IEEE AICAS 2025 Grand Challenge Technical Report

Team Name:显存去哪儿了(Video Memory Go Where)  
Participant Names：Yizhe Wang, Dong hui Xue, Licheng Wu, Jiawei Xiang(UESTC)  
Superviosr Names:Liang Chang(UESTC),Hang Lu(ZGC, LAB)  
Affiliations: University of Electronic Science and Technology of China (UESTC)

# Introduction

## Overall architecture

In this experiment, we aim to deploy and optimize the Qwen2.5-0.5B-Instruct large language model (LLM) on the KV260 platform, utilizing the on-chip ARM processor and FPGA resources. The primary optimization strategies include 8-bit quantization and hardware-software co-design. On the hardware side, we accelerate the matrix computations using a matrix multiplication module. However, due to time and resource constraints, hardware simulation has been completed, but the interaction between hardware and software has not yet been fully debugged.

## Testing Methods

The testing process begins by navigating to the AICAS directory.

1. Accuracy Testing: *make acc\_eva*
2. Throughput Testing: *make thpt\_eva*
3. Compression Rate Testing: *make comp\_eva*,

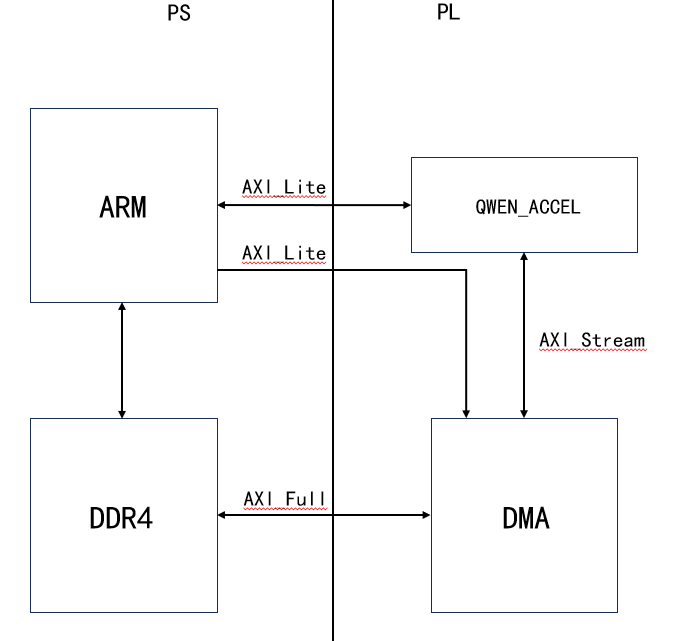
with the compressed model saved in the directory ‘*./model\_quantized/model.safetensors*’.

# Methods

## Overall architecture

The overall system architecture is shown in Fig. 1. The inference main program of the Qwen large model is executed on the ARM core, which is responsible for building the Transformer model, including tasks such as tokenizer and sampler construction. The forward pass of the Transformer is handled in the ARM processor, but data is transferred directly to the Qwen acceleration module through the AXI4 bus via Direct Memory Access (DMA). This enables efficient memory access and offloads the heavy computation tasks to the hardware accelerator. The Qwen acceleration module processes the data and then sends it back to the DDR memory through DMA for further usage.

The DMA module and the Qwen accelerator both feature AXI\_Lite interfaces, allowing the ARM processor to configure the modules dynamically to meet various computation requirements. These interfaces provide an easy way for software to communicate with hardware, ensuring seamless integration between the processing system (PS) and the programmable logic (PL). Currently, the hardware acceleration module only supports matrix multiplication, a crucial operation for speeding up the model's inference. However, future enhancements are planned, including the integration of parameter pruning, sparse matrices, and other advanced techniques, which will further optimize the system's performance and resource utilization.



*Fig. 1. Overall architecture*

In this architecture, the ARM processor handles high-level control, management, and pre-processing tasks, while the FPGA-based Qwen accelerator performs specialized computations, leveraging its parallel processing capabilities. The communication between the two via the AXI interfaces ensures high throughput and minimal latency, making it ideal for deploying large language models like Qwen on embedded platforms with limited resources.

## 8bit-quantization

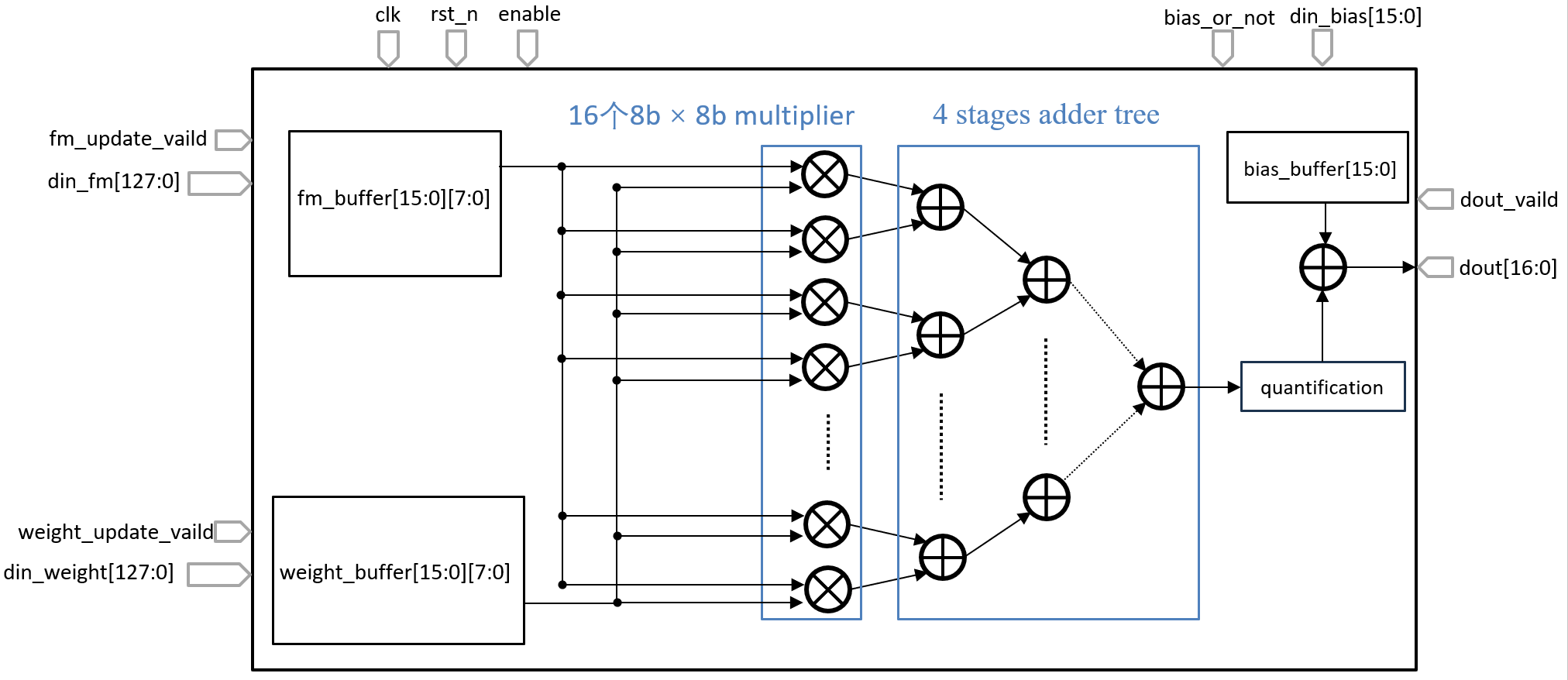
The original model where the entire calculation of the forward pass is kept in bfloat16 is so large that it can’t be employed and run efficiently on FPGA, because the model checkpoint files are very large (it takes 2 bytes per every individual weight), and the forward pass is relatively slow. The storage and time overheads are enormous. So we need to accelerate the forward pass and make the checkpoint files smaller.

Our optimization strategy is to quantize the model parameters to lower precision, giving up a little bit of correctness in return for smaller checkpoint sizes and faster forward pass. We decide to quantize the model to the precision of int8 so that we can get the benefits of smaller checkpoint sizes and faster forward pass and don’t sacrifice too much of the model accuracy. Because some layers are very sensitive to precision, such as scales and bias in RMSNorm, we only quantize the weights participating in matmuls and the activations in the forward pass. And we dynamically quantize and dequantize between bfloat16 and int8 at run time, which adds overheads. But we can get the benefits that the majority of the calculations are using pure integer arithmetic, especially the matmuls in the process of calculating attentions. The quantization of weights is symmetric around 0 and the weights are quantized to the range [-127,127]. The steps of quantizing are as followed. Firstly, we divide tensors which will be quantized into some groups with a group size of 64 and we will quantize tensors group by group. Secondly, we find out the max value in every group and then divide the maximum value by 127 so that we can get scale of every group. Thirdly, divide every value in every group by scale in order to get the weights in the range of [-127,127] and we save these weights with the precision of int8. Finally, we save scales of every group in checkpoint file with the precision of bfloat16.

## Design of matrix multiplication

The matrix multiplication module is designed to accelerate the attention computations in the Transformer model by decomposing input matrices into row and column vectors for parallel processing. Each pair of vectors undergoes element-wise multiplication using a dedicated Processing Element (PE) unit, which employs 16 parallel 8-bit multipliers to process 16 elements simultaneously. The partial products are then accumulated through a pipelined adder tree with log2(k) levels, minimizing latency while ensuring high throughput. Feature and weight buffers optimized for high-bandwidth access feed data to the PE units, while a finite state machine coordinates data flow and computation sequencing. This architecture leverages hardware parallelism and pipelining to overlap memory access with arithmetic operations, effectively hiding latency.

The design achieves a throughput of 16 multiplications and 15 additions per clock cycle per output element, with total latency reduced to 1+ log2(k) cycles. By reusing PE units and optimizing buffer utilization, the module balances resource efficiency and performance scalability. Future enhancements include sparse matrix support via zero-skipping logic and adaptive quantization group sizing to further improve memory and computational efficiency for embedded deployment.



*Fig. 2. PE unit design*

# Results

The original checkpoint file is 942MB. With the help of our quantizing, the checkpoint file is compressed to 490MB, which is almost half of the original file.

The throughput and accuracy results presented here are based on the quantized model tested on the ARM side. The accuracy of the quantized model is 0.5228, which represents a decrease of approximately 1% compared to the model before quantization.

In terms of throughput, the prefill throughput has increased to 4.8081 token/s, up from 3.1111 token/s before quantization, showing a notable improvement. Similarly, the decoding throughput has reached 4.3586 token/s, compared to 2.8423 token/s before quantization, demonstrating a significant enhancement in performance.

These results highlight the tradeoff between accuracy loss and the substantial improvement in throughput after applying quantization, which is beneficial for deploying the model on hardware with limited resources.