14th International Workshop on OpenCL and SYCL

IWOCL 2026

Advancing OpenCL-Based LLM Inference in Llama.cpp on Qualcomm® Adreno™ GPUs

Hongqiang Wang, Qualcomm

Li He · Shangqing Gu · Shaofei Qi · Yunjie Xu · Alex Bourd

Qualcomm Technologies, Inc.

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

Agenda

- Introduction
  - Team overview
  - llama.cpp and the OpenCL backend
- Progress Over the Past Year
- Current Work and Priorities
  - Ongoing development and near-term focus areas
- MoE Models on Snapdragon X2
  - High level introduction
  - Optimization strategies and performance results
- Backend Comparison: OpenCL vs. CUDA, Metal, and Vulkan
- OpenCL Ecosystem
  - Extensions used today
  - Extensions critical or promising for future work
- Summary

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.*

Resources / References

- IWOCL 2025 talk: Introduction to the Adreno-optimized OpenCL backend for llama.cpp
- Public Llama.cpp git repo: https://github.com/ggml-org/llama.cpp
  - 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
  - Qualcomm fork of Llama.cpp at github.com/qualcomm
- Blogs about the initial 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
- Adreno OpenCL SDK & Adreno OpenCL programming guide
- Khronos OpenCL specification & extension registry: https://registry.khronos.org/OpenCL/
- Model files: https://huggingface.co/ ; OpenAI gpt-oss-20b: https://openai.com/index/introducing-gpt-oss/ ; Qwen model: https://qwen.ai/home
- Flash Attention: https://arxiv.org/abs/2205.14135
- Speculative Decoding: https://arxiv.org/abs/2211.17192
- Google TurboQuant: https://github.com/0xSero/turboquant
- DDTree: https://arxiv.org/abs/2604.12989
- D-Flash: https://arxiv.org/abs/2602.06036

Introduction

Who Are We?

- From Qualcomm GPU Research Team (GRT)
  - Work on GPGPU and AI/ML projects for Adreno GPUs
  - Focus on architecture, programming models, performance, and power optimization
- Lead the open-source GenAI projects for Adreno GPUs
  - Contribute to the popular open-source projects like TVM/MLC and llama.cpp
  - Help shape the future of GPU GenAI for edge computing
- Long-term contributors to IWOCL
  - Talks on OpenCL topics over years
  - Two technical talks at IWOCL 2026
    - This one — llama.cpp on Adreno GPUs
    - **Open-Source Deep Learning Compiler Powering GenAI on Adreno GPU**

Authors / key contributors

HW **Hongqiang Wang**
Engineer, Principal/Manager

LH **Li He**
Senior Staff Engineer

SG **Shangqing Gu**
Senior Engineer

SQ **Shaofei Qi**
Engineer

YX **Yunjie Xu**
Principal Engineer

AB **Alex Bourd**
Senior Director, Technology

All authors: Qualcomm Technologies, Inc.

What is Llama.cpp?

- An open-source project written in C/C++ for inference of Large Language Models (LLMs):
  - *"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." – from https://github.com/ggml-org/llama.cpp*
  - 100K+ stars at GitHub as of today.
  - Mainly text generation; multi-modal too — LLaVA, Gemma.
- Pluggable backends — CPU, CUDA, OpenCL, Metal, Vulkan, SYCL, HIP, Ascend NPU, MUSA, …
  - Some hand-tuned assembly, e.g., ARM CPU, Qualcomm NPU
  - Runs on Android, Linux, macOS, Windows (x86 / WoS), Auto platforms

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

Why Running Large Language Models (LLM) on Local?

Advantages of Running LLMs Locally

- Privacy, low latency, and deep personalization
- Near-zero marginal cost once models are deployed locally
- Agentic applications (e.g., OpenClaw) are rapidly gaining adoption, but cloud-based usage can become prohibitively expensive due to token costs

Momentum Toward Local LLMs

- Running open-source LLMs on local machines is becoming increasingly popular
- Modern local hardware is highly capable (GPUs, NPUs, high-bandwidth memory)
- Models are increasingly designed and optimized for on-device inference

Hybrid Is the Future of Edge AI

- A local-first + cloud-fallback architecture balances perf, cost, and scale
- Innovation is accelerating in open-source models:
  - DeepSeek, Qwen, Gemma, Phi, OpenAI GPT-OSS-20B
- Emerging techniques are lowering the bar for local deployment:
  - Flash Attention, Speculative decoding, TurboQuant, D-Flash, DDTree, and related long-context opts

Closing the Quality Gap

- Quality of local and open-source models is now close to top closed-source models
- Example:Qwen3.6-35B-A3B: 35B total parameters, 3B active (MoE);
  - Optimized for agentic coding;
  - Delivers performance competitive with 100B+ models or dense 30B+ models, at a fraction of the compute cost

Why Llama.cpp?

- Simplicity and flexibility
  - Easy to work with — both runtime and kernels, entirely implemented in C/C++
  - Hand-written kernels (OpenCL C, CUDA, Metal Shading Language, GLSL, SYCL)
  - Library support (cuBLAS, oneMKL, accelerate frameworks, etc.)
  - Support many ways of deployment, e.g., a server that runs llama.cpp for multi-users.
- Cross-platform support
  - Built on Android, Windows, Linux, and macOS via CMake with minimal dependencies
- Industry traction — very active and popular
  - Thousands of contributors
  - Direct contributions from AMD, Arm, Intel, Huawei, Qualcomm, and many others
- Foundation of other frameworks:
  - Whisper.cpp, Ollama, etc.
- Competition
  - MLX, vLLM, etc. Mostly target very specific APIs

Why OpenCL for Llama.cpp, History, and Goal

- OpenCL is a mature, broadly-supported API
  - Easy to program, debug, profile, and port
  - Best-in-class for mobile and embedded heterogeneous systems
  - Supported on all Snapdragon SoCs
- GPUs are available on all Snapdragon SoCs
  - Supported on every tier of Snapdragon (handsets, XR, auto, IoT, compute)
  - Other accelerators may vary — GPU is the common denominator
- Strong customer interest
  - Many ISVs already ship OpenCL pipelines (vision, imaging, classical ML)

- **Brief history**
  - Started as a side project and gained traction across products from IoT, automotive, embedded, and Windows on Snapdragon
  - Now a multi-engineer effort with regular upstream cadence
- **Goal**
  - An OpenCL backend optimized for Adreno GPUs, with a clean path for other vendors to extend
- **Note on Vulkan**
  - Vulkan/OpenCL backends serve different customers.
  - OpenCL exposes Adreno features that Vulkan cannot today (ILA, on-chip global memory, image-backed weights), and provides more flexibility
  - We are also working on Vulkan backend for Adreno

Llama.cpp Workflow

- **Converter tool**
  - Written in Python; parses Torch checkpoints / Hugging Face models, outputs GGUF in FP16
- **Quantization tool**
  - C++ — quantizes FP16 GGUF to Q4_0, Q4_K_M, Q6_K, MXFP4, …
- **llama.cpp executable**
  - Loads quantized GGUF and runs inference via the selected backend (OpenCL on Adreno)

Updates Over Last Year

General Workflow to Upstream Our Work to Mainline of Public llama.cpp Repo

- Not all work is available at the public repo
  - Public repo has the optimized OpenCL C kernels
- Some optimized kernels in-line assembly (ILA) are only available internally for now
  - Plan to have them upstreamed either via Qualcomm Package Manager (QPM) or via OpenCL vendor extensions or KHR extensions
  - Work in progress

Our Contributions to Llama.cpp — One Year Later

A more complete OpenCL backend

50+ hand-tuned kernels covering all ops needed for typical LLM inference.
Active maintenance and regular upstream PRs into mainline

Quantization coverage expanded

Beyond Q4_0 (2025) — now includes Q4_K_M, Q5_0/1, Q8_0, MXFP4 (gpt-oss), etc.
FP16 and FP32 paths for accuracy-sensitive layers

Multiple platforms

Adreno 7xx and 8xx (Snapdragon 8 Gen 2/3) and Snapdragon X Elite and X2 Elite WoS platforms
Optimization for older Adreno A6x, Internet-of-Things (IoT) and Auto platforms

Internal ILA path for flagship models

Inline-assembly kernels, MatMul, GEMV, and MoE MXFP4 on gpt-oss-20B and Gemma-class models
Plan to make them available via QPM, OpenCL vendor extensions, or Khronos extensions

llama.cpp OpenCL — Upstream Activity (ggml) over Last 90 Days

github.com/ggml-org/llama.cpp • path: ggml/src/ggml-opencl/ • snapshot May 3, 2026

Recently Merged PRs (26 merged)

Quant kernel buildout Q4_K, Q5_K, Q8_0, GEMM/GEMV kernels for Adreno; flattened-tensor variants for matvec

MoE & MXFP4 MoE-aware kernels with MXFP4 router-table reorder; Adreno-specific path

Op surface expansion CUMSUM, L2_NORM, NEG, EXP, DIAG, SET; i32 CPY support

Memory & dispatch large-buffer support on Adreno; refactored host-side dispatch for Q8_0; transpose buffer resize

Stability fixes Q8_0 leak fix; L2_NORM fix; rms_norm_mul fix; KQV mm refinement

Opening PRs (18 with label of OpenCL)

Flash Attention multiple FA optimization paths in review — quantized KV, BLOCK_N tuning, native exp, Adreno xmem path, MXFP SoA reference

Portability compat layer for non-subgroup devices; OpenCL 1.2 + 2.0 fallback paths; broader OpenCL 3.0 feature gating

Quant SoA refactors Q4_K batch GEMM with SoA layouts for OpenCL 3.0; legacy qst/packed paths for older devices; Q4_0 cleanup

Runtime efficiency on-disk cl_program binary cache to eliminate JIT cold-start; kernel arg-init refactor

llama.cpp OpenCL Backend — Active Work

github.com/qualcomm/llama.cpp • target: Adreno 800/X, X2 Elite • filtered: not yet upstream, active in last 30 days • **May 2026**

Flash Attention

New kernels: flash_attn_f32_q4_0, flash_attn_f32_q8_0, etc.

Quantized KV: Q4_0 / Q8_0 K-V consumed inside FA, no separate dequant pass

Flash-decoding: split-K over KV; pre-kernel computes partial m/l, final reduces

Tuning: BLOCK_N=64, native exp(), N_SPLIT; BM=16/NS=8 retune for DK=256

DK=256 path: prefill override (NS=8) for long-context heads

Status: not yet upstream

New Model Architecture

ssm-scan: Mamba-2 / SSM scan; lifted d_state cap to max workgroup size

gated-delta-net: full coverage — permuted, v_repeat>1 (Qwen3-Next)

vision-ops: POOL_2D, CONV_TRANSPOSE_1D, IM2COL_3D — multimodal preconditions

Speculative decoding: MUL_MAT view-on-SoA-quant fix → unblocks draft-model path

New ops

- **Small-ops:** ARGMAX, ARANGE, ROLL
- **Top-k:** bitonic sort + top-k slice — needed for sampling on-device
- **Vision-ops:** POOL_2D, CONV_TRANSPOSE_1D, IM2COL_3D (also under arch)
- **Context**: FILL, EXPM1, SOFTPLUS, TRI, SOLVE_TRI, CUMSUM and Q4_1 / Q5_K / Q8_0 GEMM; some may be upstreamed already.

LLM Models Supported with the Backend

Model family	Variants run	Architecture	Notes
Llama 3 / 3.1 / 3.2	1B · 3B · 8B	Dense	Reference workload; Q4_0 / Q4_K_M
Qwen 2.5 / Qwen 3	0.5B · 1.5B · 7B · 14B	Dense	Long-context tested up to 32K
Phi-3 / Phi-3.5/ Phi 4 mini	Mini · Medium	Dense (SWA)	Sliding window; small-model latency
Mistral 7B	v0.1 · v0.2 · v0.3	Dense (SWA v0.1)	Classic SWA validation
DeepSeek R1 / V3-lite	1.5B · 3B · Lite	Dense / MoE / MLA	MoE routing exercised
Mixtral 8×7B	Q4_K_M	MoE (8 experts)	Memory-bound on mobile; works
LLaVA / Gemma 3 vision	7B class	Dense + ViT	Conv2D + image embedding via OpenCL
OpenAI gpt-oss-20B	MXFP4, Q4_0	MoE, Linear Attention	Internal ILA path; flagship demo
Qwen 3.5 and 3.6 A3B	35B, 36B, MXFP4	Linear Attention, MoE	Internal ILA path; flagship demo
Gemma 4	27B	Hybrid SWA, MoE	Internal ILA path; flagship demo

Optimizations of the OpenCL Backend

Host: minimize data movement, better memory management	Vector data types — half4 / half8 in dequant + GEMV	Flatten quantized struct fields into separate arrays	Image / texture-backed weights for hot GEMV (Q4_0, Q4_K_M)
Loop unrolling tuned per kernel (no blanket #pragma unroll)	Mixed precision FP16 / FP32: careful with denorm sensitivity	Tile / blocking parameters swept per kernel	Constant memory for routing tables and small uniform data
Subgroup functions — reductions, broadcast, shuffle	Careful MatMul / MatVec design - wave-aware, coalesced loads	On-chip global memory residency for KV, embeddings, rotating weights	Kernel program cache -- amortize OpenCL compile cost

Mixture of Expert (MoE) Models on Snapdragon X2 Elite Adreno GPU

What Is MoE and Why It Is Popular

- MoE Models in the Prefill Stage
  - During the prefill stage, each transformer layer consists of two main components:
    - Attention + Feed-Forward Network (FFN)
- Feed-Forward Network (FFN)
  - Dense model:
    - Every token is processed by the same FFN
    - Each token performs matrix multiplications with the entire set of FFN parameters.
  - Mixture-of-Experts (MoE) model: only a subset of experts is selected to process each token.
    - A router (gating network) is computed on the fly based on the token's input representation.
    - The MoE FFN routes each token to the top-k experts out of N total experts.
    - Computation is performed only on the selected experts, reducing overall compute while increasing model capacity.
    - It may be hard for hardware where selectively launching kernels at runtime is inefficient
- Recent models such as GPT-OSS-20B, Qwen, Gemma, and DeepSeek adopt MoE.
  - MoE improves scalability, computational efficiency, and overall model performance.
  - For example, a 20B-parameter MoE model can achieve inference latency comparable to that of a 6B dense model, with much higher quality

Case Study: Optimizing gpt-oss-20B on Snapdragon X2 Elite

OpenAI released gpt-oss-20B in 08/2025

- Their first open-source LLM
- Rapidly gained significant industry traction
- Built on a Mixture-of-Experts (MoE) architecture with MXFP4 data type

Performance Bottleneck Analysis

- The MoE MXFP4 kernel is the primary bottleneck in gpt-oss-20B
- Initial profiling on Snapdragon X2 GPU shows:
  - 90%+ of execution time spent in the MoE component
  - Prefill throughput limited to ~120 tokens/sec

Perf Breakthrough via Systematic Optimization

- Hand-written ILA kernel specifically designed for MoE execution
- Key optimizations:
  - Activation reordering
  - MXFP4 → FP16 dequantization
  - Tiled GEMM
  - Shared inner-product computation
  - Precision-aware iteration and accumulation tuning
- Unified design
  - A single kernel handles MoE computation across all selected experts

Results

- Achieved 600+ tokens/sec prefill with a 1024-token prompt (GPU only and single stream)
- Measured without attention-layer optimizations
- ~5x improvement over baseline MoE performance
- Similar performance gains observed across other MoE models
- Quality: MMLU benchmark scores almost no drop vs original model: 86%

Availability

- Optimized binary kernels will be released via QPM
- Potential future delivery as a Qualcomm extension
- Performance has more room with FA and other techniques.

Ongoing Efforts and Priorities

Short Term Priority : Long-Context Performance

- Agentic Workloads
  - Continuous, interleaved input and output
  - Both prefill throughput and token generation latency matter
  - Prefill computation scales as $O(N^2)$ with sequence length
  - Token generation performance is bounded by *Effective bandwidth / (Model weights + KV cache)*
  - As conversations grow longer, both prefill and token generation slow down significantly
- How to address the challenges
  - Deeper optimization of the attention path
    - GEMM / MUL_MAT dominates LLM inference cost
  - ILA kernels to enable efficient Flash Attention
  - *cl_khr_cooperative_matrix* (drafted Apr 29, 2026) as an enabling standard
  - KV cache quantization to reduce memory BW pressure
  - Exploration of new techniques and algorithms to break current scaling limits

A List of Options

- KV Cache Quantization & Compression
  - Flash attention with Q8/Q4 of KV quantization
  - Google TurboQuant for KV-cache quantization: under 3-bit with zero loss on fidelity.
- Decoding and Attention Strategies
  - Speculative decoding for improved throughput
  - DDTree/D-Flash Tree Attention
- Recordable Command Queue/Buffers
  - cl_qcom_recordable_queue enables replay of per-layer launches with minimal CPU overhead
  - cl_khr_command_buffer is expected to replace the vendor extension once it is available on Adreno
  - Reduced CPU overhead and improved GPU utilization
  - Up to ~10% performance uplift observed on selected large models (MoE models)

Ongoing Optimization and Algorithms for Long Context

Flash Attention: IO aware attention optimization for GPUs

- Targets long-context summarization and multi-turn chat
- KV quantization, Q8/Q4 instead of FP16 to save BW
- Implemented on qualcomm fork: but still a lot of rooms for optimization

Speculative Decoding: main model vs draft model

- Host-level llama.cpp feature — backend-agnostic in principle
- Small "draft" model proposes N tokens; large "target" model verifies in one batch: Llama 3.2 1B draft + 8B
- If draft tokens are accepted, you get N tokens for the cost of ~1 forward pass on the target
- Currently functionality working but need more optimization

KV cache quantization: essential for long context token generation

- KV cache becomes bottleneck for long context, while for shorter chats weights dominate the perf
- Work with flash attention, and new techniques like Google TurboQuant targets more aggressive KV cache compression

Backend Comparison and Key OpenCL Extensions

Gaps Between OpenCL Backend vs CUDA/Metal/Vulkan: Lots of Things to Do

Capability	CUDA	Metal	Vulkan	OpenCL	Closure path
Dense MUL_MAT (GEMM/GEMV)	full	full	full	full	*Caught up for key quantizations.*
Wide quantization coverage	full	full	full	Partial	*Q4/Q5/Q6/Q8 K-quants, MXFP4 all running.*
Flash Attention — decode	full	full	full	partial	*Stable on FP16; needs prefill + bf16 to match.*
Flash Attention — prefill	full	full	full	wip	*Tiled FA2 branch in flight on the qualcomm fork.*
MoE (mul_mat_id)	full	full	full	partial	*Routing works; per-expert tuning ongoing.*
Multi-modal / vision	full	full	full	partial	*Conv2D + im2col on Adreno; LLaVA / Gemma 3 vision.*
Cooperative matrix	full	full	full	missing	*Blocked on cl_khr_cooperative_matrix landing.*
Inline assembly / ILA	full	n/a	n/a	partial	*Adreno ILA explored; public path via QPM/Extension.*
On-chip / device memory	partial	partial	partial	wip	*OpenCL-only differentiator on Adreno.*
Pre-recorded cmd buffers	full	full	full	partial	*cl_khr_command_buffer adoption.*
Ollama integration	full	full	wip	wip	*Fast deployment*
Robust auto-tuning	full	full	partial	missing	*Kernel cache + sweep-driven default selection.*

full partial wip missing n/a

Khronos Extensions — Leverage and Push

Extension	Status	Used by	Notes
cl_khr_fp16	shipping	Everywhere	Native FP16 + half4/half8 vector loads
cl_khr_subgroups	shipping	Reductions, FA, RMSNorm	sub_group_reduce_*, broadcast, shuffle
cl_khr_subgroup_extended_types	shipping	Mixed-precision GEMV	FP16 / INT subgroup ops
cl_khr_integer_dot_product	shipping	Q4 / Q8 dequant + accum	INT8 dot-product accelerator path
cl_qcom_reqd_sub_group_size	shipping	Wave-aware kernels	Required subgroup size for Adreno
cl_qcom_recordable_queues	shipping	Latency-critical loops	Pre-built command-stream replay
cl_qcom_subgroup_uniform_load	shipping	Routing tables, masks	Hint compiler for uniform values
cl_qcom_image_array_indexing	shipping	Image-backed weight banks	Index into texture arrays in-kernel
cl_qcom_priority_hint	shipping	Mixed UI + LLM workloads	Avoid jank under inference
cl_khr_command_buffer	shipping	Not available on Adreno	Pre-recorded launches, low CPU overhead
cl_khr_cooperative_matrix	WIP	Future GEMM / FA	Tensor-core-class throughput

Summary

- We have continued to **advance the OpenCL backend for llama.cpp** on Adreno GPUs
  - Expanded operator coverage and quantization support(**Q4_0, Q4_K, Q4_K_M, MXFP4,** and more)
  - Enabled support for a **broader set of model architectures**
  - Brought up key features with ongoing optimization, including **Flash Attention** and **KV-cache quantization**
  - Delivered **high performance MoE execution** using hand-tuned **ILA** kernels
- Ongoing Efforts and Near-Term Priorities
  - Focus on **long-context performance** to support agentic workloads
  - Release of key optimized kernels via QPM and future extensions
  - Adoption of **cl_khr_cooperative_matrix** to close the peak-perf gap vs OpenCL C and other GPU APIs
  - Enablement of **Ollama on OpenCL**
- The LLM ecosystem is moving fast, with **significant momentum and opportunity**
  - New contributors are warmly welcomed
  - **Together, let's make OpenCL great again**

Questions? wangh@qti.qualcomm.com

Thank you

Follow us on:

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