

MELTing point: Mobile Evaluation of Language Transformers

Stefanos Laskaridis
 mail@stefanos.cc
 Brave Software
 London, UK

Kleomenis Katevas
 kkatevas@brave.com
 Brave Software
 London, UK

Lorenzo Minto
 lminto@brave.com
 Brave Software
 London, UK

Hamed Haddadi
 hhaddadi@brave.com
 Brave Software &
 Imperial College London
 London, UK

ABSTRACT

Transformers have recently revolutionized the machine learning (ML) landscape, gradually making their way into everyday tasks and equipping our computers with “sparks of intelligence”. However, their runtime requirements have prevented them from being broadly deployed on mobile. As personal devices become increasingly powerful at the consumer edge and prompt privacy becomes an ever more pressing issue, we explore the current state of mobile execution of Large Language Models (LLMs). To achieve this, we have created our own automation infrastructure, MELT, which supports the headless execution and benchmarking of LLMs on device, supporting different models, devices and frameworks, including Android, iOS and Nvidia Jetson devices. We evaluate popular instruction fine-tuned LLMs and leverage different frameworks to measure their end-to-end and granular performance, tracing their memory and energy requirements along the way.

Our analysis is the first systematic study of on-device LLM execution, quantifying performance, energy efficiency and accuracy across various state-of-the-art models and showcases the state of on-device intelligence in the era of hyperscale models. Results highlight the performance heterogeneity across targets and corroborates that LLM inference is largely memory-bound. Quantization drastically reduces memory requirements and renders execution viable, but at a non-negligible accuracy cost. Drawing from its energy footprint and thermal behavior, the continuous execution of LLMs remains elusive, as both factors negatively affect user experience. Last, our experience shows that the ecosystem is still in its infancy, and algorithmic as well as hardware breakthroughs can significantly shift the execution cost. We expect NPU acceleration, and framework-hardware co-design to be the biggest bet towards efficient standalone execution, with the alternative of offloading tailored towards edge deployments.

KEYWORDS

Machine Learning, Mobile Systems, Large Language Models

1 INTRODUCTION

Our devices are getting increasingly more capable in performing tasks that have traditionally required human intelligence [12, 90]. The proliferation of capable on-device hardware has enhanced their capabilities in areas such as vision [25, 82], language [84, 103] and sensor understanding [112]. Convolutional [53] and recurrent architectures [35] have undoubtedly been fueling intelligence over the past decade, with significant investment towards their performant execution, via hardware [102], algorithmic [55, 58] and EfficientML [29, 43, 61] optimizations, along with hardware and software co-design [1, 52].

Lately, however, transformers [101] have become the go-to architecture for deep learning models, with attention mechanisms offering unparalleled performance and the ability to model long-sequence data with fewer inductive biases. Applied across modalities, including vision, speech and text [25, 83, 84], these models have demonstrated significant performance benefits, across modeling and generation tasks. Another key benefit of these models has been their ability to scale to very large sizes, both in terms of data ingestion and parameter size, without their performance plateauing [99]. This has given birth to “foundation models”, large models that are trained on large corpora of data and act as universal backbones for a series of downstream tasks. For example, Large Language Models (LLMs) [17, 44, 99] have been trained on large parts of the Internet [19, 85] and are able to tackle downstream tasks without explicit training [24, 103, 105].

Despite their accuracy benefits and enabling unprecedented use-cases, such models have been pushing the computational boundaries of cloud systems, both in terms of training [20] and deployment [54]. As a result, specialized systems and hardware have been developed for the cloud to accelerate such demanding workloads. Typically deployed in large data centers, this poses questions both in terms of the sustainability of such deployments [76, 77, 108, 118], as well as the privacy and custody of user data [18]. We recognize that it is not always necessary to deploy a highly over-provisioned network to solve the task at hand [26]. Given that model performance, even for smaller models, does not saturate quickly, i.e., more data gives performance gains [78],

and the need for user privacy [109], we focus our attention to the study of deploying LLMs at the edge [57], with particular emphasis on the mobile execution of chat assistants.

To this end, we have created our own infrastructure, named MELT¹, designed to interact, trace and benchmark LLMs across ML frameworks, devices, and ecosystems. With our tool, we automate the interaction with instruction fine-tuned models and capture events and metrics of interest at a granular level, both in terms of performance as well as energy. While in this paper MELT is deployed for LLMs inference, our infrastructure is general enough to support the tracing of any workload, without user intervention. To the best of our knowledge, our tool is the first to support granular on-device energy measurements across device targets (i.e., Android, iOS, Linux) with realistic interactions.

Research Questions. Given the constant developments in mobile and embedded System-On-Chips (SoCs) and the meteoric rise of LLMs, we aspire to measure the deployability of Large Language Models at the consumer edge and identify the bottlenecks that prevent the broad deployment of such workloads. Specifically, the research questions we aim to answer with this study are the following:

- (1) Is it feasible to deploy LLMs locally on device in a private yet efficient manner?
- (2) What is the state of inference execution across a heterogeneous ecosystem of consumer edge and mobile devices in terms of performance and energy demands?
- (3) What are the current limiting factors and bottlenecks for deploying LLMs on device?
- (4) What is the impact of quantization to the performance and accuracy of the network?
- (5) Given the abundance and heterogeneity of smart devices, can such workloads be realistically offloaded to ambient devices at the consumer edge?

Concretely, our paper makes the following contributions:

- We gather the most popular open-source LLMs and benchmark them across mid and high-tier mobile and edge devices of different manufacturers, including iOS and Android-based phones as well as Nvidia Jetson edge devices. Our goal is to explore the deployability of broadly available LLMs on broadly available consumer hardware.
- To this end, we have developed the first mobile LLM evaluation suite, called MELT, responsible for downloading, quantizing, deploying and measuring the performance and energy of an LLM across heterogeneous targets.
- Through MELT, we trace specific events during inference and pinpoint their computational and energy impact. We also evaluate the continuous runtime of LLMs and their impact on battery life and user's Quality of Experience.

- We further quantify the impact of quantization on the accuracy of models, over different datasets and tasks.
- Last, we pinpoint bottlenecks in deployment and explore alternative avenues for edge deployment.

2 BACKGROUND & MOTIVATION

2.1 Transformer Preliminaries

Transformers [101] were introduced back in 2017 as an alternative architecture for NLP tasks, providing better performance and scalability than their recurrent counterparts and fewer inductive biases than convolutional networks. Since then, they have been expanded to more tasks, including vision [25] and multi-modal use-cases [82]. In this paper, we are focusing our attention on large-scale language transformers.

The original transformer comprises an *encoder-decoder* architecture, where the *encoder* digests tokens from the input sequence, whereas the *decoder* digests tokens from the output in an *autoregressive* manner. Each part of the architecture consists of multiple attention blocks. There are also encoder and decoder-only model variants, which include the respective part of the architecture. Tokens are (sub-)word representations, generated by a *tokenizer* model, embedded into a subspace (e.g., WordPiece [23] or BytePair [84] encoding).

The main contribution of transformers has undoubtedly been the attention mechanism, which captures the relationship between tokens in a sequence from a single source (self-attention) or multiple sources (multi-head attention). Attention is calculated as $A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, where Q , K , V represent the query, key, and value matrices, respectively, and d_k the dimensionality of the key matrix. The inner product boosts closer query-key vectors (relevance), softmax normalizes the dot-product and the multiplication with the value results in the relevant value scores being retrieved. The quadratic complexity of attention, with respect to the sequence length (i.e., the prompt or intermediate tokens), is one of the main bottlenecks of deployment, which has given way to alternatives such as sparse [11] or approximate [2, 106] attention mechanisms, as well as attention-free variants as of lately [38]. *Context size* refers to the maximal window of tokens a transformer block can pay attention to, whereas the *maximum generated length* refers to the maximum number of tokens generated as output. Generation ends when an <EOS> (end-of-sequence) token is generated. The auto-regressive nature of decoding means that given an input sequence $X = \{x_1, x_2, \dots, x_t\}$, the model generates x_{t+1} , which is fed to the next generation step. Key-Value cache [80] optimizes this by storing intermediary attention states.

¹Our code can be found at: github.com/brave-experiments/MELT-public.

2.2 Large Language Models

What has made Transformers an instant success has been their applicability to various modalities and their scalability to very large parameter sizes without saturating accuracy [99]. This phenomenon has given birth to Foundation Models (FMs), *pretrained* on huge corpora of data, i.e., text in our case, and act as a great tool for modeling language and a starting point for fine-tuning on downstream tasks. The task of pretraining usually comprises masked or next-word prediction (self-supervised), whereas downstream tasks can include anything from translation to summarization. *Instruction fine-tuning* [75] refers to a specific form of fine-tuning where the model is trained on pairs of input-output instructions. Last, *alignment* is usually the final step of model tuning, typically through reinforcement learning from human [75] or automated [9] feedback, to promote a certain style or content or response that “aligns” with values of the creator (e.g., safety). Training cost generally scales down as we move from pretraining downstream, as do data ingestion needs [120].

2.3 Current State and Motivating Factors

Centralization and privacy. Training a large-scale LLM is a costly effort, and many models are only offered as black-box solutions to users, such as ChatGPT and GPT-4 by OpenAI, Claude by Anthropic or Gemini by Google. These are offered as-a-service, which means that user prompts are transmitted to the provider, thereby compromising user-privacy. At the same time, users lack control over whether their data get incorporated in the training set of models without their explicit consent [18], making them amenable to various attacks [74]. Additionally, these tools remain accessible and operational only under an active internet connection.

LLMs democratization. Nevertheless, more and more models offer openly their weights, including models from Meta [98, 99], Mistral AI [47], Google [36] and Microsoft [46]. This creates an excellent opportunity for users to deploy their models locally and even personalize them to their preferences, without data ever leaving their device premises. However, such models remain significantly smaller in scale and still require considerable resources to deploy. Toward this end, new frameworks are emerging for enabling local execution of LLMs across different targets [3, 34, 42, 70, 95, 97]. In this effort, quantization [32, 61, 93] is one of the most prominent out-of-the-box solutions for reducing their footprint. Yet another enabler towards this democratization is the broad availability of capable SoCs at cost. Indicatively, from our measurements, a recent M2-based Mac Studio can run Llama-2 [99] 7B model (4-bit quantized) at a sustained 46.8 tokens/sec.

Sustainability. Last but not least, the issue of sustainability becomes ever more pronounced [76, 77, 108, 118], since the

Table 1: Device Farm of MELT

| Device Model | SoC | Mem. | Battery | OS version | Year | Tier |
|-----------------------------------|--------------------------------------|--------|----------|---------------------------|------|------|
| Co-ordinator & Builder | | | | | | |
| Raspberry Pi 4 | Broadcom BCM2711 | 8GB | - | RPi OS 11.9 | 2019 | - |
| Mac Studio | M2 Max | 32GB | - | macOS 14.1.2 | 2023 | - |
| Mobile | | | | | | |
| Galaxy S23 | Snapdragon 8 Gen 2 | 8GB | 3785 mAh | Android 14 | 2023 | High |
| Pixel 6a | Tensor Core | 8GB | 4410 mAh | Android 13 | 2023 | Mid |
| iPhone 14 Pro | A16 Bionic | 6GB | 3200 mAh | iOS 17.3.1 | 2022 | High |
| iPhone SE | A15 Bionic | 4GB | 1821 mAh | iOS 17.3.1 | 2022 | Mid |
| Edge | | | | | | |
| Jetson Orin AGX | NVIDIA Carmel Ampere GPU | + 64GB | - | Ubuntu 20.04 (L4T 35.2.1) | 2022 | High |
| Jetson Orin Nano | 8-core Arm Cortex-A78AE + Ampere GPU | 8GB | - | Ubuntu 20.04 (L4T 35.4.1) | 2022 | Mid |

training and deployment of large models requires a significant amount of energy, be it inside or outside the premises of the data center. As a result, the cost is not only monetary, but also energy consumption bound.

For all the reasons above, we feel it is more critical than ever before to quantify the cost of running LLMs on mobile and edge devices, the current bottlenecks and the sustainability of this deployment model. This way we aim to fuel future research avenues for optimizing local model deployment and further democratizing their adoption.

3 MELT INFRASTRUCTURE

In order to benchmark the runtime of LLMs on edge and mobile devices, we have engineered our own device farm, which comprises a combination of hardware and software components working in tandem to automate and measure robustly the on-device behavior of the targeted use-case. Our infrastructure adopts a client-server architecture, with the *coordinating* process running on a Raspberry Pi 4 (RPi). This is responsible for *i*) organizing the execution of the benchmarking suite, *ii*) scheduling and dispatching jobs to devices, *iii*) controlling downstream interaction with the application, *iv*) monitoring their runtime, temperature and energy consumption along with the *v*) tracing the events of interest in the downstream task.

The *co-ordinator* communicates with two sets of devices, namely *PhoneLab* (Sec. 3.1) which consists of mobile devices and *JetsonLab* (Sec. 3.2), which includes Nvidia Jetson boards, as the name suggests. We support the monitoring and interaction with Graphical User Interface (GUI) and Command Line Interface (CLI) applications, so that performance can be measured in realistic settings as well as in silo.

3.1 PhoneLab

We have incorporated four smartphones into our device farm, spanning across different resource tiers (mid and high tier) and platforms (Android and iOS), as detailed in Tab. 1. These mobile devices are interfaced with a Monsoon high-voltage power monitor (model AAA10F) [73]. To facilitate accurate power measurements, we employ a battery bypass process

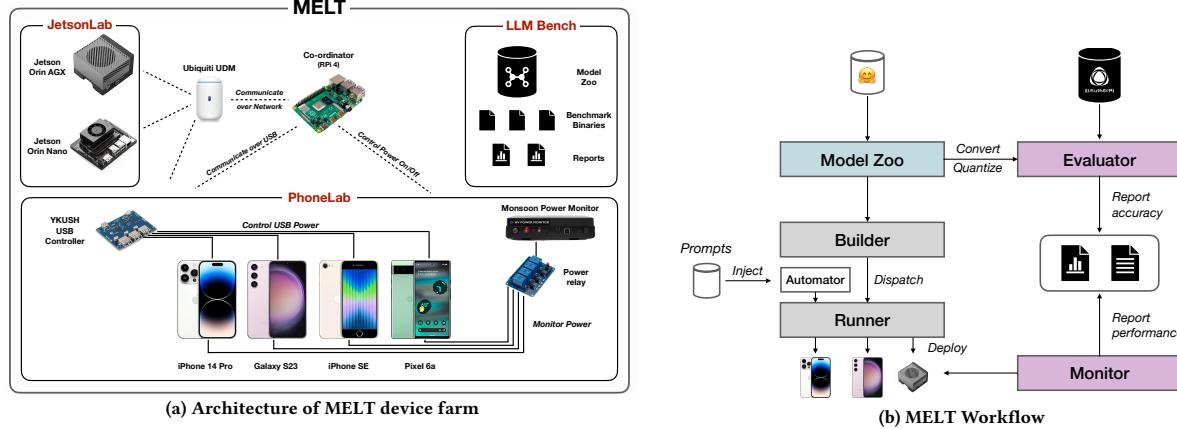


Figure 1: Architecture and workflow of MELT

that requires disassembling each device to remove its battery, extracting the internal battery controller and expose the power terminals through cables. This setup ensures precise monitoring of the devices’ power consumption directly from their power terminals [100] at a maximum frequency of 5KHz through Monsoon. In order to support the powering of multiple devices, we have a *programmable relay* that communicates over general-purpose input/output (GPIO) pins of Raspberry Pi and can selectively power on and off the devices, one at a time. The host machine initially communicates with the mobile devices via USB, connected over a *YKUSH Switchable Hub* [116]. Its purpose is to selectively disable the power lanes of the USB connection, so as not to measure USB charging draw. For monitoring the thermal behavior of the devices, we have a Flir One Edge wireless *thermal camera* positioned at 0.5-1.0m from the device whose temperature we want to measure. To minimize the influence from extraneous factors we disabled the automated OS and App updates, turned off the adaptive brightness/charging/battery features, enabled the dark mode and standardized the brightness level to 25% across devices. We call this part of the infrastructure *PhoneLab* (see Fig. 1a).

Communication to Android devices is accomplished via the Android Debug Bridge (ADB). This enables us to interact (over tap or typing events) over CLI commands with the device and application, without the need for explicit human intervention during the experiment. ADB connection is established over Wi-Fi 6 (5GHz channel) for automation, because data and power lines cannot be independently controlled over the USB channel. Interfacing with iOS is more intricate, as there is no automated toolchain for controlling the device. To achieve this, we have built a Python-based service which maps commands like touch, swipe, and text input to a virtual Human Interface Device (HID), simulating a Bluetooth keyboard and mouse that controls the device. In both cases, the baseline power draw of Bluetooth and Wi-Fi events is subtracted from the energy traces. For the compilation and deployment of apps, we have a Mac Studio

in the same network as the rest of PhoneLab, with remote access to the devices. Packages are installed through ADB and iDeviceInstaller [60] for Android and iOS, respectively.

3.2 JetsonLab

At the same time, the *co-ordinator* is connected over Ethernet to the same network as our Jetson boards with SSH access to them. We are able to take power and temperature metrics through SysFS probes available on the devices, at a frequency of approximately 100Hz². This way, not only can we calculate the power and thermal behavior of each device, but we are also able to calculate the power draw from specific components of the board (e.g., CPU, GPU, SoC, DRAM, etc.). Last, Jetson devices support a range of predefined power modes, which we control over the nvpmode. For all experiments, we used the fan speed in its maximum setting. We call this part of the infrastructure *JetsonLab* (see Fig. 1a).

Compilation of packages and models happens directly on Jetson devices over Docker images³. Automation is handled over SSH commands from RPi and results are collected immediately after execution. Both Jetsons have their Operating System (OS) installed on a high speed UHS-I SD card and have dedicated M2 SSDs for the rest of the filesystem, where models and executables reside.

4 METHODOLOGY

For the purpose of measuring LLMs performance on device, we created MELT as a benchmarking framework, which is responsible for i) the download and conversion/quantization of models, ii) the compilation of the respective benchmarking suite backend, iii) the deployment, automation and runtime of the LLM on the respective device, iv) the fine-grained monitoring of resource and energy consumption of the execution and v) the reporting of the results. The workflow of MELT is depicted in Fig. 1b.

²This granularity was explicitly tuned to capture events of interest, without interfering with the measurement itself due to I/O thrashing.

³Based on images from <https://github.com/dusty-nv/jetson-containers/>.

Algorithm 1: MELT (Experiment Process)

```

Pseudocode for MELT experiments. Functionality of undefined methods in comment. Prefixed methods run on the device in prefix (e.g., Monsoon, device).

Input: PhoneLab, JetsonLab, Monsoon, GPIO, YKUSH, device,  $Q_{\text{experiments}}$ , iterations, samplingFrequency, betweenExpSleep

1 PowerOn(device)
2 if device.platform == "ios" :
3     ConnectBT(device) # connect as HID device via Bluetooth
4     UnlockScreen(device) # unlock screen with passcode over HID
5 SyncClocks(device) # sync host and guest clocks
6 apiAddress = StartRESTServer() # start REST service on host
7 for exp in  $Q_{\text{device}}$  : # iterate over experiments in the queue
8     Push([exp.model, exp.conversations], device) # push dependencies
9     Apply(exp.conf, exp.model, device) # edit model conf and execution
   parameters on device
10    for it=0; it<iterations; ++it :
11        StartMonitoring(Monsoon, device)
12        RunExperiment(exp, device)
13        StopMonitoring(Monsoon, device) # disable monitoring
14        CollectMeasurements(exp, device) # get results from FS
15        sleep(betweenExpSleep) # sleep between runs
16 def PowerOn(GPIO, YKUSH, device):
17     if device in PhoneLab.devices :
18         GPIO.EnableRail(device.rails) # enable rail through GPIO
19         YKUSH.PowerOn(device) # enable YKUSH USB of device
20         Monsoon.SetVoutCurr(device) # configure Monsoon power out
21         Wait(device) # wait until device is responsive
22 def StartMonitoring(Monsoon, YKUSH, device):
23     if device in PhoneLab.devices :
24         YKUSH.DisableUSB()
25         Monsoon.MeasurementMode("on", samplingFrequency)
26     elif device in JetsonLab.devices :
27         Jetsonlab.ScheduleEvents(samplingFrequency)
28         Jetsonlab.Monitor("on") # poll SysFS
29 def RunExperiment(exp, device, apiAddress):
30     # open app w/ ADB, Bluetooth HID or SSH
31     app = device.OpenApp(exp.backend)
32     Automate(app, model, device) # automate interaction with app
33     http.post("start", apiAddress) # notify through REST service
34     for conversation in exp.conversations :
35         for prompt in conversation :
36             report = device.Trace(model(prompt)) # run inference
37             device.Write(report, exp.conf.outputPath) # results to FS
38             http.post("stop", apiAddress) # notify through REST service

```

4.1 Model Zoo and Evaluation

Model Zoo. As a first step, we collect the models we would like to benchmark on device from their respective sources and convert them, based on the backends available, to the respective format (e.g., GGUF - formerly known as GGML - for llama.cpp; MLC/TVM compiled files and libraries for MLC-LLM). The benchmarked models are shown on Tab. 2. Moreover, given the sheer size of the model weights, more often than not, it is necessary to quantize the models to lower precision so that their memory footprint is reduced, and the traffic between on-chip and DRAM memory is smaller. To this end, MELT’s converter is able to resolve and download models from git or huggingface and convert their weights to the respective format. This format varies both in terms of the ML framework, as well as the hardware executing the network. The supported formats and quantization methods are depicted in Tab. 3. The original models were downloaded

Table 2: Supported pretrained models

| Model Type | Size | Type | HuggingFace Repository |
|----------------|------|---------|--|
| TinyLlama [78] | 1.1B | Decoder | TinyLlama/TinyLlama-1.1B-Chat-v0.5 |
| Zephyr-3B | 3B | Decoder | stabilityai/stablelm-zephyr-3b |
| MistralAI-7B | 7B | Decoder | mistralai/Mistral-7B-Instruct-v0.1 |
| Gemma [36] | 2B | Decoder | google/gemma-2b-it |
| | 7B | Decoder | google/gemma-7b-it |
| LLama-2 [99] | 7B | Decoder | meta-llama/LLama-2-7b-chat-hf |
| | 13B | Decoder | meta-llama/LLama-2-13b-chat-hf |

directly from HuggingFace Hub and the converted models reside in MELT’s *Model Zoo*, which is a repository of converted models available to be benchmarked.

Model Evaluator. The next step is to evaluate the accuracy degradation of the model due to quantization. To accomplish this, we use MELT’s *Model Evaluator* component, which is responsible for evaluating the model⁴ on a given dataset and reporting its accuracy. We leveraged the LM-Evaluation Harness [33] and integrated a custom inference server to serve our converted models from each of the supported backends. This offers a convenient abstraction layer between the frameworks and the evaluation harness. Because of the lack of native support from the frameworks, we had to implement the extraction of token log probabilities to assess the accuracy per downstream dataset⁵. The currently supported datasets are depicted in Table 4 and the results of the evaluation are presented in Sec. 5.4.

4.2 Automated On-Device Benchmarking

Benchmark Workflow. During the execution of the respective model, we have instrumented the binaries of each framework so that we can report fine-grained timings of chat and model operations. This instrumentation includes timing of granular chat and DNN graph operations as well as calculation of performance metrics. Chat events include operations such as *prefill*, *encoding* or *decoding*, whereas graph operations refer to the LLM layers and kernel operations, which vary per framework because of optimizations happening during model conversion (e.g., operator fusion [15]). Due to the overhead of tracing very granular events (i.e., single operations), we only enable the respective flag in specific experiments (Sec. 5.3).

Builder. In order to evaluate the performance across devices, we have used two frameworks that have constituted so far the benchmarks for executing LLMs on device, namely MLC-LLM [15, 95] and llama.cpp [34] (detailed in Tab. 3). While there are increasingly more such frameworks [3, 42, 70, 72, 97], we selected the ones with the highest popularity (measured by their stars on GitHub) and widest model and

⁴We evaluate the non-finetuned variants of the models, as a typical proxy of the accuracy degradation of downstream models.

⁵Because of issues with evaluating quantized models on MLC-LLM, we evaluate AWQ [61] quantized models with autoawq package as a proxy.

platform support. We have made MELT extensible so that new frameworks can be integrated with minimal effort.

We have automated the build of the framework backends and applications for each platform (e.g., Android, iOS, Linux (CUDA)), along with the conversion binaries for the respective models. We used an M2-powered Mac Studio on the local network to build and package dependencies for mobile targets, especially since Xcode was required to sign app releases on iOS. Specifically, the Android apps were built with Android SDK v.35.0.0 and NDK v.26.1, whereas for iOS we used Xcode 15.2. Installation of packages (.apk and .ipa) was done by the co-ordinator. For the case of *JetsonLab*, the frameworks and models were compiled on device with CUDA 12.2.

Automator. In order to measure the performance of the respective model on device, we automate the interaction with the chat application. To accomplish this, we use a set of pre-canned prompts, sampled from the OAAST chat dataset [51], and interact in a multi-turn manner with the LLM. More information on the distribution of these prompts in Sec. 5.2.1.

For mobile execution, we have used custom native applications⁶ that automatically read prompts from a given file and replay the discussion with the model at hand. For edge execution and Android *llama.cpp*, we leverage the command-line interface to converse with the LLM and automate the interaction with expect scripts. These are TCL-based scripts that operate based on the text output of a binary. In the future, we would also like to evaluate guardrail chat mechanisms [87] and how the impact runtime characteristics.

For *JetsonLab*, transferring the dependencies and executing the job is accomplished over SSH commands. For *PhoneLab*, the process is more involved. For Android devices, communication and execution of jobs is mostly handled over ADB. We use the ADB as the controller for transferring files, installing and launching the application as well as automating the interaction with the app (i.e., launching a fragment or tapping on screen elements). For iOS devices, we emulate an HID Bluetooth device with the RPi that acts as a combo mouse/keyboard device. This way, we carefully script the series of actions that need to be taken so that we launch and execute a job on that device. At the end of the experiment, the co-ordinator (RPi) is automatically notified when the evaluation task is complete through a REST request. The reason behind this is for the co-ordinator to know when an experiment has finished to stop energy measurements, persist logs and continue with the next job. At the same time, we collect the generated responses and the metrics of interest. **Runner.** The runner is tasked with deploying the built application or binary, along with the associated converted models

⁶All applications have graphical user interface except for *llama.cpp* on Android, for which we used the ADB CLI interface [6].

Table 3: Frameworks and platforms supported by MELT.

| Framework | Backend | Version | Supported Platforms | Quantization |
|-----------------------|-----------------------|---------------------|--|---|
| MLC-LLM [95] | TVM [15] | 96a68e [†] | Android (GPU), iOS (Metal), Linux (CUDA) | Group Quantization [93], GPTQ [32], FasterTransformer Row-wise Quantization |
| <i>llama.cpp</i> [34] | <i>llama.cpp</i> [34] | b22022 [‡] | Android (CPU, GPU), Linux (CUDA) | k-quants [66] |
| LLMFarm [41] | <i>llama.cpp</i> [34] | 7226a8 [*] | iOS (Metal) | |

[†]We used version 784530 for supporting Gemma models and *llama-2-7B* on Android.

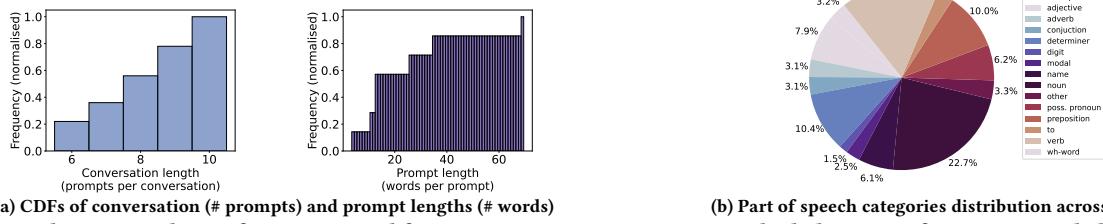
[‡]We used version d5ab29 for supporting Gemma models. * We used version 46bdb4 for supporting Gemma models.

to the respective device, running the automated interaction and gathering the reported results and logs. The experiment runtime is documented in more detail in Algorithm 1. When an experiment is run, the *co-ordinator* is responsible for powering the device if in *PhoneLab* (L.1), connecting to it (over SSH or USB), synchronizing the clocks (L.5), deploying the job dependencies (model, application, inputs) (L.8), executing the task (L.12) and gathering the outputs to return (L.14). This happens over multiple iterations, with configurable waiting times between experiments (L.15).

Monitor. Our monitoring infrastructure comprises a combination of hardware and software components. We measure *coarse* (end-to-end) and *fine-grained* (per-operator) metrics about latency and memory from the benchmark binaries. We also traced the execution through Android, Xcode and Nvidia Visual profilers for analyzing the behavior of each runtime across different platforms. These were invoked in isolation due to their overhead. These give us computational information about the LLM workload. At the same time, as aforementioned in Sec. 3, our mobile devices from *PhoneLab* are connected to a Monsoon high-voltage power monitor (AAA10F) for energy measurements, while *JetsonLab* supports power monitoring through SysFS probes. These metrics are buffered in memory and asynchronously persisted to the filesystem in a CSV timeseries file. As we have granular and synchronized timings for each operation of the LLM chat execution, we can correlate the power and thermal behavior of the device with the execution of the respective operation.

5 EVALUATION

In this section, we present results from running LLMs across devices and platforms with MELT. We start by describing our experimental setup and how we have run our experiments in Sec. 5.1. Next, we move to the on-device evaluation of various models, showcasing the computational, memory, energy and thermal behavior of these workloads. Specifically, we can distinguish our measurements in two settings: *i*) *macro-experiments*, where we measure how a chat assistant behaves on device, with real conversations (details in Sec. 5.2.1) and variable token length output, and *ii*) *micro-experiments*, where we fix the output length and disregard <EOS> tokens, so that we measure specific operations in a more controlled manner. The former setting aims to quantify the realistic behavior of chat assistants while the latter is destined for specific operation tracing. Since we heavily employ quantization to deploy LLMs on device, we also quantify its



(a) CDFs of conversation (# prompts) and prompt lengths (# words)

(b) Part of speech categories distribution across prompts

Figure 2: Qualitative analysis of prompts used for macro-experiments to assess the behaviour of LLM-powered chats on device.

impact on various language tasks in Sec. 5.4. Last, understanding that the constant release of new models and the fact that current generation of mobile hardware may not be yet optimized for this type of workloads, we explore in Sec. 5.5 the performance of running LLMs when deployed on local edge devices, i.e., Nvidia Jetson devices, and comment on the potential of edge offloading.

5.1 Experimental Setup

For our experiments, we leverage the infrastructure and methodology described in Sec. 3 and 4, respectively. For each device (Tab. 1), we tweak the model size, quantization bitwidth, context size, maximum generated length and token batch size through a grid search⁷. GPU acceleration has been used for MLC-LLM and llama.cpp when it yielded performance benefits. This was not the case for llama.cpp on Android, where the gains from running on GPU were minimal⁸. As such, for comparative performance, we have shown CPU runtimes for llama.cpp on PhoneLab. We based our infrastructure on the versions of frameworks shown on Tab. 3, but with further instrumentation and automations on our side to support the scalable evaluation of performance across platforms and devices. We used the models of Tab. 2, and converted/quantized them with the native tools of each backend. This was necessary as we needed to alter the generated libraries for instrumentation. Unless stated otherwise, all experiments were repeated three times and we report mean and standard deviation of the runs.

5.2 Macro Experiments

5.2.1 Dataset Qualitative Analysis. For macro-experiments, we used a subset of prompts from the OpenAssistant/oasst1 dataset [51]. We filtered out inputs, so that the resulting dataset has prompts in English, with at least 5 turns of interaction. We used a sample of 2k data points and ended up with a dataset of 50 conversations. We present some qualitative results on Fig. 2, where we depict the distributions of conversation lengths, prompt lengths and also part-of-speech

⁷(context size={512, 1024, 2048} \odot max gen. length={64, 128, 256}) \times batch size={128, 512, 1024}, where \odot is the Hadamard and \times the Cartesian product.

⁸Indicatively, running TinyLlama-1.1B (4-bit) on S23 resulted in 13.61 ± 0.54 vs. 13.22 ± 0.46 tok/sec on CPU and GPU, respectively. Others have also documented this: <https://github.com/ggerganov/llama.cpp/issues/5965>.

categories across prompts. We can see from Fig. 2a that the conversation length spans linearly from 6 to 10 prompts with the 80-th percentile of prompts below 36 words. Most words represent verbs, determiners and nouns, as analyzed with the nltk python package. We combined the long tail of tags of less than 1% to the category “other”. Of course, the correspondence of words to tokens depends on the tokenizer used by the respective model.

5.2.2 On-device Runtime. We start by quantifying the token throughput and efficiency per device and framework.

Computational throughput. First, we show the prefill and generation throughput of various models when used in a conversational setting. We divide our results per device tier and illustrate the average throughput (in tokens/sec) per framework in Fig. 3. Generally, we witness much higher prefill vs. generation throughput, which can be largely attributed to the usage of KV-cache [80] when encoding a sequence of tokens and the compute vs. memory boundedness of the workload [70]. Moreover, MLC-LLM generally offered higher performance to llama.cpp, but at the cost of model portability (models need to be compiled per platform). Operator fusion and TVM-based optimization play a significant role towards this result, with generation throughput difference of +4% on average for GPU execution (+28% vs llama.cpp CPU) and up to $3.53\times$ higher. Notable exceptions included TinyLlama across targets and Gemma on S23. We also noticed that 4-bit quantized models performed better than their 3-bit variants, offering 24.77% higher throughput on average. We attribute this to the effects of dequantization and better cache alignment during execution. However, there is a trade-off with memory consumption, which made certain models to run out-of-memory during runtime, especially on phones with smaller RAM sizes. Last, the Metal-accelerated iPhones seem to be offering higher throughput compared to the OpenCL-accelerated Android phones for the case of MLC, with 78.93% higher generation throughput on average. Even in the case of CPU runtime on llama.cpp, iPhones generally performed faster, but less efficiently. We can attribute this to the higher thread count that llama.cpp allowed on iOS without crashing the application. Last, our hypothesis for the relatively high variance of the results is the variable context and generation length as well as potential Dynamic Voltage and Frequency

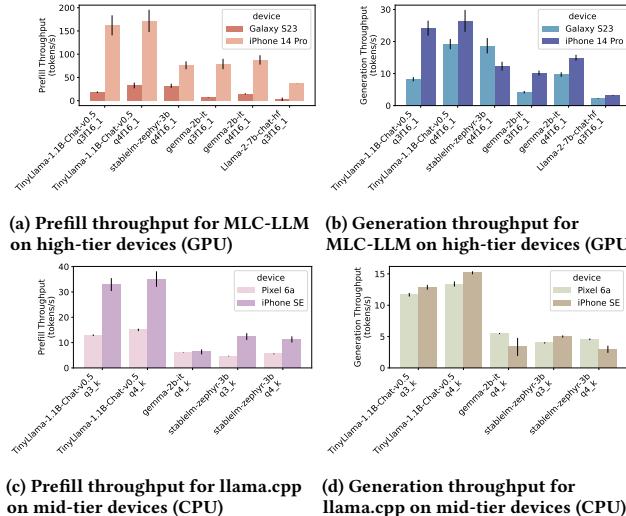


Figure 3: Throughput across frameworks and devices

Scaling (DVFS) (more in commentary of Fig. 5). We provide performance and discharge rates for GPU execution for llama.cpp on iOS devices in Appendix A.1.

Energy efficiency. Next, we take the same set of models and illustrate the energy discharge (in mAh) per token generated across devices and frameworks in Fig. 4. Overall, we noticed that the trend of larger networks (in terms of parameter size) offering larger discharge rates across devices and frameworks. This is expected as DRAM utilization and memory copies into the SoC registers consume significant energy [76]. Notable exceptions to this rule were TinyLlama (3-bit) and Gemma (4-bit), which we aim to investigate with help from upstream maintainers. Last, the CPU execution of llama.cpp offered overall lower energy efficiency, but this could also be attributed to the latency of running inference compared to when using GPU acceleration.

Power timeline. Next, we zoom into the runtime of our experiments and show the execution timeline of Zephyr-3B (4-bit quantized) running six prompts across devices (iPhone 14 Pro and Galaxy S23) and frameworks (MLC-LLM and LLMFarm). During execution, we have traced specific events of interest, that we annotate on Fig. 5, which depicts the power draw (in Watts) of the device during inference. First off, we noticed from the beginning that iPhones tend to boost their power draw very high, reaching a maximum of 13.8W of sustained (averaged) power draw and an instantaneous maximum of over 18W. The equivalent wattage from the Galaxy device only reached an instantaneous maximum of 14W with sustained power draw below 8.5W. At the given power draw, the overall power consumption during inference was 11.54, 10.43, 2.42 mWh (normalized per token: 0.21, 0.20, 0.16 mWh/token) for S23 and iPhone 14 Pro on MLCChat and LLMFarm, respectively. At that pace, each device could run 542.78, 490.05 and 590.93 prompts until its battery is depleted,

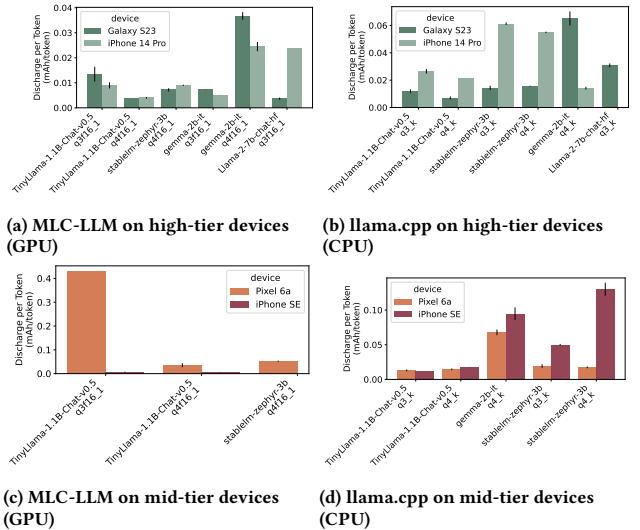


Figure 4: Discharge per token across frameworks and devices. Missing bars indicate unsuccessful runs due to OOM errors or too long runtime (>1hr per conversation).

at an average input of 40 tokens and generation length of 135 tokens. Of course, we do not account for simultaneous load and different energy modes applied by the OS here.

Model loading on Zephyr-3B (4-bit) took an average of 2.41 ± 0.09 sec, during which time the device often becomes unresponsive. We annotate with blue color five intermediary conversations and divide the sixth into prefill and generate events. In between runs, we have a sleep of 5 seconds. As also previously discussed, prefill takes only a small part of the inference which is mostly bottlenecked by the memory-bound generation process. The length of each inference is not only a function of the speed of the device, but also the number of generated tokens and context size. Therefore, we see the time length of each inference varying per device and across devices. Last, we see that the power drops gradually after an inference has completed, which is signified by the last gray spike per inference. We remind to the reader at this point that we synchronize the clocks of the co-ordinator and client device to avoid time drift issues.

5.2.3 Quality of Experience (QoE). In real-world settings, tractability does not imply deployability. What this means is that while a model can run on a device, it can adversely affect the user experience and render the device unstable or unusable. There are largely three dimensions to consider: *i*) device responsiveness, *ii*) sustained performance and *iii*) device temperature. We discuss each of them below:

Device responsiveness refers to the general stability and reliability of the device during the runtime of LLM inference. Upon deployment, factors that affected the device responsiveness included long *model loading times* (see purple areas in Fig. 5 and Fig. 6) during which the device became largely

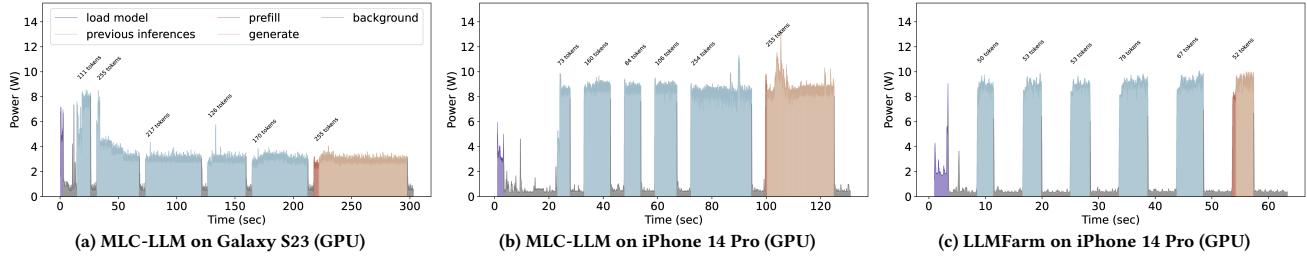


Figure 5: LLM execution timeline of Zephyr-3B (4-bit quantized) across devices and frameworks. We use a moving average of 500 points for smoothing the timeline. We annotate the number of generated tokens per inference.

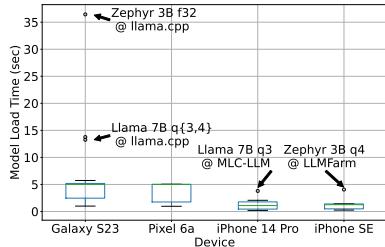


Figure 6: Model loading time per device. Each supports different set of models, based on available memory and framework. unresponsive⁹; *out-of-memory errors* (OOM), which killed the application at arbitrary times; and *device restarts*, which for undefined reasons caused Denial of Service (DoS) by rebooting the device. All these negatively affect the user experience and their frequency of appearance should be minimized. We encountered multiple such events during our benchmarks, which create the need for heterogeneous in-the-wild deployments and parameter selection (e.g., model size, quantization precision, prefetching, KV cache size, batch size, context size) based on the available device resources and use-case at hand. **Sustained performance** refers to the device’s ability to offer the same performance throughout the runtime of multiple inference requests. There are multiple reasons why this may not be stable, including DVFS, thermal throttling, different power profiles, low battery level and simultaneous workloads, among others. At this stage, we assume that our LLM is the main workload running on the device, although it has been reported that multiple DNNs reside on smartphones nowadays [57]. To quantify how, we took Zephyr-3B (4-bit) on iPhone 14 Pro and ran continuous inference over 50 prompts to check where throughput starts degrading. We repeated the experiment three times and measure the variation across runs. Results are depicted on Fig. 7a. We experience straightaway performance dropping with two bumps happening on the 20th and 32nd prompts (on average, annotated in red). Our hypothesis is that the device enters different energy and DVFS modes at these stages, with higher variation signifying that the point at which this happens is not fixed in time. The performance on Jetson AGX (50W) was much smoother (Fig. 7b), as signified by the straight line

⁹Inference impacts overall usability as GPU is also used for GUI rendering.

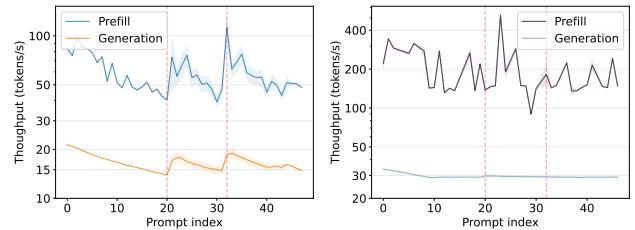


Figure 7: Continuous inference on mobile and edge devices with Zephyr-3B (4-bit).

in the generation throughput. The initial higher generation throughput can be attributed to the context not being filled. **Temperature** is yet another parameter that we briefly touched upon in the previous paragraph. Temperature does not only affect device performance, but also user comfort [107]. Devices nowadays come in various forms, but mostly remain passively cooled. Therefore, heat dissipation is mainly facilitated by the use of specific materials and heat management is governed by the OS. The power draw that was witnessed in Fig. 5b did cause temperatures to rise to uncomfortable levels, reaching 47.9°C as depicted in Fig. 9a.

5.3 Micro Experiments & Bottlenecks

In this section, we investigate deeper into on-device LLM inference, and its system bottlenecks. First, we fix the prefill and maximum generation tokens to a fixed number of 256 and remove the <EOS> token for stopping the sequence, leading to a more deterministic execution. In Sec. 5.3.1, we trace the low-level operations during different stages of inference. Next, in Sec. 5.3.2, we use profiling tools to inspect the compute and memory behavior during inference.

5.3.1 ML Operations. We start by introspecting Llama-7B (3-bit) on Android. We compile a custom version of TVM and MLC-LLM where we enable the `vm_profiler` in the backend and report kernel runtimes per operator of interest. In this section, we only measure per kernel latency, as the end-to-end latency is heavily impacted by the use of the profiler. Results are shown in Fig. 8 for the prefill, embed and decode operations. We see that most of the execution is taken up by de-quantize and matrix multiplication fused operations for the prefill and decode operations, taking

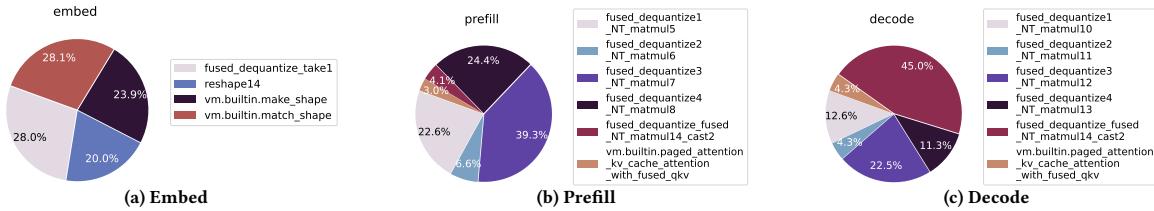


Figure 8: Per-op benchmarks of Llama-7B (3-bit) with MLC-LLM on Samsung Galaxy S23. These are operations generated by the TVM compiler. The variants may signify different implementation or hyperparameters tuned for performance.

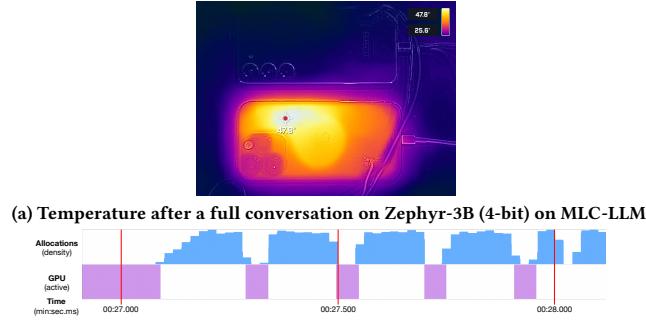


Figure 9: Thermal and memory behavior on iPhone 14 Pro

up 97% and 95.7% of the total runtime, respectively. We hypothesize that the dequantization operation is also why 3-bit quantized networks may have performed worse than their 4-bit counterparts, as we discussed in Sec. 5.2.2. On the contrary, the embed operation seems mostly to be doing tensor conversion and retrieval operations. Since the generation process is mostly bottlenecked by the decode operation (evident also in Fig. 3 and 7), we proceed to investigate the real system bottleneck during execution via profiling. Due to lack of GPU tracing via the Android GPU Inspector on Galaxy S23, we apply the analysis on the iPhone 14 Pro.

5.3.2 Memory Usage and Bottlenecks. It is known that LLM execution is bottlenecked by the memory bandwidth requirements during generation [20, 21, 54]. Effectively, inference waits for the model state and activations to be expensively transferred from main to the on-chip memory, with little reuse due to the small batch sizes and autoregressive causal generation nature of the workload. Our analysis corroborates this on the mobile side, by what is shown in the memory profiling of Fig. 9b, where we depict the memory allocations and GPU computation happening effectively one after the other. While GPU memory gets allocated, GPU compute effectively stalls, waiting for data to process. This was measured through xctrace and visualized with Apple Instruments application.

5.4 Impact of Quantization

A prominent method for reducing the memory traffic between main and on-chip memory is to decrease the precision of the weights and activations of the Neural Network [32, 61, 110]. However, this often comes at the expense of model accuracy, especially at sub 4-bit weight precision.

Table 4: Evaluation datasets description

| Dataset | Task | Size | Description |
|-----------------|------------------|------------|---|
| HellaSwag [119] | Common-sense NLI | 70k | Given an event description, select the most likely continuation. |
| Winogrande [89] | Common-sense NLI | 44k | Benchmark for common-sense reasoning, designed not to be easily solvable by statistical models and plain word associations. |
| ThuthfulQA [62] | Knowledge NLG | 817 | Benchmark for measuring truthfulness in a model’s generated answers. |
| ARC-{E,C} [16] | Reasoning NLI | 5.2k, 2.6k | Science and language exam questions from a variety of sources. E: Easy; C: Complex |

Moreover, the hardware needs to support operations at these precisions, to avoid dequantization before computation.

By leveraging the supported quantization schemes in the two LLM frameworks MELT supports (Tab. 3), we measure the impact of quantization in various tasks on the pretrained models. We use pretrained instead of fine-tuned models for this because the latter’s fine-tuning and RLHF [75] alignment can affect the original performance. A description of the employed quantization schemes is presented in Sec. 7. We use the benchmark datasets depicted in Tab. 4, which consist of Natural Language Inference (NLI) and Natural Language Generation (NLG) tasks. In the former case, it comprises multiple choice questions, and the most likely answer – expressed by cumulative log likelihood of the model’s output – is selected and matched against the correct label. In the latter case, the model’s output is evaluated against template answers over BLEURT [92] score.

Results are depicted in Fig. 10 across datasets and models. From the data we can see that the most evident performance difference comes from the *model architecture* and *parameter size*, and this performance difference persists across datasets. In terms of quantization schemes, it is obvious that bitwidth is correlated to model size, but also to accuracy, i.e., lower bitwidth means higher error rate. This was very evident in our qualitative evaluations, where some smaller models (≤ 3 B parameters) were unusable with 3-bit precision, mostly hallucinating or plainly repeating the prompt. On the other hand, there was no single quantization scheme that performed uniformly better across the board. For larger models (≥ 7 B parameters), AWQ [61] and GPTQ [32] performed slightly better, at the expense of elevated model sizes.

5.5 Runtime at the Edge

Offloading. Hitherto, we have witnessed that high-end mobile devices with more than 6GB of memory can run a chat

MELTING point: Mobile Evaluation of Language Transformers

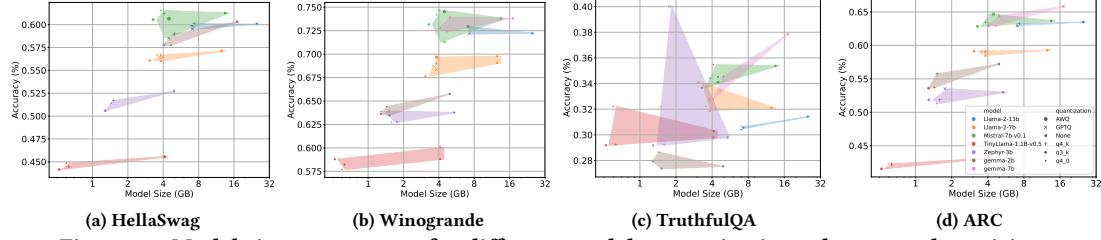


Figure 10: Model size vs. accuracy for different models, quantization schemes and precisions.

LLM at a reasonable rate. However, this comes at the cost of significant battery depletion (see Sec. 5.2.2), QoE (see Sec. 5.2.3) and end-task accuracy (see Sec. 5.4). Therefore, we envision that the future of LLM execution can be collaborative and cross-device at the edge [57, 81]. To this direction, we test to see the viability of offloading the DNN execution to a local edge device, which might be a dedicated accelerator (e.g., an Edge-AI Hub) or another edge device (e.g., a Smart TV or a high-end router). For this reason, we employ two Jetson devices, namely Nano (mid-tier) and AGX (high-tier) to check the viability of this paradigm. We assume prompt offloading happens over a Wi-Fi 6 network (9.6 Gbps) with negligible latency overhead (i.e., streamed text).

Emulating different devices Jetson devices support different energy modes, which configure the number of active cores and their frequency, along with memory frequency to provide different power envelopes. Specifically, Orin AGX supports TDPs of 50W, 30W, and 15W whereas Orin Nano supports 15W and 7W. Taking advantage of this functionality, we wanted to test how such devices can support LLM execution under different power budgets and their respective performance. This way, we first showcase the offloading viability over different ambient devices, but also give a proxy metric about potential future mobile and edge devices.

Results for both scenarios are presented in Fig. 11. Specifically, in Fig. 11a we show the generation throughput (in tokens/sec) of various models on different Jetson devices and energy profiles, as run with llama.cpp on CUDA. We see that throughputs largely follow a monotonic trajectory with respect to model size and energy modes, with the notable exception of Orin Nano and Orin AGX at 15W, with the former performing +7.89 tokens/sec better on average. Overall, generation throughput is significantly higher than the equivalent mobile runtime, and this runtime can also be sustained for longer periods, as shown in Fig. 7. Indicatively, for Zephyr-7B (4-bit), the average throughput is 3.3× and 1.78× higher, for prefill and generation respectively. CPU runtimes are also provided in Appendix A.2.

In Fig. 11b, we quantify the energy efficiency of two models (Llama-7B (3-bit) and Gemma-2B (4-bit)) running across different energy modes. Interestingly and perhaps counter-intuitively, we see that the efficiency is moving the same direction as the device’s TDP. We believe that frequency scaling of the memory subsystem from the lower power

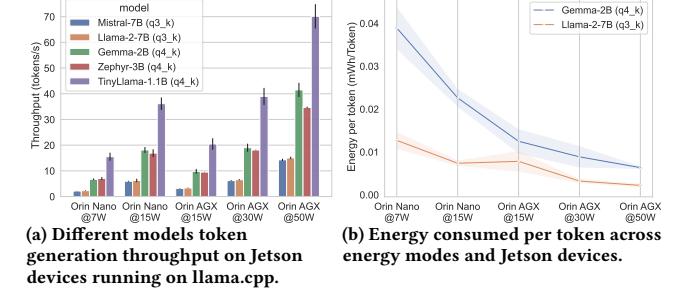


Figure 11: LLM execution on Jetsons across energy modes mode adversely affects the workload, making generation even more bottlenecked by the lower memory clock. Effectively, the GPU stalls for longer, waiting for memory I/O. When the power mode allows for higher memory frequency, there are additional efficiency gains and higher utilization.

6 DISCUSSION & LIMITATIONS

Summary of results. So far, we visited the performance and energy consumption characteristics of running LLMs on mobile and edge devices. We measured the throughput and energy efficiency of various models and showed that smaller quantized models can run sufficiently well on device at the cost of increased power consumption. Moreover, we studied the device behavior during model loading and sustained inference, along with the power variability during a conversation, witnessing high peaks and apparent consequences in user QoE. Last, we dove into the specific operator runtime and memory bottlenecks during execution and showed the memory-bound nature of generation. Recognizing that quantization is one of the main ways to drop the memory requirements, we measured the accuracy impact on various tasks, which was non-negligible in sub 4-bit precisions. Drawing from these results, we discuss their impact in LLM deployment and how they can shape future research avenues.

Hardware/Software advances While the area of generative AI has seen great acceleration the past years, so have the associated workloads. As an area of active research and industrial interest, new algorithmic methods [14, 20, 38] and hardware [27, 67] can provide non-linear scaling in how the current workloads run. Therefore, not only can current models be deployed more efficiently, but also larger models can be trained and deployed, leading to smarter models [12, 90]. **Multimodality & emergent abilities.** In terms of capabilities, the ability of models to deal with multi-modal inputs

and outputs become of great value [63, 71, 82], effectively giving assistants an extra sense. However, their overhead for deployment is non-negligible, especially on embedded hardware like smart glasses or robotics. Therefore, on-device deployment of such models emerges as an area of interest.

New use-cases. This paper is the first step towards enabling use-cases at the edge, offering metrics that can fuel algorithmic and edge hardware research, with efficiency, privacy and sustainability in mind. We envision a future where multi-modal and context-aware personalized assistants will be locally conversing with users and have long-term memory with recollection of past interactions [24]. At the same time, users will be able to interact with interfaces in natural language to accomplish tasks [59], without the need to imperatively define the individual steps [91, 105]. Last, we envision this automation expanding to interactions between humans, where individuals would be able to proxy their availability over smart assistants [10].

Organization of edge hardware resources. Last, in terms of system architecture, we foresee two major avenues of deploying intelligence at the edge. One requires SoC manufacturers to design accelerators explicitly for running LLMs in an energy efficient manner, in a way that does not hurt QoE of concurrent apps or deplete the battery in an unreasonable manner. To this direction, NPUs capable of running matrix-to-matrix multiplications efficiently with larger on-chip cache and memory throughput seems crucial. The future can also be hybrid [81] and hierarchical, with part of the workload being accelerated at the edge or cloud [56, 57, 111].

Limitations. Our study is simply the first attempt towards analyzing the on-device behavior of LLM workloads and hope can make them more accessible to the public. However, our analysis has been limited to chat fine-tuned models of 1-13B parameter size due to their broad availability and popularity. Very lately, sub-billion models have emerged [65, 96], which present their own computational interest in edge settings. Moreover, we analyzed the inference energy at a device-centric level. It is well known, though, that the consumer edge is not as green as state-of-the-art datacenters [108]. The global impact of distributing LLM computation has not been considered. Last, we only studied quantization as a way of reducing model footprint. There are various alternatives, briefly introduced Sec. 7, for further optimizing these workloads. We leave such topics as future work.

7 RELATED WORK

Benchmarking models on device. In terms of on-device DNN benchmarking, there has been a rich set of literature in the past for edge and mobile deployment. Indicatively, Ignatov et al. [45] had been one of the first in-the-wild benchmark suites for on-device benchmarking and device ranking across a multitude of downstream tasks and modalities.

Embench [5] quantified the different dynamics of model execution across various mobile, edge and desktop devices. MLPerf [88] is an industry-wide standardized ML benchmark tool. Another tangential line of work has focused on quantifying the performance of already deployed models in mobile apps, with works [5, 113] showcasing a surging trend in the deployment of on-device ML. Nevertheless, the advent of LLMs have pushed the compute requirements for executing such workloads, and thus current most deployments offload inference to the cloud [69], while on-device deployment remains limited. This phenomenon is hindered by the currently available tools and asks for better on-device measurements so that edge execution of LLMs is facilitated. To the best of our knowledge, this is the first study of LLMs on-device performance. Prior work has either focused on training efficiency [86, 118] or served inference [7, 54] in the datacenter.

Edge execution of LLMs. There have been various lines of work attempting to port LLM computation on-device. Starting with frameworks, llama.cpp [34] and MLC [95] have stood out, offering cross-platform accelerated execution and support for various LLM architectures and device targets. Other open-source frameworks include llama2.c [48], aimed at simplicity without dependencies and tinygrad [97], focused on accelerated execution, but without support quantized mobile execution. Last, TinyChatEngine [72] showcased on-device inference with compressed models, but lacks mobile support. Lately, OS providers have released their own platforms, such as Apple’s MLX [42] and Google’s AICore [8]. The former only provided support for desktop platforms (M-series SoCs) at the time of writing and the latter remains closed-source and only deployed on Pixel 8 Pro. Very recently, Google released MediaPipe [70] for on-device LLM execution.

Efficient LLMs. As we have shown, these workloads have been largely bottlenecked by the memory size and throughput of the underlying hardware. Therefore, a lot of research has focused on compressing these models to economize on their memory and bandwidth requirements. Various works have proposed quantization [22, 32, 50, 61, 64, 110] and sparsification/pruning schemes [31, 68], low-rank methods [114] and distillation-based solutions [39] aimed specifically at LLMs. Orthogonally, one can leverage secondary storage for running LLMs with limited local resources [4, 94]. The quadratic cost of attention has also been a large scalability issue. Therefore, various techniques try to address this cost, through different attention patterns [11, 20, 21, 106], token skipping [37, 40, 49] or alternative architectures [38, 79].

Employing multiple models for dropping the overall cost of inference has also been a popular approach, with techniques such as *Mixture-of-Experts* [28, 30, 117] focusing on using subsets of weights based on the input at hand. However,

these remain difficult to deploy on device, due to their memory and storage requirements. *Speculative decoding* [13, 14] has been recently introduced as a way of accelerating inference, based on the fact that not every token needs to be generated by a large LLM, but a significantly smaller draft model can be leveraged for quick token generation while the original model operates in a batched fashion. [111] proposes a distributed such setup for the edge. For a more complete overview of related work, we divert the reader to [104, 115].

8 CONCLUSION

In this work, we have made the first step towards quantifying the performance of deploying LLMs at the consumer edge. We measured the performance, memory, and energy requirements of such workloads across different model sizes and a heterogeneous ecosystem of devices, pinpointing computational, QoE and accuracy bottlenecks. We hope this study will serve as a basis for subsequent algorithmic and hardware breakthroughs that will help the realization of new use-cases and the democratization of LLMs execution in an open but privacy-preserving manner.

REFERENCES

- [1] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. 2020. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
- [2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 4895–4901.
- [3] Alibaba. 2023. MNN-LLM. <https://github.com/alibaba/MNN>
- [4] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifar, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *arXiv:2312.11514 [cs.CL]*
- [5] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I Veneris, and Nicholas D Lane. 2019. EmBench: Quantifying performance variations of deep neural networks across modern commodity devices. In *The 3rd international workshop on deep learning for mobile systems and applications*. 1–6.
- [6] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis, and Nicholas D Lane. 2021. Smart at what cost? characterising mobile deep neural networks in the wild. In *Proceedings of the 21st ACM Internet Measurement Conference*. 658–672.
- [7] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.
- [8] android.com. 2023. AICore. <https://developer.android.com/ml/aicore> Accessed: Dec 2023.
- [9] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073* (2022).
- [10] Barbara Krasnoff,. 2021. How to use Android 12’s call screening features. <https://www.theverge.com/22792060/call-screening-android-12-google-pixel-how-to> Accessed: Mar 2024.
- [11] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).
- [12] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712* (2023).
- [13] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. 2023. Medusa: Simple Framework for Accelerating LLM Generation with Multiple Decoding Heads. <https://github.com/FasterDecoding/Medusa>.
- [14] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318* (2023).
- [15] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799* 11, 20 (2018).
- [16] François Fleuret. 2019. On the measure of intelligence. *arXiv preprint arXiv:1911.01547* (2019).
- [17] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).
- [18] Matteo Ciniselli, Luca Pascarella, and Gabriele Bavota. 2022. To what extent do deep learning-based code recommenders generate predictions by cloning code from the training set?. In *Proceedings of the 19th International Conference on Mining Software Repositories (Pittsburgh, Pennsylvania) (MSR ’22)*. Association for Computing Machinery, New York, NY, USA, 167–178. <https://doi.org/10.1145/3524842.3528440>
- [19] commoncrawl.org. 2024. CommonCrawl Dataset. <https://commoncrawl.org/> Accessed: 2024-02-06.
- [20] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).
- [21] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.
- [22] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *arXiv preprint arXiv:2306.03078* (2023).
- [23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. <https://doi.org/10.18653/v1/N19-1423>
- [24] Xin Luna Dong, Seungwhan Moon, Yifan Ethan Xu, Kshitiz Malik, and Zhou Yu. 2023. Towards Next-Generation Intelligent Assistants Leveraging LLM Techniques. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (Long Beach,*

- CA, USA) (KDD '23). Association for Computing Machinery, New York, NY, USA, 5792–5793. <https://doi.org/10.1145/3580305.3599572>
- [25] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.
- [26] Ronen Eldan and Yuanzhi Li. 2023. TinyStories: How Small Can Language Models Be and Still Speak Coherent English? *arXiv preprint arXiv:2305.07759* (2023).
- [27] Hongxiang Fan, Thomas Chau, Stylianos I Venieris, Royson Lee, Alexandros Kouris, Wayne Luk, Nicholas D Lane, and Mohamed S Abdelfattah. 2022. Adaptable butterfly accelerator for attention-based NNs via hardware and algorithm co-design. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 599–615.
- [28] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research* 23, 1 (2022), 5232–5270.
- [29] Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*.
- [30] Elias Frantar and Dan Alistarh. 2023. QMoE: Practical Sub-1-Bit Compression of Trillion-Parameter Models. *arXiv preprint arXiv:2310.16795* (2023).
- [31] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *arXiv preprint arXiv:2301.00774* (2023).
- [32] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).
- [33] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation. <https://doi.org/10.5281/zenodo.5371628>
- [34] Georgi Gerganov. 2023. *llama.cpp*. <https://github.com/ggerganov/llama.cpp>
- [35] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12, 10 (2000), 2451–2471.
- [36] Google Inc. 2024. Gemma: Introducing new state-of-the-art open models. <https://blog.google/technology/developers/gemma-open-models/> Accessed: Mar 2024.
- [37] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination. In *International Conference on Machine Learning*. PMLR, 3690–3699.
- [38] Albert Gu and Tri Dao. 2023. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv preprint arXiv:2312.00752* (2023).
- [39] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge Distillation of Large Language Models. *arXiv preprint arXiv:2306.08543* (2023).
- [40] Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022. Transkimmer: Transformer Learns to Layer-wise Skim. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 7275–7286. <https://doi.org/10.18653/v1/2022.acl-long.502>
- [41] guinmoon. 2023. *LLMFarm*. <https://github.com/guinmoon/LLMFarm>
- [42] Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. 2023. *MLX: Efficient and flexible machine learning on Apple silicon*. <https://github.com/ml-explore>
- [43] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
- [44] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).
- [45] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. AI Benchmark: Running Deep Neural Networks on Android Smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.
- [46] Javaheripi, Mojtaba and Bubeck, Sébastien. 2024. Phi-2: The surprising power of small language models. <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/> Accessed: Mar 2024.
- [47] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
- [48] Andrej Karpathy. 2023. *llama2.c*. <https://github.com/karpathy/llama2.c> Accessed: Dec 2023.
- [49] Gyusan Kim and Kyunghyun Cho. 2021. Length-Adaptive Transformer: Train Once with Length Drop, Use Anytime with Search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 6501–6511. <https://doi.org/10.18653/v1/2021.acl-long.508>
- [50] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. SqueezeLLM: Dense-and-Sparse Quantization. *arXiv preprint arXiv:2306.07629* (2023).
- [51] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnosidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. 2023. OpenAssistant Conversations—Democratizing Large Language Model Alignment. *arXiv preprint arXiv:2304.07327* (2023).
- [52] Alexandros Kouris, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas D Lane. 2022. Fluid Batching: Exit-Aware Preemptive Serving of Early-Exit Neural Networks on Edge NPUs. *arXiv preprint arXiv:2209.13443* (2022).
- [53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).
- [54] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- [55] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D Lane. 2021. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*. 1–6.
- [56] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leonidas, and Nicholas D. Lane. 2020. SPINN: Synergistic Progressive

- Inference of Neural Networks over Device and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) (*MobiCom '20*). Association for Computing Machinery, New York, NY, USA, Article 37, 15 pages. <https://doi.org/10.1145/3372224.3419194>
- [57] Stefanos Laskaridis, Stylianos I Venieris, Alexandros Kouris, Rui Li, and Nicholas D Lane. 2022. The Future of Consumer Edge-AI Computing. *arXiv preprint arXiv:2210.10514* (2022).
- [58] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.
- [59] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetraault (Eds.). Association for Computational Linguistics, Online, 8198–8210. <https://doi.org/10.18653/v1/2020.acl-main.729>
- [60] libimobiledevice. 2024. ideviceinstaller. <https://github.com/libimobiledevice/ideviceinstaller> Accessed: Mar 2024.
- [61] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv preprint arXiv:2306.00978* (2023).
- [62] Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958* (2021).
- [63] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485* (2023).
- [64] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghu Ram Krishnamoorthi, and Vikas Chandra. 2023. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. *arXiv preprint arXiv:2305.17888* (2023).
- [65] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghu Ram Krishnamoorthi, et al. 2024. MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. *arXiv preprint arXiv:2402.14905* (2024).
- [66] llama.cpp Team. 2023. k-quants. <https://github.com/ggerganov/llama.cpp/pull/1684> Accessed: March 2024.
- [67] Zizhang Luo, Liqiang Lu, Yicheng Jin, Liancheng Jia, and Yun Liang. 2023. Calabash: Accelerating Attention Using a Systolic Array Chain on FPGAs. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 242–247.
- [68] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Advances in Neural Information Processing Systems*.
- [69] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials* 19, 4 (2017), 2322–2358.
- [70] Mark Sherwood. 2024. Large Language Models On-Device with MediaPipe and TensorFlow Lite. <https://developers.googleblog.com/2024/03/running-large-language-models-on-device-with-mediapipe-andtensorflow-lite.html> Accessed: March 2024.
- [71] Brandon McKinzie, Zhe Gan, Jean-Philippe Fauconnier, Sam Dodge, Bowen Zhang, Philipp Dufter, Dhruti Shah, Xianzhi Du, Futang Peng, Floris Weers, et al. 2024. MM1: Methods, Analysis & Insights from Multimodal LLM Pre-training. *arXiv preprint arXiv:2403.09611* (2024).
- [72] mit-han lab. 2023. TinyChatEngine. <https://github.com/mit-han-lab/TinyChatEngine> Accessed: Dec 2023.
- [73] Monsoon Solutions Inc. 2023. Monsoon Solutions Inc. <https://www.msoon.com> Accessed: Dec 2023.
- [74] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. 2023. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035* (2023).
- [75] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
- [76] David Patterson, Joseph Gonzalez, Urs Hözle, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. 2022. The carbon footprint of machine learning training will plateau, then shrink. *Computer* 55, 7 (2022), 18–28.
- [77] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).
- [78] Tianduo Wang Wei Lu Peiyuan Zhang, Guangtao Zeng. 2023. TinyLlama. <https://github.com/jzhang38/TinyLlama>
- [79] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albak, Samuel Arcadiño, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranti Kiran GV, et al. 2023. RWKV: Reinventing RNNs for the Transformer Era. *arXiv preprint arXiv:2305.13048* (2023).
- [80] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023).
- [81] Qualcomm. 2023. *The future of AI is hybrid*. White Paper. Qualcomm.
- [82] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
- [83] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavy, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*. PMLR, 28492–28518.
- [84] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multi-task learners. *OpenAI blog* 1, 8 (2019), 9.
- [85] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.
- [86] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) (*SC '20*). IEEE Press, Article 20, 16 pages.
- [87] Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Yansong Feng and Els Lefever (Eds.). Association for Computational Linguistics, Singapore, 431–445. <https://doi.org/10.18653/v1/2023.emnlp-demo.40>

- [88] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.
- [89] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Commun. ACM* 64, 9 (aug 2021), 99–106. <https://doi.org/10.1145/3474381>
- [90] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2024. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems* 36 (2024).
- [91] Timo Schick, Jane Dwivedi-Yu, Roberto Dessa, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).
- [92] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. 2020. BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of ACL*.
- [93] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8815–8821.
- [94] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) (ICML’23). JMLR.org, Article 1288, 23 pages.
- [95] MLC team. 2023. *MLC-LLM*. <https://github.com/mlc-ai/mlc-llm>
- [96] Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer, Michael Felsberg, Timothy Baldwin, Eric P. Xing, and Fahad Shahbaz Khan. 2024. MobiLlama: Towards Accurate and Lightweight Fully Transparent GPT. [arXiv:2402.16840 \[cs.CL\]](https://arxiv.org/abs/2402.16840)
- [97] tinygrad. 2023. Tinygrad. <https://github.com/tinygrad/tinygrad> Accessed: Dec 2023.
- [98] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [99] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahiri, Yasmine Babaee, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [100] Matteo Varvello, Kleomenis Katevas, Mihai Plesa, Hamed Haddadi, Fabian Bustamante, and Ben Livshits. 2022. BatteryLab: A Collaborative Platform for Power Monitoring. In *International Conference on Passive and Active Network Measurement*. Springer, 97–121.
- [101] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [102] Stylianos I Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 40–47.
- [103] Giorgos Vernikos, Arthur Bražinskas, Jakub Adamek, Jonathan Mallinson, Aliaksei Severyn, and Eric Malmi. 2023. Small Language Models Improve Giants by Rewriting Their Outputs. *arXiv preprint arXiv:2305.13514* (2023).
- [104] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, et al. 2023. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863* 1 (2023).
- [105] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling Conversational Interaction with Mobile UI Using Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) (CHI ’23). Association for Computing Machinery, New York, NY, USA, Article 432, 17 pages. <https://doi.org/10.1145/3544548.3580895>
- [106] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).
- [107] Graham Wilson, Martin Halvey, Stephen A. Brewster, and Stephen A. Hughes. 2011. Some like it hot: thermal feedback for mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, Canada) (CHI ’11). Association for Computing Machinery, New York, NY, USA, 2555–2564. <https://doi.org/10.1145/1978942.1979316>
- [108] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. 2022. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems* 4 (2022), 795–813.
- [109] Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870* (2023).
- [110] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.
- [111] Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. 2023. LLMcad: Fast and Scalable On-device Large Language Model Inference. *arXiv preprint arXiv:2309.04255* (2023).
- [112] Huatao Xu, Liying Han, Qirui Yang, Mo Li, and Mani Srivastava. 2024. Penetrative AI: Making LLMs Comprehend the Physical World. In *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications* (San Diego, CA, USA) (HOTMOBILE ’24). Association for Computing Machinery, New York, NY, USA, 1–7. <https://doi.org/10.1145/3638550.3641130>
- [113] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A First Look at Deep Learning Apps on Smartphones. In *The World Wide Web Conference* (San Francisco, CA, USA) (WWW ’19). Association for Computing Machinery, New York, NY, USA, 2125–2136. <https://doi.org/10.1145/3308558.3313591>
- [114] Mingxue Xu, Yao Lei Xu, and Danilo P Mandic. 2023. Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition. *arXiv preprint arXiv:2307.00526* (2023).
- [115] Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, et al. 2024. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092* (2024).
- [116] yepkit.com. 2023. YKUSH USB Controller. <https://www.yepkit.com/products/ykush> Accessed: Dec 2023.
- [117] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. 2023. EdgeMoE: Fast On-Device Inference of MoE-based Large Language Models. *arXiv preprint arXiv:2308.14352* (2023).
- [118] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2023. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 119–139. <https://www.usenix.org/conference/nsdi23/presentation/you>

- [119] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).
- [120] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems* 36 (2024).

SUPPLEMENTARY MATERIAL

A ADDITIONAL EVALUATION RESULTS

A.1 GPU Runtime of LLMFarm on iOS Devices

Table 5: LLMFarm (iOS) GPU performance and discharge rate for various models on mid-tier devices

| Device | Model | Quantization | Throughput (tokens/sec) | Discharge per Token (mAh/token) |
|----------------------|----------------|--------------|----------------------------|------------------------------------|
| iPhone 14 Pro | TinyLlama-1.1B | q3_k | 22.9826 ± 0.0557 | 0.0085 ± 0.0010 |
| | TinyLlama-1.1B | q4_k | 24.6531 ± 0.1295 | 0.0079 ± 0.0035 |
| | Gemma-2B | q3_k | 20.7053 ± 0.0949 | 0.0378 ± 0.0091 |
| | Gemma-2B | q4_k | 23.7938 ± 0.1190 | 0.0322 ± 0.0101 |
| | Zephyr-3B | q3_k | 10.5899 ± 0.2858 | 0.0130 ± 0.0088 |
| | Zephyr-3B | q4_k | 14.8165 ± 0.4524 | 0.0090 ± 0.0011 |
| iPhone SE | Llama2-7B | q3_k | 5.9889 ± 0.1374 | 0.0528 ± 0.0105 |
| | TinyLlama-1.1B | q3_k | 30.5524 ± 0.1284 | 0.0074 ± 0.0002 |
| | TinyLlama-1.1B | q4_k | 31.3951 ± 0.0936 | 0.0066 ± 0.0001 |
| | Gemma-2B | q3_k | 16.6111 ± 0.0713 | 0.0296 ± 0.0004 |
| | Gemma-2B | q4_k | 16.7310 ± 0.1718 | 0.0289 ± 0.0007 |
| | Zephyr-3B | q3_k | 13.7265 ± 0.2955 | 0.0263 ± 0.0002 |
| | Zephyr-3B | q4_k | 12.1618 ± 0.4816 | 0.0364 ± 0.0027 |

A.2 CPU Runtime of llama.cpp on Jetson Devices

Table 6: Jetson CPU performance and energy per token for various models and energy profiles.

| Device | Model | Quantization | Throughput (tokens/sec) | Energy per token (mWh/token) |
|-----------------------|----------------|--------------|----------------------------|---------------------------------|
| Orin AGX @ 50W | TinyLlama-1.1B | q4_k | 13.3085 ± 0.7917 | 0.0015 ± 0.0004 |
| | Gemma-2B | q4_k | 6.2280 ± 0.1455 | 0.0063 ± 0.0006 |
| | Zephyr-3B | q4_k | 5.4001 ± 0.2857 | 0.0033 ± 0.0013 |
| | Mistral-7B | q4_k | 2.2248 ± 0.0748 | 0.0201 ± 0.0033 |
| | Llama2-7B | q4_k | 2.3284 ± 0.0875 | 0.0204 ± 0.0035 |
| Orin AGX @ 30W | TinyLlama-1.1B | q4_k | 10.7740 ± 0.6574 | 0.0023 ± 0.0007 |
| | Gemma-2B | q4_k | 4.8950 ± 0.0925 | 0.0092 ± 0.0016 |
| | Zephyr-3B | q4_k | 4.2830 ± 0.2189 | 0.0067 ± 0.0054 |
| | Mistral-7B | q4_k | 1.7442 ± 0.0524 | 0.0181 ± 0.0056 |
| | Llama2-7B | q4_k | 1.8100 ± 0.0644 | 0.0102 ± 0.0043 |