Research for Enhancing Processing and Computational Efficiency in LLM

Yu Cong

School of Data Science, Capital University of Economics and Business, Beijing, 100070, China
32021230004@cueb.edu.cn

Abstract. In the context of current technological development, large language models (LLMs) have become a core component of artificial intelligence. This report provides an in-depth discussion of various advanced strategies and techniques to improve the processing and computational efficiency of LLMs. First, the report goes through a detailed analysis of automatic 4-bit Integer Quantization (INT4 quantization). It then discusses binarization with the Flexible Dual Binarization (FDB) fusion strategy in depth and elaborates on the principle of automatic INT4 quantization and its positive impact on computational efficiency. Furthermore, it explores the flexibility of the fusion strategy of binarization with FDB and its application. Lastly, it examines the application of Atom technology in low-bit quantization and its contribution to processing efficiency. Further, the report explores hybrid LLM inference strategies, focusing on the principles of hybrid LLM inference and the impact on efficiency. Finally, the report introduces soft prompt and decoding optimization techniques, including the principles and advantages of the MEDUSA framework and the SARATHI technique, as well as the application of the Transferable Prompt technique. By synthesizing these strategies and techniques, this report provides strong guidance and reference for the efficient deployment and application of LLM.

Keywords: Hybrid LLM inference, soft prompts, decoding optimization.

1 Introduction

Large Language Models (LLMs) are becoming an increasingly important technology in the field of Artificial Intelligence. With the continuous development of models such as OpenAI's Generative Pre-trained Transformer (GPT) series and Google's Bidirectional Encoder Representations from Transformers (BERT), they demonstrate great potential for text understanding, generation, and translation [1]. With the continuous development of models such as OpenAI's GPT series and Google's BERT, they show great potential for text understanding, generation, and translation [1]. Automatic text generation, complex problem solving, and multilingual support are some of the key applications of AI in various industries, which proves its versatility and wide applicability [2]. However, the application of LLM faces several challenges. Increasing the model size requires significant computational resources, which can increase operational costs and limit applications in resource-constrained environments [3]. Cloud computing platforms provide the necessary computational resources, but suffer from long response times, high bandwidth costs, and data privacy risks in real-time applications [4]. In addition, as the amount of model training data increases, issues such as data privacy and copyright are becoming more and more prominent, posing significant technical challenges to current LLMs.

Many research teams are currently exploring solutions to the challenges faced by LLM techniques. The development of new strategies such as model compression and advanced training techniques, as well as technological advances, can help alleviate these problems and further optimize and popularize LLM techniques. Nonetheless, to achieve widespread deployment of these techniques, interdisciplinary research and collaboration are necessary to ensure that technology development balances efficiency, cost, and privacy protection.

This study systematically analyzes the research progress in the field of LLM processing and computational efficiency improvement utilizing a literature review. The study synthesizes research literature, technical reports, and case studies from various aspects, and provides an in-depth discussion of various techniques and strategies for efficiency improvement. Special attention is paid to the application and effectiveness of techniques such as automatic INT4 quantization, binarization and FDB fusion strategies, and Atom low-bit quantization. In addition, this study analyzes the potential value of advanced techniques such as hybrid LLM inference strategies in improving LLM efficiency. Through these analyses, it aims to provide theoretical support and strategy guidance for improving LLM efficiency.

2 Fusion Strategies for Quantization, Binarization, and Atom

The INT4 quantization strategy demonstrates its unique benefits in terms of storage optimization and computational speedup; binarization, a process that reduces model weights and activation values to their most basic binary form (i.e., 1 and 0 or -1 and 1), greatly reduces the model's storage requirements and can dramatically speed up computation. Atom improves the throughput of its services through the use of low-bit quantization and dramatically reduces memory consumption through low-bit quantization. This approach employs a novel mixed-precision quantization and fine-grained group quantization to maintain high accuracy. Compared to previous methods that often lose accuracy when using low-bit representations, the Atom method performs better.

2.1 Automatic INT4 Quantization

Automatic INT4 Quantization Flow

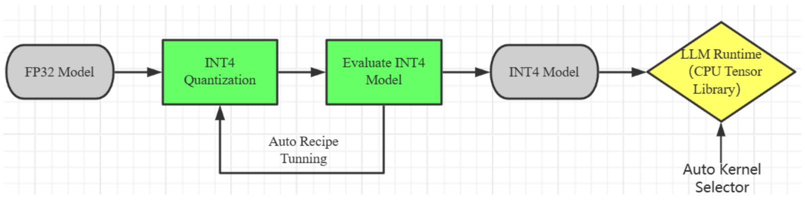

Fig. 1. Automatic INT4 quantization flow.

In the field of deep learning and machine learning, model parameters are typically represented and computed in floating-point format. The 32-bit floating-point number (FP32) format, shown on the left side of Fig. 1, represents a floating-point number with a 32-bit binary width, providing high computational accuracy and a large range of values. In the automatic INT4 quantization process, a model in FP32 format goes through the INT4 quantization step. As shown in the green "INT4 Quantization" box in Figure 1, the format is converted by applying a preset quantization recipe (including quantization scale and offset), and the quantized model is evaluated for accuracy in the "Evaluate INT4 Model" step. If the model quantized to INT4 format already meets the preset accuracy target, no further adjustment of the quantization recipe is required [5]. This process may involve "Auto Recipe Tuning" to fine-tune the quantization process. Quantization to INT4 means that each weight requires only 4 bits of representation, which is 1/8th of FP32, significantly reducing the model storage size. Once quantization is complete, the model can be further optimized for efficient execution on the target hardware using the "LLM Runtime (CPU Tensor Library)" and "Auto Kernel Selector" in Fig. 1.

The Advantages of Automatic INT4 Quantization and its Impact on Processing Efficiency. Auto Recipe Tuning: The automatic INT4 quantization process enables the intelligent selection of quantization parameters through the Auto Recipe Tuning session [5]. This ensures that the quantization model retains the performance of the original model to the greatest extent possible while reducing manual intervention and optimization time.

Accuracy assessment and optimization: after quantization, the accuracy of the model is verified by a dedicated assessment step to ensure that the quantization process has not negatively affected the model performance. If the quantized model does not meet the predefined accuracy target, an automatic recipe adjustment mechanism will be activated to further optimize the quantization parameters until the performance requirements are met [5].

After the quantization model is converted to INT4, the multiplication operation of weights and inputs is converted from floating point to integer operation. Integer operations are much less computationally intensive than floating-point operations, making them faster. At the same time, reducing the bit width of the weights means that more weights can be read from memory in the same amount of time, improving the efficiency of memory bandwidth utilization. Cache efficiency is also improved by the reduced weight volume, which reduces memory access latency and speeds up the computation of the entire model inference process.

2.2 Binarization with Flexible Dual Binarization (FDB)

Principles of Binarization. The main steps of binarization include:

Weight binarization: simplifying each weight value to -1 or 1. This is usually achieved by applying a sign function, where positive values map to +1 and negative values map to -1.

Activation binarization: the output value of the activation function is also simplified to -1 or 1 so that most of the computation in the network can be done by simple bit operations instead of complex floating-point operations.

Binarization improves processing and computational efficiency by reducing the size of the model and simplifying the computation, making the model more suitable for deployment on resource-constrained devices [6].

Flexible Dual Binarization (FDB). FDB achieves a more flexible representation by splitting the 2-bit wide quantization weights into two separate sets of binary weights while maintaining the computationally efficient advantage [6].

Dual binary weights: the 2-bit wide quantization weights are split into two sets of binary weights (each set of weights is either 1 or 0), which increases the model's representational power without sacrificing computational efficiency [6].

Tunable scaling factors: each set of binary weights has its corresponding scaling factor which is optimized during training to improve the accuracy of the model.

High sparsity: as the weights are binarized, most of the weights in the model become 0. This further increases the sparsity of the model, which helps to reduce storage requirements and accelerate model inference.

Impact on Processing and Computational Efficiency. Dramatic reduction in storage space: binarization reduces model weights to -1 or 1, significantly reducing the storage size of model parameters. In particular, the FDB technique further improves the accuracy of the weight representation by optimizing the grouping and scaling factors of the binary weights, while maintaining the storage efficiency [6].

Significant improvement in computational speed: In binarized models, the computational process is more efficient than traditional floating-point operations because only bit operations are involved [6]. This means that the model can process more data in less time while performing inference tasks.

Optimization of memory bandwidth and cache utilization: since the quantized model parameters take up less space, it makes the memory bandwidth more efficiently utilized and the CPU cache efficiency is also improved. This not only reduces the latency of memory accesses but also improves the running speed of the whole model [6].

Dynamic Adaptation and Performance Maintenance: with the dynamic adjustment capability introduced by FDB, the model can be better adapted to the data and task requirements during the training process, which improves the accuracy and efficiency of the model in various application scenarios.

2.3 Atom Low-Bit Quantization

Principles of Atom. Low-bit quantization: the Atom method significantly reduces the memory footprint and data transfer requirements of the model by quantizing the weights and activation values of the model to a lower bit width. This low-bit representation allows for more efficient integer arithmetic to be used on modern hardware, thus increasing computational speed [7].

Mixed-precision quantization: Atom employs a mixed-precision quantization strategy, meaning that different parameters and data in the model can be quantized to different levels of precision depending on their impact on model performance and demand on computational resources. This allows the maintenance of high precision in parts that are critical to the accuracy of the model while quantizing other parts to low precision to optimize efficiency [7].

The Advantages of Atom and its Impact on Processing and Computational Efficiency. The dynamic quantization strategy employed by Atom allows Atom to adjust the quantization parameters, such as the quantization scale factor and zero point, in real time according to the actual distribution of the input data [7]. This flexibility allows the quantization process to adapt more accurately to different data characteristics, thereby reducing the information loss and quantization errors associated with fixed quantization parameters.

By reducing the required memory bandwidth and storage requirements, as well as utilizing the low-bit integer arithmetic capabilities on the hardware, Atom can significantly improve the inference throughput of the model [7]. Since low-bit arithmetic is typically faster than full-precision floating-point arithmetic, especially on hardware that supports integer arithmetic acceleration. Therefore, the Atom approach can reduce the latency of model inference [7].

3 Hybrid LLM Inference

3.1 Principles of Hybrid LLM Inference

The core idea of the hybrid LLM inference is to assign queries to large or small language models through an intelligent router to reduce the reasoning cost while maintaining the quality of the response [8].

The intelligent router utilizes machine learning algorithms to predict the processing performance and output quality of different LLMs for a given query through a series of carefully designed steps. It first extracts key features from the query, such as length, lexical complexity, and syntactic structure, to assess the difficulty of the query [8]. Then, historical data is used to label the performance of each query under different models to provide a training base for machine learning models. With this training data, the smart router trains models such as decision trees or neural networks to learn to predict response quality and then ensures the accuracy of the predictions through performance evaluation. Once deployed, the router evaluates each received query in real-time and decides to route it to the most appropriate model - simple queries are handled by small models to save resources, while complex queries are handled by large models to ensure quality [8].

3.2 Impact on Processing and Computational Efficiency

Optimal allocation of computational resources: By dynamically choosing between small and large models, the system can optimize the use of computational resources based on the complexity of the task and the need for accuracy.

Improve response speed: Dynamic reasoning with hybrid LLM inference allows the system to use faster models for simple queries, thus improving response speed without sacrificing much accuracy.

Reduce Unnecessary Computational Overhead: By avoiding the use of overly complex models for simple tasks, dynamic inference methods help to reduce unnecessary computational overhead, thereby improving computational efficiency at the system-wide level.

4 Soft Prompt and Decoding Optimization

MEDUSA significantly reduces the number of steps required for decoding by adding additional decoding heads and utilizing a tree attention mechanism to process and validate multiple candidate sequences in parallel. SARATHI improves Graphics Processing Unit (GPU) utilization and reduces inference latency by using Chunked-prefills and Decode-Maximal Batching, enabling efficient use of computational resources even at smaller batch sizes. Transferable Prompt improves the performance and enhances the adaptability of compression models by introducing learnable embedding vectors as part of the model input.

4.1 MEDUSA Framework

Principles of MEDUSA. MEDUSA enhances the efficiency of LLM inference by introducing multiple decoding heads capable of predicting multiple sequent tokens in parallel. This approach differs from speculative decoding strategies that rely on a separate, smaller draft model to generate candidate sequels [9]. Instead, MEDUSA adds these decoding heads directly on top of the final hidden state of an existing model, allowing for simultaneous prediction of multiple tokens without the need for an additional model [9].

Tree Attention. In the MEDUSA framework, the Tree Attention mechanism is an innovative application. The core of the Tree Attention mechanism lies in its ability to assign attentional weights hierarchically, allowing the model to not only focus on a single element but also to gain insight into the relationships between the parts of the sequence represented by the different levels of the tree [9].

By constructing a hierarchical tree structure, MEDUSA can deal with elements in a sequence, such as words or characters, in a highly organized manner. This hierarchical representation not only enhances the model's understanding of sequence information, but also optimizes the information processing flow, allowing each node to represent a specific part of the sequence or a different level of information abstraction.

MEDUSA achieves parallel decoding by adding additional decoding heads at the top level of the model that is capable of processing and verifying multiple candidate continuation paths simultaneously. This parallel processing capability stems from the Tree Attention mechanism, which allows the model to generate and examine multiple continuation branches at each decoding step, greatly improving the efficiency of decoding [9].

MEDUSA's Advantages. Through this efficient parallel decoding and information aggregation mechanism, the application of tree attention in MEDUSA not only significantly reduces the number of decoding steps required to generate a complete sequence, but also maintains the high quality of text generation. This shows that despite the significant improvement in inference speed, the generated text remains coherent and highly relevant, demonstrating the great potential of the MEDUSA framework to improve the efficiency of LLM inference while still ensuring the quality of the output.

4.2 SARATHI

Principles of SARATHI. SARATHI works to address two major bottlenecks in LLM reasoning: underutilization of GPU computational resources in the pre-filling phase and inefficiency in the decoding phase. Traditionally, the pre-filling phase can fully utilize the GPU when processing all input tokens, but the decoding phase generates only one token at a time, resulting in decreased GPU utilization. SARATHI balances the computational loads of the pre-filling and decoding phases by intelligently reorganizing the inference computation flow, thus significantly improving the inference efficiency and GPU resource utilization while maintaining the inference quality [10].

Chunked-Prefills. Chunked-prefills is a key strategy in SARATHI designed to optimize the computational efficiency of the pre-population phase. This strategy splits large pre-filling requests into multiple small chunks, each containing a portion of the input sequence that can be processed independently and in parallel [10]. The advantages of this approach include:

Improved GPU utilization: after dividing large input sequences into small chunks, more data can be processed in parallel, which allows efficient GPU utilization even in small batches [10].

Reduces inference latency: chunking allows the model to start processing input data faster, without waiting for the entire large input sequence to be processed, especially important for applications that require fast responses.

Optimize memory usage: Chunked-prefills when processing long sequences reduce memory pressure because only a portion of the sequence is processed at a time.

Decode-Maximal Batching. Another innovative strategy of SARATHI is Decode-Maximal Batching, which predicts the next marker for all sequences in the whole batch simultaneously at each decoding step by combining multiple sequences to be decoded or different parts from the same sequence into one large batch [10]. It has the following advantages:

Parallel processing: this approach makes full use of parallel computing resources, such as GPUs, and can predict the next token for multiple sequences at the same time, which significantly improves the decoding speed and efficiency.

Batch optimization: by intelligently constructing batches containing multiple decoded sequences, the utilization of computational resources is maximized, which is especially effective in multi-GPU environments.

Combination with Chunked-prefills: Decode-Maximal Batching in combination with Chunked-prefills strategy can further optimize the computational efficiency by including both the pre-populated context information and the part of the sequence to be generated when constructing decoding batches [10].

4.3 Transferable Prompt

Principles of Transferable Prompt. The core principle of the Transferable Prompt is to enhance the performance of a model in a compressed state by adding learnable soft prompts to the input of LLMs [11]. These soft prompts consist of a series of trainable embedding vectors that, along with the actual input data, form the input to the model [11]. The key to this method lies in the optimization of these embedding vectors during the model's training or fine-tuning stages, aiming to reduce the model size and inference time as much as possible without sacrificing output quality.

Soft Prompts. Soft prompts are a series of learnable embedding vectors that serve as part of the model's input, provided to the model along with the actual input data. These vectors are optimized during the training process to enhance the predictive performance of the compressed model.

The learning objective of soft prompts is to maximize the probability of correctly predicting the next token in a sequence given the preceding tokens. This allows the learned prompts to be aware of the compressed weights, and during the optimization process, gradients pass through these compressed weights, enabling the model to consider the effects of compression while generating responses, potentially leading to improved performance [11].

Transferable Prompt's Advantages. Firstly, soft prompts enable the model to form a more accurate understanding of specific tasks or content at the initial stage [11]. This efficiency enhancement reduces the computational burden on the model when processing input data.

Secondly, the transferability of soft prompts further extends this efficiency gain. A set of prompts optimized for specific tasks can be reused across different datasets and tasks, meaning that the model can quickly adapt to new tasks without needing to learn from scratch, thus saving substantial computational resources and time [11]. Additionally, this transferability reduces the dependency on large volumes of training data, as pretrained prompts can help maintain high-performance levels even with fewer data.

Finally, when combined with model compression techniques, the Transferable Prompt method also ensures that the performance of the compressed model does not suffer significantly [11]. By reducing unnecessary computational overhead, enhancing the model's adaptability, and mitigating the impacts of model compression, it offers an effective way to improve the processing and computational efficiency of LLMs.

5. Conclusion

This report reviews advanced technologies and strategies to enhance the processing and computational efficiency of LLMs, focusing on automatic INT4 quantization, binarization combined with FDB, and Atom low-bit quantization. Automatic INT4 quantization optimizes the quantization process, boosting computational efficiency; the integration of binarization with FDB, featuring flexible dual-binary weights and adjustable scale factors, improves model expressive power while maintaining efficiency. Atom's low-bit quantization reduces model size significantly without compromising performance.

In subsequent sections of the report, particular emphasis is placed on the significance of hybrid inference methods in improving the adaptability and efficiency of LLMs. By dynamically allocating computational resources based on the complexity of tasks and real-time demands, and selecting the most suitable model or model components for inference, it optimizes resource consumption while ensuring performance.

Moreover, the report focuses on the segments concerning soft prompts and decoding optimization techniques. The MEDUSA framework, by introducing the Tree Attention mechanism, optimizes the model's attention structure, enhancing processing efficiency and accuracy. The SARATHI strategy, through Chunked-prefills and Decode-Maximal Batching techniques, significantly improves the efficiency of the model's decoding stage, reducing latency and enhancing response speed. Transferable Prompt technology, by introducing transferable soft prompts, enhances the model’s adaptability and flexibility across different tasks and domains, allowing for more efficient processing of various types of inputs and further boosting overall computational efficiency.

In conclusion, the development and application of these technologies and strategies not only enhance the performance of LLMs in specific scenarios but also greatly increase their adaptability and flexibility in the face of varying demands. This provides new directions and inspiration for future research and development efforts, heralding the vast potential of LLM technologies for widespread application across various industries.

References

1. Meyer, J. G., Urbanowicz, R. J., Martin, P. C. N. et al.: ChatGPT and large language models in academia: opportunities and challenges. BioData Mining, 16(20) (2023)
2. Xiaoliang, C.: Challenges and contributing factors in the utilization of large language models (LLMs). arXiv preprint arXiv:2310.13343 (2023)
3. Hadi, M. U., Qureshi, R., Shah, A., et al.: A survey on large language models: Applications, challenges, limitations, and practical usage. Authorea (2023)
4. Lin, Z.: Pushing large language models to the 6g edge: Vision, challenges, and opportunities. arXiv preprint arXiv:2309.16739 (2023)
5. Haihao, S.: Efficient llm inference on cpus. arXiv preprint arXiv:2311.00502 (2023)
6. Hong, C.: DB-LLM: accurate dual-binarization for efficient LLMs. arXiv preprint arXiv:2402.11960 (2024)
7. Yilong, Z.: Atom: Low-bit quantization for efficient and accurate llm serving. arXiv preprint arXiv:2310.19102 (2023)
8. Dujian, D. Hybrid LLM: Cost-efficient and quality-aware query routing. The Twelfth International Conference on Learning Representations (2023)
9. Tianle, C.: Medusa: Simple llm inference acceleration framework with multiple decoding heads. arXiv preprint arXiv:2401.10774 (2024)
10. Agrawal, A.: Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. arXiv preprint arXiv:2308.16369 (2023)
11. Zhaozhuo, X.: Compress, then prompt: Improving accuracy-efficiency trade-off of llm inference with transferable prompt. arXiv preprint arXiv:2305.11186 (2023)