DEPLOYING LLM ON IPHONE USING LLAMA.CPP AND METAL

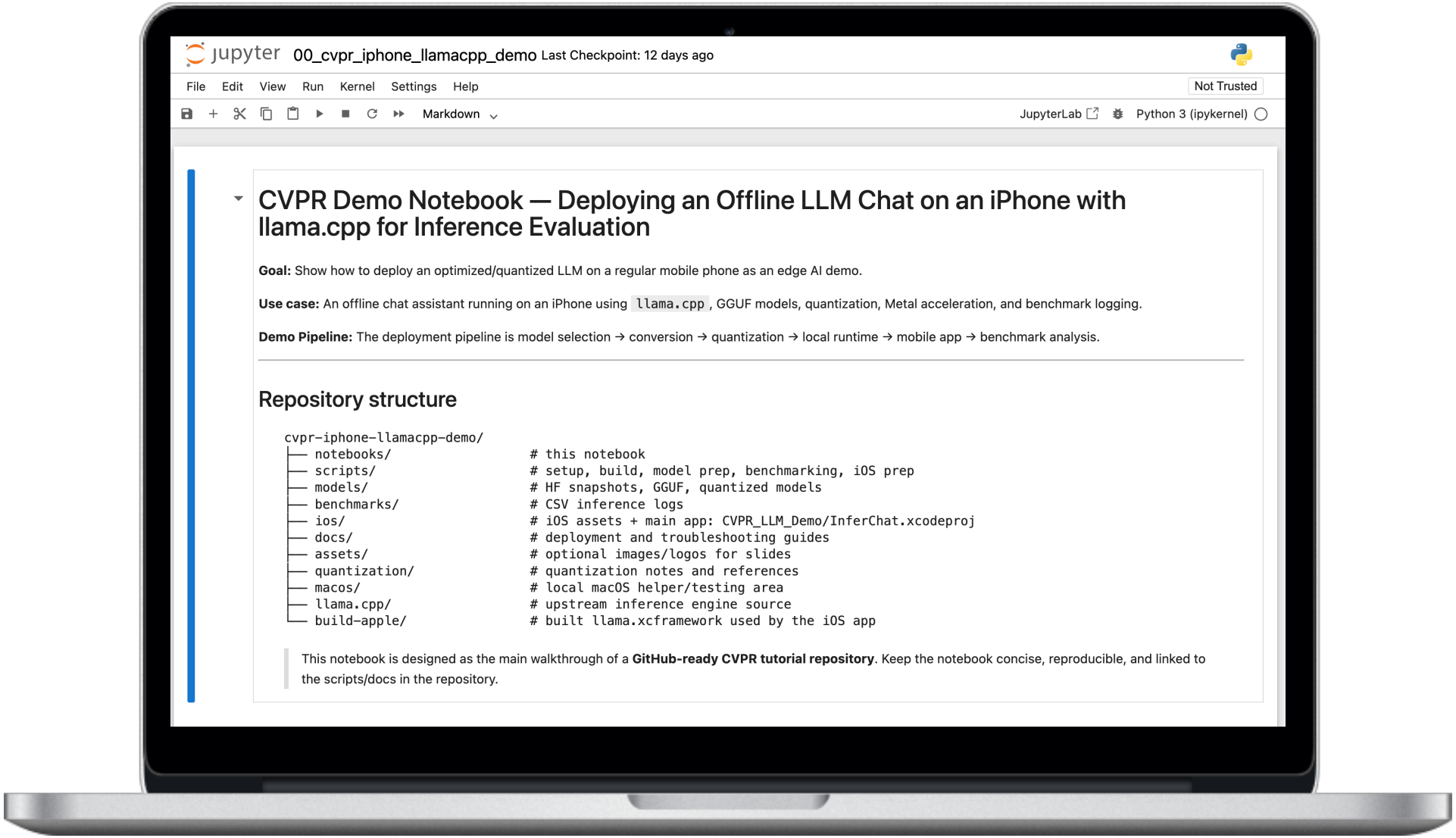
CVPR 2026 Tutorial

The IEEE/CVF Conference on Computer Vision and Pattern Recognition 2026

Denver, CO, USA

THE NOTEBOOK DEMO

http://localhost:8888/lab/tree/notebooks/00_cvpr_iphone_llamacpp_demo.ipynb



PROJECT WORKFLOW

Main Steps

Prepare the Environment

Install Apple tooling and Python prerequisites, then run the bootstrap script to create .venv and install dependencies.

Prepare models for mobile inference

Either download ready GGUF models or quantize your own from Hugging Face to Q4_K_M (or another preset).

Build llama.cpp with Metal

Validate before deploy and run local chat/inference to confirm model quality and runtime stability before moving to iPhone.

Deploy and run on iPhone

Prepare iOS build artifacts, open the existing Xcode project, add/bundle or download models in-app, and run on device.

Benchmark, export, and analyze results

Collect inference metrics in app, export CSV, replace the template CSV content, and run notebook analysis for latency, tokens/s, memory, and CPU/GPU comparisons.

PREPARE THE ENVIRONMET

Requirements for the MAC OS Development

Apple Dev Stack

Xcode App
Xcode Command Line Tools (`xcode-select --install`)
IOS SDK and signing support through Xcode

Native Toolchain

`git` for clonning/updating repos
`cmake` For configuring native builds
`ninja` (or compative) for fast C7C++ builds
`make` ans Apple clang toolchain

Phython Runtime

Python 3.11+
`pip` and virtual environment tooling (`venv`) for script execution

iPhone 14
128GB A15

Inference Runtime Framework

`llama.cpp` (repository subfolder) as local inference runtime
Metal backend support for Apple GPU acceleration

Python Libraries

Model/download: `huggingface_hub`, `transformers`, `sentencepiece`
Data/analysis: `pandas`, `matplotlib`, `psutil`
Serialization/helper: `nbformat` (from conda/pip env. setup)

Developer Workflow Tools

VS Code
Homebrew

Notebook: Use 00_setup_mac.sh and Section 1 in 00_cvpr_iphone_llamacpp_demo.ipynb.

PREPARE MODELS FOR MOBILE INFERENCE

Using Ready-to-Run Quantized Models

GGUF MODEL

GGUF is the model file format used directly by llama.cpp at inference time. It can be in quantized K-Quants variants (such as: Q4_K_M). For example: Qwen2.5 0.5B Instruct Q4_K_M

Q4_K_M is a GGUF quantization format commonly used with llama.cpp. It means:

Q4

4-bit quantization (smaller and faster than FP16/FP32).

K

Improved quantization grouping scheme.

M

Medium balanced configuration between: *Speed*, *RAM Usage*, and *Model Quality*.

PREPARE MODELS FOR MOBILE INFERENCE

Quantizing a Hugging Face Model

DOWNLOADING DATASET

Download the model snapshot from Hugging Face.

CONVERTING MODELS

Convert source weights to an intermediate GGUF F16 format using llama.cpp utilities:

```
$ "$PYTHON_BIN" “$LLAMA_DIR/convert_hf_to_gguf.py" \
    "$HF_DIR" --outfile "$GGUF_F16" --outtype f16
```

QUANTIZING MODELS

Quantize GGUF F16 to the target format (for example, Q4_K_M) using llama.cpp.

```
$ "$LLAMA_DIR/build/bin/llama-quantize" \
  "$GGUF_F16" "$GGUF_Q" "$QUANT_TYPE"
```

OUTPUT ORGANIZATION

Save quantized models in consistent locations under models/hf and models/quantized.

BUILDING LLAMA.CPP WITH METAL

Validate Before Deploy

After a successful run, you should have compiled `llama.cpp` artifacts and no build errors in the cell output.

Build llama.cpp on Mac first to verify your environment and Metal runtime, then move to iPhone with a known-good inference stack.

LLAMA.CPP — Clones or updates `llama.cpp` in the repository.

METAL — Configures a build with Apple/Metal support.

BINARIES — Compiles binaries and supporting artifacts used in later validation steps.

ARTEFACTS — Leaves build outputs in predictable folders for reuse.

Notebook: Use 01_clone_build_llamacpp.sh and 04_run_mac_chat.sh.

DEPLOY AND RUN ON IPHONE

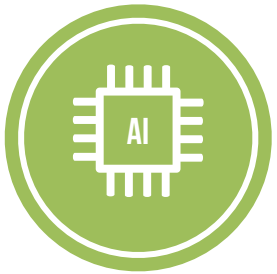
Deploy the Chat Application on the iPhone for Inference

1 - iOS Project

Prepare iOS buid artifacts and open the project on Xcode project.

2 - Deploy on Device

Build the project InferChat on the iPhone using Xcode.



3 - LLM Models

Download new models or bring your own model (BYOM) in GGUF format.

Notebook: Use 06_prepare_ios_cvpr_demo_app.sh with InferChat.xcodeproj.

BENCHMARK, EXPORT, ANALYZE

Test the Chat Application with Diverse Configuration

Prompting

Choose the models, configuration and chat with different LLMs

Export Data

Have all the inference data saved in .csv file for benchmark

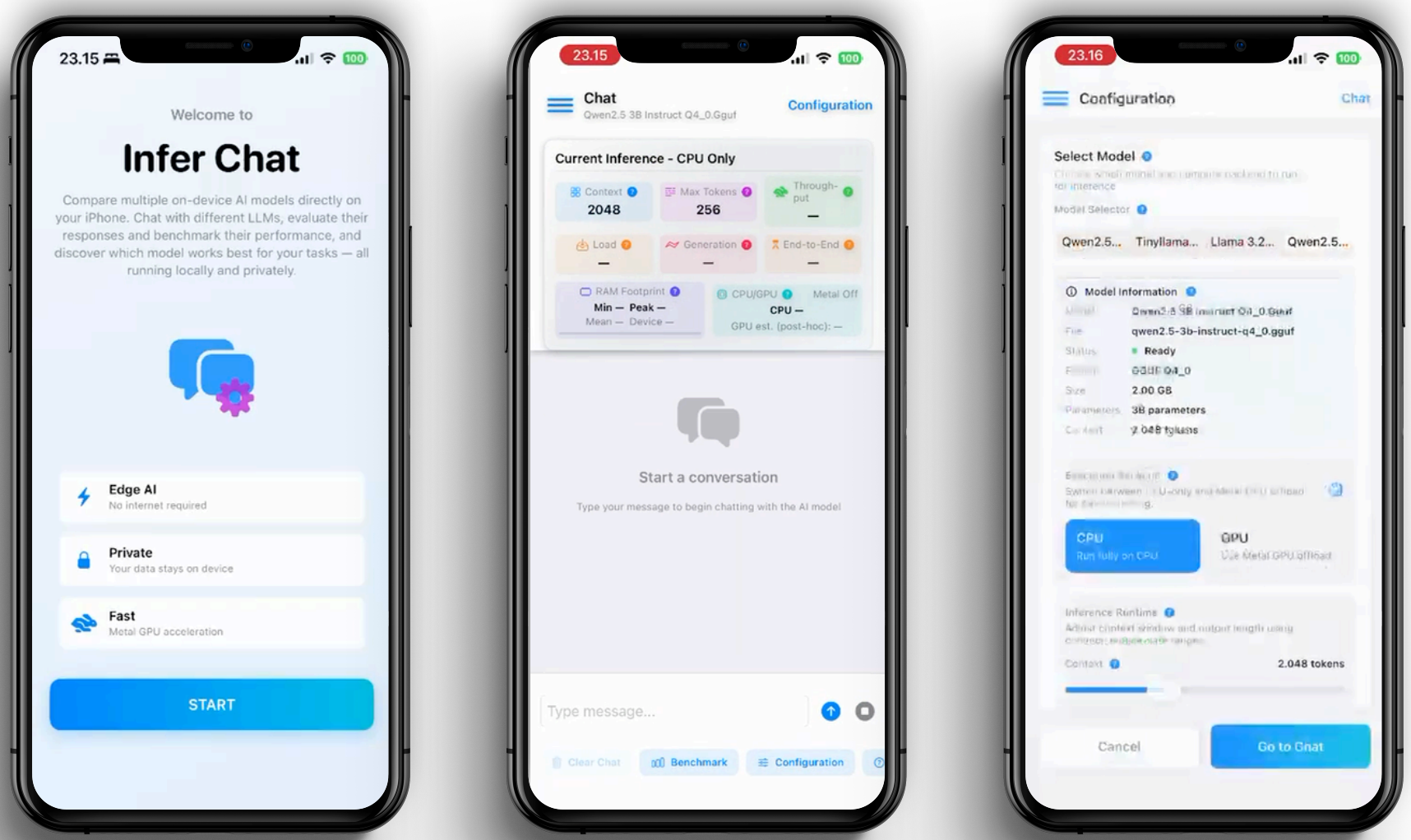
Analyse the models performace

Open the file on Notebook and analyse the benchmark

Notebook: Use 07_collect_ios_benchmark_template.csv and Section 9 in 00_cvpr_iphone_llamacpp_demo.ipynb.

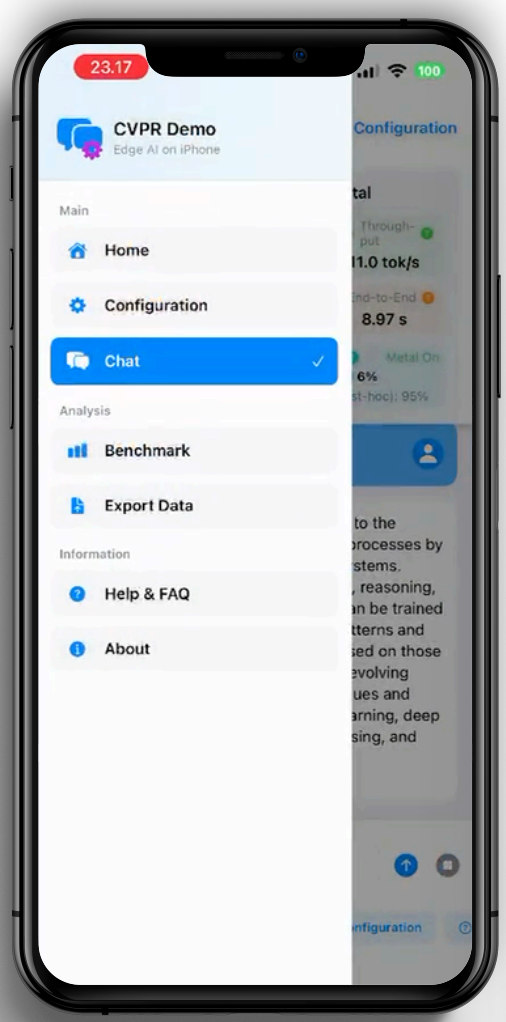
INFERCHAT APP

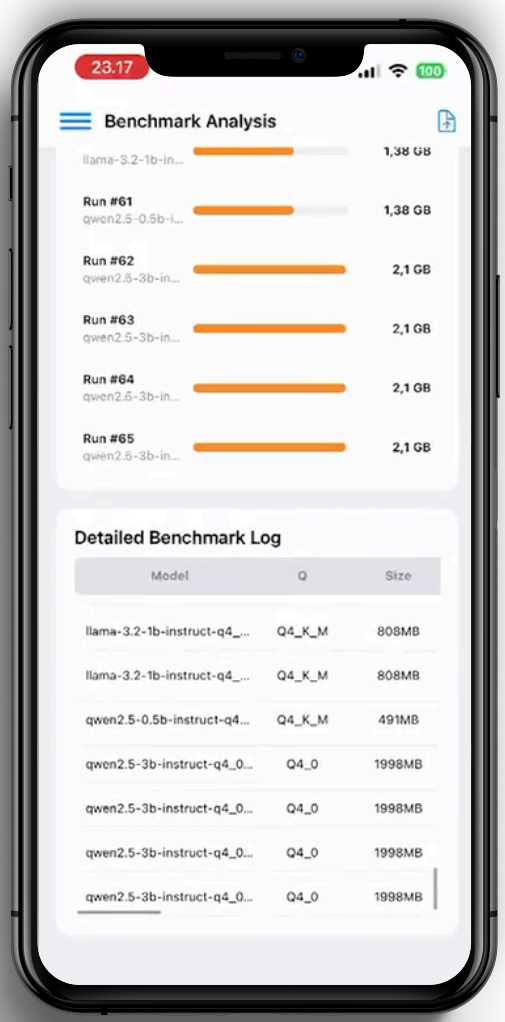
Demo



INFERCHAT APP

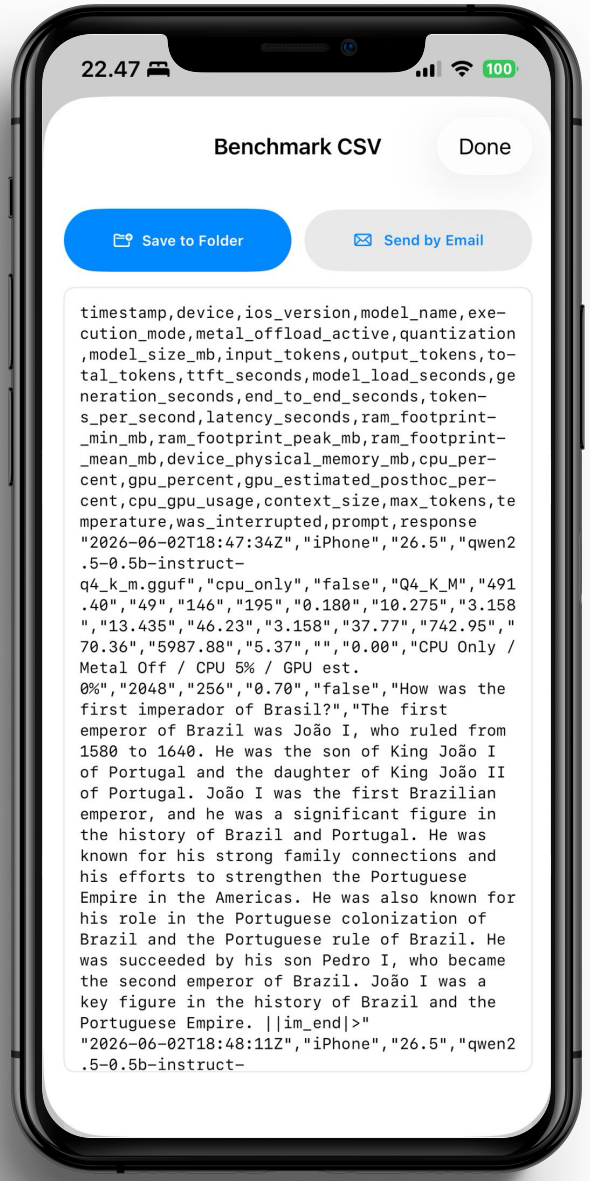
Demo



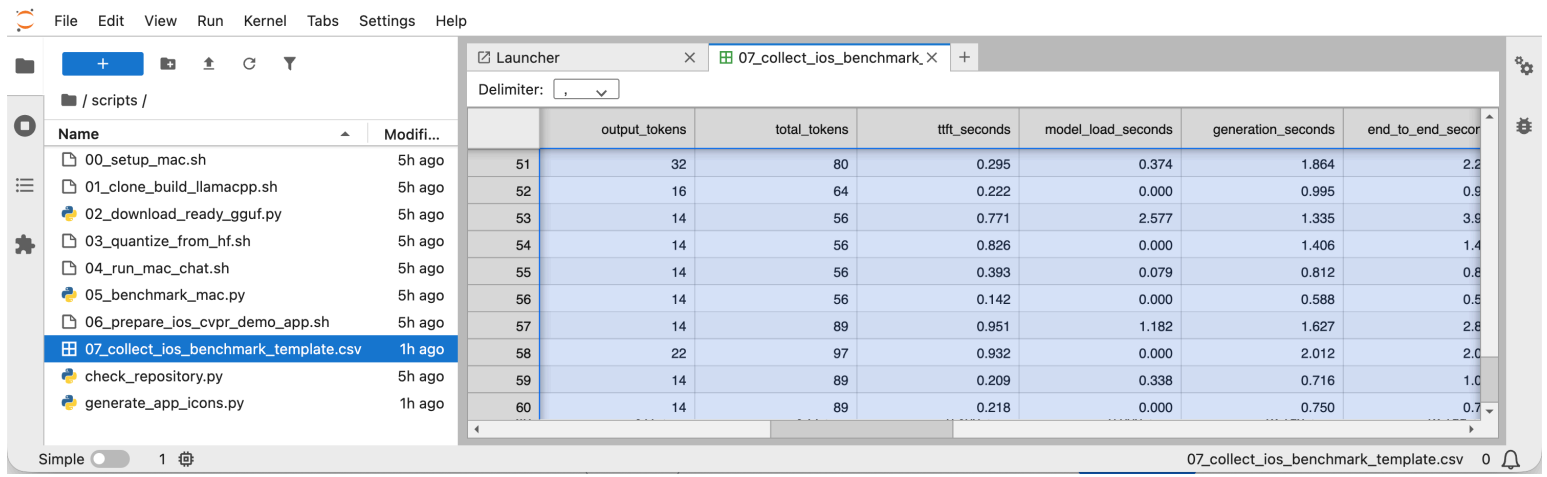


SAMPLE BENCHMARK ANALISYS

Experimental Analysis

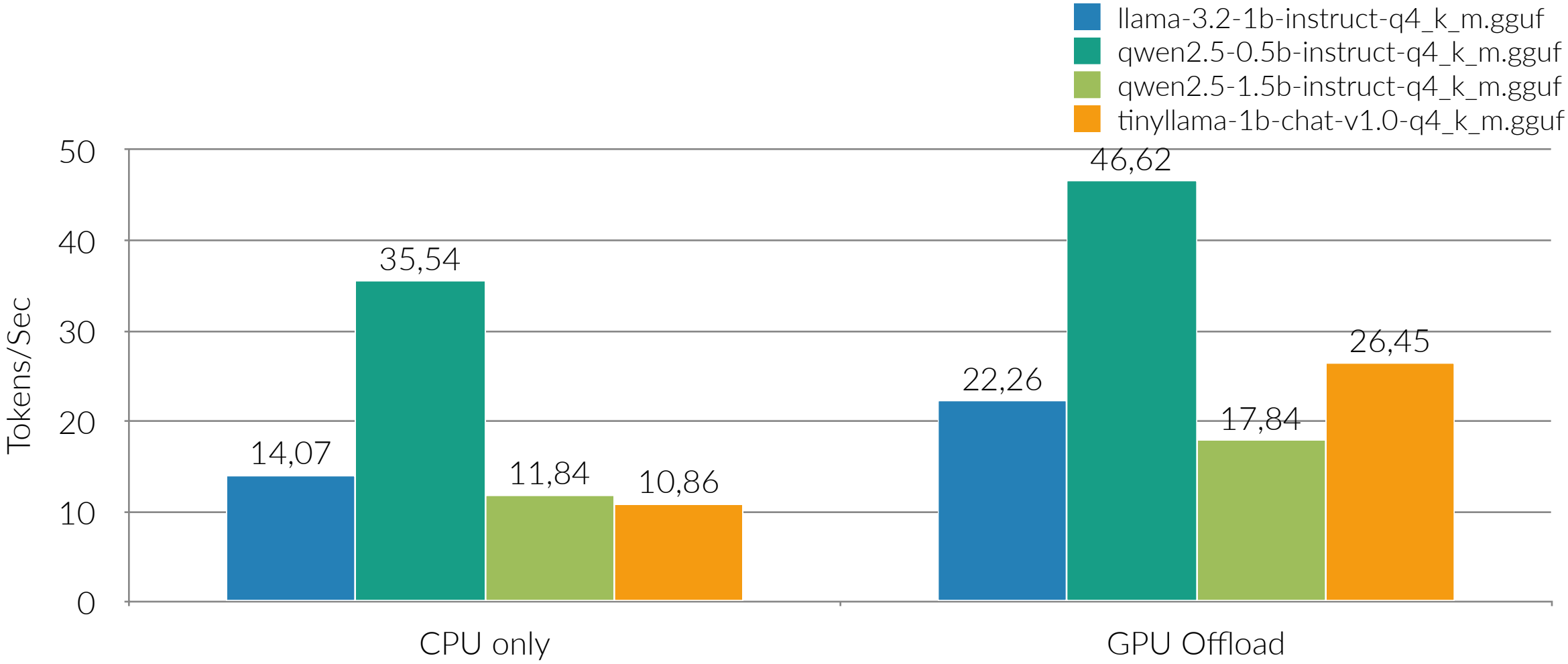


The experimental dataset is 60 rows overall, with mode-specific aggregation on those 24/36 rows.



BACKENDS COMPARISON

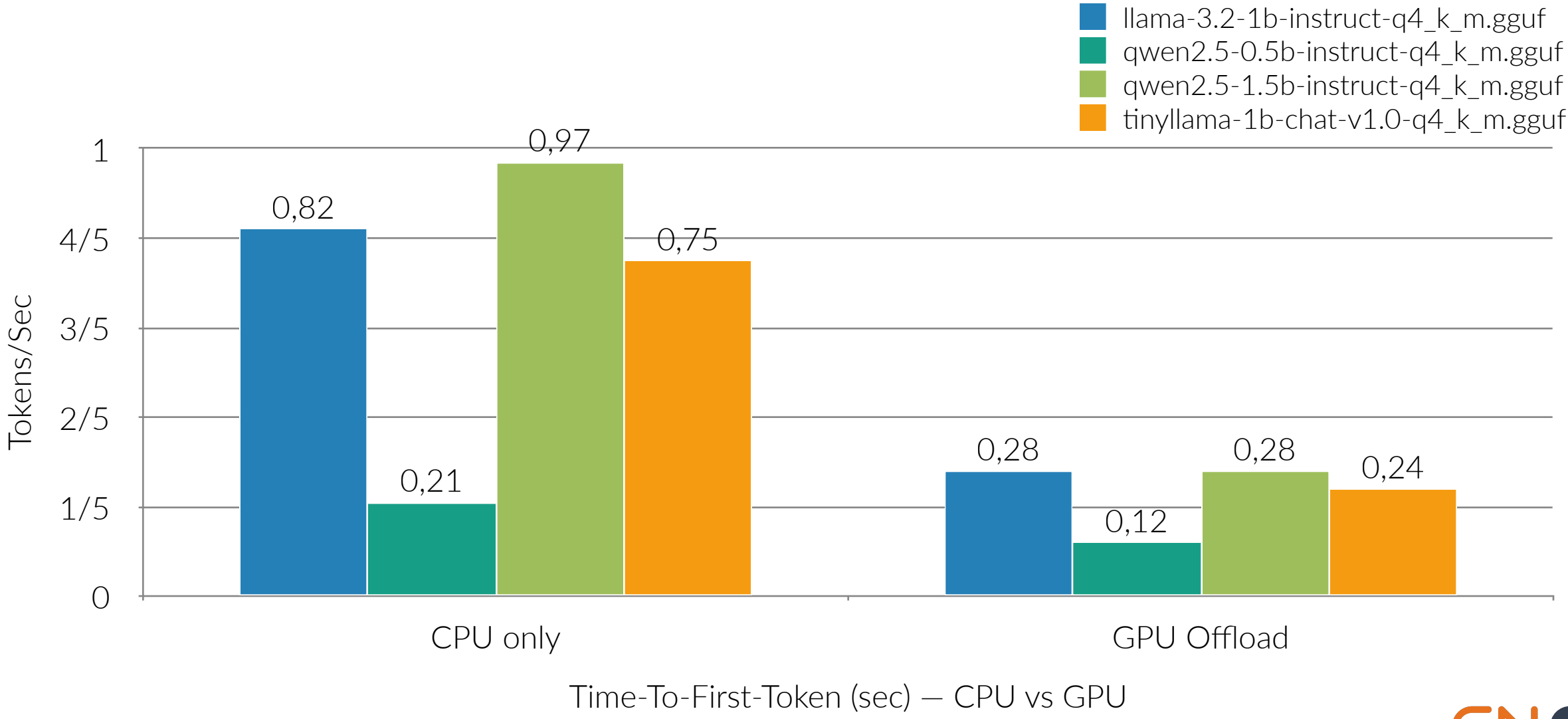
Output Throughput - Tokens per Second



Output Throughput (tokens/sec) — CPU vs GPU (Metal)

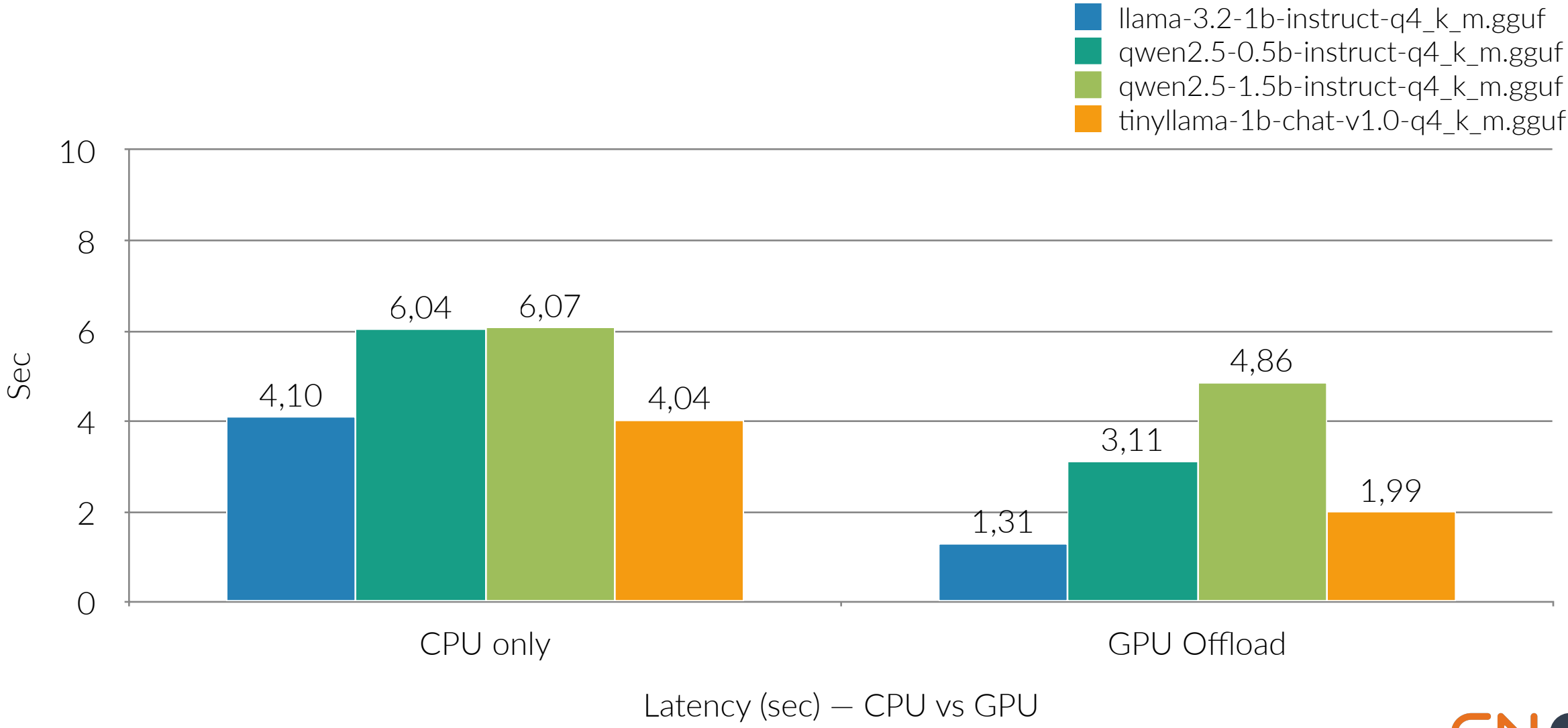
BACKENDS COMPARISON

Time-To-First-Token



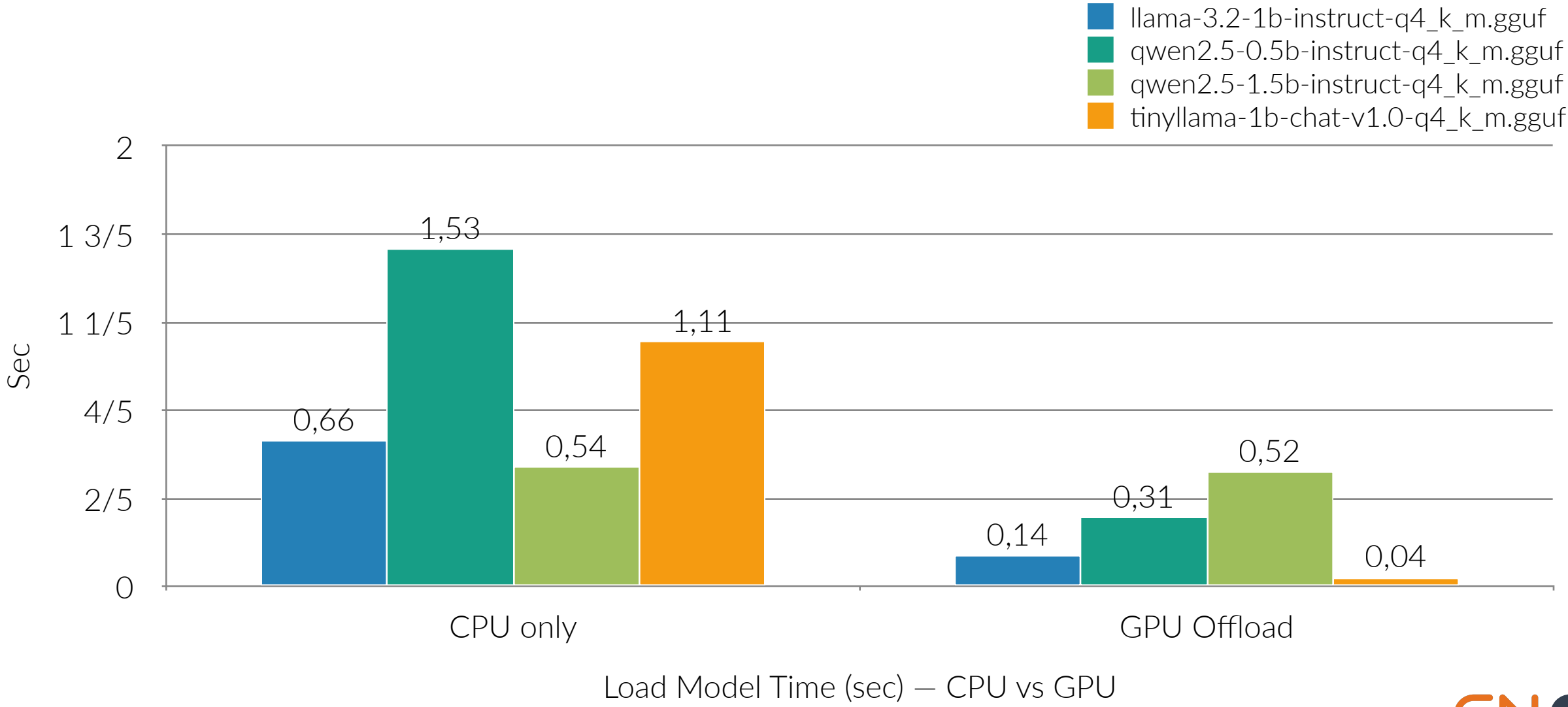
BACKENDS COMPARISON

Latency



BACKENDS COMPARISON

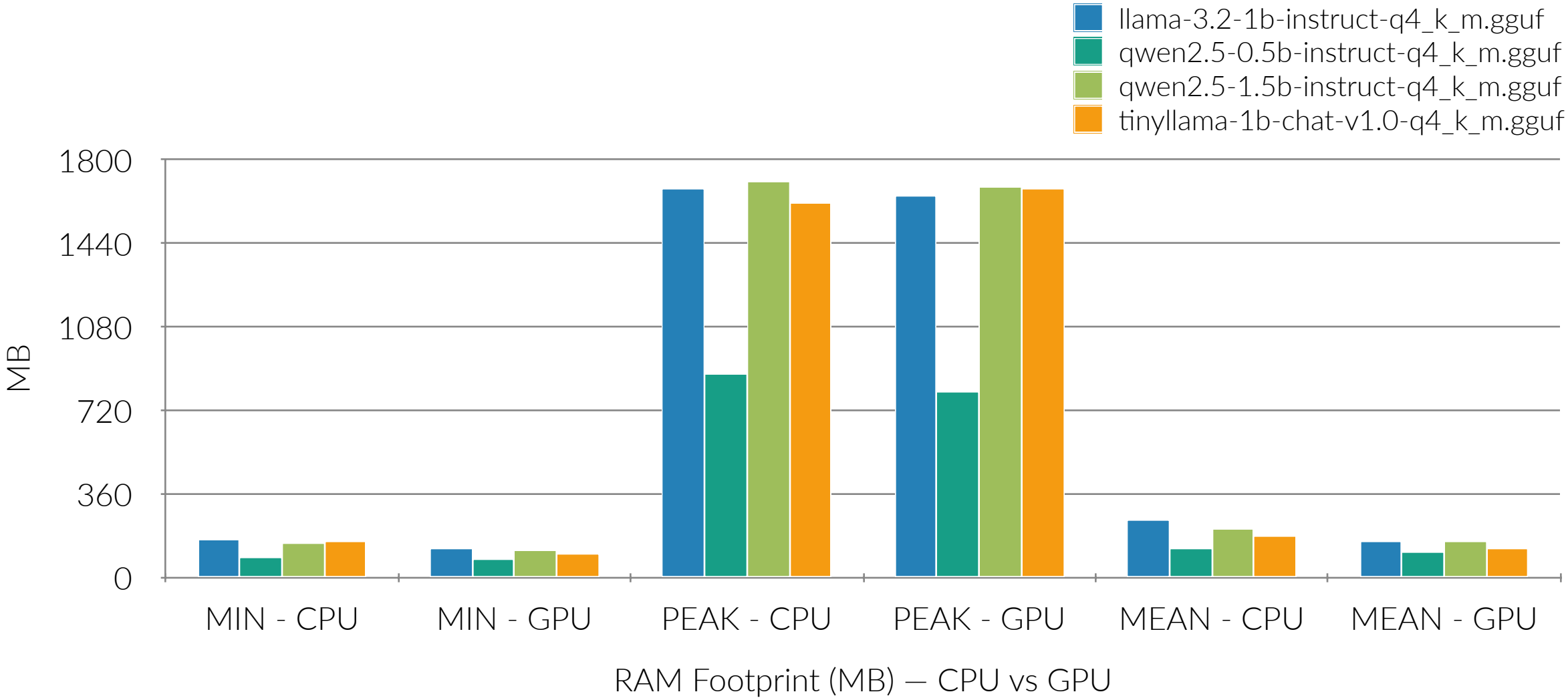
Load Model Time



Load Model Time (sec) — CPU vs GPU

BACKENDS COMPARISON

RAM Footprint



CONCLUSIONS OF THE EXPERIMENTAL ANALYSIS

Key Performance Insights from On-Device LLM Inference

tinyllama-1b-chat-v1.0-q4_k_m.gguf

- Largest GPU throughput gain: **+143.6%** vs CPU
- Largest model-load gain with GPU: **-96.1%** load time

llama-3.2-1b-instruct-q4_k_m.gguf

- Largest latency drop with GPU: **-68.1%** vs CPU

qwen2.5-1.5b-instruct-q4_k_m.gguf

- Smallest model-load time with CPU

qwen2.5-0.5b-instruct-q4_k_m.gguf

- Best throughput overall: **43.22 tokens/sec**
- Best GPU efficiency: **59.49 tokens/sec per GB peak RAM**
- Best CPU-only throughput: **35.55 tokens/sec**
- Best TTFT overall: **0.152 sec**
- Best TTFT on GPU: **0.124 sec**
- Lowest GPU peak RAM: **802.6 MB**

QUESTIONS & ANSWERS

THANK YOU!