Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)
Survey Track

Faster and Lighter LLMs: A Survey on Current Challenges and Way Forward

Arnav Chavan[1,2] , **Raghav Magazine**[1] , **Shubham Kushwaha**[1] ,
Mérouane Debbah[3] and **Deepak Gupta**[2]
[1]Nyun AI, India
[2]Transmute AI Lab (Texmin Hub), IIT (ISM) Dhanbad, India
[3]KU 6G Research Center, Khalifa University of Science and Technology, Abu Dhabi, UAE
arnav.chavan@nyunai.com, guptadeepak2806@gmail.com

Abstract

Despite the impressive performance of LLMs, their widespread adoption faces challenges due to substantial computational and memory requirements during inference. Recent advancements in model compression and system-level optimization methods aim to enhance LLM inference. This survey offers an overview of these methods, emphasizing recent developments. Through experiments on LLaMA(/2)-7B, we evaluate various compression techniques, providing practical insights for efficient LLM deployment in a unified setting. The empirical analysis on LLaMA(/2)-7B highlights the effectiveness of these methods. Drawing from survey insights, we identify current limitations and discuss potential future directions to improve LLM inference efficiency. We release the codebase to reproduce the results presented in this paper at https://github.com/nyunAI/Faster-LLM-Survey

1 Introduction

The advent of LLMs, marked prominently by models such as GPT [Brown *et al.*, 2020] and LLaMa [Touvron *et al.*, 2023a; Touvron *et al.*, 2023b] series, has paved a new revolution in language-related tasks, ranging from text comprehension and summarization to language translation and generation. These models, often consisting of billions of parameters, have shown remarkable performance in capturing intricate patterns, fine-detailed contexts, and semantic representations in natural language. As a consequence, they have become indispensable tools in various applications, leading to advancements in various domains, including artificial intelligence, information retrieval, and human-computer interaction.

Despite their unparalleled performance, widespread adoption of LLMs is hindered by their substantial computational and memory requirements, which pose challenges for deployment in resource-constrained environments. For example, loading a LLaMa-70B model requires 140GB of VRAM excluding the memory required for model inferencing. The need for efficient deployment has led to recent research into model compression as well as system-level modification techniques tailored specifically for LLMs. These early works have identified potential ways to improve the inference efficiency of LLMs. However, the current improvements are often accompanied by significant drops in the performance of the model, and novel research directions need to be identified to find the desired solutions to this problem.

A recent survey study has provided a concise overview of the recently proposed LLM compression methods, as well as the evaluation metrics and the data used to benchmark them [Zhu *et al.*, 2023]. However, to further push the frontiers of research towards practical inference improvement for LLMs, a comprehensive study is still missing. In this survey paper, we explore existing methods that aim at making LLMs efficient through model compression as well as through system-level optimizations. To fairly compare various methods, we provide empirical observations using different compression techniques applied to LLaMa(/2)-7B. Our evaluation includes methods that provide a practical advantage and include structured pruning, quantization, and system-level optimizations provided by different inference engines from the existing literature. We share valuable insights drawn from these experiments to present a useful and practical understanding of efficient LLMs. Additionally, we make the code and benchmarks associated with the experiments publicly available. We also examine the difficulties linked to current compression methods in both general deep learning and those specifically suggested for LLMs, and we discuss potential directions of research to overcome these problems.

Overall, the contributions of this paper are as follows.

- We offer a brief overview of the model compression domain, emphasizing essential methodologies that have made notable contributions to the field of lighter and faster LLMs.
- Complementary to model compression, system-level modifications have played an important role in speeding up the LLM inference, and we discuss these approaches as well.
- To provide a practical perspective, we present an empirical analysis of well-known compression methods for LLMs under a standardized setup. The insights derived can help make informed decisions about the selection of LLM compression methods based on the deployment environment.
- Drawing upon insights derived from our survey and em-

pirical analysis, we systematically pinpoint existing limitations and propose viable pathways forward for achieving optimal efficiency in LLM inference.

2 Model Compression: An Overview

Model compression techniques have emerged as a crucial area of research, offering promising solutions to enhance the efficiency of resource-intensive deep learning models. The domain of developing efficient Large Language Models (LLMs) can significantly benefit from insights and methodologies used in this field. Before diving into the topic of building efficient LLMs and the existing works around it, we provide an overview of some of the popular approaches employed in deep learning model compression. Below, we first introduce the traditional approaches of model compression and briefly discuss the development related to the traditional deep learning models. Following this, we provide an overview of the works related to compression of LLMs in the existing literature.

2.1 Compression of Deep Models

Architecture pruning refers to the process of systematically reducing the complexity of a neural network by eliminating redundant or less impactful connections, neurons, or entire layers [Janowsky, 1989]. This technique aims to enhance model efficiency, reduce computational costs, and mitigate overfitting without significantly compromising performance. Pruning involves identifying and removing connections or units based on various criteria, such as weight magnitudes [Li *et al.*, 2016], activation patterns [Molchanov *et al.*, 2016], or sensitivity analysis [Sanh *et al.*, 2020]. The pruned model retains its critical features while achieving a more compact representation, which is particularly valuable in scenarios with limited computational resources, such as edge devices or mobile applications.

Among the widely studied pruning methodologies, the lottery ticket hypothesis [Frankle and Carbin, 2019] provided fundamental insights into the impact of weight initialization and pruned network structure on neural network pruning. Network Slimming [Liu *et al.*, 2017; Chavan *et al.*, 2022] introduced a method to prune channels in CNNs and reduce the size of weight dimensions in Transformers by imposing sparsity regularization on the channel scaling factor. Movement pruning demonstrated large-scale pruning of BERT [Kenton and Toutanova, 2019] models by leveraging the first-order information i.e. retain weights moving away from zero, as compared to zero-order methods which retain weights with larger magnitudes. [Lagunas *et al.*, 2021] introduced block structures in weight matrices of transformer layers and employed movement pruning on them for practical speedups. More recently, [Jiang *et al.*, 2023a] argued that fine-tuning is redundant for first-order pruning and proposed Static Model Pruning (SMP), a fine-tuning free pruning method for language models.

Quantization reduces the precision of numerical values in a neural network, typically from 32-bit floating-point numbers to lower bit-width representations, such as 8-bit integers thus shrinking the memory footprint of the model, accelerating inference speed, and enabling more efficient deployment on hardware with limited computational resources. During quantization, weights and/or activations are rounded off to a discrete set of values, introducing a trade-off between computational efficiency and model accuracy. Even with this reduction in precision, state-of-the-art quantization methods are capable of minimizing the impact on performance.

Quantization-Aware Training (QAT) [Ni *et al.*, 2020] involves the quantization of model parameters throughout the training process, encompassing both the forward pass and backward propagation. LSQ [Esser *et al.*, 2019] proposed a learnable step size for each weight in conjunction with other network parameters. [Tailor *et al.*, 2021] introduced an architecture agnostic method for pruning graph neural networks. On the other hand, Post Training Quantization (PTQ) [Banner *et al.*, 2019] finds out the optimal clipping range and channel-bit-width settings for weights and activations. OSME [Choukroun *et al.*, 2019] proposed a PTQ method in which $l2$-distance between the quantized tensor and the corresponding floating-point tensor is minimized.

Knowledge distillation aims at training a computationally efficient model, often referred to as the student model, to mimic the predictions of a larger and more complex model known as the teacher model. This process involves transferring the knowledge embedded in the teacher model, typically characterized by its soft probabilities or intermediate representations, to the student model. Distillation is particularly useful when deploying models in scenarios with limited computational resources, as it enables the creation of smaller models that retain the performance of their larger counterparts. Additionally, distillation helps combat issues such as over-fitting, improves generalization, and facilitates the transfer of knowledge learned by deep models to simpler ones.

Knowledge distillation techniques can be divided into three classes i.e. response-based, feature-based and instance-relation based. Response-based distillation [Hinton *et al.*, 2015] trains the student model to mimic the final outputs of the teacher, while feature-based distillation [Tian *et al.*, 2022] trains the student to mimic intermediate feature maps of the teacher. Relation-based distillation takes one step further by using an objective which models the co-relation of the similarity between various feature maps of the student and teacher network. More recently, [Chen *et al.*, 2023b] used knowledge distillation during the pre-training stage and reduced the size of BERT by 40%, making it 60% faster while retaining 97% of its language understanding abilities.

Low-rank decomposition reduces the computational complexity of models by decomposing weight matrices into smaller ones with fewer dimensions which in turn approximate the initial full-rank matrix. This also reduces the number of parameters in the model and speeds up matrix multiplications hence reducing memory and latency requirements.

[Jaderberg *et al.*, 2014] proposed an architecture agnostic method of accelerating convolutional layers using tensor decomposition and discriminative fine-tuning; whereas [Denton *et al.*, 2014] proposed clustering methods with low-rank factorization for faster CNNs. [Sainath *et al.*, 2013] examined low-rank matrix factorization in acoustic models, where the factorization was applied on the final layer of the network. [Lebedev *et al.*, 2015] introduced canonical polyadic decom-

position which is calculated using non-linear least squares for speeding up CNNs. [Tai *et al.*, 2016] proposed a global decomposition optimization algorithm and thus performed better than iterative methods.

2.2 Compression of LLMs

The compression of LLMs represents a distinctive challenge compared to traditional deep learning models, primarily due to the substantial scale of the former. Many established compression methods rely on the paradigm of executing fine-tuning steps to regain lost performance during the compression stage. However, this approach encounters significant limitations when applied to LLMs owing to their considerable size, necessitating a paradigm shift in the treatment of LLM compression as an independent and new research domain.

Architecture pruning. LLM-Pruner [Ma *et al.*, 2023] used Taylor series expansion by leveraging a single gradient step to estimate important parts of a pre-trained LLM. LoRAPrune [Zhang *et al.*, 2023] outperformed LLM-Pruner by using gradients of LoRA [Hu *et al.*, 2021] weights, offering computational efficiency. LoRAShear [Chen *et al.*, 2023a] identified dependencies in LLMs, separated trainable variables into groups, and achieved compression through pruning and fine-tuning. Sheared LLaMA [Xia *et al.*, 2023] introduced targeted structured pruning and dynamic batch loading for end-to-end component removal. FLaP [An *et al.*, 2023] a fine-tuning free structured pruning method which used a fluctuation based metric to determine the importance score of various weight columns.

Unstructured pruning methods, such as SparseGPT [Frantar and Alistarh, 2023], adopted a one-shot technique without the need for fine-tuning. WANDA [Sun *et al.*, 2023] pruned weights based on the product of weight values and activation inputs, eliminating the need for fine-tuning. Another recent work suggested fusing of OBS [Hassibi *et al.*, 1993] and OBD [LeCun *et al.*, 1989] criteria for weight selection and determining layer sparsity based on sensitivities derived from Hessian matrices [S. *et al.*, 2023]. While the structured and unstructured methods mentioned above show promise, the observed performance drop for the achieved compression level remains relatively high. Further efforts are required in developing pruning methods that can lead to efficient LLMs.

Quantization. This class of methods has been relatively more successful in the compression of LLMs. LLM.`int8()` [Dettmers *et al.*, 2022] made it possible to convert the higher bit LLM weights into 8-bit without deterioration in performance post-training. They proposed a two-stage quantization scheme with vector-wise quantization and mixed-precision decomposition for outliers. SmoothQuant [Xiao *et al.*, 2023], a training-free PTQ method, reduced both weights and activations of LLMs to 8 bits. QLoRA [Dettmers *et al.*, 2023] introduced 4-bit NormalFloat (NF4) and double quantization to save memory without losing out on the performance of models. OmniQuant [Shao *et al.*, 2023] introduced Learnable Weight Clipping (LWC) and Learnable Equivalent Transformation(LET). LWC prevents weights from attaining extreme values by optimizing the clipping threshold, while LET deals with activation outliers by quantizing weights instead of activations through a LET. SqueezeLLM [Kim *et al.*, 2023] enabled compression up to 3-bit by using a sensitivity-based non-uniform quantization scheme, where second-order information is used to find the optimal bit precision. GPTQ [Frantar *et al.*, 2023] used second-order information to compress models with up to 175 billion parameters to as low as 3 bits per weight with minimal loss in accuracy, pushing the previously proposed 8-bit methods to a smaller size. [Lin *et al.*, 2023] observed that retaining 1% crucial weights can help reduce the degradation in quantization performance. They proposed Activation-aware Weight Quantization (AWQ) which finds the best channel-wise scaling, outperforming existing techniques in general language modeling and domain-specific tasks. ZeroQuant-FP [Wu *et al.*, 2023] focused on floating point quantization and found that FP8 outperforms INT8 for activations and FP4 is comparable to INT4 for weights. They also incorporated low-rank compensation into their approach for enhancement. EXL2[1] proposed a mixed-precision quantization algorithm, where different precision types for each layer are computed while measuring quantization errors. Their algorithm saves all the tries and associated error rates in the measuring pass and given a target precision, the algorithm quantizes the model by choosing, for each layer's module, a target precision with the lowest error rate. GGUF/GGML[2] proposed a mixed set of quantizations to achieve K-Quants, a *mostly K* quantization output. For example, 4bit K-Quant uses 6bit for a few of the Attention and MLP layers and the usual 4bit for others.

LLM-QAT [Liu *et al.*, 2023] proposed a data-free distillation method where they queried a pre-trained model to generate data which was used to train a quantized student model using a distillation loss. With the quantization of the KV-cache as well, apart from weights and activations, they can quantize 7B, 13B, and 30M LLaMA down to 4 bits. BitNet [Wang *et al.*, 2023a] introduced a 1-bit LLM transformer architecture. It mainly replaces the standard `nn.Linear` in PyTorch with `BitLinear` to train 1-bit weights. As the size of the models increases, it comprehensively outperforms counterparts trained on FP16. [Tao *et al.*, 2022] proposed token-level contrastive distillation and used dynamic scaling to make quantizers adaptive to different modules.

Knowledge distillation. Among the knowledge distillation methods, both white-box as well as black-box methods have been used to compress large open-source language models. Instead of solely relying on a fixed set of output sequences, Generalized KD [Agarwal *et al.*, 2023] trains the student on its self-generated output sequences by leveraging feedback from the teacher on such sequences. TED [Liang *et al.*, 2023] employs a dual-stage training process. In the first stage, task-specific loss trains filters in both student and teacher models. In the second stage, the student and its filters undergo training with a task-aware layer-wise distillation loss, alongside student-teacher and task-specific losses. In another work [Jha *et al.*, 2023], the student model is initialized with a subset of layers of the teacher and trained on the same corpus and objective as the teacher. This helps to achieve task-agnostic compression without using any distillation loss.

[1]https://github.com/turboderp/exllamav2

[2]https://github.com/ggerganov/ggml

Other distillation methods include black-box techniques such as Lion [Jiang *et al.*, 2023b], where the student network is trained using a three-stage adversarial loop consisting of an imitation, discrimination, and generation stage. In its discrimination stage, a propriety LLM is used to find hard instructions, i.e. instructions for which the student's outputs significantly differ from the teacher's outputs. As a final step, the propriety LLM generates more samples similar to the hard instructions on which the student is trained to complete the loop. DISCO [Chen *et al.*, 2023b] is a counterfactual knowledge approach in which a propriety LLM is given a prompt and is made to generate counterfactual augmentations in it. Then a task-specific teacher model filters out these augmentations, and the student model is trained on them. SCOTT [Wang *et al.*, 2023b] used contrastive decoding to generate rationale from the teacher along with the usual question-answer pair to train the student model.

Low rank approximations. TensorGPT [Xu *et al.*, 2023] compressed the embedding layer of LLMs through Tensor-Train Decomposition and stored it in a reduced Matrix Product State, which can be computed in a distributed fashion. LoSparse [Li *et al.*, 2023] approximated weight matrix in LLMs as the sum of a sparse matrix and another low-rank approximation matrix. The low-rank matrices capture the expressive features among neurons as they involve doing Singular Value Decomposition and the remaining features are captured by the sparse matrix. [Kaushal *et al.*, 2023] show that a simple decomposition of the matrices in LLMs as a product of two sparse low-rank matrices can offer noticeable compression and speedup at a small compromise of perplexity.

Overall, the research direction of using low-rank approximations to compress LLMs is new but exhibits the potential to improve inference efficiency. Two recent works have shown that low-rank approximations can often improve reasoning abilities and undergo compression through layerwise rank reduction in the weight space [Sharma *et al.*, 2023] and/or in the latent feature space [Chavan *et al.*, 2023]. These methods offer the advantage of requiring minimal computational resources for the compression process due to their layerwise approach to matrices involved. However, it should be noted that the level of lossless compression achieved using these techniques remains modest, and further improvements are needed from a practical point of view.

System level approaches. Here we highlight those methods which improve the complementary infrastructure and runtime architecture of LLMs.

Paged Attention [Kwon et al.*, 2023]* - inspired by the classical virtual memory and paging techniques in operating systems, it allows storage of continuous keys and values cached in non-contiguous memory.

Tensor Parallelism - entails dividing a tensor into shards distributed across various GPUs, processing each shard independently and in parallel, and subsequently synchronizing the results at the end of the step.

Pipeline Parallelism - allows a model to be vertically split across multiple GPUs at the layer level, where each GPU handles one or several layers, enabling parallel processing of distinct stages in the pipeline.

CPU/GPU Offloading [Song et al.*, 2023]*- involves transferring specific weight layers to GPU devices for matrix multiplication, subsequently transmitting the computed results back to the secondary device (RAM), thus optimizing parallel processing capabilities while allowing the secondary device to handle the remaining memory intensive computations.

Flash Attention(/v2) [Dao et al.*, 2022; Dao, 2023]* - optimizes attention computation by employing incremental softmax reduction through input block tiling, avoiding the need for whole-input access, and expedites the backward pass by storing the softmax normalization factor from the forward pass, eliminating the requirement to read the large attention matrix from high bandwidth memory (HBM). FlashAttention-2 minimizes non-matrix multiplication FLOPs, optimizing the online softmax technique, introducing parallelism over sequence length, and refining workload partitioning among warps within each thread block to reduce synchronization.

Fused Operations - involves consolidating multiple computational tasks, such as combining existing kernels or creating new ones, to minimize the overhead associated with multiple kernel API invocations.

Speculative Decoding [Leviathan et al.*, 2023]*- efficiently generates multiple future tokens from a chosen smaller model and verifies them in parallel using the larger model, enabling the simultaneous decoding of multiple tokens per step.

Notable implementations in this category include vLLM[3][Kwon *et al.*, 2023], Llama.cpp[4], ExLlama(/v2), TensorRT-LLM[5], MLC-LLM[6], PowerInfer[7] [Song *et al.*, 2023], among others. vLLM employs paged attention through a KV-Cache manager that separates logical and physical KV blocks, enabling dynamic growth of the KV cache. ExLlama(/v2) implements fused kernels to minimize launch overheads and API invocation overheads when operating on discontinuous blocks. Llama.cpp is a low-level C/C++ implementation of the LLaMA architecture with support for multiple BLAS backends for fast processing. It operates on the GGUF quantization scheme with CPU and GPU offloading. MLC-LLM focuses on compiler accelerations and runtime optimizations for native deployment across platforms. It encapsulates model execution logic in a container - Intermediate Representation Module (IRModule) which captures the hierarchical structure of computations for optimization and code generation. It employs Paged Attention, Fused Operators, and automatic generation of optimized kernel code for multiple hardware platforms. TensorRT-LLM implements masked multi-head attention with on-the-fly pre-processing of QKV elements. It supports Paged Attention, INT8/FP8 caches, in-flight batching, and tensor/pipeline parallelism for speedups. An additional improvement is attained due to fused in-flight batching with operation fusion. PowerInfer adopts a GPU-CPU hybrid approach, by pre-loading consistently activated hot neurons onto the GPU for fast access, computing variable cold

[3]https://github.com/vllm-project/vllm
[4]https://github.com/ggerganov/llama.cpp
[5]https://github.com/NVIDIA/TensorRT-LLM
[6]https://github.com/mlc-ai/mlc-llm
[7]https://github.com/SJTU-IPADS/PowerInfer

neurons on the CPU, and integrating adaptive predictors and neuron-aware sparse operators to optimize efficiency.

Overall, these methods work complementary to model compression methods and improve the runtime efficiency of large language models. These engines demonstrate the feasibility and benefits of optimizing the software architecture and infrastructure complementary to model compression.

3 Experimental Analysis

As discussed above, there exist several approaches for model compression, and there is no clear consensus on which method to use when or which method is superior over the others. Thus, we present here an experimental analysis of the different LLM compression methods and present important insights. For all the experiments, we provide practical inference metrics including model weight memory (WM), runtime memory consumption (RM), inference token rate and WikiText2 perplexity computed on a Nvidia A100 40GB GPU.

Method	Sparsity	RM (GB)	WM (GB)	Tokens/s	Perplexity
Baseline	-	26.16	12.55	30.90	12.62
Wanda-SP	20%	-	-	-	22.12
	50%	-	-	-	366.43
LLM-Pruner	20%	10.38	10.09	32.57	19.77
	50%	6.54	6.23	40.95	112.44
LLM-Pruner*	20%	10.38	10.09	32.57	17.37
	50%	6.54	6.23	40.95	38.12
FLaP	20%	9.72	9.44	33.90	14.62
	50%	6.26	6.07	42.88	31.80

Table 1: Performance measures for various compressed variants of LLaMA-7B model obtained using the following structured pruning methods: Wanda-SP, LLM-pruner and FLaP. Here, $*$ refers to a fine-tuned variant of LLM-pruner.

Pruning of LLaMA-7B. In this analysis, we examine the structured pruning of the LLaMA-7B model using three recent Large Language Model (LLM) pruning methods. Table 1 showcases the performance scores for these methods at sparsity levels of 20% and 50%. Notably, all compression methods exhibit effective performance in terms of perplexity at lower sparsity levels.Wanda-SP denotes Wanda adapted to structured pruning as reported in [An *et al.*, 2023]. Noticeably, Wanda-SP and LLM-Pruner impacts the model's performance and have suboptimal results at 50% sparsity. On the other hand, both FLaP and the fine-tuned variant of LLM-pruner perform well at this level. Comparing RM, WM, and Perplexity, these two methods demonstrate similar performance, with FLaP slightly outperforming the fine-tuning-based LLM-pruner. It is important to note that beyond superior performance, FLaP is also training-free, which makes it a preferred choice for LLM pruning.

Quantized LLaMA2-7B. Table 2 presents a comparative study demonstrating the efficacy of different quantization methods for improving LLM inference. For each quantization method choice, we default to Pytorch as the default inference engine and use propriety engines when Pytorch support is not available. As can be seen, the perplexity of all the models is mostly intact with only marginal degradation. As expected, lower precision leads to lower working and running memory consumption. Importantly, we see that at 4-bit, OmniQuant can maintain performance the most. However, GPTQ and AWQ have a wider support on different engines. Another interesting observation is that even though sub 4-bit quantizations lead to a drop in model performance, the resultant models are still better than those obtained from pruning at similar compression levels.

System-level optimizations for LLaMA2-7B. We also consider system-level optimization methods and improve LLM inference by employing various inference engines proposed in the existing literature. Related results are presented in Table 3. As can be seen, different methods have advantages across different performance metrics. TensorRT-LLM stands out with impressive performance across all metrics, particularly on NVIDIA GPUs. It provides the best token rate with GPTQ 4-bit quantization, however, efficient 4-bit support is only available for new hardware[8]. It can also be consistently seen that GPTQ is faster than AWQ at the same precision, however, the perplexity is slightly worse. MLC-LLM seems to demonstrate slightly lower performance compared to TensorRT-LLM, however, its compatibility with a range of hardware positions it as a favourable choice in specific scenarios.

4 Challenges and Way Forward

Large-scale pruning/distillation is computationally intensive. The strategies of architecture pruning and knowledge distillation have gained widespread popularity for compressing deep learning models. However, these techniques require several fine-tuning steps, the computational demands of which can rival or even surpass the intensity of the initial training steps. In the context of LLMs, this renders them impractical, given their already substantial computational requirements. While some efforts have been made to address this challenge, they often result in significant accuracy drops even for marginal compression gains. Possible ways to circumvent the issue could include:

- Revisiting the training-free pruning methods to explore their potential in the context of LLMs. For example, knowledge-preserving pruning, which focuses on reducing the unwanted knowledge context in a network rather than eliminating weights, can be improved and adapted for LLMs. Since such methods are mostly training-free, they could offer efficient LLMs at only a small additional computational budget.
- Exploring layerwise pruning of LLMs. A straightforward implementation of layerwise pruning would require defining localized loss functions in terms of regression loss and compressing the sub-network while ensuring that the local output is reproduced. However, in such an approach, even small errors in the early layers could easily propagate to the later layers leading to poor performance of the compressed network.
- Localized distillation of LLMs. A potential solution to overcome the issue of distillation could be to develop localized distillation methods. Instead of condensing the entire teacher LLM information into a smaller student,

[8]Ampere and newer series of GPUs support 4bit runtime

Method	Inference Engine	WM (GB)	RM (GB)	Tokens/s	Perplexity
Baseline FP16	PyTorch	12.55	26.16	30.90	5.85
GPTQ 2bit	PyTorch	2.11	2.98	20.91	NaN
GPTQ 3bit	PyTorch	2.87	3.86	21.24	7.36
GPTQ 4bit	PyTorch	3.63	4.65	21.63	6.08
GPTQ 8bit	PyTorch	6.67	7.62	21.36	5.86
AWQ 4bit GEMM	PyTorch	3.68	4.64	28.51	6.02
AWQ 4bit GEMV	PyTorch	3.68	4.64	31.81	6.02
QLoRA (NF4)	PyTorch	3.56	4.84	19.70	6.02
LLM.int8()	PyTorch	6.58	7.71	5.24	5.89
K-Quants 4bit	Llama.cpp	3.80	7.38	104.45	5.96
OmniQuant 3bit	MLC-LLM	3.20	5.10	83.4	6.65
OmniQuant 4bit	MLC-LLM	3.80	5.70	134.2	5.97

Table 2: Performance comparison of different quantization methods for the compression of LLaMA2-7B. Here, WM and RM refer to weight memory and running memory consumption, respectively.

this approach involves learning localized parts of the teacher network in smaller-scale student sub-networks. A strategy can then be devised to combine these sub-networks into a fully compressed student LLM. This approach holds promise as a potential solution to the computational challenges associated with LLM distillation.

- Growing smaller LLMs to reach the desired performance. The primary obstacle in compressing Large Language Models (LLMs) lies in the computational challenges during fine-tuning, attributed to the models' substantial size. An alternative and ambitious research direction involves growing smaller language models (SLMs) into LLMs using well-defined neural network growing strategies. This approach avoids the need to train a full-scale LLM, and the maximum computational burden is determined by the final compressed LLM obtained through the growth of the SLM.
- Using PEFT methods to fine-tune when pruning efficiency. To address the challenge of full-scale fine-tuning during pruning, an alternative approach is to employ PEFT methods. Unlike traditional methods, PEFT does not require updating the model weights; only the added masks and PEFT parameters are updated [Zhang *et al.*, 2023]. This significantly reduces the computational intensity of the fine-tuning process. However, PEFT methods currently face limitations in achieving large-scale compression of LLMs, indicating a need for further research to develop PEFT methods tailored specifically for compressing LLMs.

On-the-fly Quant-Dequant makes the inference slow. The utilization of lower-precision floating-point formats such as FP4 poses a dual challenge regarding memory efficiency and computational speed during inference. While contemporary hardware typically supports formats like FP16 and INT8, which enable substantial memory reduction, the lower precision conversions typically needs the Quantization (Quant) and Dequantization (Dequant) operations. These operations can induce computational overhead, contributing to a slowdown in the inference process compared to using higher-precision formats like FP16. Therefore, while the adoption of lower-precision formats can offer memory efficiency gains, they adversely affect the inference speed and a right balance between the two needs to be struck. A potential solution involves the development of streamlined Quant-Dequant operations, aiming to alleviate the observed overhead in inference speed. Another strategy is to tailor the choice of precision formats according to the specifications of the hardware in use. Concurrently, advancements on the hardware front are essential, necessitating support of lower precision formats to a broader range of popular hardwares.

Rank selection in the layerwise low rank approximation is hard. While low-rank approximation exhibits enormous potential for LLM compression, this approach is accompanied by a set of challenges, particularly in the determination of hyperparameters governing the rank reduction process. Deciding on a low-rank approximation strategy lacks a clear consensus for generalizing the method across different models. Moreover, the computational infeasibility of solving a system-level decomposition system adds a layer of complexity, making it challenging to achieve an optimal reduction in model size while preserving performance.

It is crucial to recognize that determining the optimal rank to retain across various layers is not easily addressed through a hyperparameter search problem. Many of these approaches are computationally expensive, particularly in the context of Large Language Models (LLMs). There is a necessity to explore and develop an effective strategy for searching for the right rank when employing low-rank approximations.

Existing evaluation metrics may not comply well. Compressing LLMs while preserving their ability to handle extensive contextual information is a challenge, and appropriate evaluation metrics need to be developed to tackle this issue. Another factor is the loss of fidelity. Aggressive compression may lead to a significant loss of model fidelity, impacting the language model's ability to generate accurate and contextually relevant outputs. Several such characteristics of LLMs need to be captured in their compressed variants, and this can only be identified by the right choice of metrics.

Python - an interpreted language leads to slower execution times. The Global Interpreter Lock (GIL) in CPython, the default Python interpreter, further restricts the concurrent ex-

Method	Hardware Support	Quantization Type	WM (GB)	RM (GB)	Tokens/sec	Perplexity
Llama.cpp	NVIDIA GPU	GGUF K-Quant 2bit	2.36	3.69	102.15	6.96
	AMD GPU	GGUF 4bit	3.56	4.88	128.97	5.96
	Apple Silicon	GGUF AWQ 4bit	3.56	4.88	129.25	5.91
	CPU	GGUF K-Quant 4bit	3.59	4.90	109.72	5.87
		GGUF 8bit	6.67	7.78	93.39	5.79
		GGUF FP16	12.55	13.22	66.81	5.79
ExLlama	NVIDIA GPU	GPTQ 4bit	3.63	5.35	77.10	6.08
	AMD GPU					
ExLlamav2	NVIDIA GPU	EXL2 2bit	2.01	5.21	153.75	20.21
	AMD GPU	EXL2 4bit	3.36	6.61	131.68	6.12
		GPTQ 4bit	3.63	6.93	151.30	6.03
		EXL2 8bit	6.37	9.47	115.81	5.76
		FP16	12.55	15.09	67.70	5.73
vLLM	NVIDIA GPU	AWQ GEMM 4bit	3.62	34.55	114.43	6.02
	AMD GPU	GPTQ 4bit	3.63	36.51	172.88	6.08
		FP16	12.55	35.92	79.74	5.85
TensorRT-LLM	NVIDIA GPU	AWQ GEMM 4bit	3.42	5.69	194.86	6.02
		GPTQ 4bit	3.60	5.88	202.16	6.08
		INT8	6.53	8.55	143.57	5.89
		FP16	12.55	14.61	83.43	5.85
TGI	AMD GPU	AWQ GEMV 4bit	3.62	36.67	106.84	6.02
	NVIDIA GPU	GPTQ 4bit	3.69	37.85	163.22	6.08
	Intel GPU	FP4	12.55	37.21	36.91	6.15
	AWS Inferentia2	NF4	12.55	37.21	36.32	6.02
		FP16	12.55	38.03	74.19	5.85
MLC-LLM	NVIDIA GPU	OmniQuant 3bit	3.2	5.1	83.4	6.65
	AMD GPU,	OmniQuant 4bit	3.8	5.7	134.2	5.97
	CPU, WebGPU,	AWQ GEMM 4bit	3.62	6.50	23.62	6.02
	Apple Silicon,	Q4F16	3.53	6.50	189.07	-
	Intel GPU,	Q3F16	2.84	5.98	185.47	-
	WASM, Adreno Mali	FP16	12.55	15.38	87.37	5.85

Table 3: Performance comparison of compressed variants of LLaMA2-7B using various inference engines, quantized for different predictions and across different hardwares. Here, WM and RM denote weight memory and running memory consumption respectively.

ecution of multiple threads, limiting the language's ability to fully exploit the potential of multi-core processors. This highlights the need to seek alternative solutions to improve the speed of deep learning workflows.

Some of these issues have been overcome with the development of optimized libraries and frameworks, such as TensorFlow and PyTorch, which incorporate high-performance kernels implemented in lower-level languages like C++ or CUDA. However, there are several other ends where Python restricts the model performance. An illustrative example is LLama.cpp, where the transition to a C++ implementation, LLaMA-7B, resulted in significantly improved speed. This shift exemplifies the impact of choosing a language optimized for performance in the context of deep learning models. Moreover, the emergence of Rust-based models has attracted attention for their superior speed. Rust, with its emphasis on both memory safety and performance, has demonstrated effectiveness in accelerating computations, particularly in scenarios where speed is of paramount importance. Thus, for optimizing the inference speed, moving away from Python to C++, Rust or other similar languages might be a future direction to pursue.

Ethical and bias considerations not necessarily maintained. LLMs are initially trained on extensive datasets, ensuring that the model remains statistically unbiased towards any specific case. However, during the process of model compression, a specific dataset is typically employed. As the LLM may lose some generic characteristics irrelevant to the target dataset, there is a potential to introduce unnoticed bias through standard evaluation practices. Consequently, there is a need to develop innovative evaluation strategies to guarantee that ethical concerns and biases are minimized in the compressed LLMs.

5 Conclusions

In conclusion, our survey extensively explores LLM compression, covering both model-level and system-level efficiency enhancements. We discuss various compression methodologies and provide practical insights from experiments conducted on LLaMA(/2)-7B, offering valuable information for optimizing LLMs. Analysis of the survey and experimental results highlights the existing bottlenecks in enhancing LLM inference, indicating the necessity for further developments to achieve efficiency. We envision this survey as a stepping stone towards advancing the field and achieving the goal of efficient LLM inference.

References

[Agarwal *et al.*, 2023] R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. Ramos, M. Geist, and O. Bachem. Generalized knowledge distillation for auto-regressive language models. *arXiv*, 2023.

[An *et al.*, 2023] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang. Fluctuation-based adaptive structured pruning for large language models. *arXiv*, 2023.

[Banner *et al.*, 2019] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv*, 2019.

[Brown *et al.*, 2020] T Brown, B Mann, N Ryder, M Subbiah, J D Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

[Chavan *et al.*, 2022] A Chavan, Z Shen, Z Liu, Z Liu, K T Cheng, and E P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *CVPR*, pages 4931–4941, 2022.

[Chavan *et al.*, 2023] A. Chavan, N. Lele, and D. Gupta. Rethinking compression: Reduced order modelling of latent features in large language models. *arXiv*, 2023.

[Chen *et al.*, 2023a] T. Chen, T. Ding, B. Yadav, I. Zharkov, and L. Liang. Lorashear: Efficient large language model structured pruning and knowledge recovery. *arXiv*, 2023.

[Chen *et al.*, 2023b] Z. Chen, Q. Gao, A. Bosselut, A. Sabharwal, and K. Richardson. Disco: Distilling counterfactuals with large language models. *arXiv*, 2023.

[Choukroun *et al.*, 2019] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev. Low-bit quantization of neural networks for efficient inference. *arXiv*, 2019.

[Dao *et al.*, 2022] T. Dao, D. Fu, S. Ermon, A. Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *NeurIPS*, 35:16344–16359, 2022.

[Dao, 2023] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv*, 2023.

[Denton *et al.*, 2014] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *arXiv*, 2014.

[Dettmers *et al.*, 2022] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv*, 2022.

[Dettmers *et al.*, 2023] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv*, 2023.

[Esser *et al.*, 2019] S. K Esser, J. L McKinstry, D. Bablani, R. Appuswamy, and D. S Modha. Learned step size quantization. *arXiv*, 2019.

[Frankle and Carbin, 2019] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv*, 2019.

[Frantar and Alistarh, 2023] E. Frantar and D. Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. *arXiv*, 2023.

[Frantar *et al.*, 2023] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv*, 2023.

[Hassibi *et al.*, 1993] B. Hassibi, D. G Stork, and G. J Wolff. Optimal brain surgeon and general network pruning. In *ICNN*, pages 293–299. IEEE, 1993.

[Hinton *et al.*, 2015] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv*, 2015.

[Hu *et al.*, 2021] E. J Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. In *ICLR*, 2021.

[Jaderberg *et al.*, 2014] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv*, 2014.

[Janowsky, 1989] S. A. Janowsky. Pruning versus clipping in neural networks. *Phys. Rev. A*, 39:6600–6603, 1989.

[Jha *et al.*, 2023] A. H. Jha, T. S., E. P. Walsh, D. Groeneveld, E. Strubell, and I. Beltagy. How to train your (compressed) large language model. *arXiv*, 2023.

[Jiang *et al.*, 2023a] T. Jiang, D. Wang, F. Zhuang, R. Xie, and F. Xia. Pruning pre-trained language models without fine-tuning. *arXiv*, 2023.

[Jiang *et al.*, 2023b] Y. Jiang, C. Chan, M. Chen, and W. Wang. Lion: Adversarial distillation of proprietary large language models. *arXiv*, 2023.

[Kaushal *et al.*, 2023] A. Kaushal, T. Vaidhya, and I. Rish. Lord: Low rank decomposition of monolingual code llms for one-shot compression. *arXiv*, 2023.

[Kenton and Toutanova, 2019] Jacob Devlin Ming-Wei Chang Kenton and L. Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[Kim *et al.*, 2023] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv*, 2023.

[Kwon *et al.*, 2023] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[Lagunas *et al.*, 2021] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush. Block pruning for faster transformers. *arXiv*, 2021.

[Lebedev *et al.*, 2015] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv*, 2015.

[LeCun *et al.*, 1989] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *NeurIPS*, 2, 1989.

[Leviathan *et al.*, 2023] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, pages 19274–19286. PMLR, 2023.

[Li *et al.*, 2016] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv*, 2016.

[Li *et al.*, 2023] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, and T. Zhao. Losparse: Structured compression of large language models based on low-rank and sparse approximation. *arXiv*, 2023.

[Liang *et al.*, 2023] C. Liang, S. Zuo, Q. Zhang, P. He, W. Chen, and T. Zhao. Less is more: Task-aware layer-wise distillation for language model compression. *arXiv*, 2023.

[Lin *et al.*, 2023] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, C. Gan, and S. Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv*, 2023.

[Liu *et al.*, 2017] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. *arXiv*, 2017.

[Liu *et al.*, 2023] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv*, 2023.

[Ma *et al.*, 2023] X. Ma, G. Fang, and X. Wang. Llm-pruner: On the structural pruning of large language models. *arXiv*, 2023.

[Molchanov *et al.*, 2016] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2016.

[Ni *et al.*, 2020] R. Ni, H. M. Chu, O. Castañeda, P. Y. Chiang, C. Studer, and T. Goldstein. Wrapnet: Neural net inference with ultra-low-resolution arithmetic. *arXiv*, 2020.

[S. *et al.*, 2023] Hang S., Bei L., and Yanmin Q. One-Shot Sensitivity-Aware Mixed Sparsity Pruning for Large Language Models. *arXiv*, 2023.

[Sainath *et al.*, 2013] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*, pages 6655–6659, 2013.

[Sanh *et al.*, 2020] V. Sanh, T. Wolf, and A. M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv*, 2020.

[Shao *et al.*, 2023] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv*, 2023.

[Sharma *et al.*, 2023] P. Sharma, J. T. Ash, and D. Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. *arXiv*, 2023.

[Song *et al.*, 2023] Y. Song, Z. Mi, H. Xie, and H. Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv*, 2023.

[Sun *et al.*, 2023] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter. A simple and effective pruning approach for large language models. *arXiv*, 2023.

[Tai *et al.*, 2016] C. Tai, T. Xiao, Y. Zhang, X. Wang, and Weinan E. Convolutional neural networks with low-rank regularization. *arXiv*, 2016.

[Tailor *et al.*, 2021] S. A. Tailor, J. F. Marques, and N. D. Lane. Degree-quant: Quantization-aware training for graph neural networks. *arXiv*, 2021.

[Tao *et al.*, 2022] C. Tao, L. Hou, W. Zhang, L. Shang, X. Jiang, Q. Liu, P. Luo, and N. Wong. Compression of generative pre-trained language models via quantization. *arXiv*, 2022.

[Tian *et al.*, 2022] Yo. Tian, D. Krishnan, and P. Isola. Contrastive representation distillation. *arXiv*, 2022.

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

[Touvron *et al.*, 2023b] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv*, 2023.

[Wang *et al.*, 2023a] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei. Bitnet: Scaling 1-bit transformers for large language models. *arXiv*, 2023.

[Wang *et al.*, 2023b] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, and X. Ren. Scott: Self-consistent chain-of-thought distillation. *arXiv*, 2023.

[Wu *et al.*, 2023] X. Wu, Z. Yao, and Y. He. Zeroquant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats. *arXiv*, 2023.

[Xia *et al.*, 2023] M. Xia, T. Gao, Z. Zeng, and D. Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv*, 2023.

[Xiao *et al.*, 2023] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv*, 2023.

[Xu *et al.*, 2023] M. Xu, Y. Lei Xu, and D. P. Mandic. Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition. *arXiv*, 2023.

[Zhang *et al.*, 2023] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, and B. Zhuang. Loraprune: Pruning meets low-rank parameter-efficient fine-tuning. *arXiv*, 2023.

[Zhu *et al.*, 2023] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. A survey on model compression for large language models. *arXiv*, 2023.