

LLM inference on the Raspberry Pi 5 via Vulkan

Chema Casanova

X.Org Developer's Conference 2026 - Toronto

Who am I?

- José María Casanova (a.K.a. Chema Casanova)
- Working at Igalia Graphics Team on Raspberry Pi graphics stack.
- Almost 10 years since my first Mesa contribution (2017-02-10)



YEARS OPEN



Me and Language Models (brief story)

- 20 years ago
- I knew about LM and Shannon information theory from Chengxiang Zhai and John Lafferty article "A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval".
- At that time I was pursuing a PhD on information retrieval and I tried to apply this technique to at sentence level instead to document level. (TREC Novelty Track).
- I forgot about the topic, and over the years LM became LLM.



Back to 2026

Everything started with:

*Have you tested the new **Ollama Vulkan backend on Raspberry Pi**? It seems that when the model starts the Vulkan check is failing and it seems to fallback to CPU.*



Edge AI on Raspberry Pi

- Running these compute **expensive workloads** locally on embedded devices like Raspberry Pi 5.
- Called **Edge AI**.
- It is already working on Pi CPU. Why not taking advantage of the V3D GPU?
- Any help is useful when you have limited resources.



Reachy Mini Powered by LiteRT and Gemma on Raspberry Pi 5

What Reachy Sees



Live Conversation

You: Hello, how are you doing?

Reachy: Hello, I am doing well, thank you!

You: What do you see at the table?

Reachy: I see a remote, a bottle, a book, and a table.

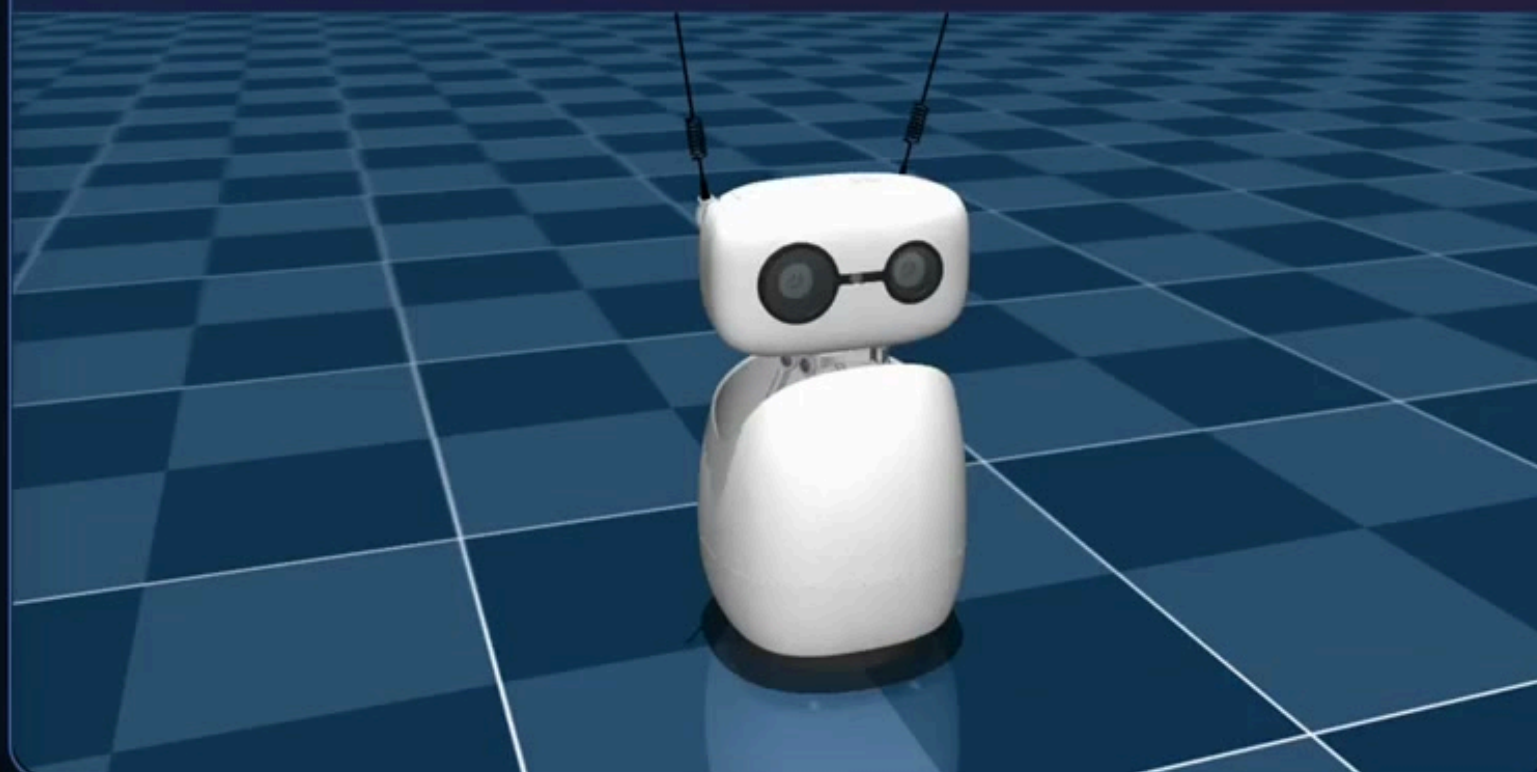
You: What do you see from the window?

Reachy: I see a car from the window.

You: Can you shake your head?

Reachy: **[shakes head]** I am shaking my head.

Reachy MuJoCo Simulation



System Logs

```
● ● ● reachy-litert-sample — python -m demo.run_demo
I heard: 'What do you see from window?'
  · car 0.65  · car 0.47  · car 0.39  · car 0.36
saying: 'I see a car from the window.'
first sound: 4216 ms

listening...
heard speech (2.2 s), looking...
I heard: 'Can you shake your head?'
  · bus 0.62  · truck 0.62  · car 0.35
gesture: shake
saying: 'I am shaking my head.'

listening...
```

LLM inference on the Raspberry Pi 5 via Vulkan

José María Casanova, X.Org Developer's Conference 2026 - Toronto



How 2026 started

Ollama and **LiteRT** couldn't take advantage of the V3D GPU.

- **Ollama**: Vulkan backend fallbacks to CPU.
 - Ollama / llama.cpp -> `ggml` -> Vulkan backend -> V3DV
- **LiteRT**: as we don't support WebGPU refuses to start with GPU backend.
 - LiteRT-LM -> WebGPU backend -> Dawn (WebGPU->Vulkan) -> V3DV



Nowadays

You can use **Ollama** and **LiteRT** GPU backends when running Raspberry Pi OS with downstream Mesa 26.2.2

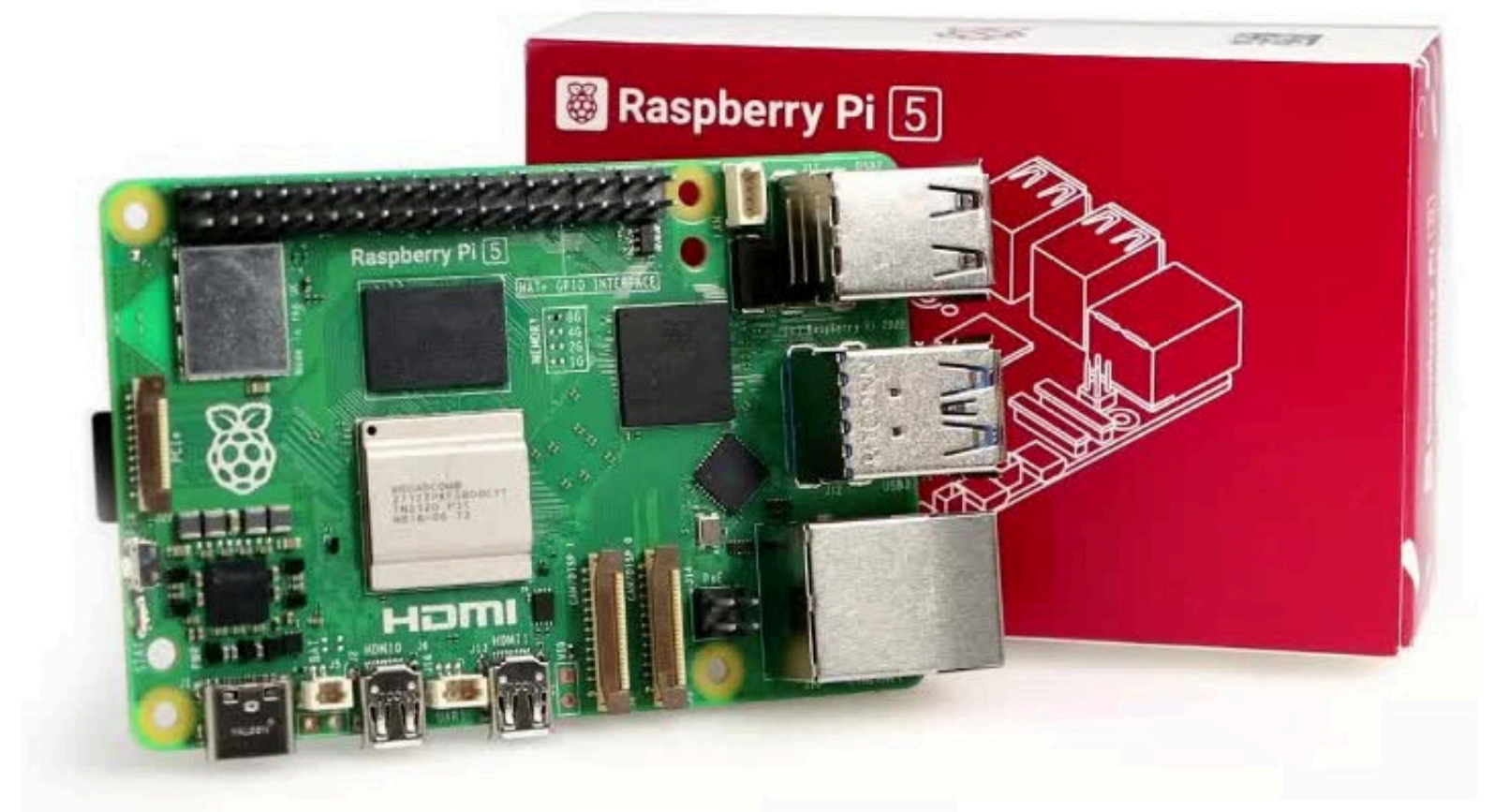
CPU is more performant than GPU in this workload. But GPU helps to free the CPU.

What follows is about what we learned and what we did in the process.



Target device: Raspberry Pi 5

- Broadcom BCM2712, quad-core Arm Cortex-A76 CPU, up to 2.4 GHz
- VideoCore V3D 7.1.7 GPU at 960 MHz,
 - OpenGL-ES 3.1 (v3d) and Vulkan 1.3 (v3dv) Mesa
 - Max workgroup = 256 invocations
 - 32-bit MMU: 4 GB GPU address.



The Ollama story



Ollama "Vulkan doesn't work on Pi 5"

- Long-standing community reports — it silently falls back to CPU.
- The symptom: shader compilation fails on the **first compute dispatch**, so Ollama disables the GPU and nobody sees why.



"Fixing" v3dv for Ollama **MR!41257**

- ggml-vulkan **hardcodes** `{512,1,1}` workgroups.
 - VVL are our friends (VUID-RuntimeSpirv-x-06432).
 - ignores `maxComputeWorkGroupInvocations = 256`. Crashes on debug build.
 - v3dv workaround is lowering oversized compute workgroups to 256 invocations (`nir_lower_workgroup_size`).
 - Used by radv/radeonsi do on older generations.
- Increase `maxComputeSharedMemorySize` to 32kb.
 - Even if Ollama doesn't reach that limit in our workloads.



Ollama helping to fix v3dv.

- After solving the `{512,1,1}` workgroups issue, the shader compilation never finished.
 - This exposed a non-convergence scenario in our optimizer loop.
 - Two nir passes were "progressing" and removing the "progress" in the same pass.
 - **MR!41256 "broadcom/compiler: move nir_lower_undef_to_zero out of the optimization loop"**.



Ollama works..., but ...

```
pi@home-rpi5-8gb:~ $ ~/devel/ollama/dist/bin/ollama run gemma3:1b "Where is Coruña?"  
Coruña is a city and municipality located in Galicia, Spain, on the west  
coast of the peninsula. More specifically, it's situated on the **Gulf of  
Atlántico**.
```

Here's a breakdown of its location:

```
* **Galicia, Spain:** It's part of the wider Galicia region, known for its  
stunning natural landscapes and rich cultural heritage.  
* **Gulf of Atlántico:** It's a major port city and a significant part of the  
coastline, offering beautiful views of the Atlantic Ocean.  
* **South of Porto:** It's roughly 60 kilometers (37 miles) south of the Porto^C
```

- But Vulkan backend has two issues:
 - Start up time: from neverending to 1h. **We are progressing!!**
 - Inference performance. lower than 1 token decode per second.



> 1 hour compiling shaders

- We have a disk shader cache. In a warm execution you don't wait much.
- Since we have disk shader cache we haven't paid enough attention to this issue.
- 1 hour for a cold run is too much. (For developers this is every time you modify mesa)
- Slow compilation **caused by shaders with hundreds/thousands of spills/fills.**



Why spilling hurts our compilation times.

- Our compiler tried an strategy ladder with 13 steps optimizing for less spills+fills.
- We spill when variables don't fit in our register budget (64 2-threads, 32 4-threads).
- We spill one register at a time -> run register allocation (RA) each iteration.
- We stop on the step where RA success without spills. Usually on the first strategy.
- For really spilling shaders, that is not possible, So we end exercising 13 strategies and the one with less spills wins.



Speeding cold compilation for heavy spillers.

- We knew there was an issue, we skipped multiple slow SSBO VK CTS tests.

```
On Pi 5: dEQP-VK.ssbo.phys...8bit.scalar.78 took ~26 min, now is 91 sec.
```

- We even had **V3D_DEBUG=opt_compile_time** to run CTS faster that stop the strategy ladder on the first one that success on RA.
- Experiment: removing all strategies and running only the conservative RA reduced Ollama start-up to ~10min.



1st step: reducing and stopping strategies

- Reduced from 13 to 6 the number of strategies.
 - Evaluated with Heavy spilling collection of fossils/shader-db.
 - Some strategies were rarely selected or with minor improvements.
- Early return when strategy spills more than the current best candidate.



2nd step: choosing better the strategy

- No spilling: you are always chosen.
- Low spilling shaders (reg-pressure ≤ 128)
 - Build all strategies and choose the less spilling compilation. On ties, the first at the ladder.
- Heavy spillers (reg-pressure > 128)
 - RA is ~90% of compile time in this kind of shaders.
 - Probe register pressure for all strategies and compile the best candidates.
 - Probe is fast. It is RA with spilling disabled to get max-temps.
 - If strategy is selected we reuse were probe first RA.
 - We define a threshold on reg-pressure when we need to eval multiple candidates.
- Included in Mesa 26.2 and implies 4.3x in some workloads.
 - CI rpi5-vk-full: 62.6 $\rightarrow$ 38.6 min (-38/-40 %).
 - Ollama first token: 1088 $\rightarrow$ 358 s (3,3x)
- Codegen neutral.



3rd step: Adaptive batch spilling !43073

- On huge over-pressure (> 32) spill batches of 16 register for top-candidates.
- Avoids RA to pick next candidate for next spill. We only update spill costs.
- On low over-pressure spill batches by a quarter of the pressure. To avoid over-spilling.

```
budget  = (PHYS_COUNT >> thread_index) + accumulators;    /* registers available to temps */
excess   = max(0, peak_pressure - budget);                  /* vir_get_max_temps() */
batch    = (excess > 2 * CAP) ? CAP : excess / 4 + 1;        /* CAP = 16 */
```

- We get a 2.3x speedup.



4th more spilling optimizations

- Peak pressure calculation at $O(n)$ (part of !43073)
- Reusing spill costs within a batch (!43159)
 - VK-CTS heavy SSBO spillers 14.8s → 14.1 s (-5%)
- Rebuild liveness once per spill batch (!43654) by JaeHoon Lee.
 - shader-db from 87.8s to 75.2s (-14.4%)
- Update liveness incremental (!44177) by JaeHoon Lee.
 - ollama-litert foz collection 10.26s -> 8.61s (-16.1%)



We have now faster shader compilation

Mesa	cold-clock	x step	x baseline
before !42889	745 s	—	1.0x
after !42889 (Mesa 26.2)	175 s	4.3x	4.3x
main (590bf21d918)	66 s	2.6x	11.3x

25 ollama gemma3:1b fossil pipelines on a Raspberry Pi 5

How to improve Ollama/llama.cpp for v3dv

- We benchmarked v3dv inference performance on LiteRT.
- There are still many optimizations opportunities in Ollama.
- **Inference engines have many codepaths gated by vendors.**
 - VK_VENDOR_ID_{INTEL,AMD,NVIDIA,APPLE,QUALCOMM}
 - No usage VK_VENDOR_ID_BROADCOM
 - Changing a 1 for a 2 in the correct location of llama.cpp can double your performance in V3D.

```
gemma3:270m      decode: 0.28 → 0.42 tok/s (+50%)  
prompt eval: 0.65 → 0.72 tok/s
```

- That was adjusting the shape of matrix to allow processing two rows per workgroup while you have registers free, adjusting to 4 is perf drop.



Performance opportunities

- Implement 16-bit support (halves TMU & ALU traffic)
- Use native v8dot for quantized matmul.
- Pre-RA pressure scheduling, loop depth scaled spill costs.
- Compute dispatch, track actual compute write targets instead of full barriers.
- TFU offload for copies/tiling.

This list of ideas was useful for our work on LiteRT GPU inference performance.



Current Ollama numbers

- Measured on RPi5 (Mesa 26.2.3, gemma3:1b Q4_K_M, ollama v0.34.3-rc1):

	CPU	GPU	CPU faster
Prefill	49.1 tok/s	5.24 tok/s	~9.4x
Decode	13.6 tok/s	0.28 tok/s	~48x

- The gap is *worse* than LiteRT-LM Gemma3-1B (8.3x decode): the Ollama path is the **least-optimised** one.



LiteRT-LM

Everything started with a lie followed by an OOM.



LiteRT-LM

- This January, Raspberry Pi involved Igalia in a collaboration with Google to enable GPU acceleration for LiteRT on RPi5.
- LiteRT for Linux GPU backend uses WebGPU via Dawn — which we didn't support.

```
litert_lm (LiteRT-LM, on-device LLM runtime + ml-drift
          |                                     + WebGPU connector)
          |
          ▼ WebGPU compute
Dawn (Chrome WebGPU) → Vulkan → V3DV (Mesa) → V3D 7.1 HW
```



WebGPU on V3D one year earlier.

- On 2025 we evaluated what would be needed to enable WebGPU on Chromium.
- Dawn is Chromium WebGPU implementation and implements WebGPU via Vulkan.
- At that time we understood that it was not possible.
- Dawn required ...
 - ... a **framebuffer size of 8192** and RPi5 only supports 7680, RPi4 4096.
 - ... **68 varyings** user components (now 72) and HW only has 64 (16 vec4)
 - ... **shader*ArrayDynamicIndexing** not implemented yet.



The Lie

- So the first step was just lying — exposing 8192 framebuffer, 72 varying components and all ArrayDynamicIndexing features.
- In the end **LiteRT was not using any of these WebGPU requirements**. We started running small vision models like PoseNet MobileNetV1 designed for edge devices.
- Performance of WebGPU backend it was extremely slow. Once it was "fast" enough we moved to LiteRT-LM to try to run gemma3:270m model.



The OOM

When we started running LiteRT-LM on GPU with the gemma3:270m model everything failed because we ran out of memory.

LiteRT-LM was dispatching a torrent of VkFill/Copy operations to reset to Zero the multiple buffers.

Why setting buffers to zero ends generating an OOM?



v3dv queues

- v3dv interacts with four job queues in the driver:
 - **CL** (Control List) — the classic 3D tiler render pipeline (binner + renderer, TLB-centric).
 - **TFU** (Texture Formatting Unit) — a fixed-function DMA/format-conversion engine.
 - **CSD** (Compute Shader Dispatch) — the actual compute queue.
 - CPU queue.
- The graphics workload uses mainly the CL queue.
- On LiteRT, LLM workload is mainly **CSD** usage, with a torrent of **fill/copy** to move data around.



Thousands of tiny buffer fills & copies

- Every LLM forward pass: zero-init scratch, copy weights/activations, repack tensors.
 - GFX workload usually does a handful of big copies.
 - LLM does **thousands of small ones** per token.
- These became **TLB render jobs** (512kb tile allocation, binning, RCL) just to clear or copy a buffer (maybe for 16 bytes). Enormous fixed overhead per op.



Our fills and copies response.

- Move as much as we can fills and copies off the render path onto TFU [MR!41725](#) extending TFU usage.
- But when we still need to TLB, stop paying for render setup reducing the over-allocation of 512kb tile_alloc to almost zero for copy operations. [MR!40554](#).
- Undocumented behaviour: we can fill zeroes with the TFU using as source a zeroed buffer with a the max_row size using stride-0 on RPi5.

With this memory pressure was reduced, and **LiteRT-LM started running**, but slow.



Making LiteRT inference faster

- Enabling V3D 7.1 ALU instructions.
- make quantized matmul fast: hardware-accelerated 4x8-bit dot products `v8dot` **MR!41255**
- emit supported packed-f16 ALU ops natively on V3D 7.1.
 - we needed to fix liveness analysis (partial register writes)
- enabled nir lowered shaderFloat16/Int16/Int8 + VK_KHR_shader_float16_int8 **MR!41810**
 - We take advantage of supported fp16 ALU operations the HW support.
 - WIP: take advantage of vectorizing fp16 ALU operations as f16vec2.



Only last bug hunting.



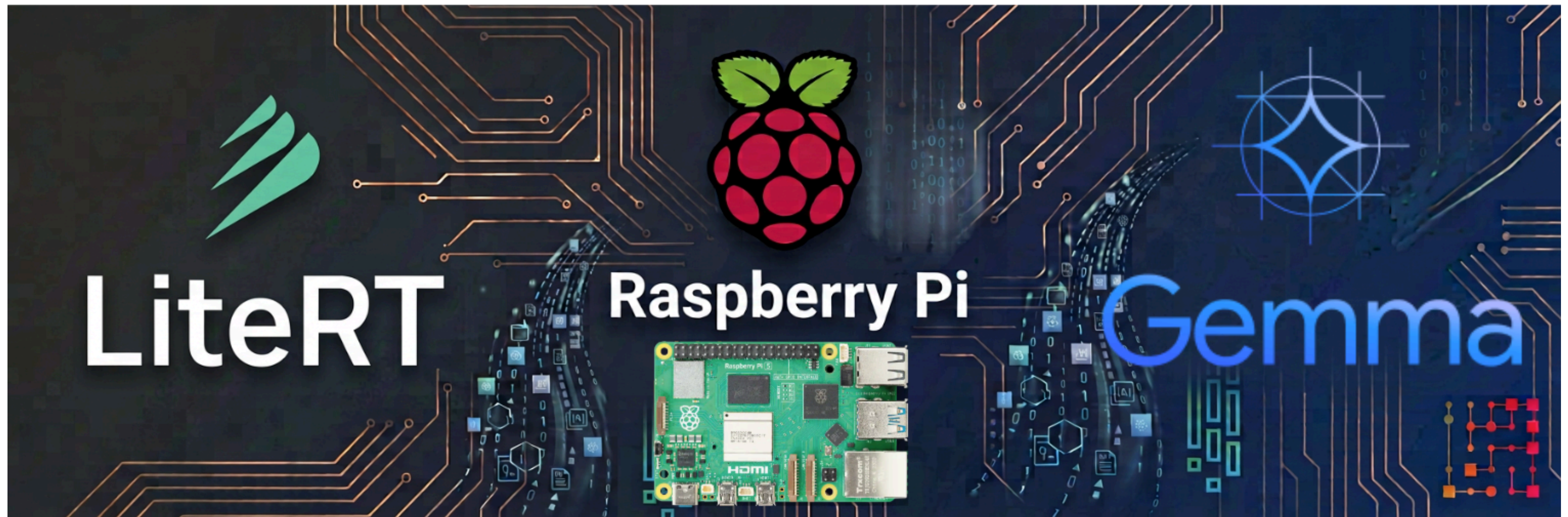
Only last bug hunting.

- Last minute fp16 precision with **Gemma4-E2B**. It was hallucinating with native fp16 ALU ops.
 - "Burj Khalifa" was not the answer, it was "Aula Iso Hungary..."
 - But lowering fp16 $\rightarrow$ f32 (f16 $\rightarrow$ f32 $\rightarrow$ f16) **worked**.
 - We forgot to restrict fsat_signed/fclamp_pos to 32-bit when we enabled fp16.

```
- (( 'fmin', ('fmax', a, -1.0), 1.0), ('fsat_signed', a), has_unpack_sat),
- (( 'fmax', ('fmin', a, 1.0), -1.0), ('fsat_signed', a), has_unpack_sat),
- (( 'fmax', a, 0.0), ('fclamp_pos', a), has_unpack_max0),
+ (( 'fmin', ('fmax', 'a@32', -1.0), 1.0), ('fsat_signed', a), has_unpack_sat),
+ (( 'fmax', ('fmin', 'a@32', 1.0), -1.0), ('fsat_signed', a), has_unpack_sat),
+ (( 'fmax', 'a@32', 0.0), ('fclamp_pos', a), has_unpack_max0),
```



LiteRT is working with V3DV



- With `V3D_WEBGPU_OVERRIDE=1` you can run LiteRT-LM on Mesa main today or in Raspberry Pi OS.



the LiteRT-LM numbers

The A76 CPU beats the V3D GPU at both phases. Gemma3-1B-int4, 128 prefill / 64 decode, current Mesa 26.2.3

	CPU (4xA76)	GPU (V3D)	CPU faster
Prefill	64.0 tok/s	20.7 tok/s	3.1x
Decode	6.88 tok/s	0.83 tok/s	8.3x
Init, warm	0.5 s	2.6 s	-

CPU is still faster, but we started being 100x slower :)

Conclusion

Work on enabling LLM inferences workload has helped us to:

- Huge reduction of compilation time.
- Enabling more codepaths blits/copy/fills to use TFU free CLE&CSD.
- We already expose native fp16.
- We enabled experimental WebGPU support.
- Ollama and LiteRT can run on Raspberry Pi 5.



What's next

- More performance.
 - dual fp16 emission. fp16vec2.
 - more performance.
- WebGPU support?
 - I can talk about it if I didn't run out of time yet.
 - enable webgpu cts.
 - maybe we can enable v3dv on dawn.



Thank You

Join us!

<https://www.igalia.com/jobs>



What about WebGPU support?

Take advantage of the extra coverage of WebGPU CTS.

- We can test two implementations (dawn or wgpu)
- When we run it ~10,000 tests failed with a 1.3 conformant driver.
- We think it is valuable the extra coverage to complement VK-GL-CTS.



Enable dawn/WebGPU on v3dv ?

- 72 varying components : 64 user + 4 Position + 4 center_pos. Position in v3dv doesn't consume user varyings.
- compat-limits would allows a compatibility framebuffer size of 4096.
- WIP enable copy/blit/clear with 8192 dimension for 2D and 16384 for 1D/3D dimensions. This would allow increasing maxImageDimension with our 7680 framebuffer size.
- We need to implement shader*ArrayDynamicIndexing.



Thank You Again

Join us!

<https://www.igalia.com/jobs>





YEARS OPEN



