Compression Strategies for Massive Natural Language Generation Models

Abhinav Dayal[1(✉)], Jayasri Angara[2], Sumit Gupta[1], and Ravi Shankar Saripalle[3]

[1] Vishnu Institute of Technology, Bhimavaram, India
abhinav.dayal@vishnu.edu.in
[2] GITAM University, Visakhapatnam, India
[3] Gayatri Vidya Parishad College of Engineering, Visakhapatnam, India

Abstract. This paper presents a comprehensive review and proposed strategy for compressing massive Natural Language Generation (NLG) models focused on enhancing the efficiency of transformer-based models for deployment in real-time online applications. The review examines a range of compression techniques, including pruning, quantization, distillation, and knowledge distillation, explaining their roles in reducing the computational and memory requirements of these models. This reduction is crucial in mitigating deployment costs and latency problems, thus improving user experience. The unique complexities of transformer-based models, such as their sequential nature and attention mechanism, necessitate that the compression strategy applied be task-specific to be effective. The outcomes of this study serve as a valuable guide for NLG model developers wanting to optimize their models for online deployment. The effectiveness of these methods is discussed in terms of numerical performance indices such as latency reduction, memory footprint reduction, and efficiency improvement. By leveraging these strategies, researchers can develop more efficient and cost-effective NLG models.

Keywords: Natural Language Generation · Compression Strategies · Massive Natural Language Generation Models · Pruning · Quantization · Distillation · Knowledge Distillation

1 Introduction

Digital technologies have witnessed a profound evolution with the advent of Natural Language Generation (NLG) models, which have proven to embed transformative interventions in numerous text-based tasks. Representing this sphere of computational linguistics, models based on Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Generative Adversarial Networks (GAN), and particularly Transformers, have revolutionized applications ranging from summarization and question generation to essay writing and dialogue systems [34]. However, the increasing size and complexity of NLG models pose

P. Pareek et al. (Eds.): IC4S 2024, LNICST 599, pp. 65–78, 2025.
https://doi.org/10.1007/978-3-031-77081-4_7

significant challenges in terms of computational resources and deployment efficiency.

The NLG landscape is dominated by voluminous transformer-based [36] models such as GPT, which are recognized for their high-performance outcomes across various natural language generation tasks. Their intrinsic ability to be fine-tuned for specific downstream tasks escalates their appeal within the scientific community [34]. The inference velocity of these models is subjected to significant strain due to the steadily increasing number of parameters and the auto-regressive decoding dependent calculus based on the text length to be generated. This has propelled researchers to venture into a myriad of compression techniques geared towards combating the re-source constraints of transformer-based models. Pruning, quantization, distillation, and knowledge distillation have been meticulously studied for their potential to reduce memory requirements and computational footprint [3].

This paper embarks on an analytical journey to provide an extensive overview of these inventive compression techniques. Moreover, it aims to present a practical, result-oriented strategy for the implementation of these techniques on novel or existing NLG tasks. The grand vision is to have transformer models that are not only efficient but also cost-effective, paving the way for their robust deployment in real-time online applications. Table 1 presents an overview of the methods for LLM compression that will be discussed in this paper.

2 LLM Compression Methods

In the field of Large Language Models, several compression methods have been proposed to address the challenges posed by their immense size and resource requirements [33]. Researchers presented a survey of compression techniques like pruning, quantization, knowledge distillation, etc. to improve performance in DNNs that either reduce the model parameters size, and/or the resource demands to allow for efficient parallel computation and deployment on limited resource platforms [26] [24] [4]. Recent works have applied these compression algorithms to transformer-based language models in order to improve their implementation efficiency. [32] [29] There are methods specific to transformers like attention head pruning, attention decomposition, replacing transformer blocks with an RNN or CNN, etc. Ganesh et al. compared the effectiveness of different model compression techniques on Transformer-based NLP models [13]. Gupta et al. presented a detailed survey on compression of deep learning models for text [14]. However, most of the methods presented in these extensive surveys are done for models less than 2B parameters. In this section, we present current strategies applied to Large Language Models (LLM).

2.1 Quantization

One effective approach to NLG Model Compression is Quantization, reducing the number of bits to represent model weights and activations, thereby reducing

Table 1. Quick glimpse of methods for LLM compression.

Method	Description
Pruning	– Unstructured • Effective Solution • Need special hardware/Libraries for speedup – Structured • Often needs re-training • Not tried on large models • Can be task specific
Distillation	– Teacher/Student Model Training • Need re-training • Not tried on large models • Often task specific
Quantization	– Uniform/Mixed/Precision Single/Floating Point • Post-Training effective • For large models' negligible accuracy loss • Need special hardware/Libraries for speedup
Compute/Memory Optimization	– Decomposition like BTD (Attention, Parameter Sharing) • Not tried on Large generative models • Can be explored – GPU Level Optimization Libraries • Not yet experimented with large models • Specific to hardware/software • Can be explored – Distributed non-blocking pipelines and memory pools • Experimented with large models with definite reduction in re-sources
Dynamic Inference	– Adaptative Inference + Remove Redundancy-Task specific and mostly used in classification – MoEfication • Not yet experimented with large models • Can be task specific • Can be explored
Other	– Predictive Look ahead • Need multiple smaller models to run in parallel • Speedup at cost of more computation – Word Piece Encoding • Reduces vocabulary by finetuning • Often task or domain specific • Not tried with large NLG models

memory use through the use of low-bit fixed-point arithmetic [2]. Quantization can be uniform or mixed (use different bit-widths for different layers of the network). And the quantization method can be post-training (PTQ) that often leads to quantization noise due to precision loss. An alternative for better accuracy preservation is quantization aware training (QAT), which has a downside of re-training the model, an expensive proposal for massive models, which are difficult to quantize in comparison to smaller BERT-based models [28].

High dynamic range of activation layers (aka large magnitude or structured outliers) make quantization difficult, in particular without retraining [2]. This has led to the emergence of various strategies for adaptive quantization with little to none post training required. Wei et al. show that the in the LayerNorm amplifies the outliers [40]. They propose gamma migration to effectively perform non scaling LayerNorm and then do a light-weight finetuning. Park et al. perform 8-bit asymmetric quantization for both weights and quantization and perform dynamic scaling of activations by introducing quadapter layers after each LayerNorm and then training those by block-wise calibration minimizing the L2 loss and then through light-weight end to end finetuning. Additional layers sometimes increase the computations [28].

Recent works, such as ZeroQuant [44], nuQmm [27], and LLM.Int8 [5], have presented post-training quantization (PTQ) approaches for very large models using group-wise quantization to 8-bit integers. ZeroQuant applies dynamic per-token activation, taking into account the drastic differences in activation ranges among different tokens. They also propose layer-wise knowledge distillation to improve quantization to 4/8-bit mixed precision, although the distillation process can be computationally expensive for massive models (e.g., 3 h of compute time reported for a model with 1.3B parameters). Both nuQmm and ZeroQuant demonstrate their approaches on models with a maximum of 20B and 2.7B parameters, respectively, and do not maintain comparable performance beyond that size. Additionally, these methods require custom CUDA kernels for implementation.

Balancing computational efficiency and model accuracy in machine learning is crucial. Optimal quantization hyperparameters for learning models can have a profound impact on generalization performance. This has been thoroughly explored, providing helpful insights into achieving the ideal trade-off between efficiency and precision [16].

Additionally, the extreme compression of large language models is another important aspect, and additive quantization has emerged as an effective technique for this purpose. Recent research work has shown the enormous potential of additive quantization in significantly optimising language models [9].

LLM.Int8 introduces mixed-precision quantization, using FP16 for outliers and INT8 for other activations, while retaining accuracy. On the other hand, GPTQ performs more extreme layer-wise quantization with 2.5 to 4 bits per weight, with negligible accuracy loss [11]. However, efficient GPU kernels are required for inference, and activations are not quantized in this approach. SmoothQuant stands out as the first proposed method for true INT8 weight and

activation quantization, with tested results on generative models up to 175B parameters [42]. The implementation of Smooth-Quant is available in Nvidia's Faster Transformer library on GitHub. Research sug-gests that 4-bit precision is generally optimal in terms of total model bits and zero-shot accuracy [6]. Table 2 demonstrates a comparison of recent quantization schemes applied to large language models.

Table 2. Comparison of Quantization techniques for massive LLMs

Original Model	Quantization	Size Reduction	Speed Improvement	Task	Eval (Compressed; Orig)	Metric
OPT-175B	Smooth-Quant	1.96x	1.51x	7 Zero-Shot benchmarks Avg	66.9%; 66.8%	Accuracy (H)
OPT-175B	Smooth-Quant	1.96x	1.51x	WikiText	10.99; 11.11	Perplexity (L)
OPT-175B	LLM.int8	2x	0.81x	WikiText	10.99; 11.10	Perplexity (L)
GPT3-1.3B	ZeroQuant	1.96x	3.67x	WikiText-2	15.3; 21.9	Perplexity (L)
GPT3-1.3B	ZeroQuant- LKD	3x	3.67x	WikiText-2	15.3; 17.6	Perplexity (L)
OPT-175B	GPTQ	4x	1.9x	WikiText-2	8.34; 8.68	Perplexity (L)

2.2 Pruning

Pruning compresses a model by removing redundant or less important weights (unstructured) and/or rows/columns/blocks or entire components (structured). Unstructured pruning offers more adaptive control and thus better accuracy but requires special hardware and libraries to allow for speedup benefits [12] [25]. Structured pruning methods target speedup as they reduce matrix dimensions or remove layers at the cost of more accuracy loss. Pruning techniques that employ retraining regularize weights to improve upon the accuracy loss. Such retraining can be expensive for LLMs. Till date, structured pruning techniques are only experimented on smaller LMs with mil-lions to rarely a billion parameters mostly on classification or NLU tasks [14] [30].

TextPruner is a PyTorch-based library to apply word embedding and transformer optimization-free structured pruning and has added support (only for vocabulary pruning) for moderately large models like T5 with 11B parameters [43]. SparseGPT offers one-shot post-training unstructured pruning for LLMs offering compressions up to 60 with minimal accuracy loss and a possible speedup of 2× using supporting hardware [10]. Theirs is the only work till date that has explored pruning on LLMs. However, recent works have introduced techniques for efficient structured pruning for generative models often combined with Knowledge Distillation (Sect. 2.2) [8] [17] [30]. However, the experiments are still for smaller models (less than 2B parameters). The positive side is that the larger the model, the relatively smaller the performance loss as more compression happens. For example, ZipLM demonstrates that the speedup due to pruning is inversely proportional to the accuracy, and the rate of de-crease lowers with the size of the model, since larger models seem to have redundant paths to infer the same information [17]. Mini-GPT leverage contextual pruning to create smaller, domain-specific language models with maintained or enhanced

performance, addressing the sustainability and practicality of large-scale language processing [35].

2.3 Architectural Efficiency

In addition to compression techniques like pruning and quantization, architectural efficiency is another key consideration in the compression of large language models. Architectural efficiency refers to the design and structure of the language model architecture itself, with the aim of improving performance and reducing computational costs. After the core transformer architecture [36], many different architectures are proposed to address various performance improvements. For example, Linformer attempts to reduce self-attention time complexity from O(n2) to O(n). Since these, in effect, require training from scratch, this is not a viable compression strategy for a pretrained model [37]. However, aspects of architectural components can be used for student models in KD as detailed in the following section.

2.4 Knowledge Distillation (KD)

KD is training a smaller model (student) using one or more larger pre-trained models (teachers). Distillation can happen at Output Logits (replace entire model keeping the softmax same), Encoder outputs (replace encoder units with similar units with less heads/layers or another arch), Attention Maps (keep the softmax attention outputs and replace the entire attention unit with something else). They can train entire or parts of the model and can distill at both the pre-training level and downstream task specific level [15] [38]. Since they require training, they incur high computational costs [17]. Till date KD is not applied to massive language models, owing to a very expensive training phase that may take weeks to months to train [10]. Liang et al. discusses data augmentation to reduce the performance gap, while it is only tested for BERT base models on GLUE tasks [20].

2.5 Dynamic Inference

It is a way to selectively omit unnecessary computation during the inference phase. For example, MoEfication partitions the FFN layer to highly compressed Mixture of Experts and then builds routes to decide the expert to use for each input [46]. Adaptive Length Reduction (AdapLeR) method accelerates inference by dynamically identifying less contributing token representations through layers of BERT-based models [22]. It adds an MLP followed by a softmax after each layer in the model to estimate the contribution score for each token representation. These scores offer a saliency map to determine what tokens to pass to the next layer and thus prune unnecessary computations hierarchically. AdapLeR report up to 22X speedup for classification tasks, while generation tasks are yet to be explored. The concepts of self-distillation and adaptive inference were first introduced in FastBERT [21]

2.6 Computation Optimization Strategies

These works aim to speed up by optimizing memory or computation resource use. Ma et al. use Block-Term Tensor Decomposition (BTD) for significantly compressing the non-linear multi-head attention to multi-linear attention, reporting no accuracy loss with 50+ compression [23]. Wang et al. offers a series of GPU optimization techniques in LightSeq to streamline computation of neural layers and reduce memory footprint [39]. It shows definite speedup improvements in NLG tasks for relatively smaller batch and sequence sizes (less than 1024) compared to contemporary ap-proaches like FasterTransformer and Tensorflow. The impact of this approach with integer quantization and sparse pruning is yet to be explored.

EET is an Easy and Efficient Transformer Library that applies a series of transformer inference optimizations at both the algorithm and implementation levels to speed up transformer inference up to 8 times, depending on the GPU hardware [19]. Energion targets efficient tensor and non-blocking pipeline parallel strategies to avoid redundant computation in a distributed setting and implements a peer memory pool to improve latency throughput [7]. It allows for inferring from larger models in a single GPU using a larger heterogeneous memory space. DeepSpeed also optimizes parallelization to enable large transformer model inferences with unimaginably few resources [1].

Speculative Decoding attempts to speed up by addressing memory bottleneck while increasing the computation costs [18]. They use lightweight models to predict the next set of tokens in a generative task while using the primary model to validate, correct, or induce the next token. So instead of predicting one token in each decoding cycle, they can do 3–4 tokens on average, resulting in speedup. Few other techniques include Parameter sharing (weight shared across all encoder units) and Word piece encoding (reducing vocabulary to reduce embedding matrix) [23] [31].

2.7 Compounding

Many of the techniques discussed above are orthogonal to each other, however, their impact when combined for NLG tasks with LLMs needs to be studied [13]. BMCook, a task-agnostic compression toolkit for big models, implements a combination of Quantification, Pruning, Distillation, and MoEfication and reports 12× compression while retaining 97 accuracy on MNLI-m and SQuAD 1.0 datasets with the T5-3B model [45]. SparseGPT suggests the possibility of integrating GPTQ with their pruning method, but the impact on performance is not clear [10].

2.8 Other Advanced Techniques

Tomut et al. introduced a groundbreaking method for the extreme compression of large language models, named CompactifAI, employing quantum-inspired Tensor Networks to maintain high accuracy while significantly reducing model size

and energy consumption [33]. In the same context, Xia et al. presents a pioneering GPU kernel design for FP6 quantization, which notably enhances inference throughput and maintains model quality, offering a crucial ad-vancement in the deployment of resource-intensive language models [41]. Eliseev and Mazur propose novel offloading strategies for running large Mixture-of-Experts language models on consumer hardware, enhancing efficiency through an LRU cache and preemptive expert loading [9].

3 Practical Recommendation

The efforts from recent research indicate adoption of integer quantization for compres-=sion with objective of less resource use while retaining most of the performance. Next comes Unstructured pruning that possibly can be combined with quantization as both require special hardware and GPU kernels to achieve speedup from the compression. Structured Pruning impacts performance while KD and Direct Inference methods are more task specific and not yet much explored for LLMs in context of NLG. There is a possibility for applying computational optimization techniques that do not require re-training in order to gain speedups, but such techniques for LLMs is yet to be explored. Libraries like fasterTransformer, LightSeq, EET or Energion / DeepSpeed (for distributed settings) can also be explored to attain speedups. For task specific down-stream tasks, approaches like KD, and structured pruning are recommended suggestions.

Table 3 presents a summary of the practical recommendations for implementing compression techniques for LLMs. It is important to experiment with different compression techniques and libraries to determine the best approach for a specific use case. Additionally, it is recommended to follow best practices for training and evaluation of compressed models to ensure optimal performance. Based on the literature survey, the following practical implementation strategy can be suggested:

1. Implement integer quantization for compression to reduce resource usage while retaining most of the performance.
2. Combine unstructured pruning with quantization as both require special hardware and GPU kernels to achieve speedup from the compression.
3. Explore the possibility of applying computational optimization techniques that do not require re-training in order to gain speedups.
4. Use libraries like fasterTransformer, LightSeq, EET, or Energion/DeepSpeed (for distributed settings) to attain speedups.
5. For task-specific downstream tasks, approaches like KD, and structured pruning are recommended suggestions.
6. Keep in mind that KD and Direct Inference methods are more task-specific and not yet much explored for LLMs in the context of NLG.
7. While implementing structured pruning, it is important to consider its impact on performance.
8. Continuously monitor the performance and accuracy of the compressed models to ensure that they meet the requirements of the downstream tasks.

Table 3. Practical Recommendations for Implementing Compression Techniques for LLMs

Method	Description
Distributed Training	Distributed training can be used to accelerate the training process for LLMs. Libraries like Horovod and DeepSpeed can be used to implement distributed training on multiple GPUs or nodes.
Knowledge Distillation	Knowledge Distillation (KD) can be used to reduce the size of LLMs while retaining their performance. In KD, a smaller student model is trained to mimic the output of a larger teacher model. This approach can be used to train smaller LLMs that can be deployed on devices with limited re-sources.
Model Pruning	Model pruning can be used to reduce the size of LLMs by removing redundant or unimportant parameters. Structured pruning and unstruc-tured pruning can be used for this purpose. Unstructured pruning can be combined with quantization to achieve further compression.
Quantization	Quantization can be used to reduce the memory footprint and computation requirements of LLMs. Integer quantization can be used to compress the weights and activations of LLMs to reduce their size.
Model Parallelism	Model parallelism can be used to distribute the memory and computation requirements of LLMs across multiple GPUs or devices. This can help to train larger models that cannot fit on a single device.
Inference Optimization	Inference optimization techniques like dynamic batching, caching and memoization can be used to speed up the inference process of LLMs. Libraries like TensorFlow Serving, TorchServe, and Triton Inference Server can be used to deploy LLMs and optimize their inference performance.

There are several evaluation metrics to assess the performance of compressed language models as presented in table 4. It is important to choose the right metrics based on the specific requirements of the problem and to use cross-validation to get a more reliable estimate of model performance. There are a few important considerations to keep in mind when using these evaluation metrics.

Table 4. Evaluation Metrics for Compressed Language Models

Metric	Description
Compression Ratio	This metric is used to measure the extent to which a model has been compressed, and is defined as the ratio of the original model size to the compressed model size.
Model Size	The size of the compressed model is an important metric to consider, as it directly affects the resource requirements of the model.
Inference Speed	The inference speed of a compressed model is another important metric to consider, as it affects the overall performance and usability of the model.
Perplexity	Perplexity is a common evaluation metric used in natural language processing (NLP) to measure the quality of language models. It measures how well a language model predicts the next word in a sequence of words.
Accuracy	Accuracy is a measure of how well a model performs on a specific task, such as question answering or text classification. It is usually measured in terms of precision, recall, and F1 score.
Retention Ratio	Retention ratio is a metric that measures the proportion of original model parameters retained after compression. It can provide insight into the level of compression achieved without sacrificing model performance.
Distortion	Distortion is a metric used to measure the amount of information lost during compression. It can help to identify which compression techniques are most effective while maintaining model performance.
Energy Efficiency	Energy efficiency is a metric that measures the amount of energy required to perform a certain computation. It is becoming increasingly important in the field of machine learning as energy consumption becomes a critical concern for many applications.

Choosing the right metrics: Depending on the nature of your problem, certain metrics may be more relevant than others. For example, accuracy may not be the best metric for imbalanced datasets, in which case precision, recall, or F1 score may be more appropriate.
Balancing trade-offs: Different evaluation metrics can be optimized to achieve different objectives. For example, optimizing for recall can lead to higher false positives, while optimizing for precision can lead to higher false negatives. It is important to balance these trade-offs depending on the specific needs of your problem.
Setting appropriate thresholds: For binary classification problems, the output of the model is typically a probability score. These scores need to be converted into class predictions by applying a threshold. The choice of threshold can have a significant impact on the performance of the model, and it should be chosen based on the specific needs of your problem.
Using cross-validation: To get a more reliable estimate of model performance, it is important to use cross-validation, in which the data is split into multiple folds, and the model is trained and evaluated on each fold. This helps to reduce the impact of random variations in the data and gives a more accurate estimate of the model's performance.

4 Conclusion

To summarize, Large Language Models (LLMs) based on transformer architecture, such as GPT and BERT, have gained popularity for natural language generation tasks, but suffer from slow inference speed due to their large number of parameters and auto-regressive decoding. To address this issue, various compression techniques like pruning, quantization, and knowledge distillation have been applied to LLMs. Quantization, which involves reducing the number of bits to represent model weights and activations, has been found to be an effective approach for LLM compression. This can be done through post-training quantization or quantization-aware training, which involves retraining the model. Adaptive quantization strategies have also been pro-posed to reduce the impact of outliers. Pruning, which involves removing redundant or less important weights, has also been applied to LLMs, with unstructured pruning offering more adaptive control and structured pruning targeting speedup. Knowledge distillation, which involves training a smaller student model to mimic the behavior of a larger teacher model, has also been found to be useful for LLM compression. Overall, these techniques have shown promising results in improving the efficiency and speed of LLMs while maintaining their accuracy, allowing for better practical applications in natural language generation tasks.

References

1. Aminabadi, R.Y., et al.: Deepspeed inference: enabling efficient inference of transformer models at unprecedented scale. In: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2022)

2. Bondarenko, Y., Nagel, M., Blankevoort, T.: Understanding and overcoming the challenges of efficient transformer quantization, pp. 7947–7969 (2021).https://doi.org/10.18653/v1/2022.findings-emnlp.185, https://aclanthology.org/2022.findings-emnlp.185
3. Cheng, P., et al.: A study of generative large language model for medical research and healthcare (2023). https://doi.org/10.1038/s41746-023-00958-w
4. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: Model compression and acceleration for deep neural networks: The principles, progress, and challenges (2018).https://doi.org/10.1109/MSP.2017.2765695
5. Dettmers, T., Lewis, M.A., Belkada, Y., Zettlemoyer, L.: Llm.int8(): 8-bit matrix multiplication for transformers at scale. CoRR abs/2208.07339 (2022)
6. Dettmers, T., Zettlemoyer, L.: The case for 4-bit precision: k-bit inference scaling laws. In: International Conference on Machine Learning, pp. 7750–7774 (2023)
7. Du, J., et al.: Energonai: An inference system for 10-100 billion parameter transformer models. arXiv preprint arXiv:2209.02341 (09 2022).https://doi.org/10.48550/arxiv.2209.02341, https://arxiv.org/abs/2209.02341
8. Edalati, A., Tahaei, M.S., Ahmad, R., Nia, V.P., Clark, J.J., Rezagholizadeh, M.: Kronecker decomposition for gpt compression **2**, 219–226 (2022)
9. Eliseev, A., Mazur, D.: Fast inference of mixture-of-experts language models with offloading. ArXiv **abs/2312.17238** (2023), https://api.semanticscholar.org/CorpusID:266573098
10. Frantar, E., Alistarh, D.: SparseGPT: massive language models can be accurately pruned in one-shot. In: International Conference on Machine Learning, pp. 10323–10337 (2023)
11. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: Gptq: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323 (2022).https://doi.org/10.48550/arxiv.2210.17323, https://arxiv.org/abs/2210.17323
12. Gale, T., Zaharia, M., Young, C., Elsen, E.: Sparse GPU kernels for deep learning, pp. 1–14 (2020)
13. Ganesh, P., et al.: Compressing large-scale transformer-based models: A case study on bert (2021). https://aclanthology.org/2021.tacl-1.63
14. Gupta, M., Agrawal, P.: Compression of deep learning models for text: a survey. ACM Trans. Knowl. Discov. Data (TKDD) **16**(4), 1–55 (2022)
15. Jiao, X., et al.: Tinybert: Distilling bert for natural language understanding, pp. 4163–4174 (2020). https://doi.org/10.18653/v1/2020.findings-emnlp.372, https://aclanthology.org/2020.findings-emnlp.372
16. Kashiwamura, S., Sakata, A., Imaizumi, M.: Effect of weight quantization on learning models by typical case analysis. arXiv preprint arXiv:2401.17269 (2024https://doi.org/10.48550/arxiv.2401.17269, https://arxiv.org/abs/2401.17269
17. Kurtic, E., Frantar, E., Alistarh, D.: ZipLM: inference-aware structured pruning of language models. Adv. Neural Inf. Process. Syst. **36** (2024)
18. Leviathan, Y., Kalman, M., Matias, Y.: Fast inference from transformers via speculative decoding. In: International Conference on Machine Learning, pp. 19274–19286 (2023)
19. Li, G., et al.: Easy and efficient transformer : Scalable inference solution for large NLP model, pp. 62–68 (2022). https://doi.org/10.18653/v1/2022.naacl-industry.8, https://aclanthology.org/2022.naacl-industry.8
20. Liang, K.J., et al.: Mixkd: Towards efficient distillation of large-scale language models (2020).https://doi.org/10.48550/arXiv.2011.00593, http://arxiv.org/abs/2011.00593v2

21. Liu, W., Zhou, P., Wang, Z., Zhao, Z., Deng, H., Ju, Q.: FastBERT: a self-distilling BERT with adaptive inference time, pp. 6035–6044 (2020).https://doi.org/10.18653/v1/2020.acl-main.537, https://aclanthology.org/2020.acl-main.537
22. M, A., Mohebbi, H., Pilehvar, M.T.: AdapLeR: speeding up inference by adaptive length reduction. In: Annual Meeting of the Association for Computational Linguistics, pp. 1–15 (2022). https://doi.org/10.18653/v1/2022.acl-long.1, https://aclanthology.org/2022.acl-long.1
23. Ma, X., et al.: A tensorized transformer for language modeling. In: NIPS 2019: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 2232–2242 (2019)
24. Marinò, G.C., Petrini, A., Malchiodi, D., Frasca, M.: Deep neural networks compression: a comparative survey and choice recommendations. Neurocomputing **520**, 152–170 (2023)
25. Mishra, A.K., et al.: Accelerating sparse deep neural networks. arXiv preprint arXiv:2104.08378 (04 2021).https://doi.org/10.48550/arxiv.2104.08378, https://arxiv.org/abs/2104.08378
26. Mishra, R., Gupta, H.P., Dutta, T.: A survey on deep neural network compression: challenges, overview, and solutions (2020)
27. Park, G., et al.: LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. arXiv preprint arXiv:2206.09557 (2022). https://doi.org/10.48550/arxiv.2206.09557, https://arxiv.org/abs/2206.09557
28. Park, M., You, J., Nagel, M., Chang, S.: Quadapter: adapter for GPT-2 quantization, pp. 2510–2517 2022).https://doi.org/10.18653/v1/2022.findings-emnlp.185, https://aclanthology.org/2022.findings-emnlp.185
29. Park, S., Choi, J., Lee, S., Kang, U.: A comprehensive survey of compression algorithms for language models. arXiv preprint arXiv:2401.15347 (2024). https://doi.org/10.48550/arxiv.2401.15347, http://arxiv.org/abs/2401.15347
30. Santacroce, M., Wen, Z., Shen, Y., Li, Y.: What matters in the structured pruning of generative language models? arXiv preprint arXiv:2302.03773 (2023). https://doi.org/10.48550/arxiv.2302.03773, https://arxiv.org/abs/2302.03773
31. Shi, K., Yu, K.: Structured word embedding for low memory neural network language model. Interspeech 1254–1258 (2018). https://doi.org/10.21437/Interspeech.2018-1057
32. Tang, Y., et al.: A survey on transformer compression. arXiv preprint arXiv:2402.05964 (2024).https://doi.org/10.48550/arxiv.2402.05964, https://arxiv.org/abs/2402.05964
33. Tomut, A., et al.: CompactifAI: extreme compression of large language models using quantum-inspired tensor networks (2024)
34. Topal, M.O., Bas, A., van Heerden, I.: Exploring transformers in natural language generation: GPT, BERT, and XLNet (2021)
35. Valicenti, T., Vidal, J.J., Patnaik, R.: Mini-gpts: Efficient large language models through contextual pruning. arXiv preprint arXiv:2312.12682 (12 2023). https://doi.org/10.48550/arxiv.2312.12682, https://arxiv.org/abs/2312.12682
36. Vaswani, A., et al.: Attention is all you need. Adv. Neural Inf. Proc. Syst. (2017)
37. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity (2020).https://doi.org/10.48550/arXiv.2006.04768, http://arxiv.org/abs/2006.04768v3
38. Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: MiniLM: deep self-attention distillation for task-agnostic compression of pre-trained transformers. In:

Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)
39. Wang, X., Xiong, Y., Yang, W., Wang, M., Li, L.: LightSeq: a high performance inference library for transformers, pp. 113–120 (2021).https://doi.org/10.18653/v1/2021.naacl-industry.15, https://aclanthology.org/2021.naacl-industry.15
40. Wei, X., Zhang, Y., Zhang, X., Gong, R., Zhang, S., Zhang, Q., Yu, F., Liu, X.: Outlier suppression: pushing the limit of low-bit transformer language models. Adv. Neural. Inf. Process. Syst. **35**, 17402–17414 (2022)
41. Xia, H., et al.: Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design. arXiv preprint arXiv:2401.14112 (01 2024).https://doi.org/10.48550/arxiv.2401.14112, https://arxiv.org/abs/2401.14112
42. Xiao, G., Ji, L., Seznec, M., Demouth, J., Han, S.: SmoothQuant: accurate and efficient post-training quantization for large language models, pp. 38087–38099 (2023)
43. Yang, Z., Cui, Y., Chen, Z.: Textpruner: A model pruning toolkit for pre-trained language models, pp. 35–43 (2022).https://doi.org/10.18653/v1/2022.acl-demo.4, https://aclanthology.org/2022.acl-demo.4
44. Yao, Z., Aminabadi, R.Y., Zhang, M., Wu, X., Li, C., He, Y.: Zeroquant: efficient and affordable post-training quantization for large-scale transformers. Adv. Neural. Inf. Process. Syst. **35**, 27168–27183 (2022)
45. Zhang, Z., et al.: BMcook: a task-agnostic compression toolkit for big models, pp. 396–405 (2022). https://doi.org/10.18653/v1/2022.emnlp-demos.40, https://aclanthology.org/2022.emnlp-demos.40
46. Zhang, Z., Lin, Y., Liu, Z., Li, P., Sun, M., Zhou, J.: Moefication: transformer feed-forward layers are mixtures of experts, pp. 877–890 (2022). https://doi.org/10.18653/v1/2022.findings-acl.71, https://aclanthology.org/2022.findings-acl.71