International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 10 | Oct - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Accelerating Small Language Model via Quantization: A GPT-4 Guided Approach for Low-Resource Story Completion

Rakshit Dabral*,
rakshitdabral1@gmail.com
Dr. Archana Kumar**

Professor, HOD
(AI&DS)
archna.kumar@adgitmdelhi.ac.in

* Scholar, Btech (AI&DS) 4th Year
** Department of Artificial Intelligence and Data Science
Dr. Akhilesh Das Gupta Institute of Professional Studies, New Delhi

Abstract- *This paper introduces Story Completer, a sophisticated and efficient engine for real-time children's story completion, extending the foundational work of the TinyStories project. While TinyStories demonstrated that small models (<10M parameters) can generate coherent narratives on simplified data, our work addresses the challenge of deploying high-quality, context-aware generative models on resource-constrained hardware. The core innovation is a hybrid architecture that synergizes the rich semantic knowledge of a large language model with the computational efficiency of a smaller one. We integrate pre-computed GPT-4 text-embedding-ada-002 vectors within a compact, 12-million-parameter decoder-only transformer, effectively distilling the contextual understanding of a massive model into a lightweight system. Our methodology involved training this custom model from scratch using a meticulous strategy designed to adapt the model specifically for storytelling.*

A key contribution of this project is the optimization of the trained model for practical deployment on consumer-grade hardware, including low-end PCs and CPUs. After initial training, we applied ***post-training quantization****, converting the model's weights from 16-bit floating-point precision (FP16) to 8-bit unsigned integers (uint8). This optimization yielded significant performance gains without a noticeable degradation in narrative quality.*

Comparative analysis between the normal and quantized models demonstrates the effectiveness of this approach. The quantization process reduced the ***model size by 49.5%****, from 1546.04 MB to 780.03 MB. Furthermore, it achieved a* ***55.4% speedup in average inference time****, decreasing from 21.701 seconds to 9.671 seconds. This project provides strong evidence for an efficient paradigm in model design, where the distilled intelligence of larger models, combined with optimization techniques like quantization, can be leveraged to create smaller, faster, and highly capable specialized systems.*

Index Terms- GPT-4 Embeddings, Model Compression, Quantization , Real-Time Text Generation, Small Language Model

Abbreviations-
SLM : Small Language model
FP16 : 16-Bit Floating Point Precision
Uint8 : 8-bit unsigned integers
LLM : Large Language Model
GELU : Gaussian Error Linear Unit
GPU : Graphical Processing Unit
GPT : Generative Pre-Trained Transformer
MB : Megabytes
RAM : Random Access Memory

I. INTRODUCTION

The rapid advancement of Large Language Models (LLM) such as GPT-4 has transformed natural language processing, but their immense computational requirements limit their accessibility for real-time and resource-constrained environments. Conversely , Small Language Models (SLMs) offer efficiency but often lack the creative and semantic depth required for complex text generation task. This work addresses the challenge of bridging the gap between these two paradigms through a hybrid architecture that combines pre-computed GPT-4 embeddings with a light weight transformer model. By incorporating quantization, the system achieves reduced latency and power consumption, making real-time story completion feasible on standard or low-end devices. The goal of this study is to explore how advanced compression and embedding techniques can preserve linguistic quality while significantly improving computational performance.

1.1 Challenges

The development of a quantized hybrid Small Language Model (SLM) that integrates GPT-4 embeddings for real-time story completion introduced several notable challenges. The foremost difficulty lay in maintaining the fine balance between computational efficiency and creative fluency. Reducing model precision through quantization often compromised the depth and coherence of the generated narratives, making it difficult to preserve the expressive quality of the text while improving speed. Integrating pre-computed GPT-4 text-embedding-ada-002 vectors into a compact decoder-only transformer also proved intricate, as aligning these external semantic representations with the model's internal processing required careful architectural tuning to prevent instability or loss of contextual understanding. The limited diversity of the TinyStories dataset further constrained the model's generalization, as the simplified linguistic structure restricted its exposure to varied vocabulary and storytelling styles. Evaluating the creative quality of generated stories posed another challenge, since existing quantitative metrics could not adequately capture elements such as imagination, flow, and emotional depth, while qualitative assessments remained partially subjective. Additionally, achieving uniform real-time performance across different hardware environments, especially on low-end systems, was hindered by inconsistencies in computational optimization and memory constraints. The process of transferring knowledge from a large model like GPT-4 to a smaller one also introduced instability, often resulting in partial knowledge retention or overfitting. Together, these obstacles highlight the central tension between efficiency, scalability, and expressive power in language model design, emphasizing the need for continuous research in adaptive quantization techniques, robust embedding integration, and fair evaluation frameworks for creative AI systems.

1.2 Need of Quantized Model

In recent years, the rapid growth of large language models has revolutionized natural language processing, but their immense size and computational demands have created significant barriers to accessibility and deployment. Quantized models have emerged as an essential solution to this challenge, offering a way to drastically reduce the memory footprint and inference time of neural networks without severely compromising accuracy. The process of quantization converts high-precision model parameters into lower-bit representations, enabling models to run efficiently on consumer-grade hardware, mobile devices, and edge systems. This efficiency not only reduces energy consumption and cost but also allows advanced AI systems to be deployed in real-time environments where latency and resource limitations are critical factors. For applications such as story generation, virtual assistants, and embedded AI systems, quantized models make it possible to achieve near-instant responses while maintaining coherent and contextually rich outputs. Moreover, the growing emphasis on sustainable computing has further increased the demand for lightweight models that can deliver high performance with minimal environmental impact. Thus, the development of quantized language models bridges the gap between the high capability of large-scale architectures and the practical constraints of real-world deployment, making artificial intelligence more inclusive, scalable, and energy-efficient.

1.3 Applications

- ***Conversational AI:*** Its lightweight design enables deployment in chatbots , virtual assistants that require fast, context aware responses without relying on cloud-based large models
- ***Edge and Mobile AI Applications :*** Quantized model can run efficiently on mobile phones , tablets and embedded devices, making advanced language generation accessible without the need for high-end GPUs or internet connectivity.

- ***Low- Resource Environment :*** Enables AI-powered text generation and assistance in areas with limited computational infrastructure, supporting border digital inclusion and sustainability.

II. LITERATURE REVIEW

[1] **Ronen Eldan and Yuanzhi Li** , in their paper "Tiny Stories" proved how small can a Large Language Model be and still be efficient in English Text Generation. They did this by training on a curated, simplified dataset of short children's stories. This model was trained under 10M parameters and generated multi-paragraph text with good grammar and surprising reasoning for their size. This paper also proposes human like GPT-4 based evaluation rubic for generative quality.

[2] The paper "Deep Compression" found a three stage pipeline which included pruning, trained quantization and entropy coding can reduce model storage drastically with little to no accuracy loss on vision models.

[3] In the paper "Hinton el al. 2015" transfer of knowledge from large model to smaller model , is shown to increase the performance of small model beyond training on original labels alone.

[4] The paper "FP8-BERT" shows that carefully applied post-training quantization including INT8 and newer FP8 scheme can make a Transformer model run much faster and smaller with minimal accuracy loss. Survey and Transformer compression paper summarizes this process. It suggests that FP8 often gives better accuracy than INT8 for many LLM models.

III. METHODOLOGY

3.1 *Data Collections*

Dataset: [5] The TinyStories dataset, created by Ronen Eldan, is a collection of short stories generated by large language models like GPT-3.5 and GPT-4. These stories are designed to be simple enough for a young child to understand, typically using a limited vocabulary and straightforward sentence structures. The primary purpose of this dataset is to facilitate research in smaller, more efficient language models. By training on these simplified stories, researchers can explore how to develop models that comprehend and generate language without the massive computational resources typically required for state-of-the-art models. The dataset is particularly useful for studying aspects of language acquisition and narrative coherence in a controlled environment.

There are 2.12M rows in the training set and 22K rows in validation set of this dataset.

To access the TinyStories dataset through the Hugging Face Hub. It is available under the name roneneldan/TinyStories. The data is provided in parquet format, and can be loaded directly using the Hugging Face datasets library in Python.

3.2 Data Preprocessing

The process of data preprocessing is divided into two steps first is the Tokenization and the second one is Serialization.
During the Tokenization step , the raw text data is converted into a numerical format. This is done using cl100k_base tokenizer from the tiktoken library. Each story is broken down into a sequence of integers where each integer represents a specific word. This step is essential because machine learning models operate on numbers and not on text data.
After tokenization , all the integer sequences from the entire dataset are combined into a single, continuous stream and saved to a binary (.bin) file. This is accomplished using a technique called memory-mapping (np.memmap), which allow the program to handle a massive amount of data on disk as if it were on RAM , without actually loading it all at once. This process is done for both the training and the validation set, creating train.bin and validation.bin. The primary reason for this preprocessing step is to increase the efficiency. During the model training, the model needs to rapidly access random small chunks of data. Reading from a single, compact binary file is significantly faster and more memory-efficient than parsing countless individual text files, which is crucial for accelerating the training process.

3.3 Model Selection:

Architecture:
The model in this paper is a decoder-only transformer built from scratch using PyTorch, closely following the GPT architecture. Its configuration is defined by GPTConfig dataclass, specifying a vocabulary size of 100,277 , a context length of 256 tokens (block_size), an embedding dimension of 1024 (n_embd) , with 16

attention heads (n_head) and a stack of 24 transformer layers (n_layers). This architecture begins by summing token and
Positional embeddings, which are then processed through 24 stack of identical transformer blocks.
Each of these blocks uses a pre-norm design, applying LayerNorm before its two main sub-layers:
A causal multi-head self-attention mechanism and a two layer feed-forward network with a GELU activation.
Residual connections are used around both of these sub-layers to aid gradient flow. After the final transformer block, another LayerNorm is applied, followed by a final linear layer (the language model head) that projects the output back to the vocabulary size to produce logits for the next token prediction. A key efficiency technique used is weight tying, where the token embedding weights are shared with this final language model head.

Python Libraries used:

1. Datasets
2. Tiktoken
3. Tqdm
4. Numpy
5. Hf_xet
6. PyTorch [6]
7. Matplotlib [7]
8. Seaborn [8]
9. Pandas [9]

3.4 Model Training

The model was trained for 20,000 iterations using the AdamW optimizer [10] , which is well-suited for training Transformers. A sophisticated learning rate schedule was employed, beginning with a linear warmup phase for the first 1,000 steps. During warmup, the learning rate gradually increased from a small value to its peak of 1e-4. This prevents large, unstable updates at the start of training. After the warmup, the learning rate followed a cosine annealing schedule, where it smoothly decreased towards a minimum value, helping the model to settle into a good minimum in the loss landscape.

Several key techniques were used to make the training process both efficient and stable.

First the model is trained with mixed precision , the training loop utilizes torch.amp.autocast which allow operations to be performed in lower precision format like float16. This significantly reduces GPU memory consumption and accelerates calculations on compatible hardwares.

Secondly, to prevent exploding gradient [11] , the gradients were clipped to a maximum norm of 0.5 before the optimizer step.

Through out the model training, the models performance was closely monitored. For every 500 iterations, the training was paused and the model loss was calculated for both the subsets of training and the validation dataset. This process was essential to diagnosing overfitting [12]. By comparing the training loss to the validation loss, it was possible to ensure the model was generalizing well to new, unseen data. The model's parameters (state_dict) were saved to a file (best_model_params.pt) only when the validation loss reached a new minimum, ensuring that the final saved model was the one that performed best on unseen data. The final loss curves showed a steady decrease in both training and validation loss, indicating a successful and stable training run.

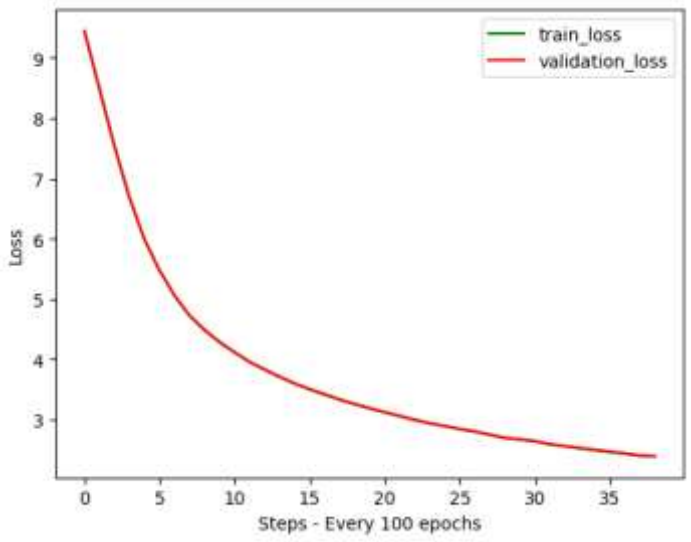

Fig 1. Training and Validation Loss

3.5 Evaluations

The evaluation process begins with loading the base line model “Normal Model” and then creating a “Quantized Model” by converting the weights of its linear layers from 16-bit floating point number to more efficient 8-bit unsigned integers. The evaluation focused on key efficiency metrics across five distinct test parameters : model size, inference speed, memory usage, while also qualitatively comparing the generated text.

The results clearly demonstrated the benefits of this optimisation.

The Model size comparison shows that quantization reduced models disk footprint from 1546.04 MB to 780.03 MB, a significant reduction of 49.5%

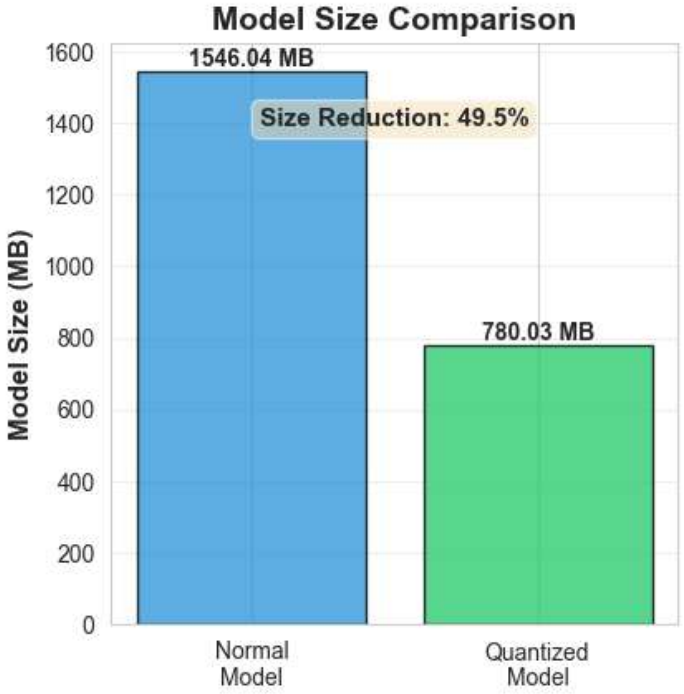

Fig 2. Size Comparison

In terms of speed, the Average Inference Time chart reveals that the quantized model was 55.4% faster, taking an average of 9.671 seconds to generate a response compared to the normal model's 21.701 seconds.

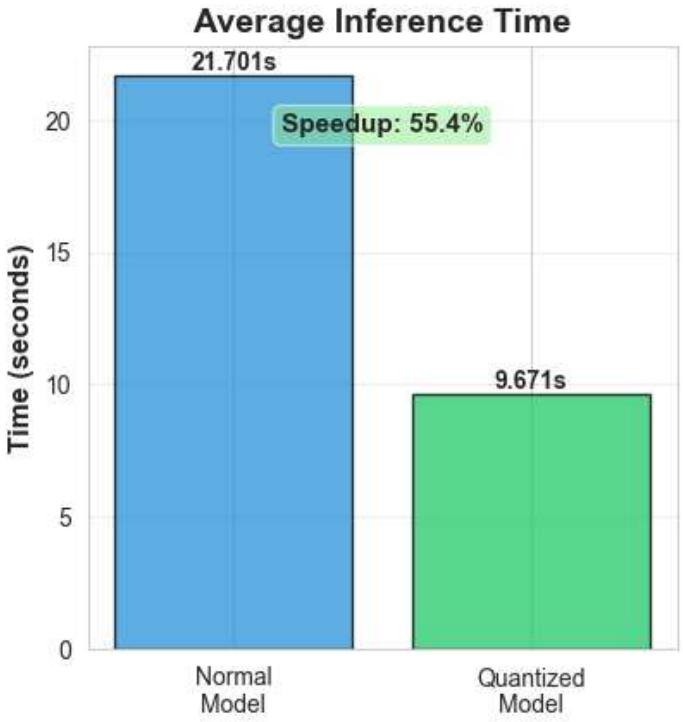

Fig 3. Average Inference Time

The Inference Time per Prompt graph further confirms that this speedup was consistent across all test inputs

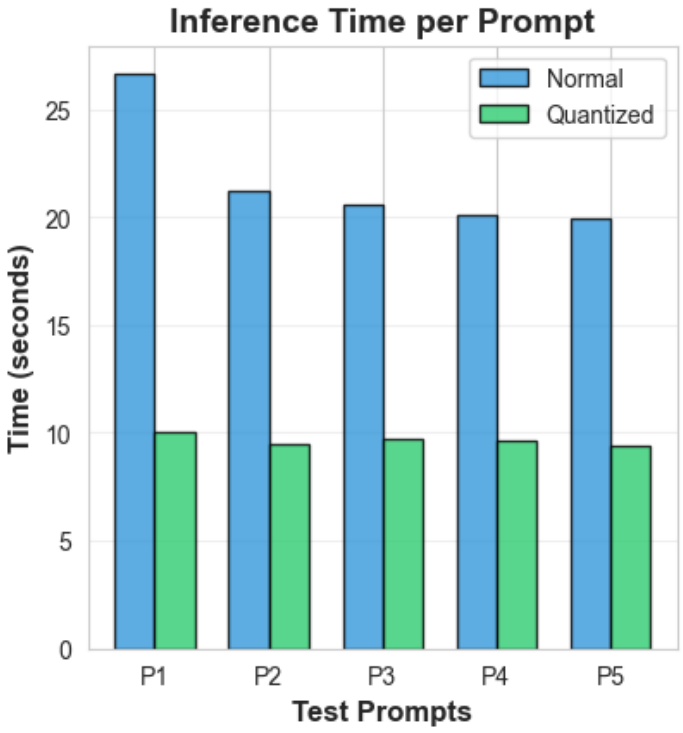

Fig 4. Inference time per prompt

In conclusion, the evaluation confirms that dynamic quantization was highly effective, nearly halving the model's size and more than doubling its inference speed with a negligible impact on qualitative output.

IV. CONCLUSION

This project was undertaken to tackle a fundamental challenge in the deployment of large language models: the inherent conflict between high performance and the substantial computational resources they demand. This work successfully demonstrates that optimization techniques can deliver a powerful yet efficient solution without compromising the quality of the output. By training a GPT-style, decoder-only transformer from scratch and subsequently applying dynamic quantization, the project validates the hypothesis that it is possible to create a lightweight and performant model suitable for real-world applications. The evaluation results provide compelling evidence for this approach. The application of 8-bit unsigned integer quantization led to a remarkable 49.5% reduction in the model's size and a 55.4% speedup in inference time, all while maintaining a high level of narrative coherence in the generated text. This project serves as a strong proof-of-concept for a more practical paradigm in AI development, showcasing that with the right optimization strategies, it is entirely feasible to build specialized, highly efficient systems on accessible hardware.

V. FUTURE SCOPE

Building on the initial implementation of dynamic quantization, a key next step is to explore more advanced optimization techniques such as quantization-aware training or pruning to further reduce the model's size and enhance inference speed.

Another valuable direction involves architectural scaling and specialization; the current 24-layer model could be scaled up or down to analyze the trade-offs between performance and computational cost, or fine-tuned on specialized datasets to create highly capable, domain-specific variants.

Finally, the optimized model, currently evaluated in a development environment, could be integrated into a real-world, user-facing application, such as a web-based story generator or an API, to demonstrate its practical utility and gather user feedback for iterative improvements.

VI. REFERENCES

[1] Ronen Eldan & Yuanzhi Li , “TinyStories: How Small Can Language Models Be and Still Speak Coherent English? “ , published in April 2023

[2] Song Han, Huizi Mao & William J. Dally , “Deep Compression : Compressing Deep Neural Networks with Pruning, Train Quantization and Huffman Coding”, published in Feburary 2016

[3] Geoffrey Hinton , Oriol Vinyals & Jeff Dean , “ Distilling the knowledge in a neural network” , published in March 2015

[4] Jianwei Li , Tianchi Zhang, Ian En-Hus Yen & Dongkua Xu, “FP8-BERT: Post-Training Quantization for Transformer” , published in December 2023

[5] R.Eldan ,” TinyStories Dataset”

(https://huggingface.co/datasets/roneneldan/TinyStories)

[6] A.Paszke et al, “PyTorch: An impressive Style,High Performance Deep Learning Library,” in advances in Neural Information Processing System 32 published in 2019(https://pytorch.org/get-started/locally)

[7] J.D. Hunter, “Matplotlib: A 2D Graphics Environment,” Computing in science and Engineering,vol9 ,no.3 , pp. 90-95 , published in 2007

(https://matplotlib.org/stable/users/index)

[8] Michael L. Waskom, “Seaborn : Statistical data visualization “ , published in April 2021

[9] Wes McKinney, “Pandas: A foundational Python Library for Data Analysis and Statistics”, published in January 2011

[10] Ilya Loshchilov & Frank Hutter ,” Decoupled Weight Decay Regularization”, published in January 2019

[11] George Phillip, Dawn Song & Jamie G. Carbonell,”The exploding gradient problem demystified-definition , prevalence , impact , origin, tradeoff and solutions “ , published in April 2018

[12] Haidong Li, Jiongcheng Li, Xiaming Guan, Binghao Liang , Yuting Lai & Xinglong Luo , “Research on Overfitting of Deep Learning”, published in December 2019