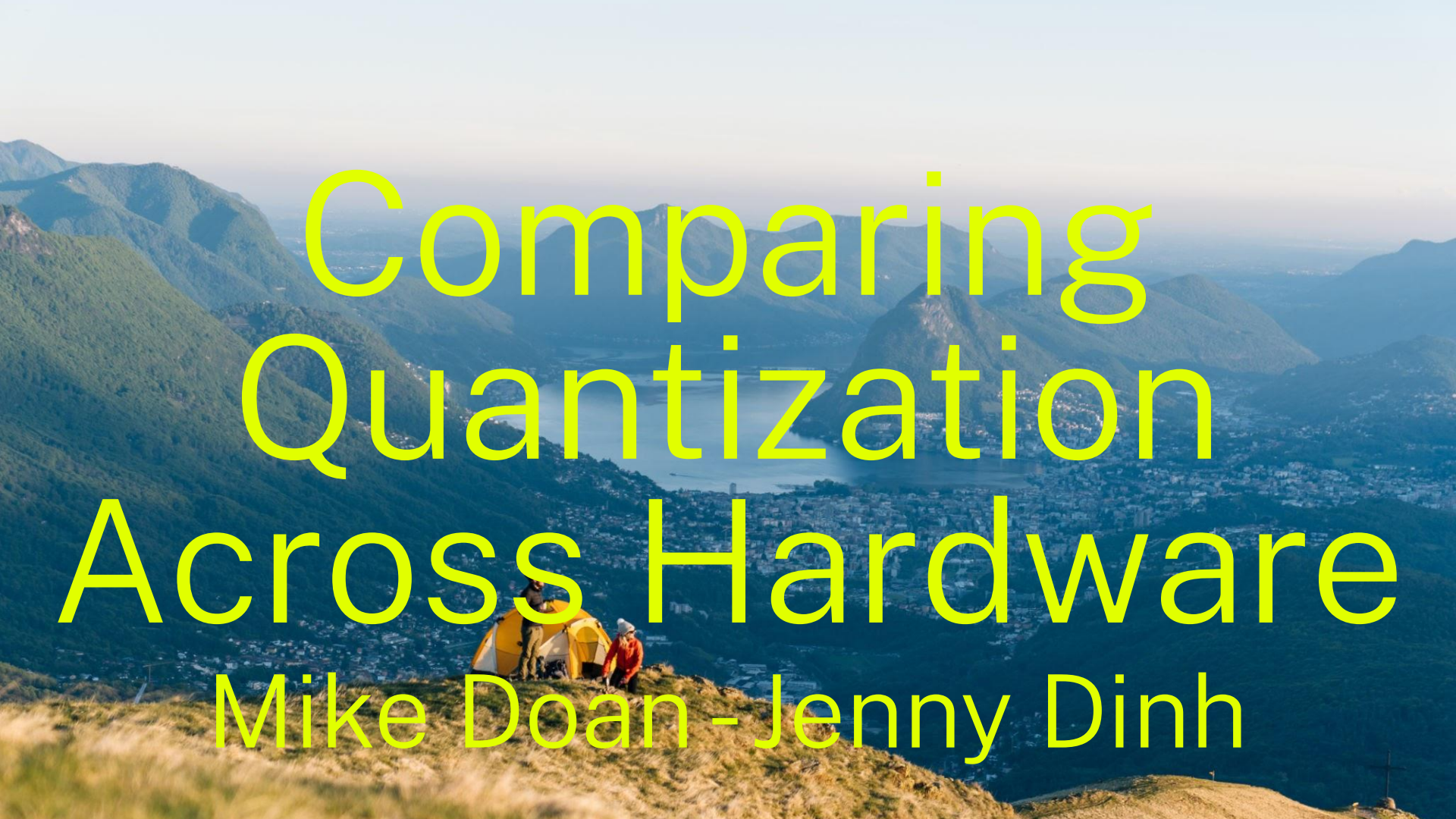


Comparing Quantization Across Hardware

Mike Doan - Jenny Dinh

Abstract

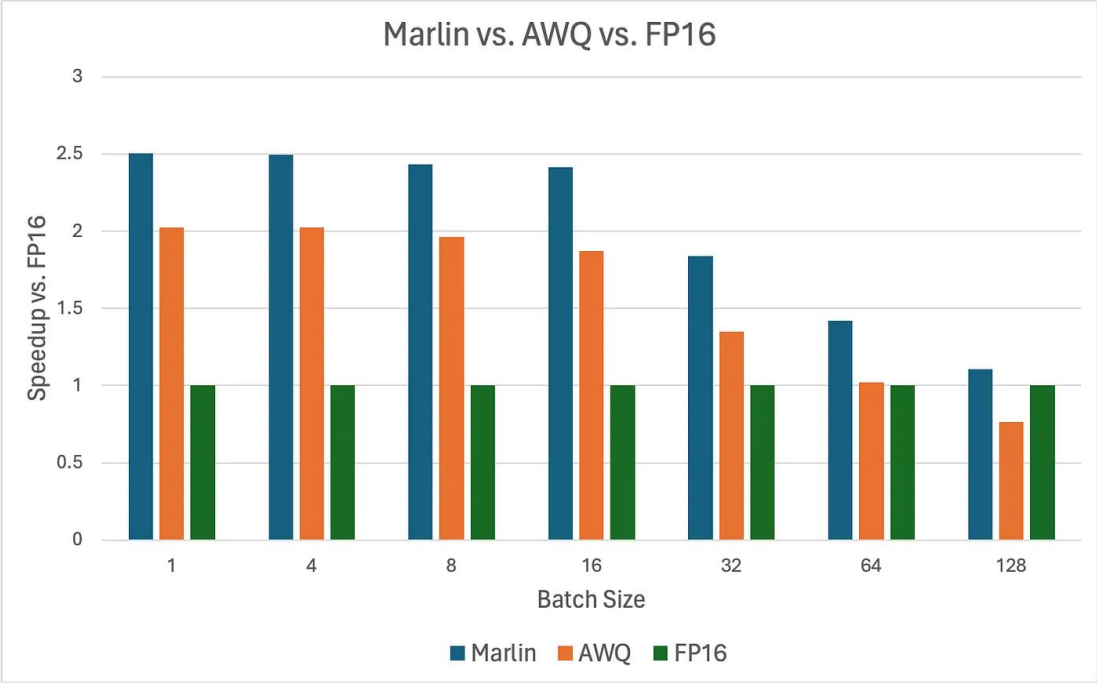
There is a growing interest in running LLMs locally, either on dedicated devices like desktop / laptop gpus, or on edge devices like smartphones or in embedded environments.

However, as LLMs contain not only massive weights that have to be stored in (at least partially) memory, but also large activations, there have been many techniques developed to attempt to minimize this requirement. One such technique is quantization

This work aims to investigate the differences of quantization methods, on a multitude of devices ranging from android phones, to gaming laptops, and enterprise grade GPUs.

In particular, we aim to investigate how different quantization standards and strategies impact accuracy, latency, throughput, energy, and memory usage.

Abstract



https://medium.com/incodeherent/llm-quantization-performance-0887e9fbd06c

Metrics

We measure following metrics across a range of devices (android phone, laptop, and desktop gpus)

Accuracy

Against benchmarks, or by using raw PPL on a representative dataset like Fineweb or WikiText

Latency

Measured as time until first token from token submission, both including and not including tokenization and preprocessing.

Throughput

Raw token/s. Measured across seqlens, kvcache sizes and other inference configs.

Energy

Watt/Tok. Measured as delta between sampled idle watt consumption and max load. Also interesting is seeing how throttling impacts perf.

Memory

Raw MB used. Both on model load, and during peak utilization. (activations are large :( )

Core Technologies

Models

- Phi-3 Mini (~3.8B)
- Qwen2.5-3B (and Qwen 3)
- Llama-3.1-8B-Instruct

Runtimes

- llama.cpp (GGUF)
- MLC LLM (TVM-based)

Quantization strategies

- FP16 (or BF16) – reference
- INT8 (W8A8)
- INT4 (W4A8 / W4A4)
- KV-cache quantization (INT8/INT4)
- GPTQ/AWQ group-wise quant

Workloads

- Prefill tokens: 256, 1024, 2048
- Decode tokens: 128, 512 (batch=1)
- Prompt mix: short QA, reasoning, few-shot

Extending scope

In addition to the default project goals we also would like to accomplish the following for potential lasting impact:

Durable workflows

There are few **plug-and-play** toolkits available on the market for sweeping across quantization strategies.

We would like to **automate** as much of the process as possible

Others can use the code to benchmark their devices / find optimal quantization strategies.

Standardized Data

We aim to use a **well thought out data format,** suitable for public indexing and searching.

This will make it easy for us to **collate** and **illustrate** out results, and also make it simple for others to append their devices onto a shared data bank.

Benchmarking

In addition to measuring raw perplexity, we also aim to measure **downstream performance on actual benchmarks.**

For this, we will use the **verifiers framework** https://github.com/willccbb/verifiers, which has a library of premade, publicly maintained plug and play benchmarks.

Enterprise GPUs

As mentioned in previous slides, we also aim to target **enterprise GPUs** as part of our investigation.

This is because we believe that most models, even local ones, will still have the highest return on investment when deployed on chips with proper **compute / ram balancing,** and this will benefit from a cohesive sweep.

Expected challenges

Software incompatibility

Since we are targeting multiple backends (ARM for phones, x86/CUDA for desktops and laptops), we expect some software to not natively work.

Implementation inconsistencies

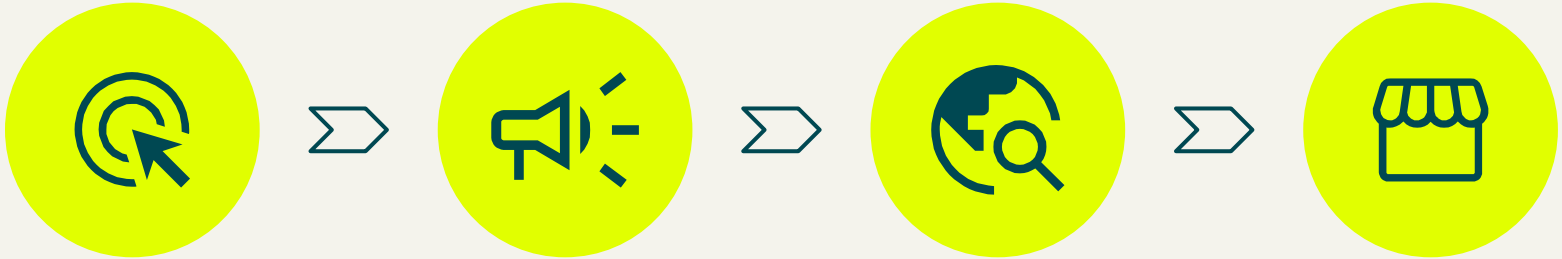
It is possible that different quantization standards are not implemented the same way across backends.

This will likely lead to discrepancies in measured performance.

Engineering challenges

Because we are extending the scope of the project to the creation of a durable and generic benchmarking suite, we expect to face some design / implementation challenges.

Research outputs



Research writeup

We will write a detailed report of our work in standard arxiv preprint format. This will contain the fine details of our techniques, and analysis of our results.

Presentation

TBA

Public codebase

We will release the code under a permissive MIT license.

It will be well tested, easily reproducible, and target as many backends as we can manage.

Any suggestions?

TBA

Timeline

- Week 1: Sanity checks: focus on llama.cpp, get inference running on pixel phone, macbook laptop, x86 desktop on cpu and gpu.
- Week 2: Data format definition, trivial sampling: (note: hard coded inputs for data simplicity)
- Week 3: End to end automation. CI backend., Correct sampling: wikitext, fineweb sampling. Results for llama.cpp should be collected.
- Week 4: MLC backend.. New results.
- Week 5: Verifiers benchmarking. Work on writeup.
- Week 6: Full sweeps, scale up. (Sourcing data publicly?)
- Week 7: Headroom for project scope.
- Week 8: Headroom for project scope.

Thank You!