Quantization and Deployment of Qwen2.5 on ARM CPUs

*Note: Sub-titles are not captured in Xplore and should not be used

Team Name：冬瓜皮
Participant Names：Nengzhen Fang
Affiliations：Fuzhou University

I. INTRODUCTION

The rapid advancement of large language models has significantly enhanced natural language processing applications, including text understanding, text generation, sentiment analysis, machine translation, and interactive question-answering. However, the enormous parameter sizes of modern LLMs, often ranging from billions to trillions of parameters, pose substantial challenges for efficient deployment on edge devices. Furthermore, the growth rate of model parameters surpasses hardware performance improvements, necessitating innovative approaches to optimize LLM inference under constrained hardware resources.

To address these challenges, academia and industry have increasingly focused on software-hardware co-optimization strategies, including model compression, data flow optimization, and efficient operator scheduling. Given the rising demand for LLM deployment on edge devices and the widespread adoption of Arm-based CPUs in these environments, this competition aims to explore the systematic optimization of LLMs for efficient inference on Armv9 architecture CPUs.

The 2025 AICAS competition is designed to promote research on LLM inference optimization on Arm-based CPU architectures. Specifically, participants will deploy and optimize the Qwen2.5 large language model on the Alibaba Cloud platform, utilizing the Armv9-based Yitian 710 processor. The competition encourages innovative techniques to enhance inference efficiency while maintaining model accuracy, leveraging hardware-specific optimizations such as BF16 computation, vectorized matrix multiplications, and the Arm Compute Library.

To tackle this challenge, our approach primarily revolves around the following three strategies:

1. Quantization Techniques: We leverage quantization methods to reduce the memory footprint and computational complexity of the model. For the transformer blocks, we employ the AutoRound[1] technique, which facilitates optimal weight rounding, while for the embedding layers, we utilize the OMSE[2] approach to minimize quantization errors.

2. Deployment with llama.cpp: We utilize the llama.cpp deployment framework to efficiently run large language models on CPU platforms. This framework is particularly well-suited for optimizing inference performance on Arm architectures, as it takes advantage of hardware-specific features such as vectorized computations and low-level performance tuning.

3. Parameters Tuning: We conduct extensive parameter tuning tailored to the specific hardware characteristics of the Yitian 710 CPU. This process involves experimenting with various configurations to identify the optimal balance between inference speed and memory usage, ensuring that our deployment is both efficient and effective.

Through the combination of these techniques, we aim to achieve high-performance LLM inference on Arm-based CPU platforms, contributing to the advancement of efficient LLM deployment in edge and resource-constrained environments.

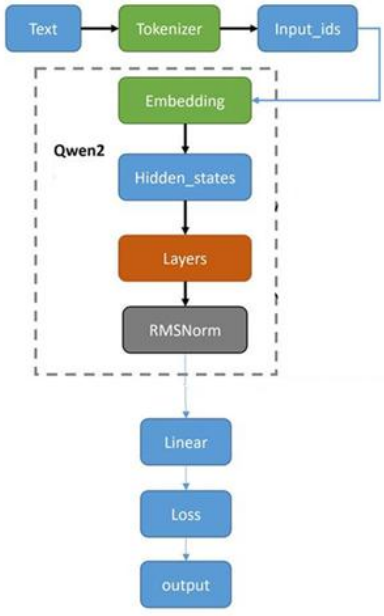

Figure 1. The structure of Qwen2.5 model.

II. METHODS

In this section, we primarily introduce the quantization techniques employed in our approach. The rounding optimization technique[3], originally proposed by Qualcomm, has been widely adopted in the field of computer vision model quantization. It formulates the rounding problem in quantization as an unconstrained binary quadratic optimization problem:

$$\underset{v}{\operatorname{argmin}} ||wx - \widetilde{\boldsymbol{w}}x||_F^2 + \lambda f_{reg}(v)$$

$$\widetilde{\boldsymbol{w}} = s \cdot clip(\left\lfloor \frac{W}{s} \right\rfloor + h(v),\ n,\ p)$$

$$f_{reg}(v) = \sum_{i,j} \left(1 - \left|2h(v_{i,j}) - 1\right|^{\beta}\right)$$

$$h(v) = clip(\sigma(v)(\zeta - \gamma) + \gamma,\ 0,\ 1)$$

where w represents the original model weights, and $\hat{w}$ represents the quantized model weights.

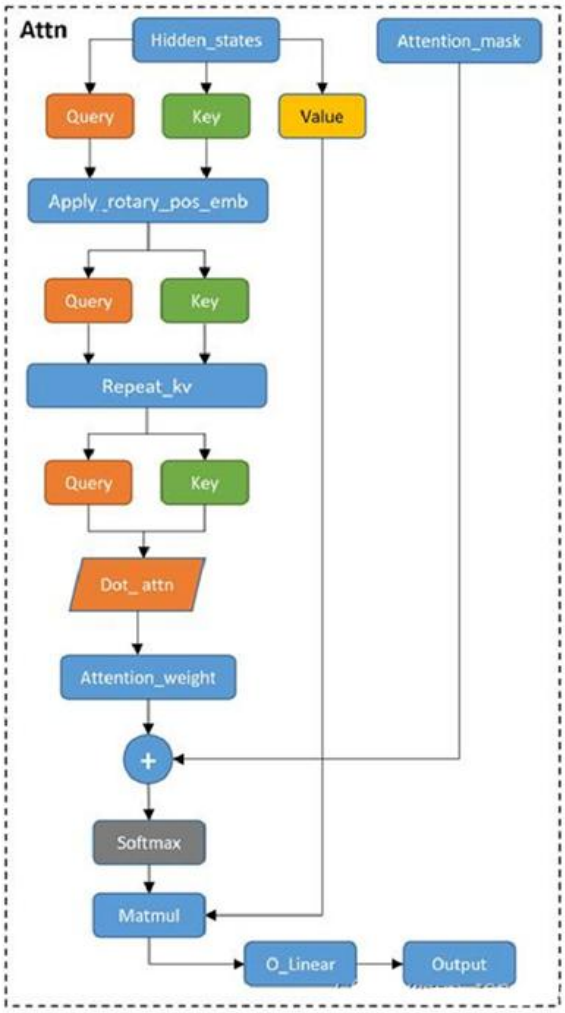

Figure 2. The transformer block of Qwen2.5 Model.

The advantage of this method lies in its ability to mitigate the accumulation of quantization errors during the quantization process, preventing the quantization error from being excessively amplified in subsequent computations. However, as it relies on gradient backpropagation, it requires approximately 20,000 iterations using the Adam optimizer for each layer. Moreover, since the rounding optimization is performed at the element level, the optimization process itself is resource-intensive and time-consuming. This makes it challenging to apply to LLMs, where the number of parameters is significantly greater compared to CNNs with parameter sharing.

Recently, researchers proposed AutoRound[1] to address the rounding optimization problem for LLMs. It uses symbolic gradient descent to optimize the rounding process, achieving optimal rounding in just 200 iterations. This significantly reduces the optimization cost of rounding on LLMs and achieves state-of-the-art accuracy.

This approach helps preserve model accuracy while reducing the memory footprint and computational burden during inference.

OMSE is a widely-used quantization method in computer vision models, known for its ability to minimize quantization error by optimizing the scale factor through a grid search process. However, due to the computational cost associated with this search process, OMSE has not been extensively applied in LLMs.

In our approach, OMSE was particularly effective for quantizing the embedding layer. Unlike the linear operations in transformer blocks, the embedding layer involves a table lookup process, making it incompatible with optimization algorithms such as GPTQ[4] and AutoRound[1]. Additionally, since we employed the llama.cpp backend, the Q8_0 format imposes a fixed groupsize of 32. This limitation renders techniques like AWQ[5], which adjust weight magnitudes per channel, ineffective within this deployment environment.

By default, llama.cpp performs weight quantization using a simple min-max approach. However, as Figure 3. shows, OMSE[2] has demonstrated that the quantization function is non-convex and requires a grid search process to identify the optimal scale. To address the computational bottleneck of OMSE, we implemented a custom kernel function using GPU parallel processing. This significantly accelerated the grid search process, enabling the practical application of OMSE for embedding layer quantization in our LLM deployment.

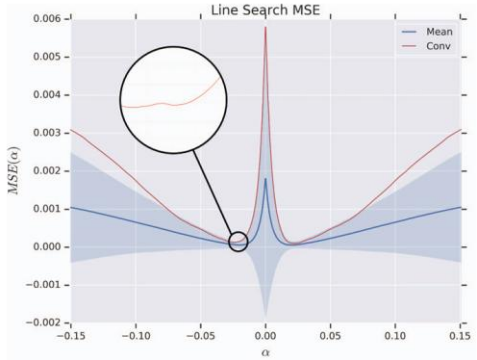

Figure 3. MSE as a function of the scaling factor α.

Combining AutoRound for transformer blocks and OMSE for the embedding layer with 8-bit quantization allowed us to achieve a balance between model accuracy and inference efficiency on the ARM cloud platform.

III. Results

We have developed an interface to export the model quantized with AutoRound into the GGUF format, so that llama.cpp's quantization no longer relies on the basic methods it provides. Additionally, we have written a script that allows for direct evaluation of the GGUF model. As a result, we directly perform accuracy evaluation on the quantized GGUF model file.

We evaluated the impact of quantizing the embedding layer on the model accuracy. Min-Max represents the quantization method defaulted by llama.cpp, while OMSE is the method we used. As results shown in Table 1, OMSE helps reduce the accuracy loss caused by embedding quantization.

Method	Arc Challenge	HellaSwag	PIQA
FP16	29.52	40.61	70.23
MinMax	29.35	40.60	70.18
OMSE	**29.43**	**40.69**	**70.29**

Table 1. Embedding layer quantization Results.

Subsequently, we evaluated the performance of the default llama.cpp quantization method and compared it with our approach that uses OMSE for quantizing the embedding layer and AutoRound for quantizing the transformer block. As Table 2. shows, the results demonstrate a substantial improvement with our method, achieving nearly lossless quantization.

To enhance the inference speed of the LLM model deployed via llama.cpp, we systematically optimized both the number of threads and the batch size. Increasing the number of threads fully exploits the advantages of multi-core parallel computing, enabling simultaneous execution of multiple tasks and thereby reducing the waiting time during individual inferences. Concurrently, enlarging the batch size permits the processing of a larger number of data samples per operation, effectively leveraging the CPU's vectorized computing capabilities and cache resources, which further enhances overall computational efficiency. It is noteworthy, however, that both adjustments lead to a significant increase in memory usage, necessitating a careful balance between performance gains and resource consumption.

Based on these considerations, the final experimental configuration on the Alibaba Cloud general-purpose ARM CPU platform with 16 vCPUs employed a batch size of 128 and 16 threads. This configuration achieved a considerable improvement in inference speed while maintaining adequate system resource allocation, thereby demonstrating the effectiveness of optimizing LLM model inference performance through the judicious adjustment of parallelism and batch processing under hardware performance constraints.

Method	Arc Challenge	HellaSwag	PIQA
FP16	29.52	40.61	70.23
Llama.cpp Q8_0	29.18	40.63	70.18
AutoRound w/o EmbedQuant	29.43	40.63	**70.34**
AutoRound w/ EmbedQuant	**29.43**	**40.77**	70.29

Table 2. Qwen2.5 8bit Quantization Results.

IV. REFERENCES

[1] Cheng, Wenhua, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, Kaokao Lv, and Yi Liu. "Optimize weight rounding via signed gradient descent for the quantization of llms." arXiv preprint arXiv:2309.05516 (2023).

[2] Choukroun, Yoni, Eli Kravchik, Fan Yang, and Pavel Kisilev. "Low-bit quantization of neural networks for efficient inference." In 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 3009-3018. IEEE, 2019.

[3] Nagel, Markus, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. "Up or down? adaptive rounding for post-training quantization." In International Conference on Machine Learning, pp. 7197-7206. PMLR, 2020.

[4] Frantar, Elias, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. "Gptq: Accurate post-training quantization for generative pre-trained transformers." arXiv preprint arXiv:2210.17323 (2022).

[5] Lin, Ji, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration." Proceedings of Machine Learning and Systems 6 (2024): 87-100.