IEEE AICAS 2025 Grand Challenge Technical Report

Hust\_Ody  
Zhiwei Zhou, Tong Hu  
Yi Li  
Huazhong University of Science and Technology

# Introduction

Generative models inspired by Transformer family have gained massive attention in the field of large language models (LLMs). As popularity rises, there’s a great interest in deploying LLMs on edge devices. However, the high computational and storage cost remains an obstacle to implementing LLMs at edge. The parameters of GPT3-175B occupy 326GB of memory when stored in a compact Float16 format, such a large model is unable to be allocated fully to commercial GPU memory, let alone the resource limited edge device. Moreover, the computational cost of LLM inference is still massive at edge, leading to unacceptable energy and latency consumption. Thus, it is necessary to implement optimization methods to alleviate computation and memory burden of LLMs, which is of great importance of LLM deployment at edge.

In this research, we deployed pre-trained Qwen2.5-0.5B-Instruct model on KV260 FPGA board with INT8 quantization. To further reduce computational cost, we accelerated vector-matrix computation using ZYNQ DSP. Generally, due to quantization and operator fusion, we achieved significant improvements in both efficiency and performance.

# Methods

A. Model quantization

In this paper, a post-training quantization (PTQ) method is presented for model compression. Here, symmetric quantization is implemented, where all the weights of attention layers and FFN layers are quantized into INT8, while the scaling factors, bias and RMSNorm are kept in Float32, since these layers are quite sensitive. During inference, the functions of vector-matrix multiplication (e.g. matmul) are modified accordingly to accommodate the quantization. It is noted that the vectors are quantified in batches, each group has a shared scaling factor. Group size of quantization batches is set at 64. The detailed quantization procedure is introduced at Algorithm 1. Here, each element w in a group is divided by its maximum w\_max, and quantized to INT8 format afterwards. The scaling factor here is set at w\_max/127 for each group.

B. Hardware Accelerator

At the inference stage of Qwen2.5 model, the vector-matrix multiplication (VMM) consumes much of the computation time and resources, as depicted in Fig 1. The VMM takes over 87% of the time in forward() function, suggesting that VMM is the core bottleneck of LLM inference.

Here, we developed a vector-matrix multiplication hardware acceleration kernel on FPGA. The size of VMM in the kernel is w(d,n) \* x(n). Here, d is set at 128, n is 896.

The VMM acceleration is performed in following stages, as shown in Fig. 2: i)Weight loading. The long communication latency between PL and DRAM would encumber the computation at PL, causing memory bottleneck. To minimize the loading latency, we first load all the inputs to BRAM and URAM at PL through axi. To fully exploit the bandwidth of axi channel, we loaded 2\*64 elements of w(128,896) in parallel from 2 axi ports (w0 and w1), maximizing axi bitwidth at 512. ii) Vector-matrix computation. In a iteration I, For each vector-vector multiplication, i.e. w(i,n) \* x(n), the multiplication is done in groups, so GS=64 multiplications are performed at same time. For each iteration i, the computation is done in pipeline, which further expands the parallelism of computation.

|  |  |
| --- | --- |
| **Algorithm 1**: Symmetric Quantization | |
| **Input:** Original weight parameter **W = (** , the group size **GS**.  **Output:** Quantized weight **q =** (, and its corresponding scaling factor **s =** (.  Q = [1..*n*];   1. **while** ( **do** 2. w\_max = max (; 3. scale = w\_max / 127; 4. Q[1,…,*n*]= (/scale; 5. ← scale; 6. **end while** 7. **return q** ← Q, **s**; | |
|  |

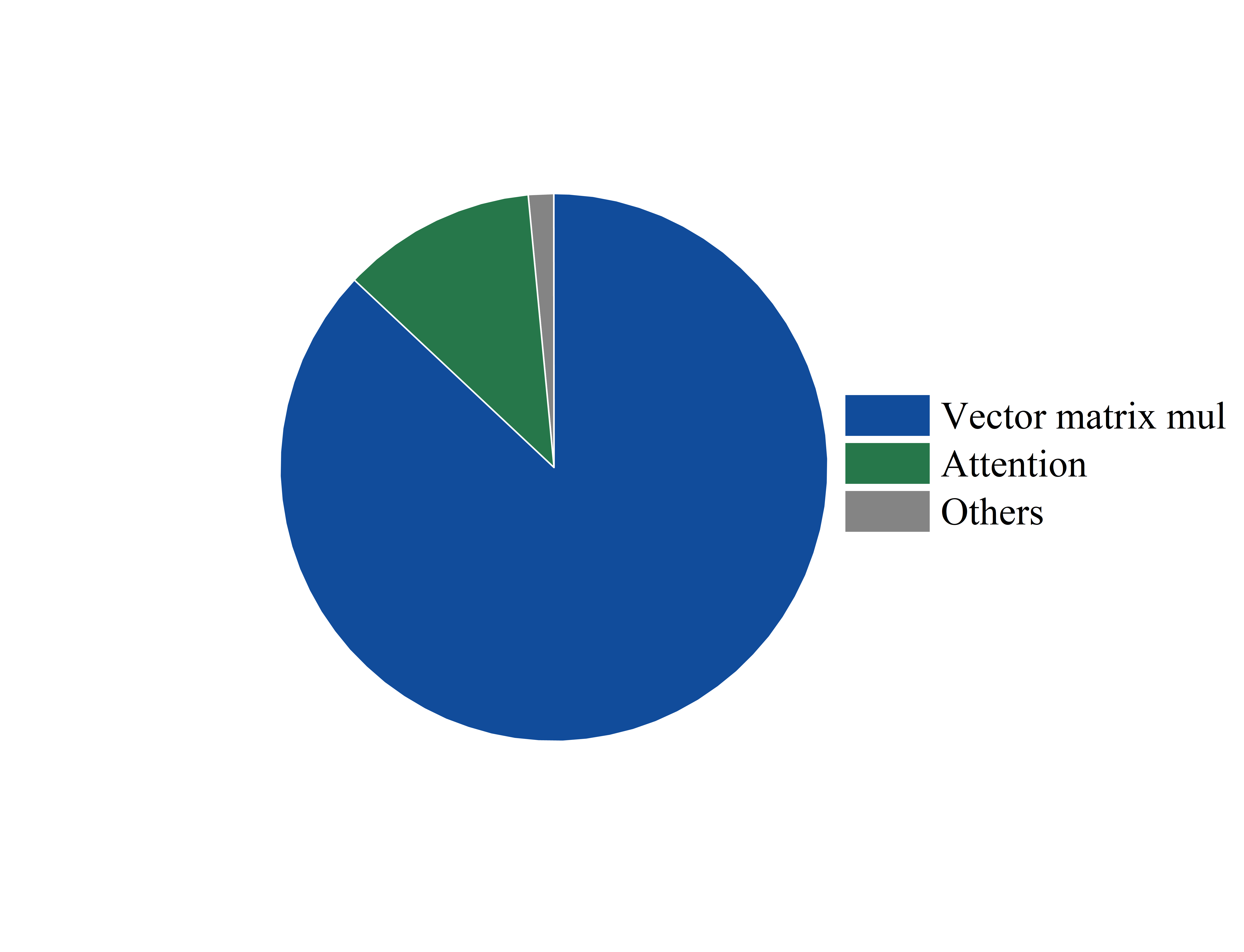


Fig. 1. Time breakdown of forward() during inference.

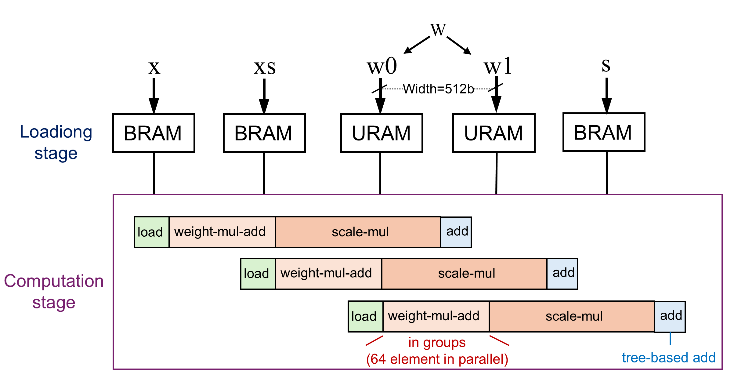


Fig. 2. VMM stages at PL.

# Results

We validated our PTQ and operator accelerator design on 3 aspects: accuracy , compression ratio , throughput and . The experiments set up are as follows: we use Qwen2.5-0.5B-Instruct as baseline model, where model inference is completed on C++ language file, similar to llama2c project.

**Hardware Kernel and application**. Our Hardware kernel can deal with input of (128,896) \* (896) VMM in parallel and pipeline. The kernel consumes 14 BRAM, 48 URAM, 488 DSP, 49424 FF and 51918 LUT. The kernel operates at 200MHz, and the total execution time is estimated to 7.7us. The Kernel is compiled on a customized vitis hardware platform, and is complied to .xclbin in vitis. We use XRT to run the kernel at the host application file (fpga\_runq.cpp), and pre-allocate the weights to XRT buffers during loading model transformer weights stage.

**Accuracy**. We evaluated the model’s performance on WNLI dataset, a popular nature language inference task. Upon 635 selected questions, our model achieves 51.81% accuracy.

**Compression ratio**. By quantization at int8, the model is compressed to 525164136 bytes, compression ratiois 0.469.

**Throughput**. The prefill rate and decode rate are 1.93 and 1.85 tokens/s, separately. It is noted that due to XRT limitations, the majority of time is consumed at “run.wait()”(~0.12ms) while the actual kernel execution time when running .cpp program is <0.014ms.

# Conclusion

In conclusion, our research successfully deployed a pre-trained Qwen2.5-0.5B-Instruct model on a KV260 FPGA board with INT8 quantization, achieving significant improvements in computational efficiency and memory usage. Through the use of post-training quantization and hardware acceleration, we demonstrated the feasibility of deploying large language models on edge devices. Future work may focus on further optimizing the quantization process and exploring more efficient hardware acceleration methods to enhance the performance of LLMs on edge devices.