Technical Report

Team Name: 简单随机抽样的1/7192  
Participant Names: 卢林强 罗荣鑫 梁锦文 李昊凌  
Superviosr Names: 刘杰  
Affiliations: Nanjing University

# Introduction

Qwen-2.5 is a large language model officially released by Alibaba Cloud in September 2024. It comes in various sizes, ranging from 0.5B to 72B parameters. Currently, this model demonstrates comprehensive performance across mainstream benchmark datasets. All models were pre-trained on a massive dataset containing up to 18 trillion tokens. Compared to Qwen-2, Qwen-2.5 boasts significantly more knowledge. Among them, the 500-million parameter general-purpose model, Qwen-2.5-0.5B, is open-sourced and free. Alibaba Cloud provides global multi-channel access to the model, along with related training, deployment, and inference services.

Participating teams can independently select CPU platforms to verify large model optimization methods for Qwen-2.5, conduct overall performance evaluations, compare the "Ability" and "Efficiency" before and after deployment hardware optimization. This technical report present our optimization method and test results.

# Methods

The main method we use to optimize Qwen2.5-0.5B is deploying the Qwen2.5 - 0.5B model to llama.cpp.

llama.cpp (<https://github.com/ggml-org/llama.cpp>) is an open - source project. The main goal of llama.cpp is to enable LLM inference with minimal setup and state-of-the-art performance on a wide range of hardware - locally and in the cloud. Some advantages of llama.cpp:

### Plain C/C++ implementation without any dependencies

### Apple silicon is a first-class citizen - optimized via ARM NEON, Accelerate and Metal frameworks

### AVX, AVX2, AVX512 and AMX support for x86 architectures

### 1.5-bit, 2-bit, 3-bit, 4-bit, 5-bit, 6-bit, and 8-bit integer quantization for faster inference and reduced memory use

### Custom CUDA kernels for running LLMs on NVIDIA GPUs (support for AMD GPUs via HIP and Moore Threads MTT GPUs via MUSA)

### Vulkan and SYCL backend support

### CPU+GPU hybrid inference to partially accelerate models larger than the total VRAM capacity

We deployed the Qwen-2.5-0.5B model to llama.cpp. We use the Q4\_0 quantization in llama.cpp project. This type of quantization quantize f16 weights to INT4, preserve f32 bias and output-tensor(f16->INT8). Specifically, Q4\_0 quantization use "type-0" quantization, where weights w are obtained from quantizing q using w = d \* q, where d is the block scale. Q4\_0 quantization of llama.cpp wraps 32 weights into one block. Llama.cpp writes special operator to accelerate Q4\_0 quantized model.

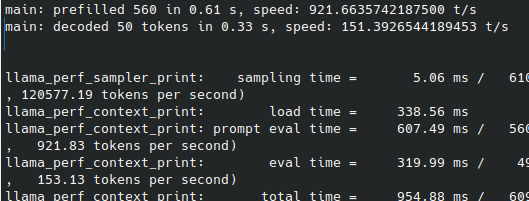
After quantization, we used Linux perf\_events (aka "perf") and FlameGraph (<https://github.com/brendangregg/FlameGraph>) systematically identify and address performance bottlenecks. By capturing and analyzing execution profiles during model inference, we observed significant cycles consumed by quantized matrix multiplication kernels, particularly within low-bit arithmetic operations. We converted certain core functions (ggml\_vec\_dot\_f16) from NEON to SVE2 ISA to improve the performance floating-point arithmetic.

To get a better performance, we optimized the code of llama.cpp by integrating the pull request ([https://github.com/ggerganov/llama.cpp/pull/8878](https://github.com/ggerganov/llama.cpp/pull/8878" \t "_blank)). And we remove the support for other LLM architectures in llama.cpp. This simplifies the logic and decreases the memory cost.

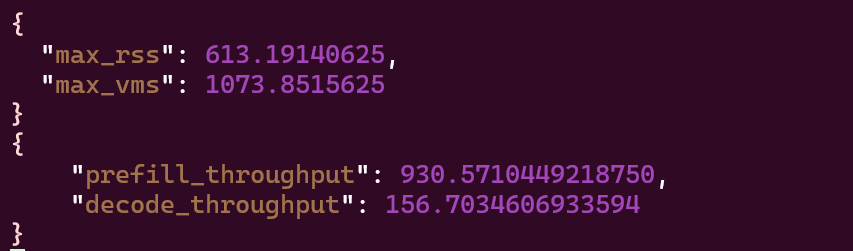
To maintain accuracy after quantization, we use LLama-factory to finetune our raw model. ([hiyouga/LLaMA-Factory: Unified Efficient Fine-Tuning of 100+ LLMs & VLMs (ACL 2024)](https://github.com/hiyouga/LLaMA-Factory)) LLama-factory is a tool for llms’ finetune. We mainly use alpaca\_gpt4\_data and alpaca\_gpt4\_data\_zh(<https://github.com/Instruction-Tuning-with-GPT-4/GPT-4-LLM/tree/main>) to improve the overall performance.

# Results

We embedded our prefill and decode test into the llama-cli program(see below). The memory test script is slightly modified from your script.



This is our final prefill, decode and memory results.



For accuracy, we rewrite a API model in lm-eval to test accuracy. This API model is relied on llama-cpp-python.

This is our accuracy final result:

