Mobile and Edge Evaluation of Large Language Models

Stefanos Laskaridis [1] **Kleomenis Katevas** [1] **Lorenzo Minto** [1] **Hamed Haddadi** [1 2]

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

1. Introduction

Our devices are getting increasingly capable in performing tasks that have traditionally required human intelligence (Bubeck et al., 2023; Schaeffer et al., 2024). The proliferation of capable on-device hardware has enhanced their capabilities in areas such as vision (Radford et al., 2021; Dosovitskiy et al., 2021), language (Radford et al., 2019; Vernikos et al., 2023) and sensor understanding (Xu et al., 2024a). Lately, transformers (Vaswani et al., 2017) have become the go-to architecture for deep learning models, with attention mechanisms offering unparalleled performance and scalability, along with the ability to model long sequence data with fewer inductive biases across modalities (Dosovitskiy et al., 2021; Radford et al., 2019; 2023). This has given birth to "foundation models", large models

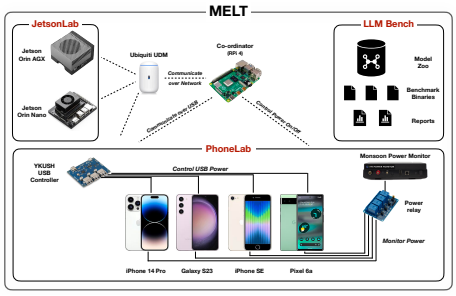


Figure 1: Architecture of MELT device farm

that are trained on large corpora of data and act as universal backbones for a series of downstream tasks. Despite their accuracy benefits, such models have been pushing the computational boundaries of cloud systems, both in terms of training (Dao, 2023) and deployment (Kwon et al., 2023), which poses questions both in terms of the sustainability (Wu et al., 2022; Patterson et al., 2022; 2021; You et al., 2023), as well as the privacy and custody of user data (Ciniselli et al., 2022). We recognize that it is not always necessary to deploy a highly over-provisioned network to solve the task at hand (Eldan & Li, 2023).

Given that model performance, even for smaller models, does not saturate quickly, i.e., more data gives performance gains (Zhang et al., 2023), and the need for user privacy (Xiao et al., 2023a), we focus our attention to the study of deploying LLMs at the edge (Laskaridis et al., 2022), with particular emphasis on the mobile execution of chat assistants. To this end, we have created our own infrastructure, named MELT (Mobile Evaluation of Language Transformers), 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. To the best of our knowledge, our tool is the first to support granular on-device energy measurements across targets (i.e., Android, iOS, Linux) with realistic interactions.

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

[1]Brave Software, London, UK [2]Imperial College London, London, UK. Correspondence to: Stefanos Laskaridis <mail@stefanos.cc>.

ES-FoMo workshop at the 41st International Conference on Machine Learning, Vienna, Austria. 2024.

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	-
PhoneLab (Mobile devices)						
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
JetsonLab (Edge devices)						
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

non-negligible accuracy cost. Last, drawing from its energy and thermal behavior, the continuous execution of LLMs remains elusive, as both negatively affect user experience.

2. 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 *co-ordinating* process running on a Raspberry Pi 4 (RPi). The *co-ordinator* communicates with two sets of devices, namely *PhoneLab* (Sec. D.1) which consists of mobile devices and *JetsonLab* (Sec. D.2), which includes Nvidia Jetson boards. The architecture of our device farm is shown on Fig. 1 and includes devices of Tab. 1. Additional details about the infrastructure can be found in Appendix D.

3. 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, v) the *evaluation* of the LLM accuracy and vi) the *reporting* of the results. The workflow of MELT is depicted in Fig. 2, while details about each component can be found in Appendix E.

4. Evaluation

In this section, we present results from running LLMs across devices and platforms with MELT. Detailed setup and results are included Appendix F. We also include an analysis on the accuracy impact of quantization of various models, precisions and quantization methods in Appendix F.3.

4.1. Macro Experiments

In *macro-experiments*, we measure how a chat assistant behaves on device, with real conversations (details in Sec. F.2.1) and variable token length output.

4.1.1. On-device Runtime

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 (Pope et al., 2023) when encoding a sequence of tokens and the compute vs. memory boundedness of the workload (Mark Sherwood, 2024). 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 and up to 3.53× 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 27.19% 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, by +78.93% on average.

Table 2: Supported pretrained models

Model Type	Size	Type	HuggingFace Repository
TinyLlama (Zhang et al., 2023)	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 (Google Inc., 2024)	2B	Decoder	*google/gemma-2b-it*
	7B		*google/gemma-7b-it*
Llama-2 (Touvron et al., 2023b)	7B	Decoder	*meta-llama/Llama-2-7b-chat-hf*
	13B		*meta-llama/Llama-2-13b-chat-hf*

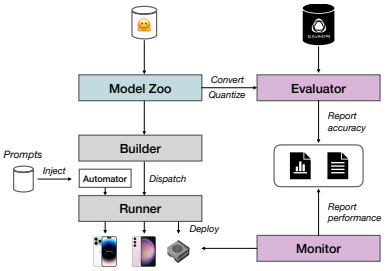
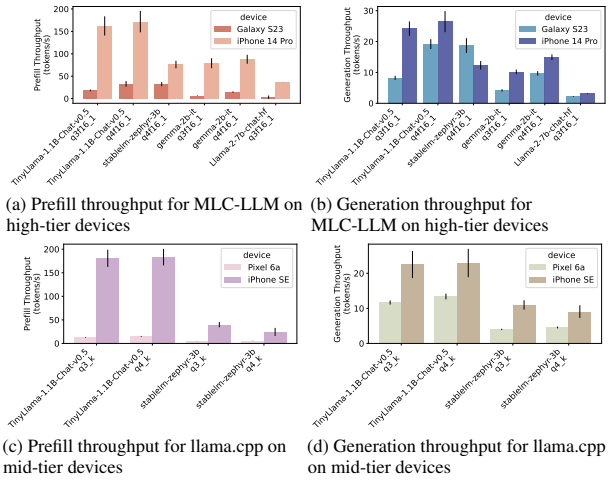


Figure 2: MELT Workflow

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 (Patterson et al., 2022). 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 efficiency, but this could be attributed to the increased inference latency compared to LLMFarm's Metal acceleration (CPU experiments in Appendix F.2.3).



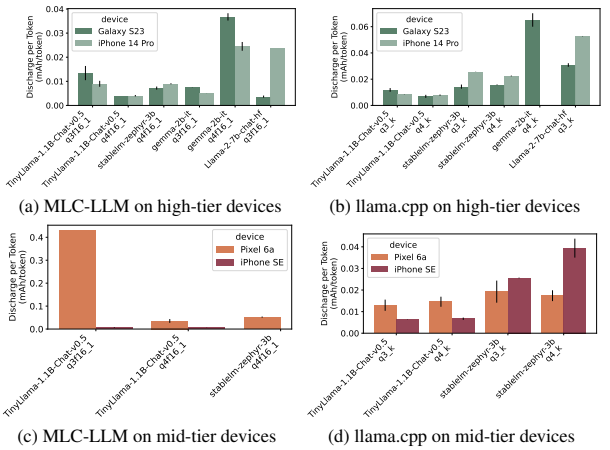
(a) Prefill throughput for MLC-LLM on high-tier devices

(b) Generation throughput for MLC-LLM on high-tier devices

(c) Prefill throughput for llama.cpp on mid-tier devices

(d) Generation throughput for llama.cpp on mid-tier devices

Figure 3: Throughput across frameworks and devices



(a) MLC-LLM on high-tier devices

(b) llama.cpp on high-tier devices

(c) MLC-LLM on mid-tier devices

(d) llama.cpp on mid-tier devices

Figure 4: Discharge per token across frameworks and devices. Missing bars indicate unsuccessful runs (OOM or time limit).

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 8.5W and 14W, respectively. 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, at an average input of 40 tokens and generation length of 135 tokens, not accounting for simultaneous workloads.

4.1.2. 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 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. 9 in Appendix) during which the device became largely unresponsive; *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.

ii) 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. 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. Results are depicted on Fig. 6a. 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. 6b), as signified by the straight line in the generation throughput. The initial higher generation throughput can be attributed to the context not being filled.

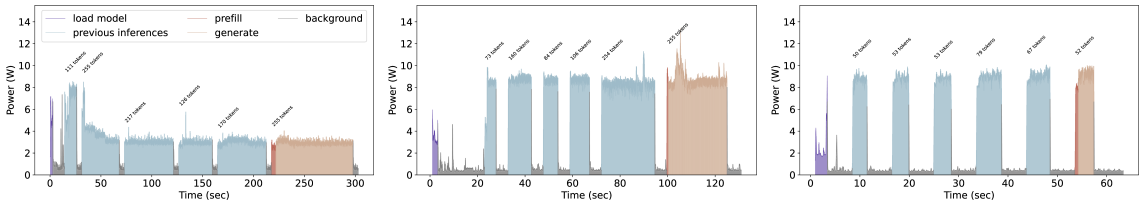
iii) Temperature does not only affect device performance, but also user comfort (Wilson et al., 2011). Devices nowadays come in various forms, but mostly remain passively cooled. Therefore, heat dissipation is mainly facilitated by the use of specific materials and heat management is governed by the OS. The power draw that was witnessed in Fig. 5b did cause temperatures to rise to uncomfortable levels, reaching 47.9°C as shown in Fig. 12a of the Appendix.

4.2. Micro Experiments & Bottlenecks

In *micro-experiments*, we fix the output length and disregard <EOS> to measure specific ops in a controlled manner.

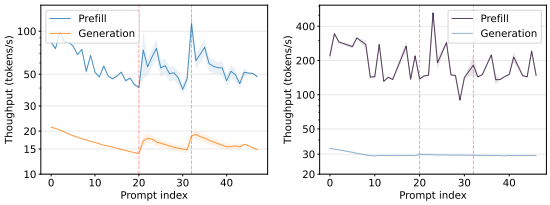
4.2.1. ML Operations

We start by introspecting Llama-7B (3-bit) on Android. We compile a custom version of TVM and MLC-LLM where



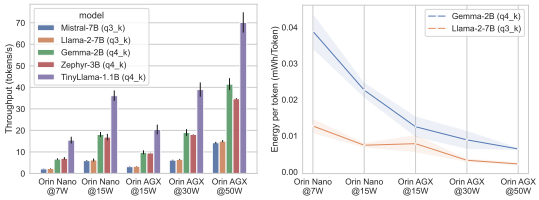
(a) MLC-LLM on Galaxy S23 (b) MLC-LLM on iPhone 14 Pro (c) LLMFarm on iPhone 14 Pro

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.



(a) iPhone 14 Pro on LLMFarm (b) Jetson AGX (50W) on llama.cpp

Figure 6: Continuous inference on mobile and edge devices with Zephyr-3B (4-bit).



(a) Generation throughput. (b) Energy consumed per token.

Figure 7: LLM execution on Jetson devices across energy modes with llama.cpp.

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. 11 for the `prefill`, `embed` and `decode` operations. Most of the execution is taken up by de-quantize and matrix multiplication fused operations for the `prefill` and `decode` operations, taking 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. 4.1.1. 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 6), 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.

4.2.2. Memory Usage and Bottlenecks

It is known that LLM execution is bottlenecked by the memory bandwidth requirements during generation (Kwon et al., 2023; Dao et al., 2022; Dao, 2023). Our analysis corroborates this on the mobile side, by what is shown in the memory profiling of Fig. 12b, 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` tool.

4.3. Runtime at the Edge

Offloading. Hitherto, we have witnessed that high-end mobile devices with more than 6GB of memory can run a chat LLM at a reasonable rate. However, this comes at the cost of significant battery depletion (see Sec. 4.1.1), QoE (see Sec. 4.1.2) and end-task accuracy (see Sec. F.3). Therefore, we envision that the future of LLM execution can be collaborative and cross-device at the edge (Laskaridis et al., 2022; Qualcomm, 2023). 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) under various energy modes, which configure the number of active cores and their frequency, along with memory frequency to provide different power envelopes.

In Fig. 7a 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. 6. In Fig. 7b, we quantify the energy efficiency of two models (Llama-7B (3-bit), Gemma-2B (4-bit)) running across different energy modes. Interestingly, efficiency moves in the same direction as device TDP, probably due to bottlenecked generation from the lowered memory frequency.

5. 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.

Impact Statement

Our work aims to benchmark and assess the feasibility of running large language models (LLMs) at the edge, with the objective of promoting a fairer, more private and sustainable deployment method. We identify three key aspects where our research impacts the current landscape. Below, we provide a brief overview of each area, with a more detailed discussion available in Appendix B.3 and G.

Privacy. The predominant approach to using LLMs today involves black-box access through providers such as ChatGPT, Anthropic, and Gemini. This method requires transmitting user prompts over the wire, thereby compromising their privacy. By enabling local deployment of LLMs, our work aims to enhance privacy by eliminating the need for data transmission to external servers.

Democratization. The high cost of training LLMs currently restricts access to a few dominant players who control and influence the technology. However, the availability of models with open weights provides an opportunity for broader access. Local deployment allows users to customize models, promoting a more equitable use of LLMs.

Sustainability. While not all tasks necessitate multi-billion parameter models, which opens the door for more sustainable edge deployments, it is important to consider the macroscopic environmental impact. Edge energy sources are typically less green compared to optimized datacenters (Wu et al., 2022; Patterson et al., 2022), a discrepancy which should be considered when evaluating the overall energy impact at a larger scale.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics.

Alibaba. MNN-LLM, 2023. URL https://github.com/alibaba/MNN.

Alizadeh, K., Mirzadeh, I., Belenko, D., Khatamifard, K., Cho, M., Mundo, C. C. D., Rastegari, M., and Farajtabar, M. Llm in a flash: Efficient large language model inference with limited memory, 2023.

Almeida, M., Laskaridis, S., Leontiadis, I., Venieris, S. I., and Lane, N. D. 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*, pp. 1–6, 2019.

Almeida, M., Laskaridis, S., Mehrotra, A., Dudziak, L., Leontiadis, I., and Lane, N. D. Smart at what cost? characterising mobile deep neural networks in the wild. In *Proceedings of the 21st ACM Internet Measurement Conference*, pp. 658–672, 2021.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

android.com. Aicore, 2023. URL https://developer.android.com/ml/aicore. Accessed: Dec 2023.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Barbara Krasnoff,. How to use Android 12's call screening features, 2021. URL https://www.theverge.com/22792060/call-screening-android-12-google-pixel-how-to. Accessed: Mar 2024.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Cai, T., Li, Y., Geng, Z., Peng, H., and Dao, T. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. https://github.com/FasterDecoding/Medusa, 2023.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chen, T., Moreau, T., Jiang, Z., Shen, H., Yan, E. Q., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. Tvm: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, 11(20), 2018.

Chollet, F. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Ciniselli, M., Pascarella, L., and Bavota, G. 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*, MSR '22, pp. 167–178, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393034. doi: 10.1145/3524842.3528440. URL https://doi.org/10.1145/3524842.3528440.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Dong, X. L., Moon, S., Xu, Y. E., Malik, K., and Yu, Z. Towards next-generation intelligent assistants leveraging llm techniques. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 5792–5793, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599572. URL https://doi.org/10.1145/3580305.3599572.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Fan, H., Chau, T., Venieris, S. I., Lee, R., Kouris, A., Luk, W., Lane, N. D., and Abdelfattah, M. S. Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 599–615. IEEE, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Frantar, E. and Alistarh, D. SparseGPT: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023a.

Frantar, E. and Alistarh, D. Qmoe: Practical sub-1-bit compression of trillion-parameter models. *arXiv preprint arXiv:2310.16795*, 2023b.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

Gerganov, G. llama.cpp, 2023. URL https://github.com/ggerganov/llama.cpp.

Google Inc. Gemma: Introducing new state-of-the-art open models, 2024. URL https://blog.google/technology/developers/gemma-open-models/. Accessed: Mar 2024.

Goyal, S., Choudhury, A. R., Raje, S., Chakaravarthy, V., Sabharwal, Y., and Verma, A. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pp. 3690–3699. PMLR, 2020.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gu, Y., Dong, L., Wei, F., and Huang, M. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Guan, Y., Li, Z., Leng, J., Lin, Z., and Guo, M. Transkimmer: Transformer learns to layer-wise skim. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pp. 7275–7286, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.502. URL https://aclanthology.org/2022.acl-long.502.

guinmoon. LLMFarm, 2023. URL https://github.com/guinmoon/LLMFarm.

Hannun, A., Digani, J., Katharopoulos, A., and Collobert, R. MLX: Efficient and flexible machine learning on apple silicon, 2023. URL https://github.com/ml-explore.

Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., and Van Gool, L. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

Javaheripi, Mojan and Bubeck, Sébastien. Phi-2: The surprising power of small language models, 2024. URL https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/. Accessed: Mar 2024.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Karpathy, A. llama2.c, 2023. URL https://github.com/karpathy/llama2.c. Accessed: Dec 2023.

Kim, G. and Cho, K. 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)*, pp. 6501–6511, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.508. URL https://aclanthology.org/2021.acl-long.508.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Köpf, A., Kilcher, Y., von Rütte, D., Anagnostidis, S., Tam, Z.-R., Stevens, K., Barhoum, A., Duc, N. M., Stanley, O., Nagyfi, R., et al. Openassistant conversations–democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., and Lane, N. D. Spinn: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370851. doi: 10.1145/3372224.3419194. URL https://doi.org/10.1145/3372224.3419194.

Laskaridis, S., Venieris, S. I., Kouris, A., Li, R., and Lane, N. D. The future of consumer edge-ai computing. *arXiv preprint arXiv:2210.10514*, 2022.

Li, Y., He, J., Zhou, X., Zhang, Y., and Baldridge, J. Mapping natural language instructions to mobile UI action sequences. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J. (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8198–8210, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.729. URL https://aclanthology.org/2020.acl-main.729.

libimobiledevice. ideviceinstaller, 2024. URL https://github.com/libimobiledevice/ideviceinstaller. Accessed: Mar 2024.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Lin, S., Hilton, J., and Evans, O. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023a.

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023b.

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*, 2024.

llama.cpp Team. k-quants, June 2023. URL https://github.com/ggerganov/llama.cpp/pull/1684. Accessed: March 2024.

Luo, Z., Lu, L., Jin, Y., Jia, L., and Liang, Y. Calabash: Accelerating attention using a systolic array chain on fpgas. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 242–247. IEEE, 2023.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4):2322–2358, 2017.

Mark Sherwood. Large Language Models On-Device with MediaPipe and TensorFlow Lite, March 2024. URL https://developers.googleblog.com/2024/03/running-large-language-models-on-device-with-mediapipe-andtensorflow-lite.html. Accessed: March 2024.

McKinzie, B., Gan, Z., Fauconnier, J.-P., Dodge, S., Zhang, B., Dufter, P., Shah, D., Du, X., Peng, F., Weers, F., et al. Mm1: Methods, analysis & insights from multimodal llm pre-training. *arXiv preprint arXiv:2403.09611*, 2024.

mit-han lab. Tinychatengine, 2023. URL https://github.com/mit-han-lab/TinyChatEngine. Accessed: Dec 2023.

Monsoon Solutions Inc. Monsoon Solutions Inc., 2023. URL https://www.msoon.com. Accessed: Dec 2023.

Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A. F., Ippolito, D., Choquette-Choo, C. A., Wallace, E., Tramèr, F., and Lee, K. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., and Dean, J. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Patterson, D., Gonzalez, J., Hölzle, U., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D. R., Texier, M., and Dean, J. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

Qualcomm. The future of ai is hybrid. White paper, Qualcomm, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pp. 28492–28518. PMLR, 2023.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Rebedea, T., Dinu, R., Sreedhar, M. N., Parisien, C., and Cohen, J. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In Feng, Y. and Lefever, E. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 431–445, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.40. URL https://aclanthology.org/2023.emnlp-demo.40.

Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.-J., Anderson, B., Breughe, M., Charlebois, M., Chou, W., et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459. IEEE, 2020.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, aug 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

Schaeffer, R., Miranda, B., and Koyejo, S. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2024.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

Sellam, T., Das, D., and Parikh, A. P. Bleurt: Learning robust metrics for text generation. In *Proceedings of ACL*, 2020.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8815–8821, 2020.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

team, M. MLC-LLM, 2023. URL https://github.com/mlc-ai/mlc-llm.

Thawakar, O., Vayani, A., Khan, S., Cholakkal, H., Anwer, R. M., Felsberg, M., Baldwin, T., Xing, E. P., and Khan, F. S. Mobillama: Towards accurate and lightweight fully transparent gpt, 2024.

tinygrad. Tinygrad, 2023. URL https://github.com/tinygrad/tinygrad. Accessed: Dec 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Varvello, M., Katevas, K., Plesa, M., Haddadi, H., Bustamante, F., and Livshits, B. Batterylab: A collaborative platform for power monitoring: https://batterylab. dev. In *International Conference on Passive and Active Network Measurement*, pp. 97–121. Springer, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vernikos, G., Bražinskas, A., Adamek, J., Mallinson, J., Severyn, A., and Malmi, E. Small language models improve giants by rewriting their outputs. *arXiv preprint arXiv:2305.13514*, 2023.

Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 1, 2023.

Wang, B., Li, G., and Li, Y. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3580895. URL https://doi.org/10.1145/3544548.3580895.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wilson, G., Halvey, M., Brewster, S. A., and Hughes, S. A. Some like it hot: thermal feedback for mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pp. 2555–2564, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450302289. doi: 10.1145/1978942.1979316. URL https://doi.org/10.1145/1978942.1979316.

Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., et al. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813, 2022.

Xiao, G., Lin, J., and Han, S. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*, 2023a.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023b.

Xu, D., Yin, W., Jin, X., Zhang, Y., Wei, S., Xu, M., and Liu, X. Llmcad: Fast and scalable on-device large language model inference. *arXiv preprint arXiv:2309.04255*, 2023a.

Xu, H., Han, L., Yang, Q., Li, M., and Srivastava, M. Penetrative ai: Making llms comprehend the physical world. In *Proceedings of the 25th International Workshop on*

Mobile Computing Systems and Applications, HOTMOBILE '24, pp. 1–7, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400704970. doi: 10.1145/3638550.3641130. URL https://doi.org/10.1145/3638550.3641130.

Xu, M., Liu, J., Liu, Y., Lin, F. X., Liu, Y., and Liu, X. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, WWW '19, pp. 2125–2136, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313591. URL https://doi.org/10.1145/3308558.3313591.

Xu, M., Xu, Y. L., and Mandic, D. P. Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition. *arXiv preprint arXiv:2307.00526*, 2023b.

Xu, M., Yin, W., Cai, D., Yi, R., Xu, D., Wang, Q., Wu, B., Zhao, Y., Yang, C., Wang, S., et al. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*, 2024b.

yepkit.com. Ykush usb controller, 2023. URL https://www.yepkit.com/products/ykush. Accessed: Dec 2023.

Yi, R., Guo, L., Wei, S., Zhou, A., Wang, S., and Xu, M. Edgemoe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352*, 2023.

You, J., Chung, J.-W., and Chowdhury, M. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 119–139, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/you.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama, Sep 2023. URL https://github.com/jzhang38/TinyLlama.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

Appendix

Contents of the Appendix

A Contributions **11**

B Background & Motivation **11**
B.1 Transformer Preliminaries . . . 11
B.2 Large Language Models . . . 12
B.3 Current State and Motivating Factors . . . 12

C Related Work **13**

D Infrastructure **13**
D.1 PhoneLab . . . 13
D.2 JetsonLab . . . 14

E MELT **Workflow Components** **14**
E.1 Model Zoo and Evaluation . . . 14
E.2 Automated On-Device Benchmarking . . . 15

F Evaluation **17**
F.1 Experimental Setup . . . 17
F.2 Macro-experiments . . . 17
F.2.1 Dataset Qualitative Analysis . . . 17
F.2.2 Model Loading Latency . . . 17
F.2.3 CPU runtimes . . . 18
F.3 Impact of Quantization . . . 18
F.4 Micro-benchmarks . . . 19
F.4.1 ML Operations . . . 19
F.4.2 Memory Usage and Bottlenecks . . . 19

G Discussion & Limitations **20**

A. Contributions

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.

B. Background & Motivation

B.1. Transformer Preliminaries

Transformers (Vaswani et al., 2017) 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 (Dosovitskiy et al., 2021) and multi-modal use-cases (Radford et al., 2021). 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 as subspace (e.g., WordPiece (Devlin et al., 2019) or BytePair (Radford et al., 2019) 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 $A(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, 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 (Beltagy et al., 2020) or approximate (Wang et al., 2020; Ainslie et al., 2023) attention mechanisms, as well as attention-free variants as of lately (Gu & Dao, 2023). *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 input sequence $X = \{x_1, x_2, \ldots, x_t\}$, the model generates x_{t+1}, which is fed to the next generation step. Key-Value cache (Pope et al., 2023) optimizes this by storing intermediary attention states.

B.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 (Touvron et al., 2023b). 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 (Ouyang et al., 2022)* 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 (Ouyang et al., 2022) or automated (Bai et al., 2022) feedback, to promote a certain style or content or reply 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 (Zhou et al., 2024).

B.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 (Ciniselli et al., 2022), making them amenable to various attacks (Nasr et al., 2023). 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 (Touvron et al., 2023a;b), Mistral AI (Jiang et al., 2023), Google (Google Inc., 2024) and Microsoft (Javaheripi, Mojan and Bubeck, Sébastien, 2024). 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. Towards this end, new frameworks are emerging for enabling local execution of LLMs across different targets (Gerganov, 2023; team, 2023; Hannun et al., 2023; Mark Sherwood, 2024; Alibaba, 2023; tinygrad, 2023). In this effort, quantization (Shen et al., 2020; Frantar et al., 2022; Lin et al., 2023) 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 (Touvron et al., 2023b) 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 (Wu et al., 2022; Patterson et al., 2021; 2022; You et al., 2023), since the 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 or 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. i

C. 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. (Ignatov et al., 2018) 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 (Almeida et al., 2019) quantified the different dynamics of model execution across various mobile, edge and desktop devices. MLPerf (Reddi et al., 2020) 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 (Almeida et al., 2019; Xu et al., 2019) 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 (Mao et al., 2017), 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 faciliated. To the best of our knowledge, this is the first study of LLMs on-device performance. Prior work has either focused on training efficiency (You et al., 2023; Rajbhandari et al., 2020) or served inference (Kwon et al., 2023; Aminabadi et al., 2022) 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 (Gerganov, 2023) and MLC (team, 2023) have stood out, offering cross-platform accelerated execution and support for various LLM architectures and device targets. Other open-source frameworks include llama2.c (Karpathy, 2023), aimed at simplicity without dependencies and tinygrad (tinygrad, 2023), focused on accelerated execution, but without support quantized mobile execution. Last, TinyChatEngine (mit-han lab, 2023) showcased on-device inference with compressed models, but lacks mobile support. Lately, OS providers have released their own platforms, such as Apple's MLX (Hannun et al., 2023) and Google's AICore (android.com, 2023). The former only provides support for desktop platforms (M-series SoCs) and the latter remains closed-source and only deployed on Pixel 8 Pro. Very recently, Google also released MediaPipe (Mark Sherwood, 2024) for running LLMs on device.

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 (Lin et al., 2023; Frantar et al., 2022; Xiao et al., 2023b; Liu et al., 2023b; Dettmers et al., 2023; Kim et al., 2023) and sparsification/pruning schemes (Ma et al., 2023; Frantar & Alistarh, 2023a), low-rank methods (Xu et al., 2023b) and distillation-based solutions (Gu et al., 2023) aimed specifically at LLMs. Orthogonally, one can leverage secondary storage for running LLMs with limited local resources (Sheng et al., 2023; Alizadeh et al., 2023). The quadratic cost of attention has also been a large scalability issue. Therefore, various techniques try to address this cost, through different attention patterns (Wang et al., 2020; Beltagy et al., 2020; Dao et al., 2022; Dao, 2023), token skipping (Guan et al., 2022; Kim & Cho, 2021; Goyal et al., 2020) or alternative architectures (Peng et al., 2023; Gu & Dao, 2023).

Employing multiple models for dropping the overall cost of inference has also been a popular approach, with techniques such as *Mixture-of-Experts* (Frantar & Alistarh, 2023b; Yi et al., 2023; Fedus et al., 2022) 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* (Chen et al., 2023; Cai et al., 2023) 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. (Xu et al., 2023a) proposes a distributed such setup for the edge. For a more complete overview of related work, we divert the reader to (Wan et al., 2023; Xu et al., 2024b).

D. Infrastructure

In this section, we expand on the infrastructure overview of Sec. 2, by providing details on *PhoneLab* and *JetsonLab*.

D.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) (Monsoon Solutions Inc., 2023). To facilitate accurate power measurements, we employ a battery bypass process 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 (Varvello et al., 2022) 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* (yepkit.com, 2023). 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. 1).

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` (libimobiledevice, 2024) for Android and iOS, respectively.

D.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 100Hz[1]. 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. 1).

Compilation of packages and models happens directly on Jetson devices over Docker images[2]. 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.

E. MELT Workflow Components

In this section, we move to the workflow of MELT, as introduced in Sec. 3 and depicted in Fig. 2. This workflow is used for our LLM evaluation process, as described later in Sec. F.1.

E.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 directly

[1]This granularity was explicitly tuned to capture events of interest, without interfering with the measurement itself due to I/O thrashing.
[2]Based on images from `https://github.com/dusty-nv/jetson-containers/`.

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 model[3] on a given dataset and reporting its accuracy. We leveraged the LM-Evaluation Harness (Gao et al., 2021) 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 dataset[4]. The currently supported datasets are depicted in Table 6 and the results of the evaluation are presented in Sec. F.3.

E.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 (Chen et al., 2018)). Due to the overhead of tracing very granular events (i.e., single operations), we only enable the respective flag in specific experiments (Sec. 4.2).

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 (team, 2023; Chen et al., 2018) and llama.cpp (Gerganov, 2023) (detailed in Tab. 3). While there are increasingly more such frameworks (Alibaba, 2023; Hannun et al., 2023; Mark Sherwood, 2024; tinygrad, 2023; mit-han lab, 2023), we selected the ones with the highest popularity (measured by their stars on GitHub) and widest model and 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 in 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 precanned prompts, sampled from the OAAST chat dataset (Köpf et al., 2023), and interact in a multi-turn manner with the LLM. More information about the distribution of these prompts in Sec. F.2.1.

For mobile execution, we have used custom native applications[5] 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 (Rebedea et al., 2023) 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

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

[4]Because of issues with evaluating quantized models on MLC-LLM, we evaluate AWQ (Lin et al., 2023) quantized models with `autoawq` package as a proxy.

[5]All applications have graphical user interface except for llama.cpp on Android, for which we used the ADB CLI interface (Almeida et al., 2021).

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.

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, $Q^{\text{device}}_{\text{experiments}}$, 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^device_experiments :  # 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
```

Runner. The runner is tasked with deploying the built application or binary, along with the associated converted 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. 2, 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.

F. Evaluation

In this section, we provide additional details about the setup and experiments of Sec. 4 that could not be presented in the main text.

F.1. Experimental Setup

Table 3: Frameworks and platforms supported by MELT.

Framework	Backend	Version	Supported Platforms	Quantization
MLC-LLM (team, 2023)	TVM (Chen et al., 2018)	`96a68e`†	Android (GPU), iOS (Metal), Linux (CUDA)	Group Quantization (Shen et al., 2020), GPTQ (Frantar et al., 2022), FasterTransformer Row-wise Quantization
llama.cpp (Gerganov, 2023)	llama.cpp (Gerganov, 2023)	`b22022`‡	Android (CPU, GPU), Linux (CUDA)	k-quants (llama.cpp Team, 2023)
LLMFarm (guinmoon, 2023)	llama.cpp (Gerganov, 2023)	`7226a8`	iOS (Metal)	

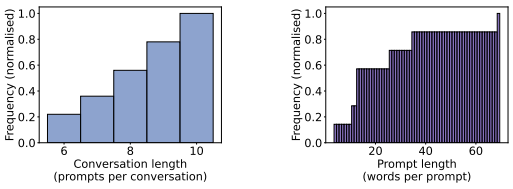
† We used version `784530` for supporting Gemma models and Llama-2-7B on Android.
‡ We used version `d5ab29` for supporting Gemma models.

For our experiments, we leverage the infrastructure and methodology described in Sec. 2 and 3, 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 search[6]. We always run on GPU, except for the case of llama.cpp for Android, where the gains from running on GPU were minimal[7]. 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.

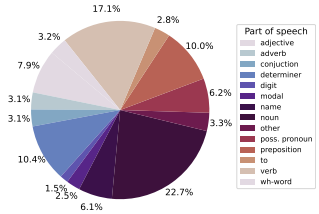
F.2. Macro-experiments

F.2.1. Dataset Qualitative Analysis

For macro-experiments, we used a subset of prompts from the OpenAssistant/oasst1 dataset (Köpf et al., 2023). 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. 8, where we depict the distributions of conversation lengths, prompt lengths and also part-of-speech categories across prompts. We can see from Fig. 8a 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.



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



(b) Part of speech categories distribution across prompts

Figure 8: Qualitative analysis of prompts used for macro-experiments to assess the behaviour of LLM-powered chats on device.

F.2.2. Model Loading Latency

In this section, we have analysed the model loading latency per device for various frameworks, which we depict as a boxplot in Fig. 9. We see that most models are loaded in less than 5 seconds, with significant outliers when model sizes get too

[6](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.

[7]Indicatively, 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`.

large. While iPhones show lower loading latencies, this is a repercussion of also supporting only smaller models due to their limited RAM size.

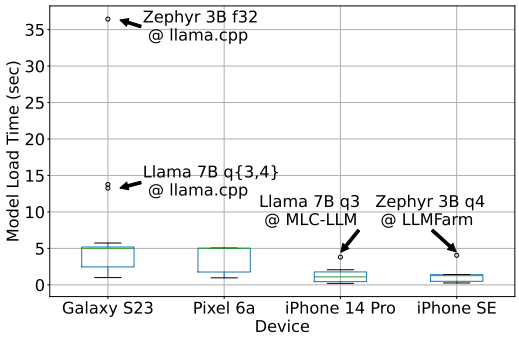


Figure 9: Model loading time per device. Each supports different set of models, based on available memory and framework.

F.2.3. CPU RUNTIMES

In the main text, we provided GPU runtimes for all but android on Llama.cpp due to its abysmal performance. To further complement our results, we also showcase below results from CPU execution of llama.cpp on iOS (Tab. 4) and Jetson devices (Tab. 5). Overall, performance is significantly lower than the equivalent GPU execution, and consumes more energy for the same workload.

Table 4: Generation throughput and energy of iOS devices on llama.cpp

Device	Model	Throughput	Discharge (mAh/token)
iPhone-14-Pro	TinyLlama-q4	15.5054±1.2012	0.0252±0.0039
	Zephyr-q3	6.4820±0.4226	0.0587±0.0039
	Gemma-2B-q4	12.4338±0.1512	0.0141±0.0018
iPhone-SE	TinyLlama-q4	13.4730±0.8370	0.0185±0.0004
	Zephyr-q3	5.0990±0.7038	0.0482±0.0043
	Gemma-2B-q4	3.3561±0.5763	0.0946±0.0036

Table 5: Generation throughput and energy of Jetson AGX on llama.cpp

Device	Model	Throughput	Energy (mWh/token)
OrinAGX-50W	TinyLlama-q4	13.3085±0.7917	0.0015±0.0004
	Zephyr-q4	5.4001±0.2857	0.0033±0.0013
OrinAGX-30W	TinyLlama-q4	10.7740±0.6574	0.0023±0.0007
	Zephyr-2B-q4	4.2830±0.2189	0.0067±0.0054

F.3. Impact of Quantization

Table 6: Evaluation datasets description

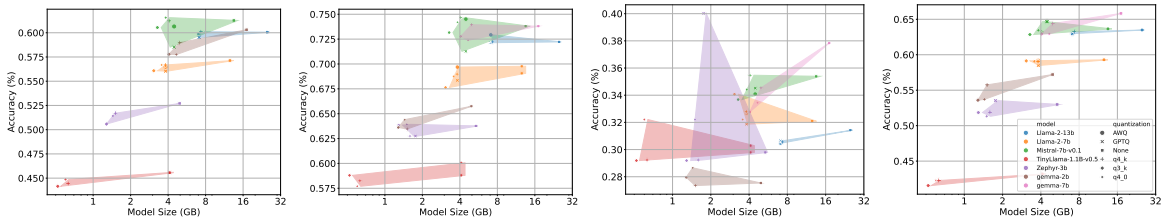
Dataset	Task	Size	Description
HellaSwag (Zellers et al., 2019)	Common-sense NLI	70k	Given an event description, select the most likely continuation.
Winogrande (Sakaguchi et al., 2021)	Common-sense NLI	44k	Benchmark for common-sense reasoning, designed not to be easily solvable by statistical models and plain word associations.
ThuthfulQA (Lin et al., 2021)	Knowledge NLG	817	Benchmark for measuring truthfulness in a model's generated answers.
ARC-{E,C} (Chollet, 2019)	Reasoning NLI	5.2k, 2.6k	Science and language exam questions from a variety of sources. E: Easy; C: Complex

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 (Frantar et al., 2022; Xiao et al., 2023b; Lin et al., 2023). However, this often comes at the expense of model accuracy, especially at sub 4-bit weight precision. 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 (Ouyang et al., 2022) alignment can affect the original performance. A description of the employed quantization schemes is presented in Sec. C. We use the benchmark datasets depicted in Tab. 6, 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 (Sellam et al., 2020) 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 ($\leq$3B parameters) were unusable with 3-bit precision, mostly hallucinating or plainly repeating the prompt. On the other hand, there was no single quantization scheme that performed uniformly better across the board. For larger models ($\geq$7B parameters), AWQ (Lin et al., 2023) and GPTQ (Frantar et al., 2022) performed slightly better, at the expense of elevated model sizes.



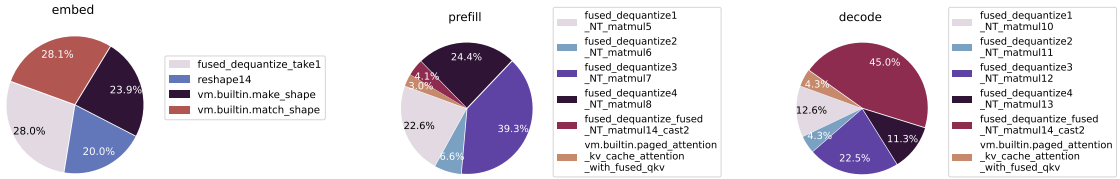
(a) HellaSwag (b) Winogrande (c) TruthfulQA (d) ARC

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

F.4. Micro-benchmarks

F.4.1. ML Operations

Here, we provide a visualization fused operators and their percentage of the computation for three LLM operations, namely `embed`, `prefill` and `decode` for Samsung Galaxy S23 on MLC-LLM. We comment on the results in Sec. 4.2.1.

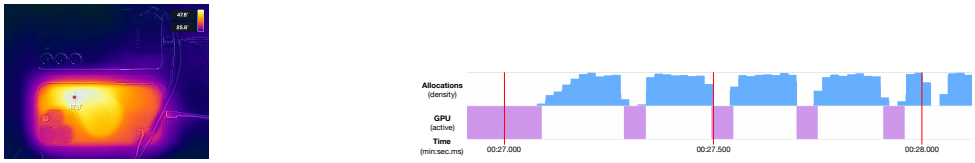


(a) Embed (b) Prefill (c) Decode

Figure 11: 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.

F.4.2. Memory Usage and Bottlenecks

Here, we provide details of the thermal and memory behaviour of LLM runtime on iPhone 14 Pro, as discussed in Sec. 4.1.2 and Sec. 4.2.2, respectively. Specifically, Fig. 12a showcases the temperature of the device after a full conversation on Zephyr-3B (4-bit) on MLC-LLM while Fig. 12b depicts the GPU stalls and memory allocations as measured through `xctrace` and visualized with the Apple Instruments application.



(a) Temperature after a full conversation on Zephyr-3B (4-bit) on MLC-LLM (b) Memory trace when running Zephyr-3B (4-bit) on LLMFarm

Figure 12: Thermal and memory behavior on iPhone 14 Pro

G. Discussion & Limitations

Summary of results. In the main evaluation, 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 (Dao, 2023; Chen et al., 2023; Gu & Dao, 2023) and hardware (Fan et al., 2022; Luo et al., 2023) 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 (Bubeck et al., 2023; Schaeffer et al., 2024).

Multimodality & emergent abilities. In terms of capabilities, the ability of models to deal with multi-modal inputs and outputs become of great value (Liu et al., 2023a; Radford et al., 2021; McKinzie et al., 2024), 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 (Dong et al., 2023). At the same time, users will be able to interact with interfaces in natural language to accomplish tasks (Li et al., 2020), without the need to imperatively define the individual steps (Schick et al., 2024; Wang et al., 2023). Last, we envision this automation expanding to interactions between humans, where individuals would be able to proxy their availability over smart assistants (Barbara Krasnoff,, 2021).

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 (Qualcomm, 2023) and hierarchical, with part of the workload being accelerated at the edge or cloud (Xu et al., 2023a; Laskaridis et al., 2022; 2020).

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 (Thawakar et al., 2024; Liu et al., 2024), 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 (Wu et al., 2022). 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. C, for further optimizing these workloads. We leave such topics as future work.