Qualcomm

For 13th International Workshop on OpenCL and SYCL (2025 IWOCL)

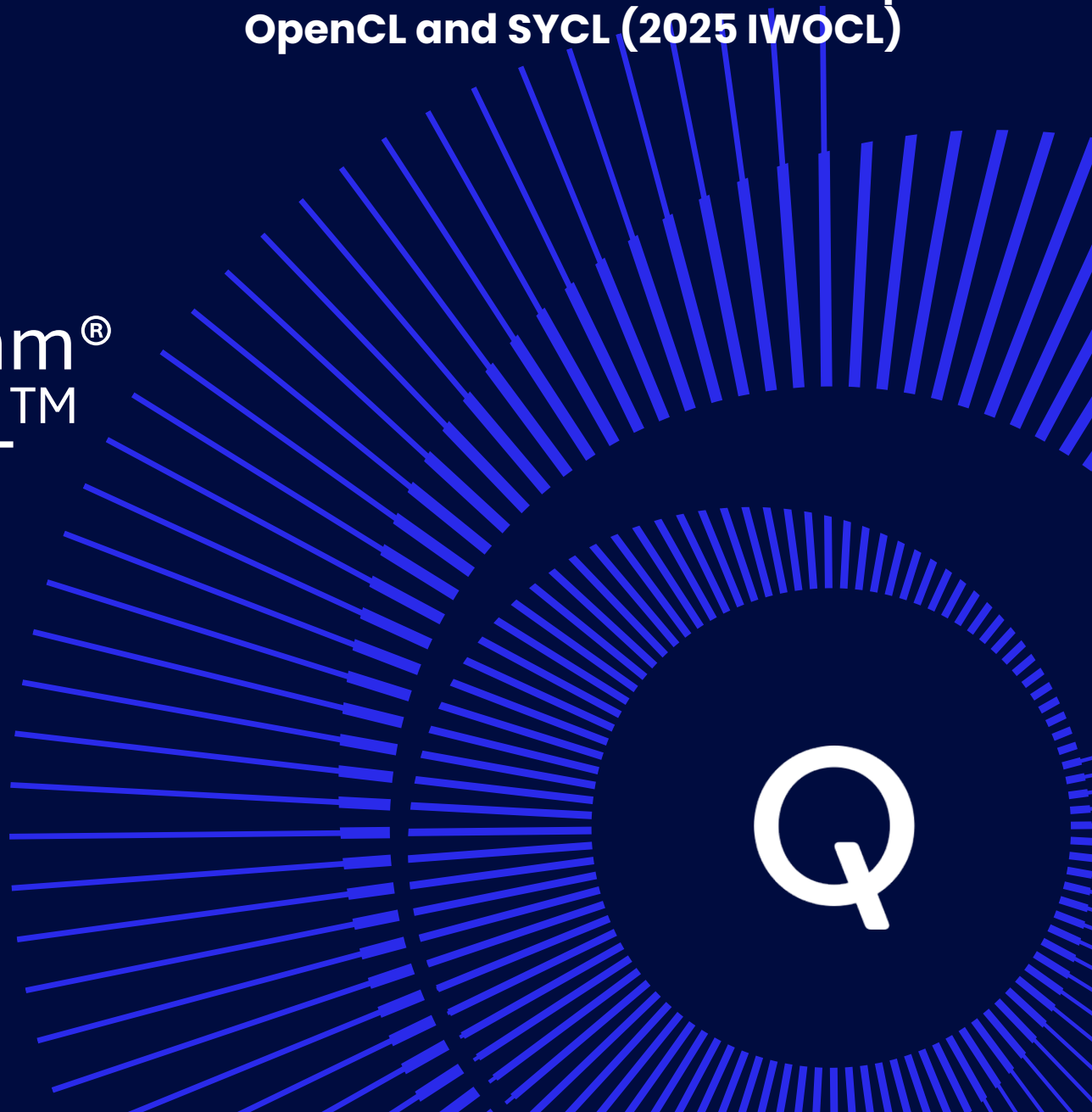
Introduction to the Qualcomm® Adreno™ Optimized OpenCL™ backend for Llama.cpp

Hongqiang Wang

Engineer, Principal/Manager

Qualcomm Technologies, Inc.

@qualcomm



Agenda

Who are we?

What and why llama.cpp + OpenCL?

Llama.cpp workflow and software arch

Our contributions: kernels, ops, and perf

Status and work in progress

Future work

Summary

Who Are We?

Our team

Part of Qualcomm GPU Research Team (GRT)

- Work on GPGPU and AI/ML projects for Adreno GPUs
- Focus on architecture
- Participate in open-source projects like TVM/MLC and Llama.cpp

Two technical presentations for IWOCL 2025

- This one: Llama.cpp on Adreno GPUs
- TVM/MLC for llm on WoS (coming next)

Long term contributors for IWOCL

Key contributors for the project

Li He
Staff Engineer

Alex Angus
Senior Engineer

*former team member

Skyler Szot
Engineer

Shangqing Gu
Senior Engineer

Shaofei Qi
Engineer

Alex Bourd
Senior Director, Technology

Resources/References

- Public Llama.cpp git repo: https://github.com/ggerganov/llama.cpp
  - OpenCL PR: Introducing experimental OpenCL backend with support for Qualcomm Adr… · ggerganov/llama.cpp@a76c56f
  - How to build: https://github.com/ggml-org/llama.cpp/blob/master/docs/backend/OPENCL.md
  - OpenCL kernels: https://github.com/ggml-org/llama.cpp/tree/master/ggml/src/ggml-opencl/kernels
- Llama.cpp at CodeLinaro: typically, first upstreamed here and then merged into Llama.cpp mainline
- Blogs about the work:
  - Introducing the new OpenCL™ GPU backend in llama.cpp for Qualcomm Adreno GPUs
  - How to run DeepSeek models on Windows on Snapdragon – Llama.cpp and MLC-LLM tutorial
- Adreno OpenCL SDK, and programming guide and best practices: https://qpm.qualcomm.com/#/main/tools/details/Adreno_OpenCL_SDK

What is Llama.cpp

- An open-source project written in C/C++ for inference of Large Language Models (LLM):
  - ***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.***
  - Support many models: 80K+ models from Huggingface (UUGF formats) as of today.
  - Mainly for text generation, also support multi-modal (vision), e.g., LLaVA
- Supports GPU/CPU/NPU via backends – CUDA, OpenCL, Metal, Vulkan, SYCL, etc.
  - Backend like Huawei Ascend NPU
  - Some assembly optimized code (ARM CPU assembly)
- Runs on Android, Linux, MacOS, Windows (x86/WoS)

Backend	Target devices
Metal	Apple Silicon
SYCL	Intel and Nvidia GPU
CUDA	Nvidia GPU
HIP	AMD GPU
Vulkan	Mostly discrete GPU
OpenCL	Adreno GPUs
MUSA	Moore Threads MTT GPU
BLAS	General
BLIS	General
CANN	Ascend NPU

Why Llama.cpp?

- Simplicity and flexibility
  - Easy to work with – both runtime and kernels, as it entirely implemented in C/C++
  - Handwritten kernels (OpenCL C, CUDA, Metal shading language, GLSL, SYCL)
  - Support of libraries, e.g., cuBLAS
- Cross platform support
  - Can be easily built on Android, Windows, Linux and MacOS via CMake with minimal dependencies
- Industry traction: very active and popular project
  - 77.8k stars at GitHub, as of 4/8/25.
  - Support from AMD, ARM, Intel, Huawei, Qualcomm, etc.
- Decent performance

Why OpenCL and Adreno for Llama.cpp?

Why OpenCL?

Mature APIs

- Supported by almost all major GPU vendors
- Easy to program, debug, profile, and port
- Best for mobile and embedded systems

Availability of GPU on Snapdragon SoCs

- Supported on all tiers of Snapdragon SoCs
- Availability of other computing cores may vary
- GPU is often idle or under-utilized

Llama.cpp had an OpenCL backend

- Only MatMul is offloaded to GPU via CLBlast
- Kernels are not highly optimized for some GPUs
- Deprecated by the community last year

How we started the project?

Brief history:

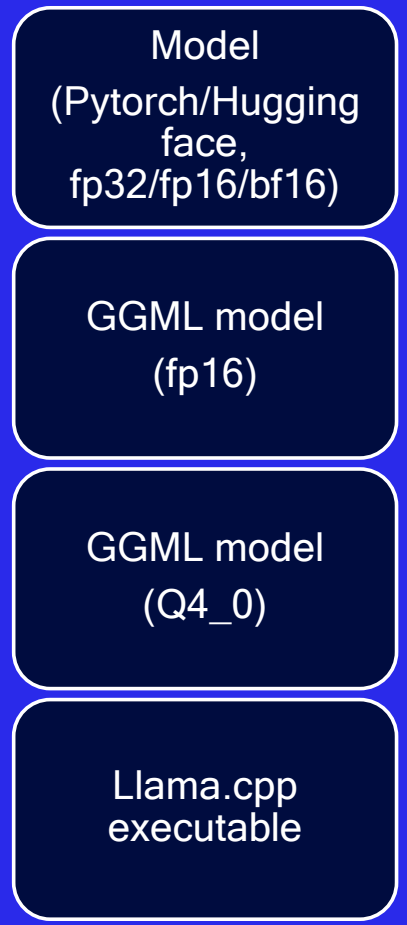
- Started as a side project by one engineer
- Gain tractions gradually
- Lately, strong interest from customers in IoT, automobile, embedded, and compute (Windows on Snapdragon)

Our goal:

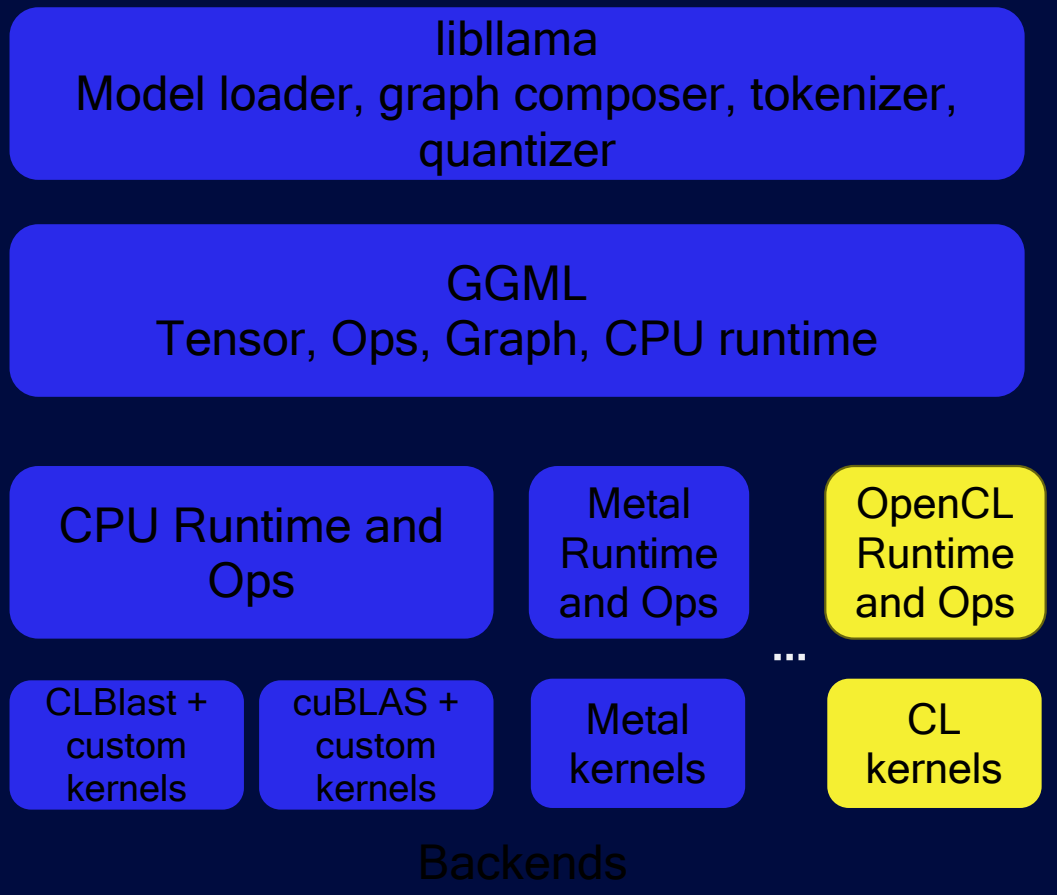
- Have an OpenCL backend optimized for Adreno GPUs
- Can be easily extended to other vendors' GPUs

Llama.cpp Workflow

- Converter tool
  - Written in Python
  - Parses Torch checkpoint files and Hugging face model files, converts GGML format in fp16
- Quantization tool
  - Written in C++ (uses the same infrastructure as llama.cpp)
  - Quantizes fp16 GGML model files using specified quantization (e.g., Q4_0)
- Llama.cpp executable
  - The llama.cpp main executable runs the quantized model



High level Llama.cpp Software Architecture



- llama.cpp
  - Shared logic for LLMs, e.g., model loader/writer, tokenizer, quantizers
  - Builds graph for various LLM models
- GGML – an ML library in C
  - Tensors, memory management interface, threading
  - Ops and ops dispatch, CPU runtime
- Backends
  - Backends implement (all or part of) the ops and target specific memory management
- OpenCL
  - Adreno backend supports all the ops for typical LLMs

Our Contributions to the Llama.cpp Project

A new OpenCL backend upstreamed to the mainline Llama.cpp project	• Full set of OpenCL kernels (50+) and the host code • A lot of debug and optimization work done • Primarily optimized for Q4_0 quantization format
Performance tuned for premium Adreno GPUs (Adreno 700 and 800): Snapdragon Gen 1, 2, 3, Elite and X Elite (WoS)	• Support Android/Linux and Snapdragon® Elite and Snapdragon X Elite (WoS) • Support selected Adreno 600 devices (like the Qualcomm Robotics RB5) • Ongoing optimization and porting for low tiers
Options to use different version of key kernels	• Smaller models are more sensitive to numerical errors caused by precisions: FP32, FP16, denorm
An internal version that uses inline assembly (ILA)	• How to expose ILA to customer is under discussion • Need the Khronos' OpenCL Working Group to move forward the extension like cooperative matrix.

Optimizations of the OpenCL Backend

Host: minimizing data movement, better memory management	Use vector data types	Flatten structs that represent quantized values into separate arrays	Use of image vs buffer	Unroll loops
Mixed precision: use of FP16 vs FP32	Tune parameters like blocking size	Good use of constant memory	Subgroup functions	Careful design of MatMul/MatVec kernels

Adreno OpenCL Programming guide: https://qpm.qualcomm.com/#/main/tools/details/Adreno_OpenCL_SDK

LLM Models Supported with the Backend

Model	Company	Sizes
DeepSeek Distilled R1	DeepSeek	1.5B, 3B
Llama 2, 3	Meta	7B, 3B
Qwen	Alibaba	7B, 3B, 1.5B, 0.5B
Phi-2/3/4 mini	Microsoft	3.8B, 5.6B
Mistral	Mistral AI	7B
Gemma	Google	2B

Metrics for Measuring the Perf of LLM Models, and Bottleneck

	Encoder	Decoder
Terms	Pre-fill, prompt processing, encoding	Token generation, decoding
Acronym	TTFT=Time To First Token	TPOT=Time Per Output Token
Meaning	How long it takes to process the prompt/file before generating new text.	Total tokens generated divided by the total time to generate the output
Numbers	The higher the better	The higher the better
Example	Prompt of 100 tokens, perf is 200 tokens/sec, it takes 0.5 seconds to generate the 1st token	Total tokens generated is 1000, total time is 100 seconds, speed is 10 tokens/second
Important when:	For long prompt use cases: email summary, continuous chat with previous output	After 1st token is generated
Main bottleneck	Generally, compute bound, a lot of matrix multiplication ops.	Generally, memory bound, a lot of matrix vector multiplication: 7B model: With 4-bit quantization, 3.5GB is required for every token, e.g., for a device with memory BW of 60GB, 17 tokens/sec is the peak perf.

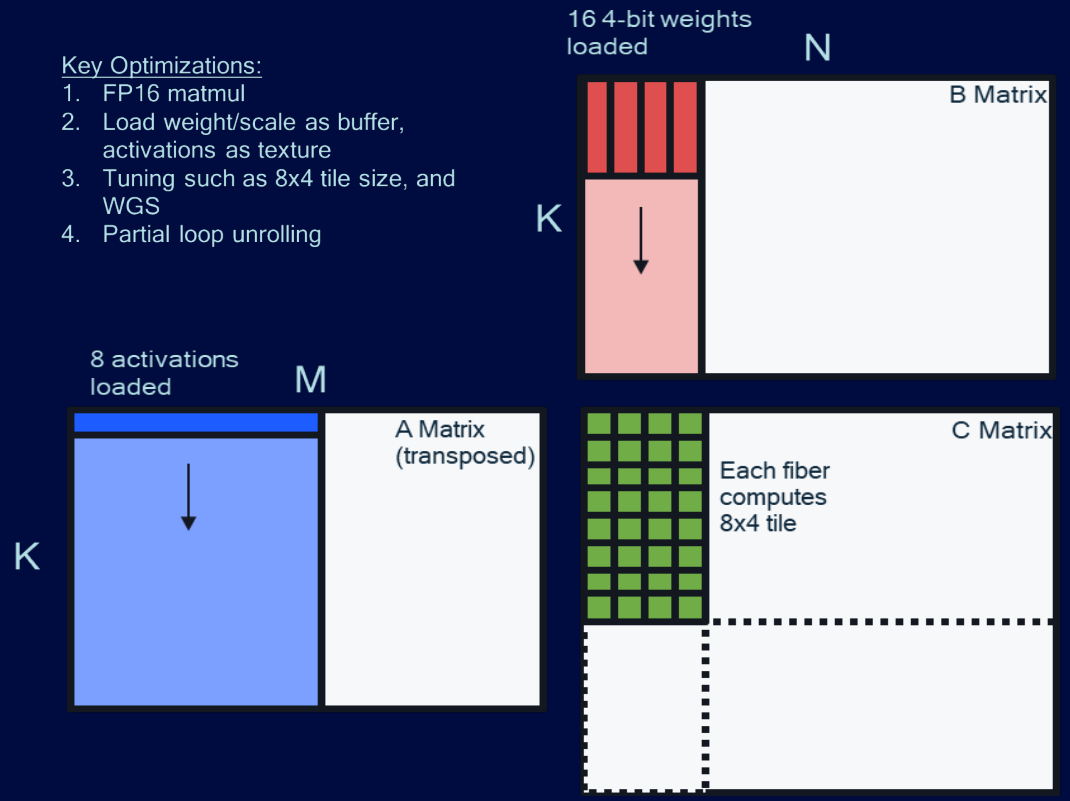
Performance of Selected Models

Model	Adreno GPUs/Device	Pre-fill (tokens/sec)	Token Generation (tokens/sec)
Llama-2 7B	**Adreno in Snapdragon X Elite**	**151**	**18**
	Adreno 830 (Snapdragon 8 Elite)	**155**	**15**
	Adreno 750	**120**	**12**
Llama-3 8B	**Adreno in Snapdragon X Elite**	116	13
	Adreno 830 (Snapdragon 8 Elite)	114	11
	Adreno 750	107	10
Gemma 2 2B	**Adreno in Snapdragon X Elite**	345	27
	Adreno 830 (Snapdragon 8 Elite)	314	21
	Adreno 750	198	9
Deepseek R1 1.5B	**Adreno in Snapdragon X Elite**	510	62
	Adreno 830 (Snapdragon 8 Elite)	384	43
	Adreno 750	495	42

Key Kernel for Pre-fill and Token Generation

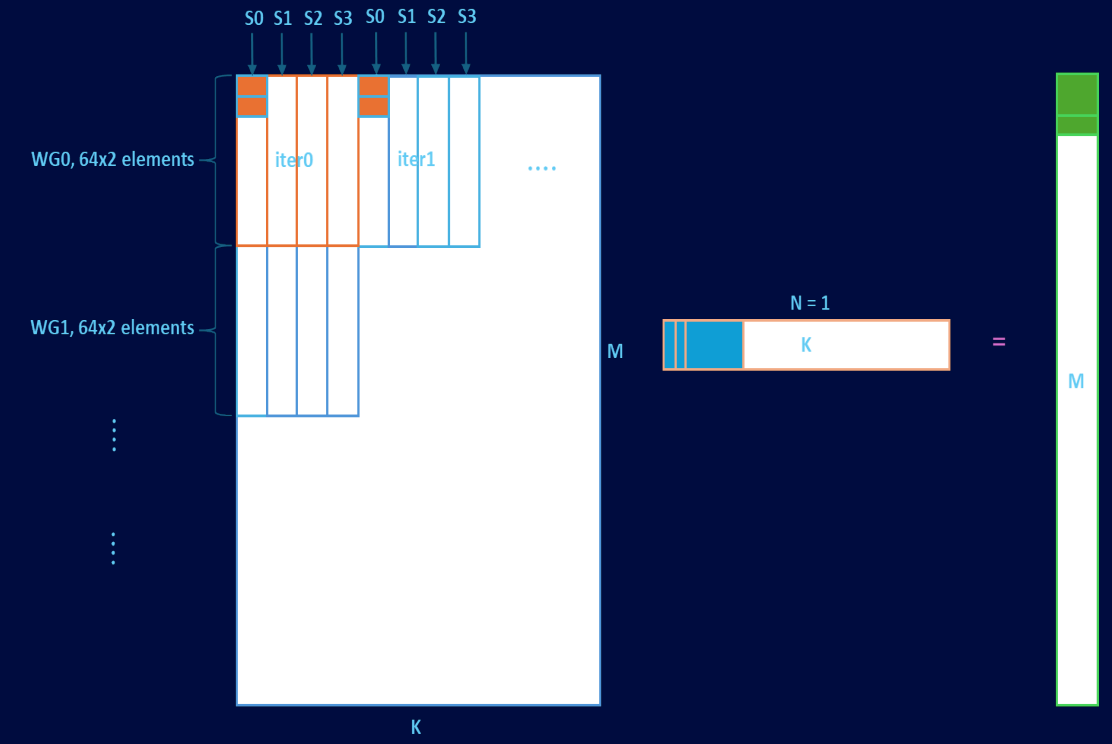
Pre-fill/encoding: Matrix Multiplication (MatMul)
Typically, compute/ALU bound

Key Optimizations:
1. FP16 matmul
2. Load weight/scale as buffer, activations as texture
3. Tuning such as 8x4 tile size, and WGS
4. Partial loop unrolling



Link to the MatMul kernel

Token gen/decoding: Matrix vector Multiplication (MatVec)
Typically, memory bandwidth bound



Link to the MatVec kernel

Discussions

Quality

- Perplexity score (not always accurate)
- Math 500 Benchmark with CPU vs GPU
  - Closely matching
  - Distill Qwen2 1.5b v3:
    - CPU: 83.1%, 232/279
    - GPU: 83.09%, 231/278

OpenCL Versions and Features

- The backend targeted OpenCL 2.0/3.0
- Require the subgroup features
- Could port to OpenCL 1.x without subgroups

Data type, quantization, precision

- Depends on models and sizes.
- Some models are more sensitive than the others.
- Ideally should have different kernels for different models with different sizes
- FP16 denorm

Experience with open-source project

- Pro: big community support, very actively updated, mature framework, lots of features/functionality
- Con: sometime not flexible to add features, e.g., debug features, or kernel/ops fusion

For OpenCL standard

- Missing cl_float support for read_imageh function;
  - Proposal to add it as an KHR extension
  - Work in progress
- Need to have cooperative matrix extension

Ongoing Efforts/Plans for the Project

Ollama

- Greatly facilitate the deployment of Llama.cpp
- Lack of OpenCL support today
- Upstreaming in progress

Multi-Modal support

- Hybrid of language and vision models
- Stable diffusion: Unet or transformer based

Algo/API/SW features

- Flash attention
- Dynamic quantization
- Dynamic kernel selection
- MoE (Mixture of Experts)
- Other key features that Metal/CUDA supports

OpenCL extensions

- Int8 dot product
- On-Chip global memory
- Recordable command buffer
- Constant memory (small model)

Other topics

- How to expose ILA kernels: Qualcomm OpenCL ML(CLML)?
- **Optimized Vulkan backend for Adreno?**

Two Open-Source Projects on LLM for Adreno GPUs

	MLC based (TVM derived)	**Llama.cpp**
Links	https://llm.mlc.ai/ https://github.com/mlc-ai/mlc-llm	https://github.com/ggerganov/llama.cpp
Programming model/language	Machine learning compiler and deployment engine for LLM	Pure C/C++ based. Manually written/optimized kernels.
Backend	OpenCL, CUDA, Vulkan, Apple's Metal, WebGPU	CUDA, OpenCL, Metal, Vulkan, SYCL, HIP, BLAS, etc.
Format	MLC-LLM format	GGUF format
Qualcomm CLML integration	Yes, via BYOC (Bring your own codegen)	Not yet
Features, Characteristics	*The AI compiler does a lot of things automatically *Have graph abstraction and auto kernel fusions *Enablement is quick. *Do not need to understand low level HW details.	*Purely C/C++ based *No graph handling or auto kernel fusion *Can easily plug in highly optimized kernels *More features, e.g., more quantizations

Summary

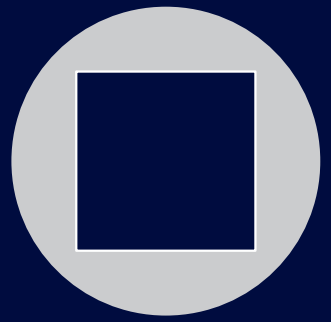
Our contribution:

- We contributed a new OpenCL backend to Llama.cpp
- Well-optimized for high end Adreno GPUs: decent performance for pre-fill and token generation
- Ported to old devices: performance being optimized (Adreno 600 serials)

A lot of features are in plan

- Try to use more OpenCL extensions, including KHR and vendor extensions
- Advanced algorithms/applications: flash attention, MoE, multi-modal etc.
- Coverage and optimization for low end and old devices

Final Words



Llama.cpp project is very active; Welcome contributions from all GPU vendors to enhance the OpenCL backend

A great opportunity to demonstrate OpenCL's capability and potentials

Disclaimer

- *OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.*
- *Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc.*
- *SYCL and the SYCL logo are trademarks of the Khronos Group Inc.*
- *Khronos and the Khronos Group logo are registered trademarks of the Khronos Group Inc.*

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog