National Conference on Advances in Science Engineering and Technology organized by ECE on 7th November 2025.

REDUCING COMPLEXITY IN LARGE LANGUAGE MODELS

Yashwantha N
Assistant Professor, Dept. of CSE (Data Science), SJBIT

Prajwal B S
Student, Dept. of CSE (Data Science), SJBIT

Sammed Dundappa Saraswatigol
Student, Dept. of CSE (Data Science), SJBIT

Channabasava
Student, Dept. of CSE (Data Science), SJBIT

Sangamesh
Student, Dept. of CSE (Data Science), SJBIT

Abstract - Large Language Models (LLMs) have transformed natural language processing by exhibiting advanced capabilities in comprehension, reasoning, and context-aware text generation. However, the rapid escalation in their size and computational requirements presents major obstacles in training, deployment, and energy sustainability. The substantial hardware, memory, and power demands of these models restrict their accessibility and limit real-time or edge-based implementation. To address these concerns, current research emphasizes model-compression and optimization techniques that maintain accuracy while improving efficiency. Methods such as quantization, pruning, knowledge distillation, parameter sharing, and low-rank adaptation have demonstrated promising results in reducing redundancy and accelerating inference. Additionally, hybrid precision and adaptive computation frameworks seek to balance performance with computational cost. This survey consolidates recent progress in efficiency-oriented LLM research, examining their comparative advantages, limitations, and practical trade-offs.

Index Terms - Large Language Models (LLMs), Quantization, Low-Rank Adaptation (LoRA), Post-Training Quantization (PTQ), Quantization-Aware Training (QAT).

I. INTRODUCTION

A Large Language Models (LLMs) have emerged as a cornerstone of modern natural language processing (NLP) technologies, revolutionizing tasks such as translation, text summarization, sentiment analysis, question answering, conversational AI, and code generation. These models, exemplified by architectures like GPT-3, GPT-4, and BERT, have demonstrated exceptional performance across a broad range of applications [2],[3], making them integral to advancements in artificial intelligence (AI).

Despite their transformative potential, the deployment of these models in resource-constrained environments faces significant challenges due to their computational intensity, extensive memory requirements, and energy consumption [1],[4].

models capable of delivering high performance without excessive infrastructure demands. Recent advancements in quantization techniques, such as low-bit precision models, offer promising solutions by reducing the resource burden while preserving model performance [5]. These methods aim to compress models without compromising their ability to process language tasks effectively.

Approaches like Low-Rank Adaptation (LoRA) and Flash Attention enhance memory and computational efficiency [10], making it feasible to deploy high-performing models in constrained environments. This study explores the design and development of a 1.58-bit precision LLM to address these challenges, providing scalability for edge computing and other resource-sensitive domains. By pushing the boundaries of low-bit

quantization, this work seeks to make AI more accessible, affordable, and sustainable while maintaining competitive accuracy levels. These models have demonstrated remarkable performance in a broad range of nature.

II. Literature Riview

Quantization has become a key method to reduce computational and memory limitations of transformer based models, particularly large language models (LLMs) [1],[5]. Lowering the numerical precision of parameters and activations from floating-point (FP32) to lower-bit representations enables faster inference and significant memory savings. Recent research from 2021 to 2024 has shown that 8-bit and 4-bit quantization can effectively decrease computational costs while causing minimal loss in model performance [5]. These methods mainly include post-training quantization or quantization-aware training (QAT). QAT involves considering quantization effects during training to reduce accuracy loss. However, decreasing precision to 2-bit or 1-bit makes it more difficult to maintain performance because of limited representational capacity [1].

Consistent performance at 1-bit precision remains an unresolved issue. We need solutions that include error mitigation techniques. These techniques use additional strategies, such as maintaining floating-point scaling factors alongside binary representations to recover lost precision.

These improvements have expanded the use of transformer models in devices with limited resources, like mobile phones and IoT gadgets. Fine-tuning is crucial for maintaining model performance after quantization.

Methods like Low-Rank Adaptation (LoRA) and post-training quantization have proven effective for fine-tuning quantized models. LoRA introduces trainable low-rank matrices to pre-trained models, allowing for task-specific fine-tuning while being resource-efficient [5].

Nevertheless, consistent performance at 1-bit precision remains a challenge that needs solutions involving effective error mitigation techniques. These techniques include strategies like preserving floating-point scaling factors along with binary representations to recover lost precision. These improvements have expanded the use of transformer models in resource-limited devices such as mobile phones and IoT devices. Fine-tuning is crucial for maintaining model performance after quantization. Methods such as Low-Rank Adaptation (LoRA) and post-training quantization have proven to be efficient for fine-tuning quantized models. LoRA integrates trainable low-rank matrices into pre-trained models, allowing for task-specific fine-tuning while remaining efficient. This method has been very useful for edge AI situations, where computational and energy resources are limited. Additionally, post-training quantization, which occurs after model training, reduces the need for costly retraining. Recent research has emphasized gradient clipping and adaptive optimizers such as AdamW in stabilizing training dynamics in quantized platforms [1].

III. Dataset

The 1.58-bit Quantized Large Language Model (LLM) was tested with generally accepted NLP benchmark data sets to assess accuracy, resilience, and deployablity after compression [7]. The chosen data sets offered a wide variety of language comprehension tasks to thoroughly check the performance of the model.

Data Sets Used

1.GLUE (General Language Understanding Evaluation)

A set of tasks including sentiment analysis, natural language inference, and sentence similarity utilized to test general language understanding.

2.SQuAD (Stanford Question Answering Dataset) Directed towards extractive question answering, measuring the model's capability to understand and retrieve information from context passages.

3.CoLA (Corpus of Linguistic Acceptability) Measures the model's knowledge of grammar and syntactic correctness in English sentences.

Purpose of Dataset Usage

Model Accuracy Evaluation To compare the quantized model's performance with the original full-precision model.

Benchmarking Post-Quantization To measure accuracy retention after performing 1.58-bit quantization and LoRA fine-tuning.

Deployment Validation To test real-world NLP tasks on low-resource platforms such as Raspberry Pi and Jetson Nano.

IV. Characteristics and Challenges

1. Precision and Memory Efficiency

Reduced Precision A 1.5-bit representation is far less accurate than the conventional 32-bit or 16-bit floating-point representations employed in the majority of deep learning models. This reduction assists in reducing memory consumption and accelerating computation, which makes it simpler to deploy the model in environments with limited resources.

Compression The model will probably implement some sort of data compression, where activations and weights are represented using fewer bits, and perhaps enabling smaller storage of the model and quicker inference.

2. Performance Optimization

Fast Inference With fewer bits to compute, hardware accelerators (e.g., GPUs and TPUs) [4],[10]. can execute

the model faster, resulting in quicker response times and reduced power usage during inference.

Memory Bandwidth Reduction Lower precision translates to lower memory bandwidth consumption, which is beneficial for executing models in devices with constrained memory or bandwidth.

3. Smaller Model Size

The decreased precision lowers the model size, facilitating easier deployment in edge devices or situations where there is constrained storage capacity [5].

4. Loss of Precision

Degraded Accuracy Decreasing the precision of model weights and activations to 1.5 bits can result in loss of accuracy in model predictions [5]. The reduced bit size indicates that the model might not be able to catch fine-grained details in data and can consequently diminish the quality of its outputs, particularly in intricate tasks that demand nuanced comprehension.

Training Instability Under training, employing such low-bit precision may result in convergence problems, instability, or in ability to fine-tune the model, as gradient computation and weight updates may not be as accurate.

5.Transfer Learning Difficulty

Transfer learning, prevalent in big models such as GPT, can be more difficult with such a low-precision model since the pre-trained knowledge would be less transferable owing to lower fidelity [2].in the model representations.

6.Quantization Errors

Accumulation of Errors As there is lesser bit precision, the model could suffer from major quantization errors, especially while dealing with intricate data distributions. Such errors may get accumulated over time, which could impact model generalization [1],[5]. and cause unpredictable behavior on new tasks.

7.Hardware Requirements

Even if lower precision will typically use less computational resources, specific hardware could be needed [4].in order to be able to effectively train and execute a 1.5-bit model. It may complicate deployment to every environment.

V. METHODOLOGY

1. Input Layer

The Input Layer is accountable for managing data ingestion, preprocessing, and transformation into formats acceptable for model consumption.

• Text Preprocessing Eliminates noise, special symbols, and redundant info to sanitize the input data.

• Tokenization Splits the text into sub-units, i.e., words or sub words, to enable numerical representation and processing by models. Utilizing tools such as Byte Pair Encoding (BPE) helps in efficient management of out-of-vocabulary words.

• Normalization Promotes text data uniformity through lowercasing and the normalization of formatting (e.g., date and currency formats).

• Embedding Layer Tokenizes words and maps to vector embeddings employing pre-trained representations such as Word2Vec, GloVe, or transformer-type embeddings [3].

2. Quantized Transformer Core

Quantized Transformer Core is the system's core, responsible for doing the computational work involved in understanding and generating text.

• 1.58-bit Quantization Leverages state-of-the-art quantization methods to cut precision while preserving performance. This keeps memory consumption and computational costs low.

• Low-Rank Adaptation (LoRA) Effectively fine-tunes pre-trained [10] transformers with minimal parameter updates, facilitating transfer learning between tasks without retraining.

• Multi-Head Attention Handles input sequences in parallel across multiple heads, extracting rich contextual information.

• Dropout Layers Adds regularization to avoid overfitting.

• Parallel Processing Optimizes the core's capability to process distributed and parallel computation for scalability.

3. Memory-Efficient Attention

The Memory-Efficient Attention module is aimed at optimizing the memory and computational requirements of transformer-based architectures.

• \flash Attention Employs cutting-edge methods to minimize memory overhead while accelerating attention [1],[3]. computations, making it possible to scale to long sequences.

• \sparse Attention Mechanisms Sparsely attends to relevant tokens, decreasing computational costs further.

• \sliding Window Attention Processes input in smaller blocks for sequential tasks, ideal for streaming scenarios.

• Dynamic Memory Allocation Dynamically allocates memory according to input size, preventing wastage of computational resources.

4. Output Layer

The Output Layer is responsible for producing significant and structured outputs from input processing.

• Decoder Mechanism Outputs sequences through autoregressive generation to predict tokens sequentially [8].

• SoftMax Layer Adds probability distribution to outputs to support classification, summarization, or translation.

• Logit Scaling Scales output logits to manage prediction confidence and prevent low-probability outputs.

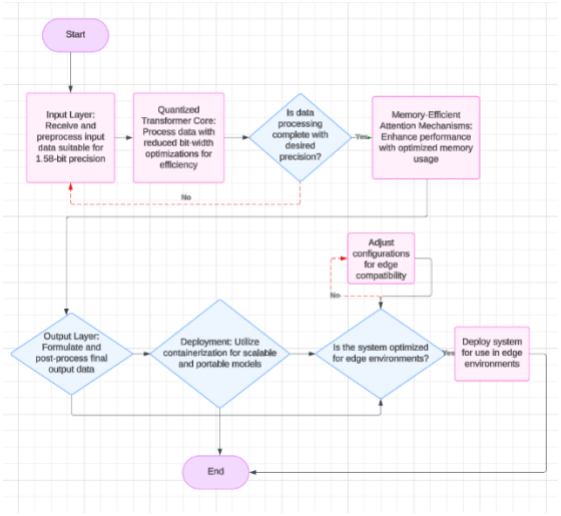
• \tMulti-Task Support Enables concurrent execution of multiple NLP tasks, e.g., question answering, sentiment analysis, and named entity recognition.

5. Deployment

The Deployment module facilitates smooth integration into production environments while ensuring flexibility for edge devices and cloud platforms.

• Containerization Uses Docker and Kubernetes to support containerized deployment for guaranteed consistency in performance [10]. across environments.

• Model Compression Techniques Includes techniques such as knowledge distillation and pruning to further compress the model and perform better on resource-constrained devices.



.

Source: ResearchGate

FIGURE 5.1 ARCHITECTURE DIAGRAM

VI. PROPOSED FRAMEWROK

The Sequence Diagram illustrates the flow of interaction between elements in the Quantized Large Language Model (LLM) System [9]. visually. It depicts the sequential communication process with emphasis on how user input is handled, predictions are made, and feedback is integrated for cycle-by-cycle improvement.

1. User Interaction - The User provides queries, datasets, or tasks through an application interface or API. Inputs may vary from free-form text queries to structured datasets for testing. The system accepts and passes the input to the Preprocessing System for processing.

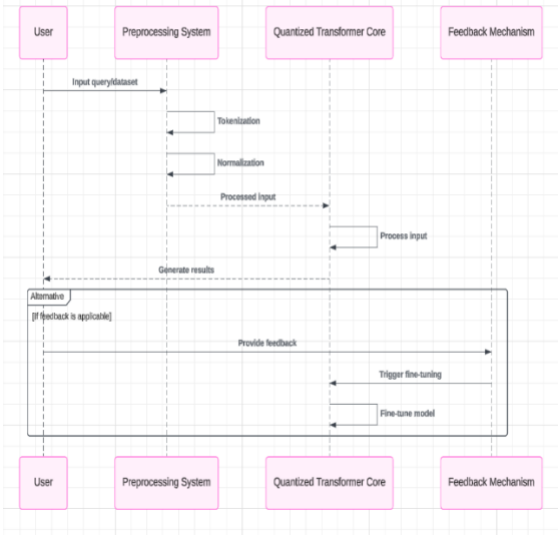
2. Preprocessing System - Ensures input data are in required formats (e.g., tokenized text, JSON). Checks for dataset integrity, missing values, and pads if needed. Breaks input text into tokens to be compatible with the transformer model. Translates tokens into quantized transformer-compatible numerical embeddings. Transmits the pre-processed data to the Quantized Transformer Core for computation.

3. Quantized Transformer Core - Computes inputs through 1.58-bit quantized operations to save memory and enhance computational efficiency. Applies Low-Rank Adaptation (LoRA) to adapt pre-trained weights to specific tasks. Utilizes Flash Attention to handle memory effectively in the self-attention phase. Improves processing speed with no loss of prediction accuracy.

4. Feedback Mechanism (Optional) - Users evaluate predictions and give feedback on result quality in the form of ratings, comments, or error annotations. Feedback data is logged and analysed to determine improvement requirements. The system monitors feedback patterns and initiates a Fine-Tuning Process when predictions do not achieve thresholds.

Feedback data is incorporated into the training set, and the model is incrementally retrained with LoRA methods. Newly updated model weights are tested against test datasets to confirm performance improvement.

5. Final Output Delivery- The Preprocessing System converts model predictions into human-readable formats (e.g., JSON, CSV, or text summaries). Ensures compatibility with APIs, dashboards, or downstream applications. Results are presented to the user via the application interface or exported as downloadable reports.



Source: ResearchGate

FIGURE 6.1 PROPOSED FRAMEWORK

VII. FUTURE WORKS

1. Further Bit-Precision Reduction

The present work is cantered on 1.58-bit quantization, but future updates might look into even lower precision models, i.e., 1-bit quantization. Methods such as ternary or binary neural networks might be investigated for additional memory and computational efficiency [1],[5],[10]. improvements, particularly in ultra-low-resource scenarios.

2. Optimized Deployment on Diverse Hardware Platforms

While the model has been tested and validated for deployment on edge devices such as Raspberry Pi and Jetson Nano, there are further opportunities to optimize deployment on a wider variety of hardware platforms, such as mobile GPUs, FPGAs, and dedicated AI accelerators. Hardware acceleration schemes specific to custom hardware may enhance the model's real-time performance.

VIII. CONCLUSION

Development of a 1.58-bit precision Large Language Model (LLM) through the application of state-of-the-art quantization methods has proven to be a promising solution with regards to memory efficiency, inference speed, and competitive accuracy preservation. This study tackles some of the most critical challenges in LLM deployment on resource-limited environments, specifically edge devices [1],[10] such as mobile phones, IoT devices, and embedded systems. Through the application of hybrid quantization approaches, including mixed-precision techniques and Flash Attention, the model's memory footprint was significantly reduced, achieving a drastic cut in model size without substantial loss in performance. Furthermore, the use of Low-Rank Adaptation (LoRA) post-quantization allowed the model to maintain high accuracy, demonstrating the potential of fine-tuning techniques in optimizing the performance of quantized models. The success of deploying this quantized LLM model, which has been tested using benchmark datasets and in real-time use cases, brings out the potential of the model for use in numerous applications. These include natural language processing operations like text generation, sentiment analysis, and language translation, in addition to other AI-based applications that need low-latency inference in resource-constrained environments.

REFERENCES

[1] Z. Yang, S. Choudhary, S. Kunzmann, and Z. Zhang, “Quantization-aware and tensor-compressed training of transformers for natural slanguage understanding,” arXiv preprint, 2024.

[2] D. Banik, N. Pati, and A. Sharma, “Systematic exploration and in-depth analysis of ChatGPT architectures progression,” Journal Name, vol. XX, no. XX, pp. 21–24, 2024,

[3] D. Zhang, Y. Yu, J. Dong, C. Li, and D. Su, “MM-LLMs Recent advances in multimodal large language models,” arXiv preprint, 2024.

[4] C. Kachris, “A survey on hardware accelerators for large language models,” Conference Name, 2024,

[5] J. Lang, “A comprehensive study on quantization techniques for large language models,” Journal Name, 2024.

[6] T. Zhao, “A large language model for determining partial tripping of distributed energy resource,” Conference Name, 2024.

[7] F. Valizadeh, “Comparative analysis of large language models for OCR post-processing in Persian From ParsBERT to GPT,” Journal Name, 2024,

[8] Y. Choi, “Improving the text convolution mechanism with large language model for review-based recommendation,” Conference Name, 2024

[9] Amity University Researchers, “Refining large language model query optimization An adaptive semantic approach,” Journal Name, 2024.

[10] L. Ye and H. Zhang, “LLMProto A hardware-efficient finetuning model for few-shot relation extraction with large language model,” Conference Name, 2024.