
MLPERF MOBILE INFERENCE BENCHMARK

Vijay Janapa Reddi¹ David Kanter² Peter Mattson³ Jared Duke³ Thai Nguyen³ Ramesh Chukka⁴
Ken Shiring⁵ Koan-Sin Tan⁵ Mark Charlebois⁶ William Chou⁶ Mostafa El-Khamy⁷
Jungwook Hong⁷ Tom St. John⁸ Cindy Trinh⁹ Michael Buch¹ Mark Mazumder¹
Relja Markovic² Thomas Atta⁴ Fatih Cakir⁷ Masoud Charkhabi³
Xiaodong Chen⁷ Cheng-Ming Chiang⁵ Dave Dexter¹⁰ Terry Heo³
Guenther Schmuelling¹¹ Maryam Shabani⁴ Dylan Zika³

ABSTRACT

This paper presents the first industry-standard open-source machine learning (ML) benchmark to allow performance and accuracy evaluation of mobile devices with different AI chips and software stacks. The benchmark draws from the expertise of leading mobile-SoC vendors, ML-framework providers, and model producers. It comprises a suite of models that operate with standard data sets, quality metrics and run rules. We describe the design and implementation of this domain-specific ML benchmark. The current benchmark version comes as a mobile app for different computer vision and natural language processing tasks. The benchmark also supports non-smartphone devices, such as laptops and mobile PCs. Benchmark results from the first two rounds reveal the overwhelming complexity of the underlying mobile ML system stack, emphasizing the need for transparency in mobile ML performance analysis. The results also show that the strides being made all through the ML stack improve performance. Within six months, offline throughput improved by 3 \times , while latency reduced by as much as 12 \times . ML is an evolving field with changing use cases, models, data sets and quality targets. MLPerf Mobile will evolve and serve as an open-source community framework to guide research and innovation for mobile AI.

1 INTRODUCTION

Mobile artificial intelligence (AI) applications are increasingly important as AI technology becomes a critical differentiator in smartphones, laptops, and other mobile devices. Consequently, laptops and smartphones have incorporated application-specific integrated circuits (ASICs) on the hardware front to support AI in an energy-efficient manner. The software front includes many code paths and AI infrastructures to support machine-learning hardware efficiently.

While support for mobile AI applications is becoming a differentiating capability, seeing through the mist of competing solutions is difficult for fairly evaluating improvements in performance and efficiency. Figure 1 shows the number of different code pathways for generating results on mobile SoCs. The dashed lines represent mere possibilities, whereas the solid lines indicate actual submissions from various organizations. Different code paths yield different

performance results. Therefore, benchmark-performance transparency is essential, as it reveals which code paths were taken, making the performance results reproducible and informative for consumers. OEMs, SoC vendors, researchers, and consumers can all benefit when mobile devices employ AI in ways they can compare transparently and fairly.

However, a key challenge for developing a robust mobile AI benchmark is, first and foremost, understanding the complex landscape of the mobile computing ecosystem. The end-user performance of a mobile AI device is more than its AI hardware capability in isolation. Instead, a more accurate measurement results from the AI hardware coupled with its ML-software framework, whose net performance is shrouded beneath layers of developer options, deployment scenarios, and the OEMs' lifecycles. This complexity is not discussed nor relevant for major server-side inference benchmarks (Gao et al., 2019; Reddi et al., 2020). As such, there is a need for a domain-specific benchmark that can critically compare and evaluate systems with mobile-specific models, numerics, frameworks, metrics, and methodology.

To address the challenge, we take an open-source, community-driven approach. A consortium of mobile vendors and academic organizations with shared interests, yielding collective expertise in mobile neural-network models, data sets, and submission rules, have developed the MLPerf

¹Harvard University ²MLCommons ³Google ⁴Intel
⁵MediaTek ⁶Qualcomm ⁷Samsung ⁸Cruise ⁹ENS Paris-Saclay ¹⁰ARM ¹¹Microsoft. Correspondence to: Vijay Janapa Reddi <vj@eecs.harvard.edu>, David Kanter <david@mlcommons.org>.

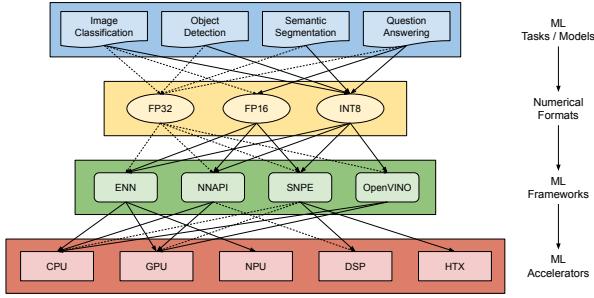


Figure 1. There are many ways to exercise a mobile SoC’s rich suite of accelerators, which is why transparency is key.

Mobile benchmark to ensure the results are relevant to the industry and academia (MLCommons, 2020a), and beneficial to consumers while being transparent and reproducible. The following five principles inform the benchmark’s design:

1. The benchmark must capture the **real-world mobile system complexity** involved in delivering AI performance to users who procure a commercial device.
2. The benchmark must **identify mobile-specific models and represent diverse tasks** that are challenging and resist model- and domain-specific optimizations.
3. Each task should have an **appropriate accuracy and minimum quality threshold** that matches and reflects the metrics that matter for mobile AI device end-users.
4. The **testing conditions must closely match the environments in which mobile devices typically serve**, such as ambient temperature and battery power.
5. Performance results must be publicly reproducible outside the submitting organization as commercial mobile devices are globally accessible and to foster generational ML advancements on prior achievements.

Our approach to addressing principles 1–5 was to develop an industry-neutral open-source MLPerf Mobile app that all benchmark tests must use. The initial version has a set of four mobile-specific neural network models for three computer vision (CV) tasks and one natural language processing (NLP) task, each with its own accuracy and minimum quality targets. The app passes these models to the back-end layer, which is an abstraction that allows different hardware (and software) vendors to optimize their implementations for neural networks. The app also has a presentation layer for wrapping the more technical benchmark layers and the Load Generator (“LoadGen”). The LoadGen allows representative testing of different inference platforms and use cases by generating inference requests in a pattern and measuring specific parameters (e.g., latency, throughput, or latency-bounded throughput). The benchmark also offers a headless version of the mobile application that enables laptops running non-mobile OSs to use the same benchmarks.

Two rounds of MLPerf Mobile submissions have been completed (MLCommons, 2021b). Comparing the results between these two generations reveals these key takeaways:

- *Benchmarking drives generational improvements.* Between two versions of the benchmark (six months), latency improved by 2× on average and by 12× in one case. Developing principled methods to measure mobile ML performance is important to drive innovation.
- *There is no one size fits all.* The results show a range of hardware and software approaches to implement neural network models efficiently on mobile devices. The approach needs to be driven by the app use case.
- *Accelerator level parallelism (ALP) is important.* Vendors exercise multiple hardware accelerators concurrently to maximize offline throughput performance. Therefore, there is a need for managing hardware ALP.
- *State-of-the-art should compare against vendor-backends.* Mobile AI accelerators often rely on vendor-specific SDKs and custom backends to unleash their full potential as more general-purpose frameworks like NNAPI can lead to 10% slower performance, or worse, be 7× slower due to buggy support (Buch et al., 2021b).
- *Numerics (still) matter.* Not all mobile ML tasks benefit from INT8 quantization. Tasks like NLP require FP16 arithmetic to be useful in real deployments, implying not everything needs a dedicated AI accelerator.

Ideally, researchers would track MLPerf Mobile’s benchmark tasks, accuracy metrics, quality thresholds, rules, etc., to present industry-relevant evaluations that practitioners can adopt to bridge the gap between research and practice. As the mobile AI landscape is vastly different from desktop and cloud AI deployments, our open-source mobile app can be a common baseline for integrating various ML frameworks and models, facilitating “out of the box” research needed for reproducibility on real devices and simulators. Supported by MLCommons (MLCommons, 2020b), MLPerf Mobile will continue to evolve and stay up-to-date.

2 MOBILE AI ECOSYSTEM CHALLENGES

Mobile AI performance is shrouded behind multiple layers of complexity. We describe these important factors that significantly impact mobile AI performance in the real world: hardware heterogeneity, software fragmentation, developer options, deployment scenarios, and OEM life cycles. Each by itself leads to performance variability, but the combination makes AI benchmarking extremely challenging.

2.1 Hardware Heterogeneity

A given device may have a spectrum of AI-performance capabilities, depending on which processing engines it uses.

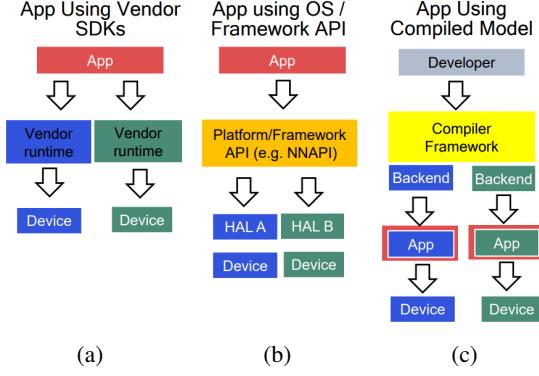


Figure 2. Application-development options.

Smartphones contain complex heterogeneous chipsets that provide many different compute units and accelerators. A typical mobile system-on-a-chip (SoC) complex includes a CPU cluster, GPU, DSP, neural processing unit (NPU), Hexagon Tensor Accelerator (HTA), Hexagon Vector Extensions (HVX), and so on. Any or all of these components can aid in machine-learning (ML) inference. Moreover, many smartphones today are Arm-based, but the CPU cores generally implement a heterogeneous “big.LITTLE” architecture (Arm, 2011). Some SoCs even have big-CPU clusters where some CPUs clock faster than others. Also, devices fall into different tiers with different hardware capabilities at different prices, varying in their memory capacity and storage features. Any processing engine can run ML workloads, but this flexibility also makes benchmarking AI performance difficult. Hence, there is a need for a transparent way to benchmark a smartphone’s AI-hardware performance.

2.2 Software Fragmentation

The mobile software ecosystem is heavily differentiated from the OS to the run-time ML framework. The diversity of different software code paths can drastically affect hardware performance. Hence, a transparent mechanism for operating, introspecting, and evaluating a mobile device is essential.

Mobile devices employ various OSs: Android, iOS, Windows, Ubuntu, Yocto, etc. Each OS has an ecosystem of ML application programming interfaces (APIs) and application-deployment options that necessitate particular solutions. Numerous APIs have served in the development of ML applications. A single SoC or OEM device will often have to support a vendor SDK and/or a plurality of frameworks.

SoC vendors offer a proprietary software development kit (SDK) that generates optimized binaries so ML models can run on SoC-specific hardware. These vendors also make engineering investments to support more generic frameworks, such as TensorFlow Lite (TFLite) (Google, 2019) and NNAPI (Google, 2020), that provide a compatibility layer to support various accelerators and device types. But as resources are limited, SoC vendors prioritize their SDKs,

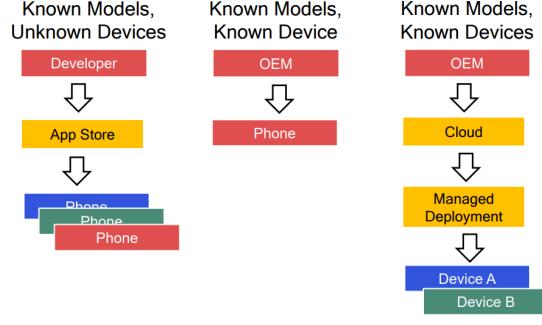


Figure 3. ML-application deployment scenarios.

resulting in less-optimal generic-framework support.

The diversity of vendor SDKs and framework-support levels are all reasons why the mobile-ML software ecosystem is fragmented. This situation complicates hardware-performance assessment because the choice of software framework has a substantial effect. A high-performance SoC, for instance, may deliver low performance, owing to an ill-matched framework. For example, even if a SoC integrates a high-performance ML accelerator, if a generic Android framework like NNAPI does not support it with high-performance driver back ends, the accelerator will function poorly when handling a network (Buch et al., 2021b).

2.3 Developer Options

The ecosystem allows application developers to choose among several different approaches to enable machine learning on mobile devices, making it necessary to have an open-source methodology for understanding performance. Application developers can work through a marketplace such as Google Play (Google, 2012) to create mobile-app variants for every SoC vendor if they follow a vendor-SDK approach (Figure 2a). However, doing so presents a scalability challenge because of the increased time to market and additional development costs. An alternative is to create an application using a native OS/framework API such as NNAPI, which provides a more scalable approach (Figure 2b). Nevertheless, this alternative has a crucial shortcoming: it is only viable if SoC vendors provide good back-end drivers to the framework, necessitating cooperation between them and the framework designers. A final alternative is to bind the neural-network model to the hardware. Doing so allows compilation of the model to a particular device, avoiding reliance on any particular run time (Figure 2c), but this lacks device portability which is needed for mobile computing.

2.4 Deployment Scenarios

Mobile ML applications have many potential uses. Details of the use case determine the extent to which an ML model is optimized for the hardware and how it runs, because of

strong or weak ties to the device. Developers primarily build applications without specific ties to vendor implementations. They may design custom neural-network models that can run on any device. Thus, mobile devices often run apps that employ unknown models for a variety of hardware (Figure 3(a)). OEMs, on the other hand, build their ML applications for their own devices. Therefore, both the models and the device targets are known at deployment time (Figure 3(b)). A service provider (e.g., Verizon or AT&T) that uses a variety of hardware solutions may, however, support its service with known models, in which case both the models and the hardware are known (Figure 3(c)). Development of the applications deployed in these scenarios may also take place in various ways. OEMs that manufacture devices can use vendor SDKs to support their applications with minimal extra effort. Given these options, it is necessary to know which of these underlying approaches is being used to produce the measured AI performance results.

2.5 OEM Lifecycle

A variety of other factors, ranging from how OEMs package software for delivery to how software updates are issued, also affect hardware-performance measurements. OEMs employ vendor SoCs and associated software releases to produce commercial mobile devices. OEMs pick up the software updates (such as framework enhancements) from the SoC vendors and bundle them with other updates for periodic release. Usually, a delay occurs between the time when an SoC vendor releases a software update and when that performance update sees deployment. The delay is months long. Moreover, commercial devices receive OEM updates only for a fixed period, so they will not benefit from software-performance enhancements afterward. Thus, getting reproducible numbers is difficult without transparency.

3 MLPERF MOBILE BENCHMARKS

To tackle the challenges, we developed MLPerf Mobile ([ML-Commons, 2020a](#)). A key aspect of our work is the methodology more so than the specifics of a benchmark version.

3.1 Benchmark Design Philosophy

The ML landscape is evolving and there are a plethora of models in the wild. Mobile ML systems include a wildly wide range of use cases, ranging from light-weight (e.g., classification on small 300x300 pixel images) to heavy-weight (e.g., 50 MP camera on Xiaomi for 5 \times zoom) tasks. Grappling with the rich diversity of tasks and models requires a long-term perspective focused on building a robust benchmark. In the initial version, we intentionally selected a few machine-learning tasks that represent light- and mid-weight mobile use cases that are stable, rather than picking models that are still evolving where there is no agreement

on which mobile ML model versions are broadly applicable. We chose networks for a small set of tasks based on their maturity and applicability to different hardware (CPUs, GPUs, DSPs, NPUs, etc.). As we show later in the evaluation section, benchmarking these initial models still yields helpful insights about hardware performance across various deployment scenarios. We discuss and present our plans to extend the benchmark suite tasks over time in Appendix E.

3.2 ML Tasks and Models

Image classification. We selected MobileNetEdgeTPU ([Howard & Gupta, 2019](#)), a well-optimized mobile model that usually provides good performance on different SoCs. MobileNetEdgeTPU is a descendent of the MobileNet-v2 family optimized for low-latency and mobile accelerators. The architecture is based on convolutional layers with inverted residuals and linear bottlenecks, similar to MobileNet v2. Still, it is optimized by introducing fused inverted bottleneck convolutions to improve hardware utilization and by removing hard-swish and squeeze-and-excite blocks. Evaluation of the MobileNetEdgeTPU reference model employs the ImageNet 2012 validation data set ([Russakovsky et al., 2015](#)) and requires 74.66% (98% of FP32 accuracy) Top-1 accuracy. Before inference, images are resized, cropped to 224x224, and then normalized.

Object detection. Our v0.7 reference model is the Single Shot Detector (SSD) ([Liu et al., 2016](#)) with a MobileNet v2 backbone ([Sandler et al., 2019](#))—a choice that is well adapted to constrained computing environments. SSD-MobileNet v2 uses MobileNet v2 for feature extraction and uses a mobile-friendly SSD variant called SSDLite ([Sandler et al., 2019](#)) for detection. SSD prediction layers replace all the regular convolutions with separable convolutions (depthwise followed by 1x1 projection). SSD-MobileNet v2 reduces latency by decreasing the number of operations; it also reduces the memory that inference requires by never fully materializing the large intermediate tensors. Two SSD-MobileNet v2 versions acted as the reference models for the object-detection benchmark, one model replacing more of the regular SSD-layer convolutions with depth-separable convolutions than the other does. We used the COCO 2017 validation data set ([Lin et al., 2015](#)) and, for the quality metric, the mean average precision (mAP). The target accuracy is an mAP value of 22.7 (93% of FP32 accuracy). The preprocessing stage resizes the image to 300x300—typical of resolutions in smartphones—and then does normalization.

In v1.0, we updated the reference model to MobileDets ([Xiong et al., 2021](#)) with the SSDLite that is more geared toward stressing mobile hardware accelerators such as mobile CPUs, GPUs, EdgeTPUs and DSP. A key feature of MobileDets is that in addition to using inverted bottlenecks as the only building block, it injects regular

Version	Area	Task	Reference Model	Data Set	Quality Target
v0.7, v1.0	Vision	Image classification	MobileNetEdgeTPU (4M params)	ImageNet 2012 (224x224)	98% of FP32 (76.19% Top-1)
v0.7, —	Vision	Object detection	SSD-MobileNet v2 (17M params)	COCO 2017 (300x300)	93% of FP32 (24.4% mAP)
—, v1.0	Vision	Object detection	MobileDET-SSD (4M params)	COCO 2017 (320x320)	95% of FP32 (28.5% mAP)
v0.7, v1.0	Vision	Semantic segmentation	DeepLab v3+ (2M params)	ADE20K (512x512)	97% of FP32 (54.8% mIoU)
v0.7, v1.0	Language	Question answering	MobileBERT (25M params)	Mini Squad v1.1 dev	93% of FP32 (93.98% F1)

Table 1. MLPerf Mobile benchmark suite. In the second version (v1.0) of the benchmark, we updated the object detection model to be more representative of industry needs and this is also reflected in the more stringent quality target requirements.

convolution operations into the neural network. Regular convolutions help improve the accuracy-latency trade-off on several hardware accelerators when placed at the appropriate positions in the network. We continue to use COCO 2017 validation set for testing with mAP as the quality metric. The image input size is different in MobileDets. It increases the input image resolution from 300x300 to 320x320. MobileDets has fewer parameters than MobileNets v2, but the high image resolution demands increased computation.

Semantic image segmentation. Our reference model for this task in the first version is DeepLab v3+ (Chen et al., 2018) with a MobileNet v2 backbone. DeepLab v3+ originates from the family of semantic image-segmentation models that use fully convolutional neural networks to directly predict pixel classification (Long et al., 2015; Eigen & Fergus, 2015) as well as to achieve state-of-the-art performance by overcoming reduced-feature-resolution problems and incorporating multiscale context. It uses an encoder/decoder architecture with atrous spatial pyramid pooling and a modular feature extractor. We selected MobileNet v2 as the feature extractor because it enables state-of-the-art model accuracy in a constrained computational budget. We chose the ADE20K validation data set (Zhou et al., 2017) for its realistic scenarios, cropped and scaled images to 512x512, and (naturally) settled on the mean intersection over union (mIoU) for our metric. Additionally, we trained the model to predict just 32 classes (compared with 150 in the original ADE20K data set); the 1st to the 31st are the most frequent (pixel-wise) classes in ADE20K, and the 32nd represents all the other classes. The mIoU depends on the pixels whose ground-truth label belongs to one of the 31 most frequent classes, boosting its accuracy by discarding the network’s bad performance on low-frequency classes.

Question answering. We selected MobileBERT (Sun et al., 2020), a BERT model that is well suited to resource-limited mobile devices. Further motivating this choice is the model’s state-of-the-art performance and task-agnostic nature: even though we consider question answering, MobileBERT is adaptable to other NLP tasks with only minimal fine-tuning. We trained the model with a maximum sequence length of 384 and use the F1 score for our metric. This task employs the Stanford Question Answering Dataset (Squad) v1.1 Dev (Rajpurkar et al., 2016). Given a question and a passage from a Wikipedia article, the model must extract a text segment from the passage to answer the question.

Others. Our current network choices reflect common use-cases. Most mobile ML use cases involve computer vision. But building on our design philosophy (Section 3.1), Appendix E discusses our plans to extend the benchmark. For example, a mobile version of RNN-T for speech is in the works and we will continue to add new models in the future.

3.3 Reference Code

We provide reference-code implementations in the TensorFlow and TensorFlow Lite (TFLite) formats. We choose TF because it is vendor-neutral and most vendor backends can easily import TF code and models into their internal optimization frameworks. The reference code’s goal is to identify the critical model-invocation stages. For instance, the reference benchmarks implement the preprocessing steps and the model’s input-generation procedure. Benchmark submitters may readily adopt the code for their submission. Or they may choose to optimize these stages (e.g., rewrite them in C instead of Python) for performance—as long as they employ all the same stages and take the same steps to maintain equivalence. All reference models have 32-bit floating-point weights, and the benchmark additionally includes an 8-bit quantized version (with either post-training quantization or quantization-aware training, depending on the tasks). The reference implementations are available as open-source and free for users to download (MLPerf, 2019).

The reference implementation is poorly optimized. Vendors that submit results to MLPerf must inherit the reference code, adapt it, and produce optimized glue code that performs well on their hardware. For example, to handle (quantized) inference, they need to invoke the correct software back end (e.g., SNPE or ENN) or an NNAPI driver to schedule code for their SoC’s custom hardware accelerators.

4 LOAD GENERATOR

This section describes how we generate load onto the system under test (SUT) and measure inference performance.

4.1 LoadGen

To enable testing of various inference platforms and use cases, we devised the Load Generator (“LoadGen”) (ML-Commons, 2019), which creates inference requests in a pattern and measures some parameters (e.g., latency, through-

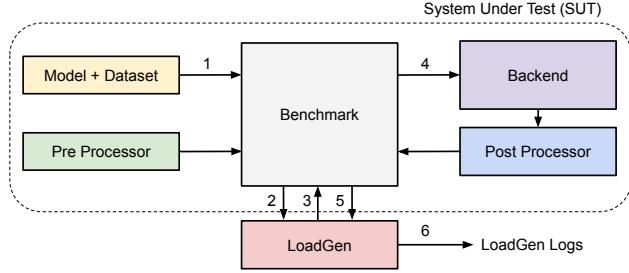


Figure 4. Load Generator (“LoadGen”) testing the SUT.

put, or latency-bounded throughput). It logs information about the system during execution to enable post-run validation. Submitter modification of the LoadGen software is forbidden to ensure the testing behavior remains intact.

As Figure 4 shows, the LoadGen uses the data sets as inputs to the SUT. It feeds the entire data set to the SUT to verify that the model delivers the required accuracy in accuracy mode. It provides only a subset of images to the SUT to measure steady-state performance in performance mode. A seed and random-number generator allows the LoadGen to select samples from the data set for inference, precluding unrealistic data-set-specific optimizations.

For pre-processing, the typical image-preprocessing tasks—such as resizing, cropping, and normalization—depend on the ML model. This stage implements data-set-specific preprocessing that varies by task, but all submitters must follow the same steps. Post-processing is a data-set-specific task that covers all the ops needed for accuracy calculations. For example, computing the Top- N results for an image classifier requires a Top-K op / layer after the softmax layer.

The backend for the reference benchmark implementation (Section 3.3) is a TFLite smartphone back end that optionally includes NNAPI and GPU delegates. We also provide a “dummy” back end as an example reference for proprietary back ends; submitters replace it with whatever corresponds to their system. For instance, Qualcomm would replace the dummy with SNPE, and Samsung would replace it with ENN. The back end corresponds to other frameworks such as OpenVINO for laptops and similar large mobile devices.

4.2 Execution Scenarios

The LoadGen provides two execution modes: *single stream* and *offline*. They reflect the operating behavior of many mobile applications in the real world. In the single-stream scenario, the application sends a lone inference query to the SUT with a sample size of one. That size is typical of smartphones and other interactive devices where, for example, the user takes a picture and expects a timely response. The LoadGen injects a query into the SUT and waits for its completion. It then records the inference run length and sends the next query. This process repeats until the LoadGen has

issued all the samples (1,024) in the task’s corresponding data set or a minimum run time of 60 seconds has passed.

In the offline scenario, the LoadGen sends all the samples to the SUT in one burst. Although the query sample size remains one, as in the single-stream scenario, the number of samples in the query is much larger. Offline mode issues 24,576 samples—enough for sufficient run time. This choice reflects applications that require multi-image processing, simultaneous processing of batched input, or concurrent application of models such as image classification and person detection to photos in an album. The implementation is usually a batched query with a batch size larger than one.

These metrics are based on best practices derived from industry feedback and open (sometimes grudging) consensus.

4.3 System Under Test

We designed the LoadGen to take advantage of any mobile device type (laptop, tablets, or smartphones). Smartphones can use the reference MLPerf Android app that supports TFLite delegates and NNAPI delegates. The app queries the LoadGen, which then queries input samples for the task, loads them to memory, and tracks the time required to execute the task. For laptops, submitters can build a native command-line application. The LoadGen integrates this application, and it supports back ends such as the OpenVINO run time. The application generates logs consistent with MLPerf rules, validated by the submission checker. The number of samples necessary for performance mode and for accuracy mode remains identical to the number in the smartphone scenario. The only difference is the absence of a graphical user interface for the laptop devices.

5 MODEL OPTIMIZATIONS

We describe the process to produce high-performance results for submission. In practice, MLPerf Mobile is a competition to produce the most competitive system performance.

5.1 Numerics

A submitter may implement minimal changes to the model if they are mathematically equivalent or approved approximations to make the model compatible with their hardware. The rules prohibit altering the AI models to reduce their computational complexity; banned techniques include channel pruning, filter pruning, and weight skipping. However, some amount of quantization techniques are permissible.

The reference models are frozen TensorFlow FP32 checkpoints, and valid submissions must begin from these frozen graphs. Submitters can export a reference FP32 TFLite model. They can generate fixed-point models with INT8 precision from the reference FP32 models using post-training

quantization (PTQ), but they cannot perform quantization-aware training (QAT). Network retraining alters the neural-network architecture, so model equivalence is difficult to verify. Also, retraining allows the submitters to use their training capabilities (e.g., neural architecture search) to boost inference throughput, changing the benchmark’s nature.

Depending on submitter needs, however, we provide QAT versions of the model. All participants mutually agree on these QAT models as being comparable to the PTQ models. In general, QAT reduces accuracy loss relative to PTQ. Therefore, we chose the minimum-accuracy thresholds (“Quality Target” in Table 1) on the basis of what is achievable through post-training quantization without any training data. For some benchmarks, we generated a reference INT8 QAT model using the TensorFlow quantization tools; submitters can employ it directly in the benchmark.

Some hardware cannot deploy TensorFlow-quantized models directly, and organizations may need different fixed-point formats to match their hardware. In such cases, we only allow post-training quantization without training data from a reference model. For each model, we specify a calibration data set (typically 500 samples or images from the training or validation data set) for calibration in the PTQ process. Submitters can only use the approved calibration data set.

5.2 Vendor Optimized Backends / SDKs

Most chipset organizations rely on optimized software backends (or libraries) to extract the best performance from their SoCs, which often support many different hardware accelerators (CPUs, GPUs, NPUs, DSPs, etc.). Smartphone companies that use proprietary vendor backends or delegates implement their back-end interface to the reference MLPerf app (Figure 5, Section 5.2). To run the neural-network models, such back ends query the correct library (TensorFlow, TFLite, the Exynos Neural Network SDK, or the SNPE SDK). For laptops, submitters can build a native command-line application. The LoadGen integrates this application, and it supports back ends such as the OpenVINO run time. The application generates logs consistent with MLPerf rules, validated by the submission checker. The number of samples necessary for performance mode and for accuracy mode remains identical to the number in the smartphone scenario. The only difference is the absence of a graphical user interface for laptop-based devices.

Figure 5 shows how we support all this flexibility. The reference TensorFlow models are at the root of the entire process, which follows one of three paths. Code path ① allows submitters to optimize the reference TensorFlow models for implementation through a proprietary back end (e.g., SNPE for Qualcomm or ENN for Samsung), then schedule and deploy the networks on the hardware. Code path ② allows

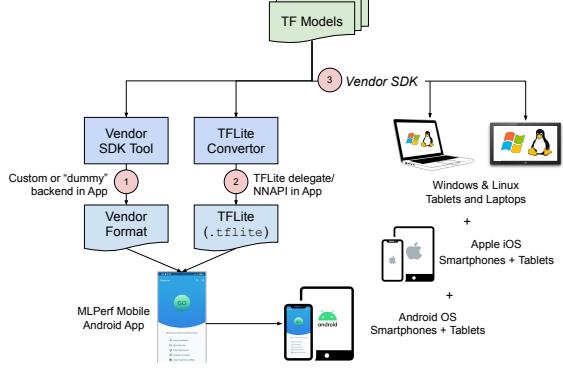


Figure 5. Benchmark supports many devices and code paths.

submitters to convert the reference TensorFlow models to a mobile-friendly format using an exporter. These models are then easy to deploy on the device, along with appropriate quantizations, using the TFLite delegates to access the AI-processing hardware. Code path ③ allows non-smartphone submitters to run the reference TensorFlow models through non-mobile back ends (e.g., OpenVINO) on laptops and tablets with operating systems such as Windows and Linux.

6 RUN RULES

We developed a strict set of run rules that allow us to reproduce submitted results through an independent third party. All of the MLPerf Mobile run rules and conditions are based on best practices derived from industry input and feedback.

6.1 Reproducibility Guidelines

Test control. The mobile app runs the models in a specific order. For each one, the model runs on the validation set to calculate the accuracy. Performance mode follows. Single-stream mode measures the 90th-percentile latency over at least 1,024 samples for a minimum run time of 60 seconds to achieve a stable performance result. Offline mode reports the average throughput necessary to process 24,576 samples; in current systems, the run time exceeds 60 seconds. These values were based on input from the member organizations.

Thermal throttling. ML models are computationally heavy and they can trigger run-time thermal throttling to cool the SoC. We recommend maintaining an air gap with proper ventilation and avoid flush contact with any surfaces. Also, we require room-temperature between 20 and 25 °C.

Coldown interval. The app provides a break setting of 0–5 minutes between the individual tests to allow the phone to reach its cooldown state before starting each one. If the suite is to run multiple times, we recommend a 10-min break.

Battery power. The benchmark runs while the phone is battery powered, but we recommend a full charge beforehand to avoid entering power-saving mode.

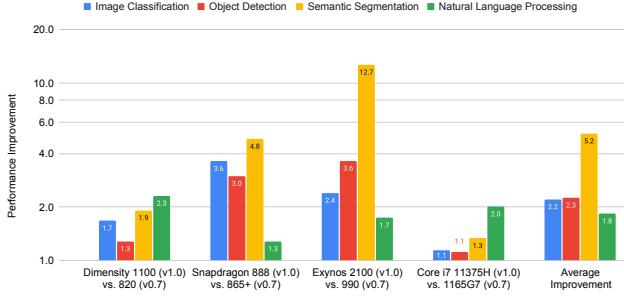


Figure 6. We see a 2× improvement between v0.7 and v1.0.

6.2 Result Validation

We require that the SUT be commercially available before publication, enabling a more tightly controlled validation, review, and audit process. The SUT includes both the hardware and software components. Submissions include all of the mobile benchmark app’s log files, unedited. Post submission, all of the results are independently audited, along with any modified models and code used in the respective submissions. The vendor backend (but not the toolchain) is included. We also evaluate any private vendor SDKs to allow auditing of the model conversion process. The audit process comprises an examination of log files, models, and code for compliance with the submission rules and verification of their validity. It includes verification of the system’s reported accuracy and latencies. To verify results, we build the vendor-specific app, install it on the device (in the factory-reset state), and reproduce the latency and/or throughput numbers, along with accuracy. The results are valid if our numbers are within 5% of the submitted scores.

7 INSIGHTS FROM BENCHMARK RESULTS

To understand what is to be gained from benchmarking, we dissect the first two rounds of MLPerf Mobile submissions. We compare version 0.7 of the results (MLCommons, 2020c) to version 1.0 (MLCommons, 2021b). SoC manufacturers submitted the results using the app (see Appendix A). In addition to the insights we present here, there are various other use cases of the benchmark (see Appendix B).

7.1 Insight 1: Benchmarking Leads to Improvements

Over about six months, the submitting organizations had new offerings with improved ML capabilities, both in terms of hardware and software. Results were collected on end-user consumer devices that incorporate the SoC chipsets. Figure 6 compares the improvement in latency for each of the ML tasks across the two generations. Results are grouped by SoCs. The figure also shows the average gain for each of the ML tasks. The performance of the ML tasks improved by ~2×. The performance improvement resulted from all smartphones SoCs advancing to a new generation.

The specific hardware improvements vary across each of the SoC families. We summarize the main reasons here and provide additional system improvement details in Appendix C.

The Samsung Exynos 2100 outperforms the 990 by 12.7× on the segmentation task. This is cause the hardware improved by more than 2×. But the software also played a crucial role—its uplift was 6×. Exynos 2100 has critical features that reduce data transfer between IP blocks, which are enabled in software through improved scheduling. The improved Qualcomm Snapdragon 888’s new Hexagon 780 can perform 26 TOPS (73% faster than 865+) along with a resigned DSP microarchitecture. The new MediaTek’s Dimensity originally had a single core MediaTek Deep Learning Accelerator (MDLA), while the Dimensity 1100 has dual MDLA cores. Associated with these changes are the software drivers. The Dimensity 1100 uses the Neuron Delegate to replace the NNAPI Delegate, when it is possible, as NNAPI has synchronization overhead due to the intermediate hardware abstraction layer (Figure 2b), and also discussed in Table 3.

On the laptop front, improvements primarily came from software enhancements and minimal hardware changes. In terms of CPU frequency, Intel’s Core i7-11375H (Intel, 2020) is 1.1× better than i7-1165G7 (Vera, 2020); in terms of GPU frequency, i7-11375H is about 1.04× better compared to i7-1165G7. For image classification and object detection, the benchmark runs on CPU. Hence, the improvements are from an increase in CPU frequency. Segmentation and NLP models need more TOPs compared to classification and detection. For this reason, Segmentation and NLP models work best on an integrated GPU (iGPU). Though there is a slight improvement in iGPU performance (4% improvement), we see a large improvement in NLP performance and a marginal increase in segmentation performance. NLP improvement is due to the OpenVINO quantized kernel.

7.2 Insight 2: No One Size Fits All Tasks & Models

No one solution dominates all the benchmarks at the overall benchmark-level and task-specific level. Figure 7 plots the single-stream results for the three smartphone chipsets on each benchmark task from the v0.7 version; the same general trend holds true for the v1.0 version. The figure shows throughput and latency results. Each chipset offers a unique differentiable value. MediaTek’s Dimensity scored the highest in object-detection and image-segmentation throughput. Samsung’s Exynos performed well on image classification and NLP, where it achieved the highest scores. Qualcomm’s Snapdragon is competitive for image segmentation and NLP.

The image-classification task also employs an offline mode for batch processing; here, Exynos delivered 674.4 frames per second (FPS) and Snapdragon delivered 605.37 FPS. These data points are not shown in Figure 7. Also, not all submitters are required to submit to this offline scenario.

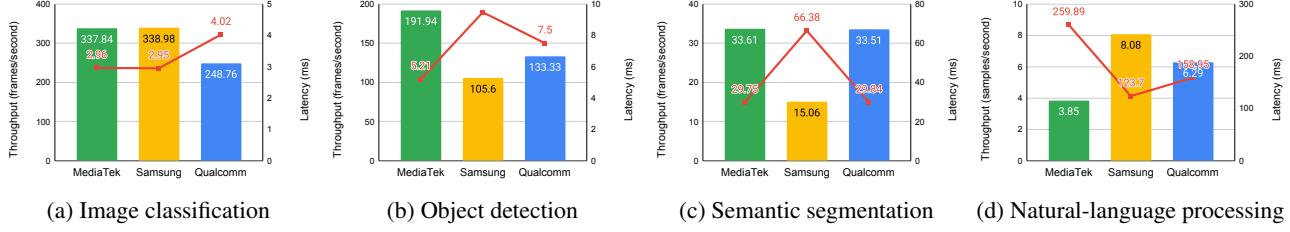


Figure 7. Results from the first round (v0.7) (MLCommons, 2020c). We observe similar trends in v1.0 submission round.

Table 2 shows the details behind Figure 7. Because MLPerf Mobile emphasizes transparency, the table includes specifics for how a benchmark is executed in the single-stream mode and the offline mode. The table shows the numerics (top of cell), framework (middle of cell), and accelerator (bottom of cell) used to produce the results. The table shows that various hardware combinations are used to achieve good mobile AI performance. No one hardware unit dominates all ML tasks. In all cases, the CPU is the backbone (not included in the table) that is orchestrating the overall execution—including doing pre- and post-processing and other tasks the benchmark does not measure. In contrast, as shown in the last row of each cell, the GPU, DSPs, NPUs, and AIPs are focused on delivering high-performance AI execution.

7.3 Insight 3: Accelerator Level Parallelism is Here

The results generally highlight another important point that to deliver the best performance vendors rely on multiple accelerators concurrently, more recently referred to as accelerator level parallelism (ALP) (Hill & Reddi, 2019). Relying on multiple accelerators allows the ML compiler framework to intelligently schedule portions of the neural network graph operations to different execution engines that best match the operation’s needs. Table 2 includes details for the image classification offline mode. In offline mode, the focal point is the throughput of the system (not latency).

Across the offline mode results (second column) in Table 2, multiple accelerators are used concurrently to maximize performance. For example, Exynos uses both the NPU and the CPU together. Similarly, on the Snapdragon chipsets, the Hexagon Tensor Accelerator (HTA) and Hexagon Vector Extensions (HVX) are used concurrently as part of the AI processing (AIP) cluster. On the core i7 SoC, both the CPU and the integrated GPU are used simultaneously. It is uncommon to exercise ALP in the latency-bounded single-stream scenario because the overhead of managing multiple concurrent accelerators can quickly become a bottleneck.

7.4 Insight 4: ML Frameworks Play a Crucial Role

Given the heterogeneity of the ecosystem, the NNAPI runtime module is a library sitting between an app and its backend drivers. It is designed to be a common baseline

for ML on Android devices and to distribute that workload across ML-processor units, such as CPUs, GPUs, DSPs, and NPUs. But nearly all submissions in Table 2 make use of proprietary frameworks (Figure 5). Vendor SDKs, such as ENN and SNPE, give SoC vendors control over their SoC’s performance. They can control which processor unit or accelerator to use for AI and what optimizations to apply.

All laptop submissions employ INT8 and achieve the desired accuracy on vision and language models. For single-stream mode, because just one sample is available per query, some models are incapable of fully utilizing the GPU’s computational resources. Therefore, the back end must choose between the CPU and GPU to deliver the best overall performance. For example, small models such as MobileNetEdgeTPU use the CPU. For offline mode, multiple samples are available as a single query, so inference employs both the CPU and GPU. This level of detailed control is strongly tied to the system’s capabilities and only the hardware vendors are well-suited for this sort of fine-grained optimization.

Table 3 shows the performance difference for v1.0 tasks between using the generic NNAPI delegate and the optimized Neuron Delegate. The latter has full support for MediaTek’s Dimensity 1100. Using the optimized framework driver can lead to over 10% difference in performance. This is because the standard NNAPI driver does not yet fully support multi-MDLA, which we discussed previously in Section 7.1.

7.5 Insight 5: Numerics Still Matter for Some Tasks

Quantization is crucial for mobile deployments because quantized inference runs faster and provides better performance and memory bandwidth than FP32 (Han et al., 2015). The accuracy tradeoff for quantized models (especially since no retraining is allowed in the benchmark) is tolerable in smartphones, which seldom perform safety-critical tasks.

However, it is not always about INT8. Mobile-device designers prefer both INT8 and FP16 (Table 2). All the mobile-vision tasks employ INT8 heavily. Most vendors rely on this format because it enables greater performance and consumes less power, preserving device battery life. NLP favors FP16, which requires more power than INT8 but offers better accuracy. More importantly, submitters use FP16 because most AI engines today lack efficient support for non vision tasks.

	Image Classification (single-stream)	Image Classification (offline)	Object Detection (single-stream)	Image Segmentation (single-stream)	Natural-Language Processing (single-stream)
	<i>ImageNet</i>	<i>ImageNet</i>	<i>COCO</i>	<i>ADE20K</i>	<i>Squad</i>
	<i>MobileNetEdge</i>	<i>MobileNetEdge</i>	<i>SSD-MobileNet v2</i>	<i>DeepLab v3+ - MobileNet v2</i>	<i>MobileBERT</i>
<i>MediaTek Dimensity 820 (smartphone)</i>	UINT8, NNAPI (neuron-ann), APU	Not applicable	UINT8, NNAPI (neuron-ann), APU	UINT8, NNAPI (neuron-ann), APU	FP16, TFLite delegate, Mali-GPU
<i>Samsung Exynos 990 (smartphone)</i>	INT8, ENN, NPU+CPU	INT8, ENN, NPU+CPU	INT8, ENN, NPU+CPU	INT8, ENN, NPU+GPU	FP16, ENN, GPU
<i>Qualcomm Snapdragon 865+ (smartphone)</i>	UINT8, SNPE, HTA	UINT8, SNPE, AIP (HTA+HVX)	UINT8, SNPE, HTA	UINT8, SNPE, HTA	FP16, TFLite delegate, GPU
<i>Intel Core i7-1165G7 (laptop)</i>	INT8, OpenVINO, CPU	INT8, OpenVINO, CPU+GPU	INT8, OpenVINO, CPU	INT8, OpenVINO, GPU	INT8, OpenVINO, GPU

Table 2. Myriad combinations of numerics, software run times, and hardware, reinforcing the need for transparency.

<i>MediaTek Dimensity 1100</i>	Image Classification	Object Detection	Image Segmentation
NNAPI Delegate	2.48 ms	5.05 ms	20.56 ms
Neuron Delegate	2.23 ms	4.77 ms	20.02 ms
% Improvement	10.08%	5.54%	2.70%

Table 3. Vendor-optimized delegates tend to perform better.

8 RELATED WORK

Compared to existing mobile AI benchmarks, MLPerf Mobile is different in the following ways, all of which we believe are *requirements* to drive the industry forward:

1. **need a system-level machine learning benchmark, designed to exercise the complete mobile system,** compared to micro-benchmarking small pieces.
2. **put accuracy first and measure performance with respect to a minimum quality target,** based on community consensus, rather than measuring performance for unchecked and arbitrary model quality targets.
3. **be an open-source benchmark that provides transparency into results and implementations** where submitters are required to submit their code for audits
4. **support many vendor backends/SDKs** and NNAPI and TFLite delegates to unleash the SoCs’ capabilities.
5. **be driven and audited by the industry,** which fosters fair and representative performance evaluations.

Due to space constraints and the lack of transparency into existing benchmarks, we indicate at least a few major requirements that are *missing* from prior art in Table 4. We provide additional details to the extent possible in Appendix D.

One crucial requirement missing from all the other benchmarks is that they are *not* driven by industry collaboration and consensus. A community-driven effort leads to the adoption of best practices that are fair to everyone. Moreover, it helps amortize the overhead of developing a complex benchmark like MLPerf to meet all five requirements. It is for this

	Req. 1	Req. 2	Req. 3	Req. 4	Req. 5
Aitutu	✓	✗	✗	✓	✗
AI-Benchmark	✓	✗	✗	✗	✗
AIMark	✓	✗	✗	✓	✗
Android MLTS	✗	✗	✓	✓	✗
GeekBenchML	✓	✗	✗	✗	✗
Neural Scope	✓	✗	✗	✗	✗
TF Lite	✗	✗	✓	✓	✗
UL Procyon AI	✓	✗	✗	✗	✗
Xiaomi	✓	✗	✓	✗	✗
MLPerf Mobile	✓	✓	✓	✓	✓

Table 4. Comparison to other mobile ML benchmarks where they are missing (✗) at least one (or more) requirement(s).

exact reason that Table 4 shows that the other benchmarks are each missing at least one major feature requirement.

Another major difference is measuring performance with respect to an minimum accuracy target. Table 1 shows the targets we set based on academic community and industry consensus, as well as ML consumers’ feedback. Our targets are all >93% FP32 (Table 1), even for int8 models. Other benchmarks show results for arbitrary accuracy targets that are less (if at all) meaningful. For example, GeekBenchML reports performance for int8 models with 52% of FP32 accuracy for object detection and 81% of FP32 accuracy for image classification (Geekbench, 2019). We leave it up to the reader to interpret if such results are practically useful.

9 CONCLUSION

This paper outlines the challenges, issues, and opportunities we faced over two years of research and development to engineer an acceptable benchmark across multiple competing organizations. We hope that the observations we make and the app we provide enables architects, designers, and developers to push the frontiers of mobile ML systems. There are several ongoing developments—expanding the suite, measuring end-to-end performance, evaluating iOS, ML framework benchmarking, measuring power and supporting rolling submissions. Details are presented in Appendix E.

REFERENCES

- AI-Benchmark. <http://ai-benchmark.com/>, 2019.
- AImark. https://play.google.com/store/apps/details?id=com.ludashi.aibench&hl=en_US, 2018.
- Antutu. Is Your Mobile Phone Smart? Antutu AI Benchmark Public Beta Is Released. <https://www.antutu.com/en/doc/117070.htm>, 2019.
- Antutu. Antutu Benchmark. <https://www.antutu.com/en/index.htm>, 2021.
- Arm. Big.LITTLE. <https://www.arm.com/why-arm/technologies/big-little>, 2011.
- Buch, M., Azad, Z., Joshi, A., and Reddi, V. J. Ai tax in mobile socs: End-to-end performance analysis of machine learning in smartphones. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 96–106, 2021a. doi: 10.1109/ISPASS51385.2021.00027.
- Buch, M., Azad, Z., Joshi, A., and Reddi, V. J. Ai tax in mobile socs: End-to-end performance analysis of machine learning in smartphones. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021b.
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.
- Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.
- Eigen, D. and Fergus, R. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture, 2015.
- Gao, W., Tang, F., Wang, L., Zhan, J., Lan, C., Luo, C., Huang, Y., Zheng, C., Dai, J., Cao, Z., et al. Aibench: an industry standard internet service ai benchmark suite. *arXiv preprint arXiv:1908.08998*, 2019.
- Geekbench. Geekbench ml - cross-platform benchmark. <https://www.geekbench.com/ml/>, 2019. (Accessed on 10/07/2021).
- Geekbench. Geekbench ML. <https://browser.geekbench.com/ml/v0/inference/94248>, 2019.
- Google. Google Play. <https://play.google.com/store>, 2012.
- Google. Neural Networks API Drivers. <https://source.android.com/devices/neural-networks#mlts>, 2017.
- Google. TensorFlow Lite. <https://www.tensorflow.org/lite>, 2019.
- Google. Neural Networks API. <https://developer.android.com/ndk/guides/neuralnetworks>, 2020.
- Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.
- He, Y., Sainath, T. N., Prabhavalkar, R., McGraw, I., Alvarez, R., Zhao, D., Rybach, D., Kannan, A., Wu, Y., Pang, R., Liang, Q., Bhatia, D., Shangguan, Y., Li, B., Pundak, G., Sim, K. C., Bagby, T., yiin Chang, S., Rao, K., and Gruenstein, A. Streaming end-to-end speech recognition for mobile devices, 2018.
- Hill, M. D. and Reddi, V. J. Accelerator-level parallelism. *arXiv preprint arXiv:1907.02064*, 2019.
- Howard, A. and Gupta, S. Introducing the next generation of on-device vision models: Mobilenetv3 and mobilenetedgetpu. <https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html>, 2019.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
- Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L., and Van Gool, L. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 International Conference on Computer Vision Workshop (ICCVW)*, 2019.
- Intel. Willow cove - microarchitectures - intel. https://en.wikichip.org/wiki/intel/microarchitectures/willow_cove, 2020.
- IRDS. Roadmap - ieee irds™. <https://irds.ieee.org/editions>, 2016.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.

- Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., and et al. The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981, Mar 2020. ISSN 1573-1405. doi: 10.1007/s11263-020-01316-z. URL <http://dx.doi.org/10.1007/s11263-020-01316-z>.
- Lin, C.-H., Cheng, C.-C., Tsai, Y.-M., Hung, S.-J., Kuo, Y.-T., Wang, P. H., Tsung, P.-K., Hsu, J.-Y., Lai, W.-C., Liu, C.-H., et al. 7.1 a 3.4-to-13.3 tops/w 3.6 tops dual-core deep-learning accelerator for versatile ai applications in 7nm 5g smartphone soc. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 134–136. IEEE, 2020.
- Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. Microsoft coco: Common objects in context, 2015.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, pp. 21–37, 2016. ISSN 1611-3349. doi: 10.1007/978-3-319-46448-0_2. URL http://dx.doi.org/10.1007/978-3-319-46448-0_2.
- Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation, 2015.
- MediaTek. MediaTek Dimensity 820. <https://www.mediatek.com/products/smartphones/dimensity-820>, 2020.
- MLCommons. LoadGen. <https://github.com/mlperf/inference/tree/master/loadgen>, 2019.
- MLCommons. MLCommons Mobile. https://github.com/mlcommons/mobile_models, 2020a.
- MLCommons. <https://mlcommons.org/en>, 2020b.
- MLCommons. MLPerf Mobile v0.7. <https://mlcommons.org/en/inference-mobile-07/>, 2020c.
- MLCommons. MLCommons Mobile App Open Source. https://github.com/mlcommons/mobile_app_open, 2021a.
- MLCommons. MLPerf Mobile v1.0 Results. <https://mlcommons.org/en/inference-mobile-10/>, 2021b.
- MLPerf. <https://github.com/mlperf>, 2019.
- NeuralScope. NeuralScope Mobile AI Benchmark Suite. <https://play.google.com/store/apps/details?id=org.aibench.neuralscope>, 2020a.
- NeuralScope. Neuralscope offers you benchmarking your AI solutions. <https://neuralscope.org/mobile/index.php?route=information/info>, 2020b.
- Qualcomm. Snapdragon 865+ 5G Mobile Platform. <https://www.qualcomm.com/products/snapdragon-865-plus-5g-mobile-platform>, 2020.
- Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text, 2016.
- Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.-J., Anderson, B., Breughe, M., Charlebois, M., Chou, W., Chukka, R., Coleman, C., Davis, S., Deng, P., Diamos, G., Duke, J., Fick, D., Gardner, J. S., Hubara, I., Idgunji, S., Jablin, T. B., Jiao, J., John, T. S., Kanwar, P., Lee, D., Liao, J., Lokhmotov, A., Massa, F., Meng, P., Micikevicius, P., Osborne, C., Pekhimenko, G., Rajan, A. T. R., Sequeira, D., Sirasao, A., Sun, F., Tang, H., Thomson, M., Wei, F., Wu, E., Xu, L., Yamada, K., Yu, B., Yuan, G., Zhong, A., Zhang, P., and Zhou, Y. Mlperf inference benchmark, 2020.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.
- Samsung. Mobile Processor Exynos 990. <https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-990/>, 2020.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.
- Schilling, A. Mlperf inference 1.1: Software-optimierungen zeigen stärkste wirkung - hardware-luxx. <https://www.hardwareluxx.de/index.php/news/allgemein/wirtschaft/57142-mlperf-inference-1-1-software-optimierungen.html>, 2021.
- Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision, 2015.

UL. UL Benchmarks. <https://benchmarks.ul.com/>, 2020a.

UL. UL Procyon AI Inference Benchmark.
<https://benchmarks.ul.com/procyon/ai-inference-benchmark>, 2020b.

Vera, X. Inside tiger lake: Intel's next generation mobile client cpu. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1–26. IEEE Computer Society, 2020.

Xiaomi. Mobile AI Bench. <https://github.com/XiaoMi/mobile-ai-bench>, 2018.

Xiong, Y., Liu, H., Gupta, S., Akin, B., Bender, G., Wang, Y., Kindermans, P.-J., Tan, M., Singh, V., and Chen, B. Mobiledets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2021.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. Scene parsing through ade20k dataset. In *Proceedings of the conference on computer vision and pattern recognition*, 2017.

APPENDIX

A MLPERF MOBILE APP

Measuring mobile AI performance in a fair, reproducible, and useful manner is challenging but not intractable.

As discussed previously in Section 2, the need for transparency owes to the massive hardware and software diversity, which is often tightly coupled with the intricacies of deployment scenarios, developer options, OEM lifecycles, and so on. MLPerf Mobile focuses on ensuring transparency for consumers by packaging the submitted code into an app. the app is readily accessible to the public ([MLCommons, 2021a](#)). Moreover, the associated “backend” code is readily available for review under the MLPerf rules.

Figure 8a shows the MLPerf Mobile startup screen. With a simple tap on the “Go” button, the app runs all benchmarks by default, following the prescribed run rules (Figure 8b), and clearly displays the results. It reports both performance and accuracy for all benchmark tasks (Figure 8c) and permits the user to view results for each one (Figure 8d). Furthermore, the configuration that generates the results is also transparent (Figure 8e). This allows users to understand what hardware and software configuration yielded the high-performance results. This transparency level is the critical missing feature from many off-the-shelf benchmarks as described in the Related Work (Section 8).

B MYRIAD USE CASES

While we highlight a few interesting observations in Section 7, there are many other use cases in the wild.

Application developers want to know what real-world performance may look like on a device. The benchmark provides insight into the software frameworks on the various “phones” (i.e., SoCs) for these application developers. More specifically, the benchmark can help them quickly identify the most optimal solution for a given platform. For application developers who deploy their products “into the wild,” the benchmark and the various machine-learning tasks offer a deep perspective on the end-user experience for an actual mobile AI application.

OEMs want to standardize the performance assessment methodology across different mobile chipset offerings. SoC vendors employ the same tasks, models, data sets, metrics, and run rules, making the results comparable and reproducible. Given the hardware ecosystem’s vast heterogeneity, the standardization that the benchmark provides is vital for progress.

Model designers want to package new machine learning models into the mobile app so that organizations can then easily share and reproduce the results. The app, coupled

with the LoadGen, allows model designers to test and evaluate the model’s performance on a real device rather than using operation counts and model size as heuristics to estimate performance. This feature closes the gap between model designers and hardware vendors—groups that have struggled to share information efficiently and effectively.

Mobile users wants to make informed purchasing decisions. Many users want to know whether upgrading their phone to the latest chipset will meaningfully improve their experience. To this end, they want public, accessible information about various devices—something MLPerf Mobile provides. In addition, some power users want to measure their device’s performance and share that information with performance-crowdsourcing platforms. Both are important reasons for having an easily reproducible mechanism for measuring mobile-AI performance.

Researchers often require reproducibility to push state-of-the-art technologies. As such, researchers can employ the mobile-app framework to test their methods and techniques for improving model performance, quality, or both. The framework is open-source and freely accessible to academia ([MLCommons, 2021a](#)). The app can enable researchers to integrate their ML optimizations and reproduce more recent results from the literature.

Technical analysts rely on reproducibility and transparency to provide an “apples-to-apples” comparison to assess generational performance improvements. MLPerf Mobile makes it easy to reproduce vendor-claimed results and interpret them because it shows how the device achieves a particular performance number and how it is using the hardware accelerator.

C SYSTEM IMPROVEMENTS: v0.7 TO v1.0

In this section, we provide a discussion about the hardware and system improvements that led to the significant performance improvement between benchmark v0.7 and v1.0.

The **Samsung Exynos** Exynos 990 ([Samsung, 2020](#)) has a dual-core neural processing unit (NPU) and an Arm Mali-G77 GPU. The Exynos 2100 has an 8-core CPU on a tri-cluster architecture with more than 30% improved multi-core performance, Arm Mali-G78 MP14 GPU with more than 40% performance improvement, and an AI engine with a triple-core NPU and a DSP, based on 5nm EUV delivering powerful performance and more than 2 \times the efficiency than the previous generation. But the software also played a crucial role. The uplift was 6 \times . Exynos 2100 has critical features that reduce data transfer between IP blocks, which are enabled in software through improved scheduling.

The **Qualcomm’s Snapdragon** 865+ ([Qualcomm, 2020](#)) uses the Hexagon 698 processor for AI acceleration. It

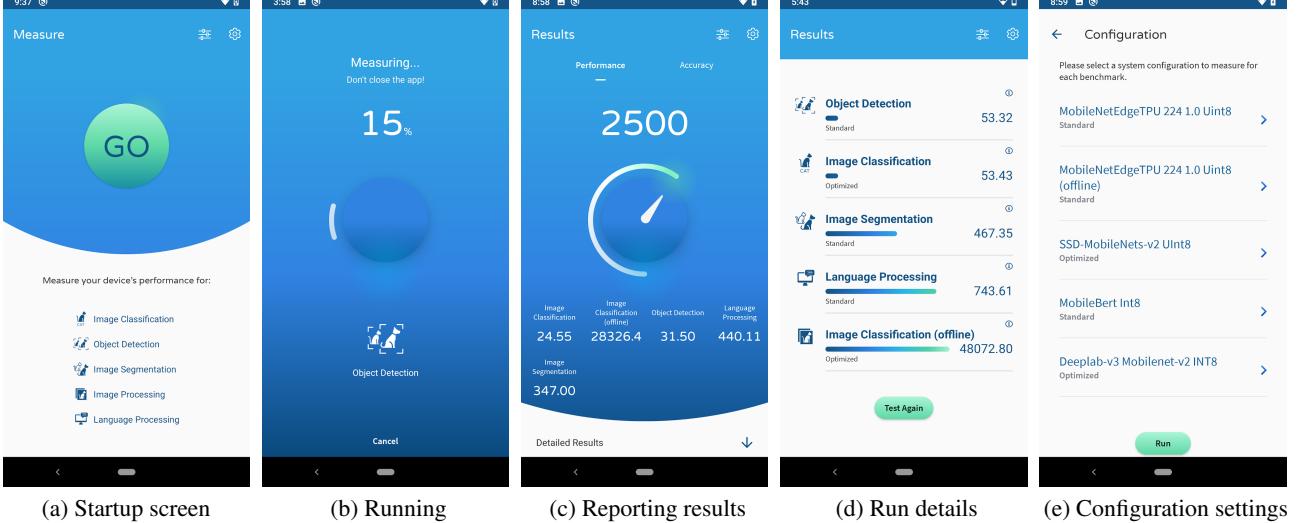


Figure 8. MLPerf Mobile app.

clocks in at 15 TOPS and has a Adreno 650 GPU. The improved Snapdragon 888’s new Hexagon 780 can perform 26 TOPS (73% faster than 865+). It also boasts a redesigned DSP microarchitecture. In the Hexagon 865 DSP, the scalar, vector and tensor execution engines were discrete independent blocks. In the Hexagon 780, the new IP block fuses all the scalar, tensor, and vector capabilities into a single monolithic IP, increasing ML workloads’ performance for the second version (v1.0).

The **MediaTek’s Dimensity 820** ([MediaTek, 2020](#)) uses an AI processing unit (APU) 3.0, an FP16 and INT16-capable accelerator optimized for camera and imaging functions ([Lin et al., 2020](#)). The SoC also has a 5-core GPU. The new Dimensity 1100 is similar in features except that it is manufactured on a 6 nm vs 7 nm process node with a more powerful GPU that is helpful for ML-task acceleration. The Dimensity 820 has single core MediaTek Deep Learning Accelerator (MDLA), while the Dimensity 1100 has dual MDLA cores. Associated with these changes are the software drivers. The Dimensity 1100 uses the Neuron Delegate to replace the NNAPI Delegate, when it is possible, as NNAPI has synchronization overhead due to the intermediate hardware abstraction layer (Figure 2b).

For v0.7, **Intel’s Willow Cove** ([Intel, 2020](#)) includes its CPU and first-generation integrated Xe-LP GPU, a Tiger Lake i7-1165G7 ([Vera, 2020](#)). For v1.0 submission, they used a TGL i7-11375H laptop. In terms of CPU frequency, i7-11375H is about 1.1 \times better than i7-1165G7; in terms of GPU frequency, i7-11375H is about 1.04 \times better compared to i7-1165G7. For image classification and object detection, the benchmark runs on CPU. Hence, the improvements are from an increase in CPU frequency. Segmentation and NLP

models need more TOPs compared to classification and detection. For this reason, Segmentation and NLP models work best on an integrated GPU (iGPU). Though there is a slight improvement in iGPU performance (4% improvement), we see a large improvement in NLP performance and a marginal increase in segmentation performance. NLP improvement is due to the OpenVINO quantized kernel.

D DETAILED PRIOR ART COMPARISON

In this section, we provide a detailed comparison of other existing mobile AI benchmarks as compared to the summary we provided in Section 8. The key differences stem from the following:

- Performance:** We support vendor SDKs. Other benchmarks can only support NNAPI and TFLite delegates (e.g., see results here: <https://browser.geekbench.com/ml/v0/inference>). While it is easier for developers to use NNAPI and TFLite delegates, they don’t showcase the hardware’s full capabilities. Table 3 shows performance varies by 10% based on delegates. But depending on the chipset, performance between delegates and vendor SDKs can vary (sometimes) by 7x due to poor/buggy op implementations (eg. Fig 5 ([Buch et al., 2021b](#)))! OEMs use vendor SDKs to speed up everyday apps like photos, camera etc. So OEMs want devices to perform well, and benchmarking vendor SDKs is essential. But this needs open-source + vendor support, which only the MLPerf Mobile benchmark supports.

- Accuracy:** In MLPerf Mobile, accuracy comes first. Performance is measured with respect to that

minimum quality target. Table 1 shows the accuracy targets we set based on community and industry consensus. Our quality targets are all >93% FP32 (Table 1). Other benchmarks show results for arbitrary accuracy targets that are less (if at all) meaningful, e.g. int8 models with 52% of FP32 accuracy for object detection and 81% of FP32 accuracy for image classification. Such models would not be deployed and will indeed mislead the industry. See e.g. <https://browser.geekbench.com/ml/v0/inference/94248>.

- **Transparency:** MLPerf is the only transparent mobile ML benchmark. Geekbench ML, AI-Benchmark, etc. do not transparently provide and/or describe their summary-score/weighting rationale, model provenance, quantization method, and optimization techniques. This makes reproducibility nearly impossible, unless with MLPerf Mobile.
- **Validity:** MLPerf has a results audit process (Section VI.B). MLPerf hires an independent auditor to review and replicate the results using the vendor-submitted code. Unlike MLPerf, none of the other benchmarks do this, nor do they make their code publicly available.

More specifically, here are the detailed comparisons against the benchmarks listed in Table 4.

Antutu employs vendor SDKs to implement image classification based on the Inception V3 neural network (Szegedy et al., 2015), using 200 images as test data (Antutu, 2021; 2019). The object-detection model is based on SSD-MobileNet (Howard et al., 2017; Liu et al., 2016), using a 600-frame video as test data. The benchmark score is a measure of speed and accuracy—faster results with higher accuracy yield a greater final score. However, *it is a closed-source application which limits result transparency*.

AI-Benchmark performs an machine-learning-performance evaluation on mobile systems with AI acceleration that integrate HiSilicon, MediaTek, Qualcomm, Samsung, and UniSoc chipsets (Ignatov et al., 2019). It evaluates 21 deep-learning tasks, including inference speed, accuracy and stability. It runs pre-selected models with various bit widths (INT8, FP16, and FP32) on the CPU and open-source or vendor-proprietary TFLite delegates. Performance-report updates appear on a website (AI-Benchmark, 2019) after each major release of TFLite/NNAPI and new SoCs with AI acceleration. However, *vendors cannot provide optimized SDKs solutions as in MLPerf, which makes a big difference*.

AImark by Master Lu (Ludashi) (AImark, 2018) is an Android and iOS application, and uses vendor SDKs to implement its benchmarks. It includes ResNet-34 (He et al., 2015), Inception V3 (Szegedy et al., 2015), SSD-MobileNet (Howard et al., 2017; Liu et al., 2016), and DeepLab v3+

(Chen et al., 2018). The benchmark judges mobile-phone AI performance by evaluating recognition efficiency, and it provides a line-test score. However, *it too is closed-sourced which limits transparency and adoptability*.

Android Machine Learning Test Suite (MLTS) is part of the Android Open Source Project (Google, 2017). MLTS includes an app that allows us to test the latency and accuracy of quantized and floating-point TFLite models (e.g., MobileNet and SSD-MobileNet) against a subset of the Open Images Dataset (Kuznetsova et al., 2020). It contains *tests to validate the behavior of the drivers in corner case conditions, not do performance benchmarking*.

GeekBenchML assesses mobile ML inference performance (Geekbench, 2019). It supports TFLite and NNAPI delegates, but it lacks supports for vendor SDKs backends which are important for OEM applications and unlocking the full SoC performance. Also, *it lacks strict minimum accuracy targets (e.g. Table 1), without which the performance results can be meaningless*.

MLPerf Inference is another industry-standard open benchmark (Reddi et al., 2020). MLPerf Mobile serves the smartphone industry (4 Billion devices), which presents its own unique challenges as discussed in Section 2. Our findings around ALP, NNAPI vs. vendor SDKs etc. are domain-specific and reveal insights that are unique to mobile device performance, which MLPerf Inference does not because *it is a benchmark for server-scale ML deployments, not mobile*.

Neural Scope from National Chiao Tung University (NeuralScope, 2020a;b) developed an NNAPI application supporting FP32 and INT8 precisions. The benchmarks comprise object classification, object detection, and object segmentation, including MobileNet v2 (Sandler et al., 2019), ResNet-50 (He et al., 2015), Inception V3, SSD-MobileNet (Howard et al., 2017; Liu et al., 2016), and ResNet-50 with atrous-convolution layers (Chen et al., 2017). Users can run the app on their mobile devices and immediately receive a cost/performance comparison, but *it lacks the open-source transparency that is direly needed*.

TensorFlow Lite provides a utility to measure the latency of any TFLite model (Google, 2019). A wrapper API is also available to reference how these models perform when embedded in an Android application. Users can select the NNAPI delegate, and they can disable NNAPI in favor of a hardware-offload back end. *It is focused on benchmarking individual TFLite operators' performance, not ML tasks with rules and metrics*.

UL Procyon AI Inference Benchmark from UL Benchmarks, VRMark (UL, 2020a;b) is an Android NNAPI CPU- and GPU-focused AI benchmark. It contains MobileNet v3 (Howard & Gupta, 2019), Inception V4 (Szegedy et al., 2015), SSDMobileNet v3 (Howard & Gupta, 2019; Liu

et al., 2016), DeepLab v3 (Chen et al., 2018), and other models. It also attempts to test custom CNN models but uses an AlexNet (Krizhevsky et al., 2012) architecture to evaluate basic operations. The application provides benchmark scores, performance charts, hardware monitoring, model output, and device rankings. But it *only compares NNAPI implementations on FP and INT-optimized models, not other frameworks.*

Xiaomi’s Mobile AI Benchmark provides an end-to-end open-source tool for evaluating model accuracy and latency (Xiaomi, 2018). The tool includes a daily performance-benchmark run for various neural-network models (mainly on the Xiaomi Redmi K30 Pro smartphone). The tool has a configurable back end that allows users to employ multiple ML-hardware-delegation frameworks (including MACE, SNPE, and TFLite). However, it is *not designed for the broad ecosystem of different vendors’ mobile devices.*

E FUTURE WORK

To enable the myriad use cases and reveal additional mobile processor insights, we are engaged in numerous activities.

Expanding the benchmark suite is an obvious area of improvement. We are working to expand the scope to include more tasks and models, along with different quality targets. Examples include additional vision tasks, such as super-resolution, as well on-device speech recognition (He et al., 2018). Our current network choices reflect common use cases, most mobile ML use cases involve computer vision. But our NLP Q&A task & Mobile BERT task are heavier. Speech RNN-T is in the works - we’re working with Google and Facebook engineers to build a mobile model version.

Mobile has more heavy-duty models than our *initial* selection. Super-resolution and high-resolution models are important use cases, but they are still evolving. However, there is no agreement on which mobile ML versions are broadly applicable for these types of use cases. Also, the metrics for evaluating these tasks are not clearly defined. The other issue is a lack of commercially distributable datasets for these tasks. Thus, to begin, we included reliable models that were stable and could be improved over time.

End-to-end performance is important as user-perceived latency includes often includes pre- and post-processing overheads, and it has been shown to be non-negligible (Buch et al., 2021a). In the future, we may consider extending the scope of measurements.

iOS support recently became available. Example uses can be found online (Schilling, 2021). Apple’s iOS is a major AI-performance player that brings additional hardware and software diversity and we expect results in the near future.

Measuring software frameworks is essential. As we de-

scribed in Section 2, software performance—and, more importantly, its capabilities—is crucial to unlocking a device’s full potential. Enabling apples-to-apples comparison of software frameworks on a fixed hardware platform has merit. The back-end code path in Figure 5 (code path 1) is a way to integrate different machine-learning frameworks to determine which one achieves the best performance on a target device.

Power measurement is a major area of potential improvement. Since mobile devices are battery-constrained, evaluating mobile AI’s power draw is important. While we currently do not account for power, most smartphone chipsets are capped at a 3W TDP, so it provides an artificial thermal/power ceiling.

Rolling result submissions are needed as new devices frequently arrive, often in between the calls for submissions. We plan to add “rolling submissions” to encourage vendors to submit their scores continuously, which would allow up-to-date and consistent reporting of the AI performance. Technical roadmaps like IRDS (IRDS, 2016) rely on this data to make informed recommendations to policymakers and funding agencies.

F ARTIFACT EVALUATION

The MLPerf Mobile App artifact we provide is a mobile app which provides latency and accuracy benchmarking capabilities for four mobile-targeted neural network models, evaluated on three computer vision tasks and one natural language processing task. The app can be run standalone to evaluate performance (“performance mode”) or optionally on full validation datasets to evaluate both accuracy and performance (“accuracy mode”). Additionally, smartphone vendors can provide optimized back-end implementations to take advantage of architecture-specific features to increase inference speed. Here, we provide instructions here for reproducing performance mode evaluations without a custom back-end. The app is open-source, and a pre-built APK is available to reviewers on request (Sec. F.2.1).

F.1 Artifact Check-list (Meta-information)

- **Program:** MLPerf Mobile App
- **Compilation:** Android Studio (version ≥ 4.0), TensorFlow Lite
- **Binary:** mlperf_app.apk
- **Datasets:** Depends on ImageNet ILSVRC2012, COCO2017, ADE20K, and SQuAD 1.1, which are available for download online.
- **Models:** Depends on MobileNetEdgeTPU, SSD-MobileNet v2, MobileDET_SSD, Deeplabv3+ - MobileNetv2, MobileBERT, which are available for download online.
- **Run-time environment:** Android 10.0+ (API 29)
- **Hardware:** Android smartphones

- **Run-time state:** On battery power, with sufficient ventilation and in a room temperature between 20-25C, with cooldown periods of 5 minutes between tests.
- **Metrics:** Queries per second, and optionally, Top-1 accuracy, mAP, mIoU, F1 score
- **Output:** Displayed numerical results
- **Experiments:** On-device benchmark measurements should be within the allowed quality targets (Table 1) and within 5% of vendor-reported metrics (Sec. VI-B).
- **How much disk space required (approximately)?:** under 100MB
- **How much time is needed to prepare workflow (approximately)?:** 2 hours
- **How much time is needed to complete experiments (approximately)?:** Under 1 hour
- **Publicly available?:** The source code for the Android APK is archived here: [<https://doi.org/10.5281/zenodo.6416132>]. The source code is also available on github at [https://github.com/mlcommons/mobile_app_open/tree/android-v2]. Build artifact is distributed only to reviewers.
- **Code licenses (if publicly available)?:** https://github.com/mlcommons/mobile_app_open/blob/master/LICENSE.md

F.2 Description

F.2.1 Build Instructions

Instructions for building the MLPerf Mobile App are provided at https://github.com/mlcommons/mobile_app_open/tree/android-v2, and in the archived version of this branch at <https://doi.org/10.5281/zenodo.6416132>. For convenience, we also have provided prebuilt Android APKs for Artifact Evaluation reviewers to use. Reviewers can email William Chou (wchou@qti.qualcomm.com) and Wookie Hong (jwookie.hong@mlcommons.org) to request access to the pre-built APK.

F.2.2 Hardware Dependencies

Smartphones equipped with the MediaTek Dimensity 820, Samsung Exynos 990, and Qualcomm Snapdragon 865+ were used in the evaluation. Some additional Android smartphone models are supported.

F.2.3 Datasets

For running benchmarks in “performance mode” no additional datasets are required to be downloaded, and all data needed is present within the APK. The following four datasets are used in the “accuracy mode” benchmark evaluation: ImageNet ILSVRC2012, COCO2017, ADE20K, and SQuAD 1.1. Instructions on obtaining and formatting the datasets are available at https://github.com/mlcommons/mobile_

[app_open/blob/android-v2/android/cpp/datasets/README.md](https://github.com/mlcommons/mobile_app_open/blob/android-v2/android/cpp/datasets/README.md) which is included as part of the source APK repository, and includes scripts for reformatting the datasets to the format expected by the MLPerf Mobile App.

F.3 Installation

For reviewers, we provide a prebuilt APK (Sec. F.2.1) which can be installed by transferring the APK to an Android phone or SDCard and selecting the APK through the file browser. To evaluate in “performance mode” only the APK itself is needed. For non-reviewers, instructions for building and installing the MLPerf Mobile App APK are provided at https://github.com/mlcommons/mobile_app_open/tree/android-v2. In particular there are multiple pathways documented for building the app, including via Bazel, Docker, and Android Studio.

F.4 Evaluation and Expected Results

Open the MLPerf Mobile App and click the gear/settings icon and ensure “Submission mode” is toggled off (this ensures the app in “performance mode” where accuracy is not evaluated on the full validation datasets). “Cooldown” can be enabled to add to include a 5 minute pause between benchmarks to avoid thermal throttling. Click back to exit the settings page, and scroll to the bottom to select the “Test” or “Test Again” button to begin the evaluation.

Once the suite of benchmarks is completed, results (as queries per second) will be displayed for Image Classification, Object Detection, Image Segmentation, Language Processing, and Image Classification (offline).

F.5 Methodology

Submission, reviewing and badging methodology:

- <http://cTuning.org/ae/submission-20190109.html>
- <http://cTuning.org/ae/reviewing-20190109.html>
- <https://www.acm.org/publications/policies/artifact-review-badging>