W2GPU: Toward WebAssembly-to-WebGPU Program Translation via Small Language Models

Breaking the Glass Ceiling From Human Memory Limits to Neural Compiler Passes

Oguz (Mehmet Oguz Derin)

LMPL 2025 Preview at GPU for the Web F2F

The Abstraction Paradox

We invent abstractions to extend limited working memory.

Yet these same abstractions impose a **glass ceiling**
on our optimizers and runtimes.

LMPL 2025 Preview

Evolution traded memory for naming

- We compress and pack context into **names**: types, functions, modules, effects.
- Working memory yields a practical *name budget* limitation: only so many chunks fit on the mental stack.
- Our tools (compilers, game, and physics engines) inherit these same constraints.
- **Consequence**: Helpful abstraction boundaries for humans can occlude long-range structure that optimizers could exploit.

Q2. What is the “glass ceiling” in compiler optimization?

- Traditional pipelines rely on human-crafted passes, rewrite rules, and heuristics—excellent locally, brittle globally.
- These rules mirror our chunking/naming tendencies and the abstractions we expose, such as functions, modules, and effects.
- Long-range, cross-stage optimizations remain latent.
- Example: Well-engineered CUDA stacks and recent IRs like MLIR still encounter plateau effects, as risks of memory management and behavior consistency increase combinatorially with heterogeneous systems

Human-designed abstractions reflect our cognitive limits

What we see:

- Named functions and kernels
- Type/module boundaries
- Interface contracts
- Explicit dependencies

What we miss:

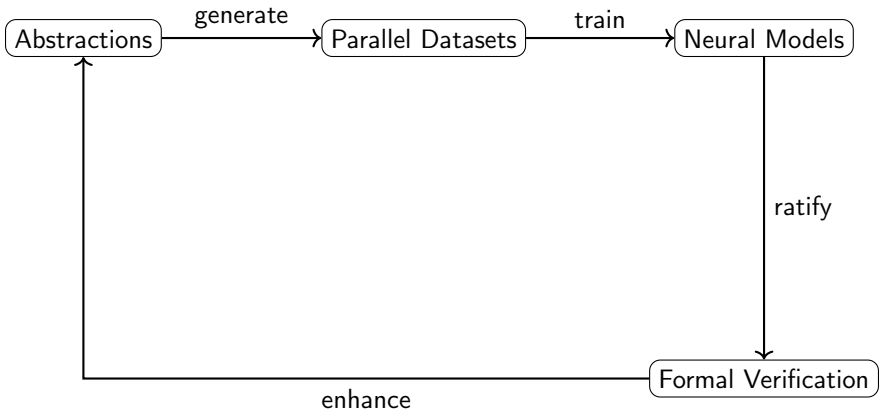
- Long-range fusion opportunities
- Cross-abstraction equivalences
- Latent parallelism and tilings
- Non-obvious host→device mappings

The ceiling: Optimizations beyond human pattern recognition.

Q3. How can we break through this ceiling?

- **Key**: Machines are not bound by human working-memory limits, naming, or procedural biases.
- Build **parallel corpora**: semantically equivalent programs with differing pragmatics and layouts yet with stable ABI
- Train small language models to learn mappings that our passes do not capture.
- **Extract** $\rightarrow$ **verify** $\rightarrow$ **ratify**: promote learned patterns into sound transformations.

Parallel datasets + neural models = jumping stones beyond human patterns



Breaking the ceiling: Learn transformations → verify → formalize → automate the loop.

Q4. Can this work for real systems? Enter W2GPU

- **Task**: WebAssembly Text (WAT, CPU) → WebGPU Shading Language (WGSL, GPU).
- Stack-based VM with linear memory → structured variables with explicit GPU types.
- Non-isomorphic semantics (no arbitrary pointer arithmetic; restricted pointer types; no native `i64` in WGSL; lack of resource annotations).
- **W2GPU Approach**: Use small language models to learn robust mappings from WAT idioms to valid WGSL kernels.

Partially yes — with validated pipelines and measurable success

Key: With a stable CPU target ABI, models recover patterns we did not hand-code.

Pipeline:

- 14,547 parallel (WAT, WGSL) pairs
- 7-stage validated compilation & execution
- Sub-2B SLMs with QLoRA-style fine-tuning
- Greedy (deterministic) decoding

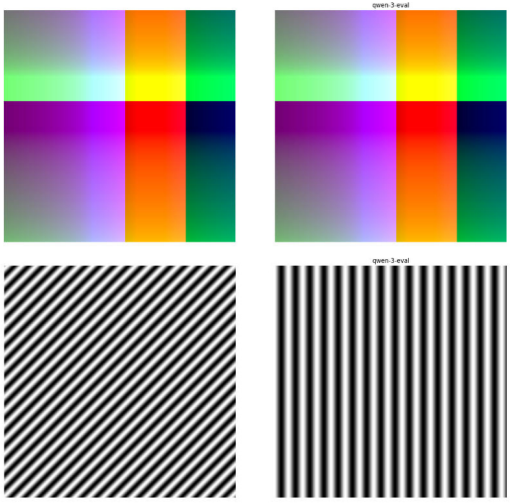
Outcomes:

- ✓ End-to-end compile success
- ✓ Image parity via PSNR on shader outputs
- ✓ Emergent GPU resource/binding patterns
- ✗ Significant problems with longer inputs

Model	Compiles	PSNR>40	20–40	<20
Qwen3-1.7B	27	12	0	15
Qwen3-0.6B	22	12	0	10
Llama-3.2-1B	18	6	0	12

LMPL 2025 Preview

Results



LMPL 2025 Preview

W2GPU: Brief History (2016–2025)

Timeline

- **2016–2017**: While learning OpenGL/Vulkan, inspired by the single-source ease of CUDA/SYCL/OpenCL; First interactions with early WebGPU efforts.
- **2018–2020**: Participation in **WebGPU**; early investigation of the *subgroup* gap within compute.
- **2021**: Focused on **WASM** as the CPU/host target: stable compiler coverage, well-defined, and tooling. Lets kernel authors keep C++/Rust (etc.) while using WASM as the common denominator.
- **2022–2024**: Conventional translation paths (wasm2c-style). Prototypes advanced but tended to be *opinionated*, needed pragmatics/comments, or depended on future feature coverage.
- **2024–2025**: Current **W2GPU** shape emerges: ABI stability via `slangc` CPU execution (as SPIRV-Cross's CPU C++ target was deprecated), WGSL generation, and small LMs learning WAT→WGSL mappings. Surpasses prototype translators on compile success, PSNR parity, and resource use; opens a path toward atomics, image objects, and subgroups without pragmas or language lock-in.

Why it crystallized now

- **Stable ABI anchor**: `slangc` *CPU execution* constrains layout/semantics; WGSL offers a clear GPU target.
- **Maturing WebGPU/WGSL stack**: Sufficient coverage to run end-to-end and validate.
- **Small LMs**: Sub-2B models with (Q)LoRA learn robust translation patterns beyond hand-coded heuristics.
- **Parallel corpora & parity checks**: Scaled (WAT, WGSL) pairs with compile-success and PSNR-based parity enable precise, iterative refinement.
- **From opinionation to generality**: Data-driven mapping helps.

LMPL 2025 Preview

LMPL 2025 Preview

The Vision: From Neural Drafts to Reliable Compilers

- **Short term**: Neural models as drafting tools; outputs require ratification.
- **Medium term**: Hybrid pipelines—neural front ends, formal back ends.
- **Long term**: Optimizers that capture intent and accelerate beyond human heuristics for both emergent empirical programming languages and existing software infrastructure.
- **Payoff**: Break ceilings in signal processing, game engines, XR, and HPC through portability promises of web stack, including native targets.

As heterogeneous neural generation improves within web sandbox,
corpora yields for ratification—paving auto-formalized transforms.

Practical Impact

For Developers:

- Start from LM drafts; keep humans in the loop
- Test & ratify kernels with validators
- Target hotspots; measure PSNR/latency/throughput
- Ship faster with confidence envelopes

For Researchers:

- Open adapters and recipes
- Reproducible pipelines and artifacts
- Validated end-to-end workflows
- Seed metrics at the PL$\times$GPGPU$\times$Web$\times$LM boundary

Key Takeaways

1. Human cognitive limits shape abstractions—and impose optimization ceilings.
2. Parallel datasets + neural models can reveal transformations beyond hand-crafted rules.
3. W2GPU shows feasibility on a real WebAssembly→WebGPU path.
4. LMPL is the right venue to standardize data, validators, and comparisons
5. Contributions to w2gpu are more than welcome to add to this approach, and repositories will make it convenient to both contribute, so that it can also serve the use case needed in a traditional CLI-like interface wrapping the open SLM model, and also jumpstart any projects as a template
6. **Next steps**: test synthesis; reward-driven tuning (e.g., GRPO); WASM-GC; atomics; workgroup/subgroup features; longer context; host+device co-transformation; perf via cycles/pressure analysis; permissive, open access.
7. **Goal**: reliable compilers that capture intent and emit performant native & web programs for humans and machines.

Acknowledgments

This work builds on open-source tools and datasets, including:

- Slang, Emscripten, WABT, Dawn, SPIRV-Cross, wgpu, shaderc, wgpu-py
- Shaders21k corpus
- Qwen3, Llama, Gemma model families
- Unsloth, TRL, and the broader ML/PL ecosystems

Special thanks to the LMPL organizers for launching this venue!

References

[1] WebAssembly Core Spec (W3C, 2019)
[2] Alpay & Heuveline. Adaptivecpp stdpar (IWOCL '24)
[3] Anthropic. Claude Opus/Sonnet 4 (2025)
[4] aONe. Keka File Archiver (2023)
[5] Baker et al. WGSL W3C CR (2025)
[6] Baradad et al. Procedural Image Programs (NeurIPS '22)
[7] Blandy et al. WebGPU W3C CR (2025)
[8] Capens. SwiftShader (Chromium, 2016)
[9] Collet & Kucherawy. Zstandard RFC 8478 (2018)
[10] Dettmers et al. 8-bit optimizers (arXiv 2021)
[11] Dettmers et al. QLoRA (arXiv 2023)
[12] Deutsch. GZIP RFC 1952 (1996)
[13] Devillers et al. No More Shading Languages (HPG '25)
[14] Döllerer & Engelke. 64-bit WASM Bounds (VMIL '24)
[15] Hugging Face. TRL (2023)
[16] Python 3.13 Release Notes (PSF, 2024)
[17] Google DeepMind. Gemini 2.5 Flash-Lite (2025)
[18] Grattafiori et al. Llama 3 Herd (arXiv 2024)
[19] Khronos. SPIRV-Cross (2018)
[20] WebAssembly CG. WABT (2018)
[21] He et al. Slang (TOG 2018)
[22] Google. Tint WGSL Compiler (2021)
[23] Jackson & Gilbert. WebGL 2.0 (Khronos, 2017)
[24] Kalajdzievski. RSLoRA (arXiv 2023)
[25] Klein et al. wgpu-py (2022)
[26] Martínez. wasm2spirv (2023)
[27] Neto & Mircevski. Shaderc (2015)
[28] OpenAI. GPT-4.1-Nano (2025)
[29] Ouyang et al. KernelBench (arXiv 2025)
[30] O'Connor. wasm-gpu (2023)
[31] Pavlov. 7-Zip (1999)
[32] Power et al. gem5-gpu (CAL 2014)
[33] Mesa Project. llvmpipe (2017)
[34] Ragan-Kelley et al. Halide (CACM 2017)
[35] Shao et al. DeepSeekMath (arXiv 2024)
[36] Gemma Team. Gemma 3 (arXiv 2025)
[37] Microsoft. DirectX Shader Compiler (2017)
[38] Unsloth AI Team (2023)
[39] Pol Vila & Quílez. Shadertoy (SIGGRAPH Asia '14)
[40] Yang et al. Qwen3 (arXiv 2025)
[41] Zakai. Emscripten (OOPSLA '11)

LMPL 2025 Preview

LMPL 2025 Preview

Thank You! Q&A

Questions & Discussion

Thank you very much for attending!

w2gpu.com - github.com/w2gpu

github.com/w2gpu/w2gpu-lmpl2025/blob/main/docs/presentation-202509.pdf

Oguz (Mehmet Oguz Derin)

@mehmetoguzderin