knowledge from both negative samples besides positive ones.

In-Context Learning Distillation

ICL employs structured prompts with task descriptions and examples for LLMs to learn new tasks without gradient updates. Work introduces a method called in-context learning distillation, which transfers in-context learning ability from LLMs to smaller models by combining in-context learning objectives with language modeling objectives. Specifically, it trains the student model to improve its generalization across various tasks by imitating the soft label predictions of the teacher model and the hard label ground truth values. Additionally, the method incorporates two few-shot learning paradigms: Meta In-context Tuning (Meta-ICT) and Multitask In-context Tuning (Multitask-ICT). In Meta-ICT, the student model adapts to new tasks with in-context learning and guidance from the teacher. Conversely, Multitask-ICT treats all target tasks as training tasks, directly using examples from them in distillation. The outcomes show that Multitask-ICT is more effective, despite its increased computational requirements. AICD leverages the autoregressive nature of LLMs to perform meta-

teacher forcing on CoTs within the context, jointly optimizing the likelihood of all in-context CoTs, thereby distilling the capabilities of in-context learning and reasoning into smaller models.

Instruction Following Distillation

IF aims to bolster the zero-shot ability of LLMs through fine-tuning using a collection of instruction-like prompt-response pairs. For instance, Lion prompts the LLM to identify and generate the "hard" instructions, which are then utilized to enhance the student model's capabilities. LaMini-LM develops an extensive collection of 2.58 million instructions, comprising both existing and newly generated instructions, and fine-tunes a diverse array of models by using these instructions. SELF-INSTRUCT uses student LMs themselves as teachers to generate instruction following dataset, and fine-tunes students themselves with the dataset. Selective Reflection-Tuning leverages the teacher LLMs to reflect on and improve existing data, while the student LMs assess and selectively incorporate these improvements, thereby increasing data quality and compatibility with the student LMs.

Remark: Black-Box Distillation uses the teacher model's outputs as supervision, but the teacher model's outputs may not cover all possible input scenarios. Thus, understanding how to handle a student model's generalization on unknown data and how to increase data diversity is an area that requires further investigation.

White-box KD

White-box KD enables the student LM to gain a deeper understanding of the teacher LLM's internal structure and knowledge representations, often resulting in higher-level performance improvements. An representative example is MINILLM, which the first work to study distillation from the Open-source generative LLMs. MINILLM use a reverse Kullback-Leibler divergence objective, which is more suitable for KD on generative language models, to prevent the student model from overestimating the low-probability regions of the teacher distribution, and derives an effective optimization approach to learn the objective. Further, GKD explores distillation from auto-regressive models, where generative language models are a subset. GKD trains the student using self-generated outputs, incorporating teacher feedback, and allows flexibility in using different loss functions when the student cannot fully replicate the teacher's distribution. Different from the above works, which focus on learning the teacher distribution, TED proposes a task-aware layer-wise distillation method, which designs task-aware filters, which align the hidden representations of the teacher and student models at each intermediate layer, to reduce the knowledge gap between the student and teacher models.

Remark: Although white-box distillation allows student LMs to learn the knowledge of teacher LLMs more deeply compared to black-box distillation, currently, open-source LLMs perform worse than closed-source ones, limiting the improvement of student LMs performance in white-box distillation. This is one of the barren factors hindering the development of white-box distillation. A feasible solution is to distill knowledge from closed-source LLMs to open-source LLMs through black-box distillation, and then use white-box distillation to transfer knowledge from open-source LLMs to student LLMs.

Remark: White-box distillation often involves understanding and utilizing the internal structure of LLMs, such as layer connections and parameter settings. A more in-depth exploration of different network structures and interactions between layers can improve the effectiveness of white-box distillation.

Low-Rank Factorization

Low-Rank Factorization reduces a large matrix into smaller ones to save space and computational effort. For example, it decomposes a large matrix W into two small matrices U and V (i.e., $W \approx UV$), where U is $m \times k$ and V is $k \times n$, with k much smaller than m and n. Recent works try to employ low-rank factorization to compress LLMs and achieve significant success in this regard. For example, LPLR compresses weight matrices of LLMs through randomized low-rank and low-precision factorization. Specifically, LPLR approximates the column space of the matrix using random sketching techniques, quantizes these columns, and then projects the original columns onto this quantized space to create two low-rank factors stored in low-precision. ASVD finds that the activation distribution has an effect on the compression performance. To solve the problem, ASVD proposes to scale the weight matrix with a diagonal matrix that contains scaling factors corresponding to the activation distribution of the input feature channels. Moreover, ASVD assigns the most suitable compression ratio to different layers by analyzing the singular values distribution in each layer's weight matrix, ensuring minimal loss of model performance during the compression process. Furthermore, research demonstrates that the performance of LLMs can be significantly improved by applying Layer-Selective Rank Reduction (LASER) to specific layers of Transformer models. LASER involves selectively reducing the rank higher-order components of weight matrices, which is shown to improve the model's handling of rare training data and its resistance to question paraphrasing.

Challenges and Future Directions

More Advanced Methods

The research on model compression techniques for LLMs is still in its early stages. These compressed LLMs, as demonstrated in prior studies, continue to exhibit a significant performance gap when compared to their uncompressed counterparts. By delving into more advanced model compression methods tailored for LLMs, we have the potential to enhance the performance of these uncompressed LLMs.

Scaling up Model Compression Methods from Other Models

In our paper, we introduce several representative model compression methods for LLMs. However, many classic model compression methods remain prevalent in traditional small models. For example, lottery tickets and parameter sharing are widely used model compression methods in small models. These methods still hold significant potential in the era of LLMs. Future work should focus on exploring how to extend these compression methods to LLMs to achieve further compression.

LLM Inference and Deployment

The efficiency of compressed LLMs during deployment is also a significant area for exploration. This involves multiple evaluation metrics, including arithmetic intensity, memory size, and throughput. Furthermore, we can use an analytical tool, the Roofline Model, to assess the resource efficiency of compressed LLMs on specific hardware. Evaluating the deployment efficiency of compressed LLMs on specific hardware can guide researchers in selecting and analyzing the advantages and disadvantages of various model compression methods and further optimizing these methods.

The Effect of Scaling Law

The scaling law underscores the significant impact of model size, dataset size, and compute resources on the performance of LLMs. However, the scaling law presents a fundamental challenge for LLM compression, i.e., there is a trade-off between model size and performance in compressed LLMs. Delving into the mechanisms and theories underpinning the scaling law is crucial for elucidating and potentially overcoming this limitation.

AutoML for LLM Compression

Existing compression techniques have made remarkable progress, but they still heavily depend on manual design. For instance, designing appropriate student architectures for knowledge distillation requires a significant amount of human effort. To reduce this reliance on manual design, a feasible solution is to combine Automated Machine Learning (AutoML) techniques such as Meta-Learning and Neural Architecture Search (NAS) with model compression. By combining with AutoML techniques, model compression can automatically select appropriate hyperparameters and tailor architectures and scales of compressed models, thus minimizing human involvement and lowering the associated costs. Furthermore, AutoML can identify optimal model compression strategies tailored to specific task requirements, thereby further enhancing compression rates without compromising model performance.

Explainability of LLM Compression

Earlier research has raised significant concerns regarding the explainability of model compression techniques applied to Pre-trained Language Models (PLMs). Notably, these same challenges extend to LLM compression methods as well. For example, CoT-distillation can enhance SLMs' reasoning performance, yet the mechanism through which it imparts CoT ability remains unclear. This challenge underscores the importance of integrating explainability with model compression approaches for the advancement of LLM compression applications. Explainability not only clarifies the changes and trade-offs in the compression process but also enhances efficiency and accuracy. Additionally, interpretability aids in evaluating the compressed model's performance to ensure it aligns with practical requirements.

Conclusion

In the survey, we have explored model compression techniques for LLMs. Our coverage spanned compression

methods, metrics, and benchmark datasets. By diving into LLM compression, we've highlighted its challenges and opportunities. This survey aims to be a valuable reference, providing insights into the current landscape and promoting ongoing exploration of this pivotal topic.

Acknowledgments

We would like to thank the anonymous reviewers and the Action Editor for their valuable feedback and discussions. The work of Jian Li is supported partially by National Natural Science Foundation of China (No. 62106257). The work of Yong Liu is supported partially by National Natural Science Foundation of China (No. 62076234), Beijing Outstanding Young Scientist Program (No. BJJWZYJH012019100020098), the Unicom Innovation Ecological Cooperation Plan, and the CCF-Huawei Populus Grove Fund.