GOSIM

GOSIM AI Paris 2025

WGML

Building a New High-performance, Cross-platform, On-device Inference Framework

https://github.com/dimforge/wgml

Get the slides

GOSIM

https://wgmath.rs/demos/wgml/gosim-paris-2025.pdf

What is WGML?

- Cross-platform local LLM GPU inference (including web) framework.
- Based on WebGPU and WgMath (more on that later).
- *Demo: https://wgmath.rs/demos/wgml/index.html*

Context

Say you are new to writing low-level AI code and want to:

1. You need to build a high-performances local LLM GPU inference (cross-platform too!)
2. But you are **not** an AI expert.
3. Hell, you don’t even know how the LLM math actually looks like…
4. … but you have a good grasp of linear algebra.
5. And you don’t want to rely on proprietary ecosystems (CUDA…)

Where do you start?

Step 1: Explore the ecosystem

candle

ONNX

LLaMA C++

And many others…

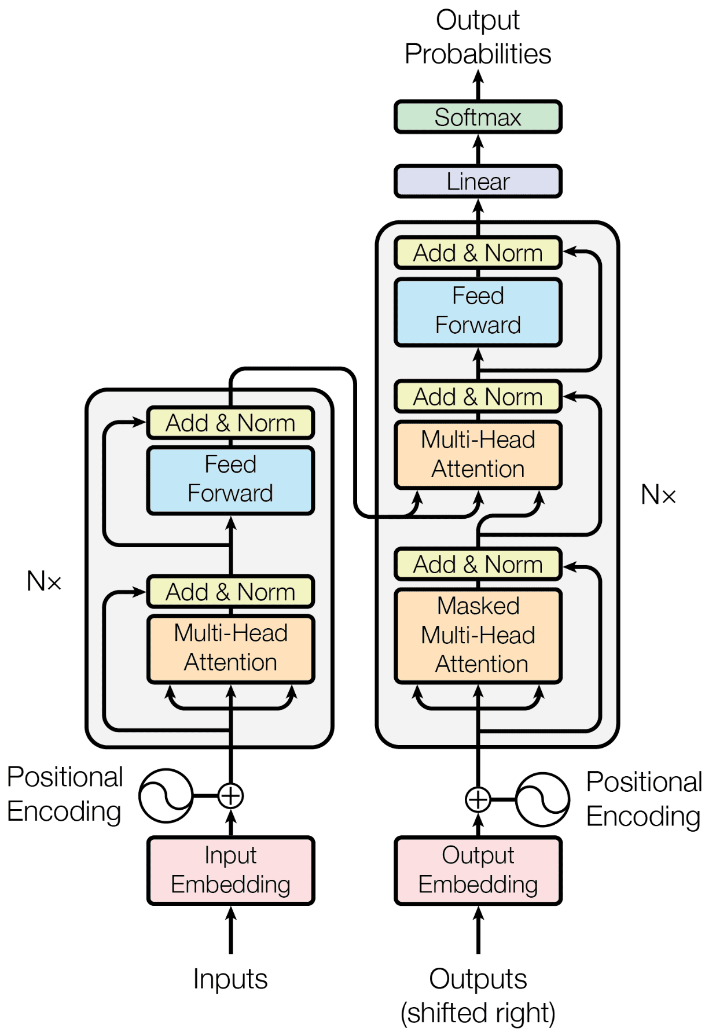
Step 2: Learning resources

Learn about LLMs, e.g., from Umar Jamil's Youtube channel:
- Transformer model: https://www.youtube.com/watch?v=bCz4OMemCcA
- LLaMA: https://www.youtube.com/watch?v=Mn_9W1nCFLo

Read code:
- llama.c: https://github.com/karpathy/llama2.c/blob/master/run.c
- Llama.cpp/ggml (not easy, take your time):
  - Metal shader (incl. matmul)
  - Dequantization
  - Transformers (link to the Qwen2 transformenr assembly).

Run some local implementations:
- Get familiar with running a couple of models (with llama.c and llama.cpp).
- Very handy for debugging!

Many other resources to checkout from. In Rust:
- https://github.com/rahoua/pecca-rs
- https://github.com/seddonm1/web-llm



The GGUF file format

- Ubiquitous and straightforward.
- Specification
- For initial debugging (and CI) generate a dummy gguf file.

Don’t forget about chat-templates !

- They are generally embedded in the GGUF file metadata as Jinja templates.
- If not, check out the model’s documentation.
- Some pre-made templates: https://github.com/chujiezheng/chat_templates/tree/main/chat_templates

Step 3: Happy coding!

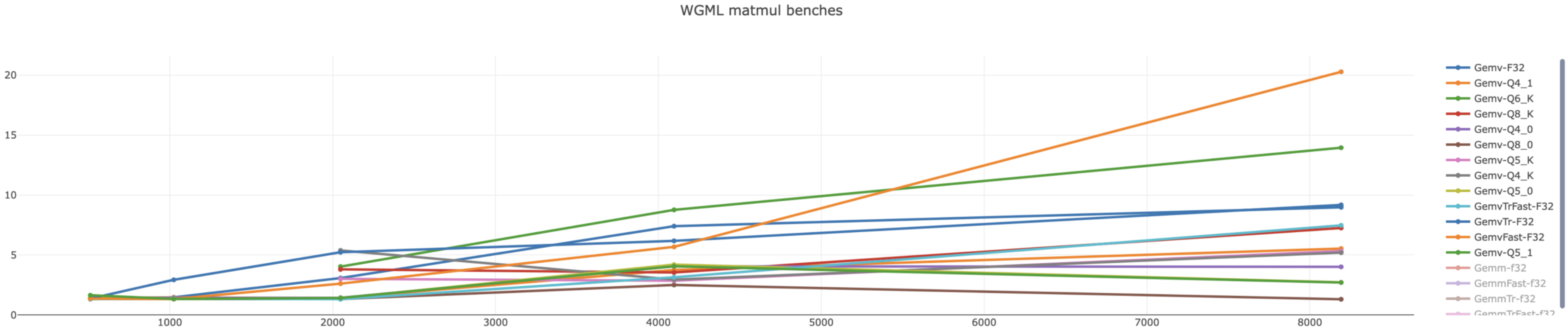
1. Start with a CPU version (easier to debug): WGML implementation with timestamps from the Umar Jamil video on llama.
2. GPU implementation with systematic unit-testing of all linear-algebra implementations.
3. Don't hesitate to use existing libraries for some difficult or boring non-gpu parts (tokenizer and sampler).

Minimal set of operations (for llama2):
- **Matrix-mul-vector**; ⚠ the matrix might be quantized.
- **Vector sum**.
- **RoPE** (special shader); ⚠ other models like Qwen rely on a slightly different variant.
- **Multi-head attention** (special shader *).
- **Silu** (special shader).

Extremely good resource: web-llm

Step 4: Work on performances

The two main bottlenecks are:
- **Matmul** = used everywhere in the transformer architecture.
- **Multihead attention** = the naïve shader has very limited scalability as the context grows.

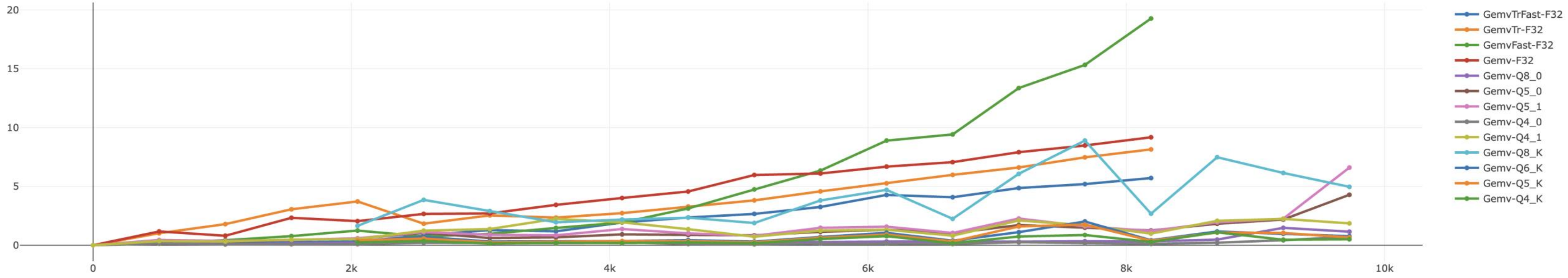


Beware of workgroups & quantization

GOSIM

Naïve strategy for dealing with quantized matrices: have each thread perform dequantize and perform all the calculations related to a given matrix (quantized) element.

=> Optimized strategy ensures multiple threads parts of the same workgroup operate on the same quantized values to preserve coalesced memory reads.



Be mindful of WebGPU limitations

- **Push-constants** are an extension = maintain a cache of uniform buffers for storing matrix sizes!
- The default **max storage buffer size** is not enough = request more when initializing the device!
- **Subgroup reductions** are an extension = write the reduction manually 😕
- **WGSL primitives** are 32 bits (no u8 type) = unpack quantized values from raw arrays of u32.

On the web: no reliable way around these limitations (currently).
On native: conditional compilation for enabling extensions based on the platform could be considered.

Step 5: Rejoice!

🎉 Add an UI (or a CLI) ahead of your transformer and you are done! 💪

DIOXUS

Web, Desktop, and Mobile apps with a single codebase.

But… how about the CPU side of WebGPU:

- *Initializing device and buffers?*
- *Instantiating shaders?*
- *Dispatching kernels?*
- *Avoiding code duplication across WGSL shaders?*

GOSIM

GOSIM AI Paris 2025

wgmath

GPU scientific computing is **hard**
(╯°□°)╯︵ ┻━┻

Dominated by ***Cuda*** 😔

Can we open this to every platform?

wgmath: ecosystem for AI, robotics, and physics

GOSIM

*(features marked **wip** are not implemented yet)*

wgcore

Macros: shaders declaration & composition
Buffers: Tensor/Matrix/Vector

↓

wgebra

Kernels: gemm, gemv, dot, max, etc.
Reusable wgsl pieces: quaternions, similarities, rotations, 2x2 and 3x3 SVD

↓

wgml

Kernels: rope, silu, rms_norm, softmax, etc.
Reusable wgsl: workgroup reductions, ML-specific transformations, activation functions.
Models: llama2, gpt2

wgparry

Kernels (wip): broad-phase, narrow-phase, composite-shape traversals, etc.
Reusable wgsl (wip): geometric types and queries.

wgrapier

Kernels (wip): islands calculation, constraints solver, integration, inverse-kinematics.
Wgsl pieces (wip): constraint resolution primitives, inertia tensors, velocities, etc.

wgsparkl

Kernels: gpu hashmap, sparse grid, g2p, p2g, etc.
Reusable wgsl: constitutive models.

THANK YOU