MLPerf Mobile Inference Benchmark

Why Mobile AI Benchmarking Is Hard and What to Do About It

Vijay Janapa Reddi[*] **David Kanter**[†] **Peter Mattson**[‡] **Jared Duke**[‡] **Thai Nguyen**[‡] **Ramesh Chukka**[§]
Kenneth Shiring[¶] **Koan-Sin Tan**[¶] **Mark Charlebois**[‖] **William Chou**[‖] **Mostafa El-Khamy**[**]
Jungwook Hong[**] **Michael Buch**[*] **Cindy Trinh**[††] **Thomas Atta-fosu**[§] **Fatih Cakir**[**]
Masoud Charkhabi[‡] **Xiaodong Chen**[**] **Jimmy Chiang**[¶] **Dave Dexter**[‡‡]
Woncheol Heo[‡] **Guenther Schmuelling**[§§] **Maryam Shabani**[§] **Dylan Zika**[††]

Abstract

MLPerf Mobile is the first industry-standard open-source mobile benchmark developed by industry members and academic researchers to allow performance/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. In this paper, we motivate the drive to demystify mobile-AI performance and present MLPerf Mobile's design considerations, architecture, and implementation. The benchmark comprises a suite of models that operate under standard data sets, quality metrics, and run rules. For the first iteration, we developed an Android app to provide an "out-of-the-box" inference-performance benchmark for computer vision and natural-language processing on mobile devices. The benchmark also supports non-smartphone devices such as laptops and mobile PCs. As a whole, the MLPerf Mobile inference benchmark can serve as a framework for integrating future models, for customizing quality-target thresholds to evaluate system performance, for comparing software frameworks, and for assessing heterogeneous-hardware capabilities for machine learning, all fairly and faithfully with reproducible results.

1 Introduction

Mobile artificial-intelligence (AI) applications are increasingly important as AI technology becomes a critical differentiator among smartphones, laptops, and other mobile devices. Many consumer applications benefit from AI: image processing, voice processing, and text interpretation. AI provides state-of-the-art solutions to these tasks with a quality that users will notice on their devices. More and more consumers are using such applications, and they expect a high-quality experience—especially for applications with video or audio interactivity.

Consequently, mobile-device and chipset manufacturers are motivated to improve AI implementations. Support for the technology is becoming common in nearly all mobile segments, from cost-optimized devices to premium phones. The many AI approaches range from purely software techniques to hardware-supported machine learning that relies on tightly coupled libraries. Seeing through the mist of competing solutions is difficult for mobile consumers.

On the hardware front, laptops and smartphones have incorporated application-specific integrated circuits (ASICs) to support AI in an energy-efficient manner. For machine learning, this situation leads to custom hardware that ranges from specialized instruction-set-architecture (ISA) extensions on general-purpose CPUs to fixed-function accelerators dedicated to efficient machine learning. Also, because mobile devices are complex, they incorporate a variety of features to remain competitive, especially those that help conserve battery life.

The software front includes many code paths and AI infrastructures owing to the desire to efficiently support machine-learning hardware. Most SoC vendors lean toward custom pathways for model compilation and deployment that are tightly integrated with the hardware. Examples include Google's Android Neural Network API (NNAPI) [15], Intel's OpenVINO [5], MediaTek's NeuroPilot [19], Qualcomm's SNPE [23] and Samsung's Exynos Neural Network SDK [21]. These frameworks handle different numerical formats (e.g., FP32, FP16, and INT8) for execution, and they provide run-time support for various machine-learning networks that best fit the application and platform.

Hardware and software support for mobile AI applications is becoming a differentiating capability, resulting in

[*]Harvard University [†]MLCommons [‡]Google [§]Intel [¶]MediaTek [‖]Qualcomm [**]Samsung [††]ENS Paris-Saclay [‡‡]Arm [§§]Microsoft

a growing need to make AI-performance evaluation transparent. OEMs, SoC vendors, and consumers benefit when mobile devices employ AI in ways they can see and compare. A typical comparison point for smartphone makers and the technical press, for example, is CPUs and GPUs, both of which have associated benchmarks [6]. Similarly, mobile AI performance can also benefit from benchmarks.

Benchmarking AI performance is nontrivial, however. It is especially challenging because AI implementations come in a wide variety with differing capabilities. This variety, combined with a lack of software-interface standards, complicates the design of standard benchmarks. In edge devices, the quality of the results is often highly specific to each problem. In other words, the definition of *high performance* is often task specific. For interactive user devices, latency is normally the preferred performance metric. For noninteractive ones, throughput is usually preferred. The implementation for each task can generally trade off neural-network accuracy for lower latency. This tradeoff makes choosing a benchmark suite's accuracy threshold critical.

To address these challenges, MLPerf (mlperf.org) takes an open-source approach. It is a consortium of industry and academic organizations with shared interests, yielding collective expertise on neural-network models, data sets, and submission rules to ensure the results are relevant to the industry and beneficial to consumers while being transparent and reproducible.

The following are important principles that inform the MLPerf Mobile benchmark:

- Measured performance should match the performance that end users perceive in commercial devices. We want to prevent the benchmark from implementing special code beyond what these users generally employ.
- The benchmark's neural-network models should closely match typical mobile-device workloads. They should reflect real benefits to mobile-device users in daily situations.
- Neural-network benchmark models should represent diverse tasks. This approach yields a challenging test that resists extensive domain-specific optimizations.
- Testing conditions should closely match the environments in which mobile devices typically serve. Affected characteristics include ambient temperature, battery power, and special performance modes that are software adjustable.
- All benchmark submissions should undergo third-party validation. Since mobile devices are ubiquitous, results should be reproducible outside the submitting organization.

MLPerf's approach to addressing the mobile-AI benchmark needs of smartphones is to build an Android app that all benchmarking must use. As of the initial v0.7 release of the MLPerf Mobile benchmark, the app employs a standard set of four neural-network models for three vision tasks and one NLP task and passes these models to the back-end layer. This layer is an abstraction that allows hardware 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") [9]. MLPerf created the LoadGen [9] to allow representative testing of different inference platforms and use cases, which generates inference requests in a pattern and measures some parameters (e.g., latency, throughput, or latency-bounded throughput). MLPerf additionally offers a headless version of the mobile application that enables laptops running non-mobile OSs to use the same benchmarks.

The first round of MLPerf Mobile submissions is complete [12]. Intel, MediaTek, Qualcomm, and Samsung participated in this round, and all passed the third-party-validation requirement (i.e., reproducibility) for their results. These results show performance variations and illustrate the wide range of hardware and software approaches that vendors take to implementing neural-network models on mobile devices. The results also highlight a crucial takeaway: measuring mobile-AI performance is challenging but possible. It requires a deep understanding of the fragmented and heterogeneous mobile ecosystem as well as a strong commitment to fairness and reproducibility. MLPerf Mobile is a step toward better benchmark transparency.

2 Benchmarking Challenges

The mobile ecosystem is rife with hardware heterogeneity, software fragmentation, developer options, deployment scenarios, and OEM life cycles. Each by itself leads to hardware-performance variability, but the combination makes AI benchmarking on mobile systems extremely difficult. Figure 1 shows the various stakeholders and explains the implementation options and challenges facing each one.

2.1 Hardware Heterogeneity

Smartphones contain complex heterogeneous chipsets that provide many different compute units and accelerators. Any or all of these components can aid in machine-learning (ML) inference. As such, recognizing the variability of SoCs is crucial.

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. Many smartphones today are Arm based, but the CPU cores generally implement a heterogeneous "Big.Little" architecture [4]. Some SoCs even have big-CPU clusters where some big CPUs clock faster than others. Also, devices fall into different tiers with

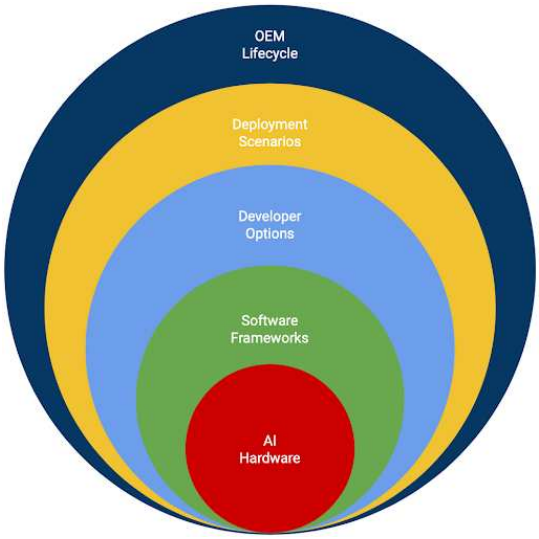


Figure 1: Mobile AI performance stakeholders.

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. A given device may have a spectrum of AI-performance capabilities depending on which processing engines it uses. Hence the need for a systematic 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 machine-learning run time. The result can be drastic hardware-performance changes or variability. Mobile devices employ various OSs: Android, iOS, Windows, Ubuntu, Yocto, and so on. Each one has an ecosystem of ML application programming interfaces (APIs) and application-deployment options that necessitate particular software solutions.

Smartphone OSs have undergone substantial consolidation. Numerous APIs have served in the development of ML applications, and often, a single SoC or OEM device will support a vendor SDK and a plurality of frameworks. SoC vendors will by default offer a proprietary 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) [24] and NNAPI [15], that provide a compatibility layer to support various accelerators and device types. Because engineering resources are limited, however, SoC vendors must prioritize their own SDKs, often resulting in partial or less optimized 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. Even for SoCs that integrate a high-performance ML accelerator, if a generic Android framework such as NNAPI does not support it (well) with high-performance driver backends, the accelerator will function poorly when handling a network.

Because software code paths can drastically affect hardware performance, a transparent mechanism for operating and evaluating a mobile device is essential.

2.3 Developer Options

Developers can choose among several approaches to enable machine learning on mobile devices. Each one has implications for achievable hardware performance on a given application. Recognizing these behind-the-scenes factors is therefore critical to maximizing performance.

Application developers can work through a marketplace such as Google Play [7] to create mobile-app variants for every SoC vendor if they follow a vendor-SDK approach (Figure 2a). Doing so presents a scalability challenge, however, 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 backend drivers to the framework, necessitating cooperation between these vendors and the framework designers.

A final alternative is to bind the neural-network model to the underlying hardware. Doing so allows compilation of the model to a particular device, avoiding reliance on any particular run time (Figure 2c).

2.4 Deployment Scenarios

Machine-learning applications have many potential uses on mobile devices. Details of the usage scenario determine the extent to which a neural-network 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) illustrates this case. 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

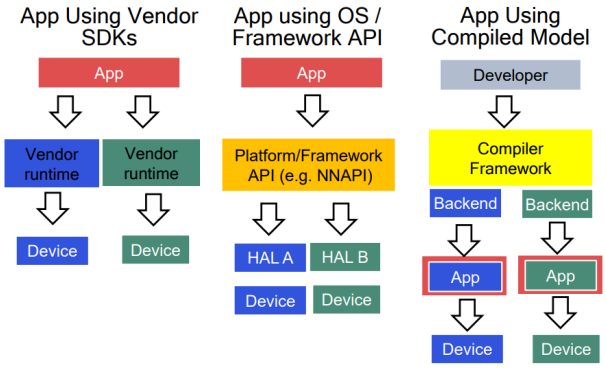


Figure 2: Application-development options.

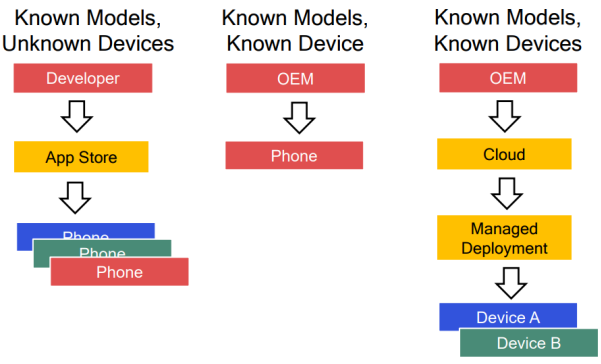


Figure 3: ML-application scenarios.

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.

2.5 OEM Life Cycle

Mobile-SoC testing often occurs on development platforms. Gaining access to them, however, is difficult. Therefore, the results of benchmark testing that employs a development platform may not be independently verifiable. For this reason, benchmarking generally takes place on commercial devices. But because of the way commercial mobile devices (particularly smartphones) operate, getting reproducible numbers can be difficult.

A variety of factors, ranging from how OEMs package software for delivery to how software updates are issued, affect hardware-performance measurements. OEMs employ vendor SoCs and associated software releases to produce commercial mobile devices. In the case of smartphones, those devices may sell unlocked or locked to a wireless carrier, in which case the carrier ultimately controls the software. OEMs pick up the software updates from the SoC vendors and usually bundle them with other updates for periodic release. If the carrier sells the device, it will likely require testing and validation before allowing any updates. This restriction can add further delays to the software-update channel. NNAPI updates, for instance, would require a new software update for the device. For a benchmark, no recompilation is necessary when using NNAPI; updates to a vendor SDK, however, may necessitate recompilation (Figure 2a).

When benchmarking a device, a newly installed software update may affect the results, and installing the same version of the software used to generate a particular result may be impossible. After a device applies a system-software update, the only way to revert to the previous configuration is to factory reset the device. But doing so also undoes any associated security fixes.

More often than not, a substantial delay occurs between the time when an SoC vendor releases new software and when that software sees deployment on user devices. The delay is usually measured in months, and it especially affects the system-API approach (e.g., NNAPI). Extensive planning is therefore necessary for a commercial phone to have all the required features for an upcoming benchmark.

Finally, commercial devices receive OEM updates only for a fixed period, so they will not benefit from additional software-performance enhancements after that time.

2.6 Legal and IP

An important yet easily overlooked aspect of ML benchmarking is the law. A chief challenge to constructing a widely used mobile benchmark is the legal and intellectual-property (IP) regime for both data sets and tool chains. Since ML tends to be open source, the rigidity and restrictions on data sets and SDKs can be surprising.

Distribution of standard ML data sets is under licenses with limited or unclear redistribution rights (e.g., ImageNet and COCO). Not all organizations have licensed these data sets for commercial use, and redistribution through an app is legally complicated. In addition, submitters to an ML benchmark may apply different legal-safety standards when participating in a public-facing software release.

Additionally, many SoC vendors rely on proprietary SDKs to quantize and optimize neural networks for their products. Although some SDKs are publicly available under off-the-shelf licensing terms, others require direct approval or negotiation with the vendor. Additionally, most forbid redistribution and sharing, potentially hindering reproduction of the overall flow and verification of a result.

Area	Task	Reference Model	Data Set	Quality Target
Vision	Image classification	MobileNetEdgeTPU (4M params)	ImageNet 2012 (224x224)	98% of FP32 (76.19% Top-1)
Vision	Object detection	SSD-MobileNet v2 (17M params)	COCO 2017 (300x300)	93% of FP32 (0.244 mAP)
Vision	Semantic segmentation	DeepLab v3+ (2M params)	ADE20K (512x512)	97% of FP32 (54.8% mIoU)
Language	Question answering	MobileBERT (25M params)	mini Squad v1.1 dev	93% of FP32 (93.98% F1)

Table 1: MLPerf Mobile v0.7 benchmark suite.

3 MLPerf Mobile Benchmarks

The MLPerf Mobile Inference benchmark is community driven. As such, all involved parties aided in developing the benchmark models and submission rules; the group includes both submitting organizations and organizations that care about mobile AI. Participants reached a consensus on what constitutes a fair and useful benchmark that accurately reflects mobile-device performance in realistic scenarios.

Table 1 briefly summarizes the tasks, models, data sets, and metrics. This section describes the models in the v0.7 MLPerf Mobile version. Rather than the models, a crucial aspect of our work is the method we prescribe for mobile-AI performance testing. Also, we describe the quality requirements during benchmark testing.

3.1 Tasks and Models

Machine-learning tasks and associated neural-network models come in a wide variety. Our benchmark's first iteration focused on establishing a high-quality method of benchmarking, rather than focusing on model quantity. To this end, we intentionally chose a few machine-learning tasks representing real-world uses. Benchmarking them yields helpful insights about hardware performance across a wide range of deployment scenarios (smartphones, notebooks, etc.). We chose networks for these tasks on the basis of their maturity and applicability to various hardware (CPUs, GPUs, DSPs, NPUs, etc.).

Image classification picks the best label to describe an input image and is commonly used for photo search and text extraction. Many commercial applications employ image classification, which is a de facto standard for evaluating ML-system performance. Moreover, classifier-network evaluation provides a good performance indicator for the model when that model serves as a feature-extractor backbone for other tasks. Image classification has a wide range of applications, such as photo searches, text extraction, and industrial automation (object sorting and defect detection).

On the basis of community feedback, we selected MobileNetEdgeTPU [28], which is well-optimized for mobile applications and provides good performance on different SoCs. The MobileNetEdgeTPU network is a descendent of the MobileNet-v2 family that is optimized for low-latency and mobile accelerators. The MobileNetEdgeTPU model architecture is based on convolutional layers with inverted residuals and linear bottlenecks, similar to MobileNet v2, but is optimized by introducing fused inverted bottleneck convolutions to improve hardware utilization, and removing hard-swish and squeeze-and-excite blocks.

The MobileNetEdgeTPU reference model is evaluated on the ImageNet 2012 validation dataset [50] and requires 74.66% (98% of FP32 accuracy) Top-1 accuracy (app uses a different dataset). Before inference, images are resized, cropped to 224x224, and normalized.

Object detection draws bounding boxes around objects in an input image and then labels the object and is commonly applied to camera input. Implementations typically use a pretrained image-classifier network as a backbone or feature extractor, then perform bounding-box selection and regression for precise localization [49, 43]. Object detection is crucial for automotive tasks, such as detecting hazards and analyzing traffic, and for mobile-retail tasks, such as identifying items in a picture.

Our reference model is the Single Shot Detector (SSD) [43] with a MobileNet v2 backbone [51]—a choice that is well adapted to constrained computing environments. The SSD-MobileNet v2 uses Mobilenet v2 for feature extraction and a mobile friendly variant of regular SSD called SSDlite for detection. In SSD prediction layers, all the regular convolutions are replaced with separable convolutions (depthwise followed by 1 x 1 projection). SSD-MobileNet v2 improves latency by significantly decreasing the number of operations, it also reduces the memory footprint needed during inference by never fully materializing the large intermediate tensors. Two SSD-MobileNet v2 versions acted as the reference models for the object-detection benchmark, where one model replaces more of the regular SSD-layer convolutions with depth-separable convolutions than the other does.

We used the COCO 2017 validation data set [42] and, for the quality metric, the mean average precision (mAP). The target accuracy is a mean Average Precision (mAP) of 22.7 (93% of FP32 accuracy). Preprocessing consists of first resizing to 300x300—typical of resolutions in smartphones and other compact devices—and then normalizing.

Semantic image segmentation partitions an input image into labeled objects at pixel granularity. Semantic image segmentation partitions an input image into labeled ob-

jects at pixel granularity. It applies to autonomous driving and robotics [38, 54, 45, 53], remote sensing [52], medical imaging [57], and also complex image manipulation such as red-eye reduction.

Our reference model for this task is DeepLab v3+ [30] 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 [44, 33] as well as to achieve state-of-the-art performance by overcoming reduced-feature-resolution problems and incorporating multiscale context. DeepLabV3+ 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 within a constrained computational budget.

We chose the ADE20K validation data set [59] 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 were the most frequent (pixel-wise) classes in ADE20K, and the 32nd represented all the other classes. The mIoU depends on the pixels whose ground-truth label belongs to one of the 31 most frequent classes, improving its accuracy by discarding the network's bad performance on low-frequency classes.

Question answering is an NLP task - responding to human-posed questions in colloquial language. Example applications include search engines, chatbots, and other information-retrieval tools. For this task, we use the Stanford Question Answering Dataset (Squad) v1.1 Dev [48]. Given a question and a passage from a Wikipedia article, the model must extract a text segment from the passage to answer the question.

Recent NLP models that rely on pretrained contextual representations have proven useful in diverse situations [31, 46, 47]. BERT (Bidirectional Encoder Representations from Transformers) [32] improves on those models by pre-training the contextual representations to be bidirectional and to learn relationships between sentences using unlabeled text. We selected MobileBERT [55], a lightweight 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.

3.2 Reference Code

MLPerf provides reference-code implementations for the TensorFlow and TensorFlow Lite benchmarks. All reference models have 32-bit floating-point weights, and the benchmark additionally provides an 8-bit quantized version (with either post-training quantization or quantization-aware training, depending on the tasks). The code for all reference implementations is open source and free to download from GitHub [11].

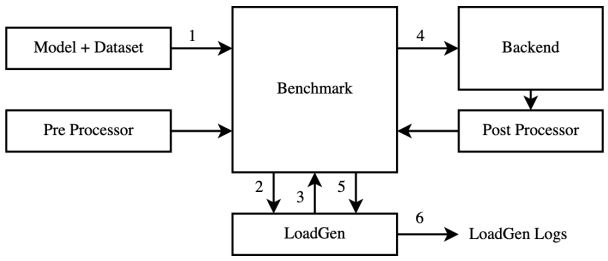


Figure 4: Load Generator ("LoadGen") testing the SUT.

The reference code's goal is to explicitly identify the critical model-invocation stages. For instance, the reference benchmarks implement the preprocessing stages and the model's input-generation procedure. Submitters may adopt the code for their submission. They may also 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.

By default, the reference code is not well-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 perform (quantized) inference, they may need to invoke the correct software backend (e.g., SNPE and ENN) or NNAPI driver to schedule code to their SoC's custom accelerators.

3.3 System Under Test

A typical system under test (SUT) interfaces with several components. Orchestrating the complete SUT execution involves multiple stages. The main ones are model selection, data-set input, preprocessing, back-end execution, and post-processing. Figure 4 shows how these stages work together.

Model selection. The first step is selection of the reference models, either TensorFlow or TFLite.

Load generator. To enable representative testing of various inference platforms and use cases, we created the Load Generator ("LoadGen") [9]. The LoadGen creates inference requests in a pattern and measures some parameter (e.g., latency, throughput, or latency-bounded throughput). In addition, it logs information about the system during execution to enable post-submission result validation. Submitter modification of the LoadGen software is forbidden.

Data-set input. The LoadGen uses the data sets as inputs to the SUT. In accuracy mode, it feeds the entire data set to the SUT to verify that the model delivers the required accuracy. In performance mode, it feeds a subset of the im-

ages to the SUT to measure steady-state performance. A seed and random number generator is used to select samples from the data-set for inference, which precludes any unrealistic data-set-specific optimizations.

Preprocessing. The typical image-preprocessing tasks—such as resizing, cropping, and normalization—depend on the neural-network model. This stage implements data-set-specific preprocessing, varies with the task and the same steps must be followed by all the submitters.

Back-end execution. The reference benchmark implementation is a TFLite smartphone back end that optionally includes NNAPI and GPU delegates. A "dummy" back end is also available as a reference for proprietary back ends; submitters replace it with whatever corresponds to their system. For instance, Qualcomm would replace the dummy with SNPE or Samsung would replace it with ENN. The back end corresponds to other frameworks such as OpenVINO for notebooks and other large mobile devices.

Postprocessing. This data-set-specific task covers all the operations necessary for accuracy calculations. For example, computing the Top-1 or Top-5 results for an image classifier requires a Top-K op / layer after the softmax layer.

A typical SUT can be either a smartphone or a laptop. We therefore designed all the mobile-benchmark components to take advantage of either one. Figure 5 shows how MLPerf Mobile supports this flexibility. The reference TensorFlow models are at the root of the entire process. The MLPerf Mobile process follows one of three paths.

Code path 1 allows submitters to optimize the reference TensorFlow models for implementation via a proprietary backend (e.g., SNPE for Qualcomm or ENN for Samsung), then schedule and deploy the networks on the hardware.

Code path 2 allows 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 quantization optimizations, using the TFLite delegates to access the AI-processing hardware.

Code path 3 allows non-smartphone submitters to run the reference TensorFlow models through nonmobile backends (e.g., OpenVINO) on laptops and tablets that run operating systems such as Windows and Linux.

3.4 Execution Scenarios

MLPerf Mobile Inference supports two modes for running ML models: single stream and offline. They reflect the typical operating behavior of many mobile applications.

Single stream. In the single-stream scenario, the application sends a single inference query to the SUT with a sample size of one. That size is typical of smartphones and other interactive devices where the user takes a picture and expects a timely response, as well as AR/VR headsets where real-time operation is crucial. The LoadGen injects a query into the SUT and waits for query completion. When

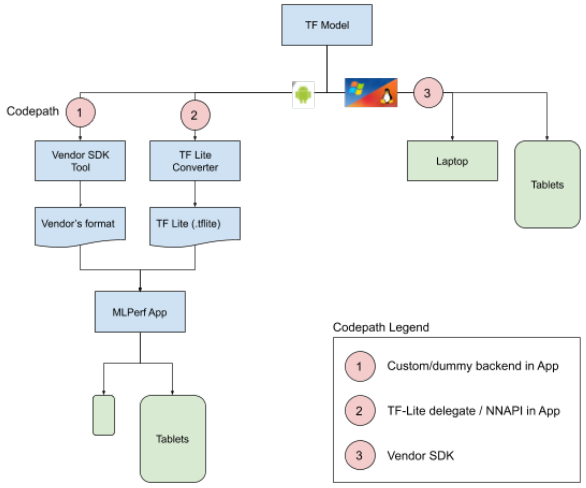


Figure 5: MLPerf Mobile benchmark code paths. The benchmarks run on smartphones and on mobile PCs, such as laptops. On smartphones, there are multiple framework options and backend codepaths that vendors can select.

the query is complete, the LoadGen 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 runtime of 60 seconds has been met.

Offline. 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 in MLPerf Mobile v0.7 issues 24,576 samples—enough to provide sufficient run time. This choice typically reflects applications that require multi-image processing, simultaneous processing of batched input, or concurrent use of models such as image classification and person detection for photos in an album. Its implementation is usually a batched query with a batch size larger than one.

4 Result Submission

This section outlines how submitters produce high-quality benchmark results for submission. We outline the submission process, the run rules, and the procedure for verifying the accuracy and validity of the results.

4.1 Submission Process

The reference models for MLPerf Mobile are provided as frozen TensorFlow FP32 checkpoints and valid submissions must start from these frozen graphs. From the frozen graph, 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 train-

ing (QAT). Network retraining typically alters the neural-network architecture therefore model equivalence is difficult to verify. Additionally, retraining allows a submitter to use their training capabilities (e.g., neural architecture search) to enhance inference performance, changing the very nature of the benchmark. Depending on submitter needs, however, MLPerf provides QAT versions of the model. All organizations 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 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 is unable to directly deploy TensorFlow-quantized models, however, and submission 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, the Mobile Working Group specified a calibration data set (typically 500 samples or images from the training or validation data set) to use for calibration in the PTQ process. Submitters can only use the approved calibration data set; but they may select a subset of the samples.

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. However, MLPerf rules strictly prohibit altering the AI models to reduce their computational complexity; banned techniques include channel pruning, filter pruning, and weight skipping.

4.2 Submission System

Smartphones and notebooks can use the mobile-benchmark suite. For smartphones, we developed a reference MLPerf Android app that supports TFLite delegates and NNAPI delegates. We benchmark the inference-task performance at the application layer to reflect latencies that mobile-device users observe and to give developers a reference for expected user-app latencies.

The MLPerf Mobile app queries the LoadGen, which in turn queries input samples for the task, loads them to memory, and tracks the time required to execute the task. Companies that used proprietary delegates implemented their backend interface to the reference MLPerf app. Such backends query the correct library (TensorFlow, TFLite, Exynos Neural Network (ENN) SDK, or SNPE SDK) to run the models on the SUT in accordance with the run rules.

For laptops, submitters can build a native command-line application that incorporates the instructions in the MLCommons GitHub repo. The MLPerf LoadGen can integrate this application, and supports backends 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 user interface for these devices.

4.3 Run Rules

In any benchmark, measurement consistency is crucial for reproducibility. MLPerf Mobile is no different. We developed a strict set of run rules that allow us to reproduce submitted results through an independent third party.

- **Test control.** The MLPerf app runs the five benchmarks in a specific order. For each one, the model first runs on the whole validation set to calculate the accuracy, which the app then reports. Performance mode then 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 and in current systems will exceed 60 seconds of run time.

- **Thermal throttling.** Machine-learning models are computationally heavy and can trigger run-time thermal throttling to cool the SoC. We recommend that smartphones maintain an air gap with proper ventilation and avoid flush contact with any surfaces. Additionally, we require normal room temperature operation—between 20 and 25 degrees Celsius.

- **Cooldown interval.** The benchmark does not test the performance under thermal throttling, so the app allows 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 benchmark suite is to run multiple times, we recommend a minimum 10-minute break between them.

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

The above rules are generally inapplicable to laptops because these devices have sufficient power and cooling.

4.4 Result Validation

MLPerf Mobile submission rules require that the SUT (smartphone or laptop) be commercially available before publication, which enables a more tightly controlled and robust validation, review, and audit process. In contrast, the other MLPerf benchmark suites allow submission of preview and research systems that are not commercially avail-

able. Smartphones should be for sale either through a carrier or as an unlocked phone. The SUT includes both the hardware and the software components, so these rules prohibit device rooting.

At submission time, each organization has no knowledge of other results or submissions. All must submit their results at the same time. Afterward, the submitters collectively review all the results in a closed setting, inspired by the peer-review process for academic publications.

Submissions include all of the benchmark app's log files, unedited. After the submission deadline, results for each participating organization are available for examination by the MLPerf working group and the other submitters, along with any modified models and code used in the respective submissions. The vendor backend (but not the tool chain) is included. MLPerf also receives private vendor SDKs to allow auditing of the model-conversion process.

The audit process comprises examination of log files, models, and code for compliance with the submission rules as well as verification of their validity. It also includes verification of the system's reported accuracy and latencies. To verify results, we build the vendor-specific MLPerf app, install it on the phone (in the factory-reset state), and attempt to reproduce latency or throughput numbers, along with accuracy. We consider the results verified if our numbers are within 5% of the reported values.

5 Performance Evaluation

The MLPerf Mobile inference suite first saw action in October 2020. Mobile submissions fall into one of two categories: smartphones and laptops. The results reveal a device's system on chip (SoC) performance for each of the machine learning tasks in version 0.7. This section assesses how the benchmark performed—specifically, whether it met expectations in being transparent and faithful, reflecting the vast diversity of AI hardware and software.

5.1 Premium ML Systems

The submitted systems include premier 5G smartphones and high-end mobile SoCs from MediaTek, Qualcomm, and Samsung. The MediaTek chipset is a Dimensity 820 [10] in the Xiaomi Redmi 10X smartphone; it contains MediaTek's AI processing unit (APU) 3.0. The APU uniquely supports FP16 and INT16 [41]. The Qualcomm chipset is a Snapdragon 865+ [22] in the Asus ROG Phone 3. It integrates Qualcomm's Hexagon 698 DSP, which consists of two engines that can handle AI processing exclusively. The first engine is the Hexagon Vector Extension (HVX), which is designed for advanced imaging and computer-vision tasks intended to run on the DSP instead of the CPU. The second, the company's AI-processor (AIP) cluster, supports the Hexagon Tensor Accelerator (HTA), which can also perform AI tasks. These engines can serve together for maximum performance, or they can operate in isolation (depending on the compiler optimizations). The Samsung chipset is an Exynos 990 [14] in the company's Galaxy Note 20 Ultra, which has a dual-core custom neural processing unit (NPU) specialized to handle AI workloads. In the laptop category, Intel submitted results for its new Willow Cove CPU [27] and first-generation integrated Xe-LP GPU. That GPU served as the AI accelerator [58]. These systems collectively reflect the state of the art in AI processors.

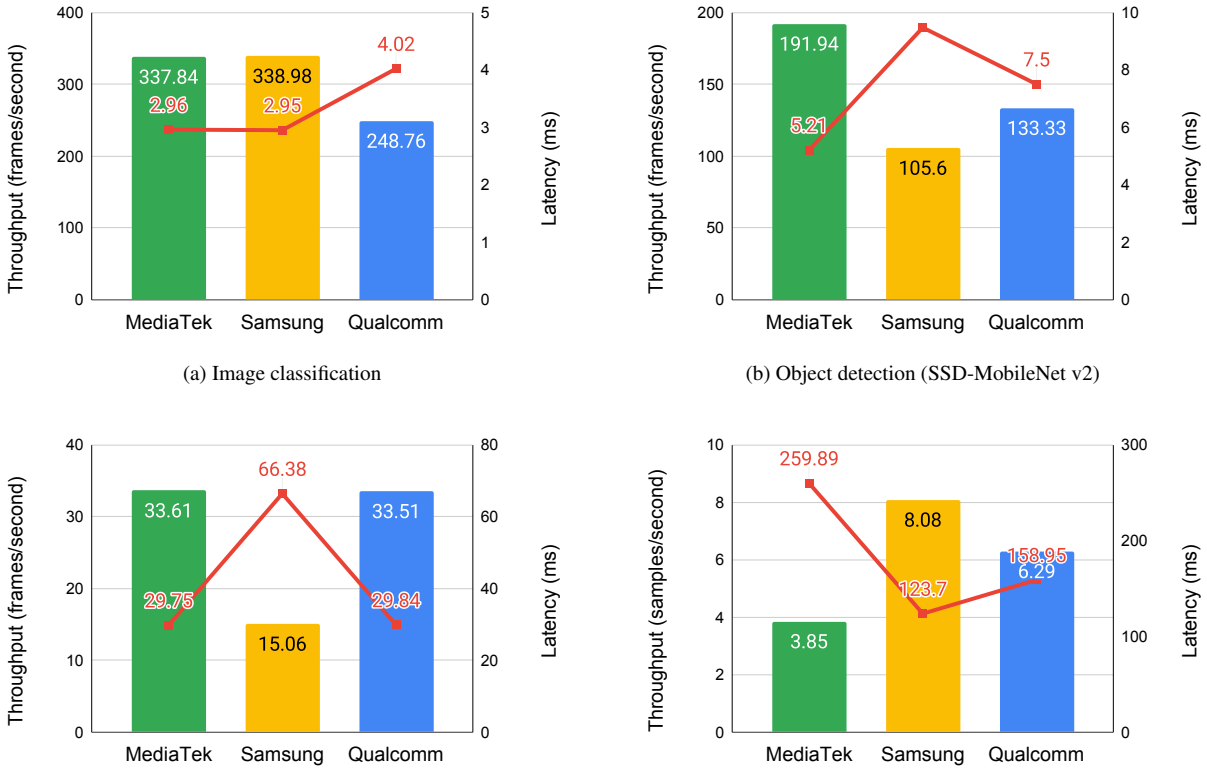
In the smartphone category, three organizations submitted a total of 14 individual results. No one solution dominates all benchmarks. Figure 6 plots the single-stream results for the three smartphone chipsets on each benchmark task. It includes both throughput and latency results. Each chipset offers 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 employs offline mode, which allows batch processing; here, Exynos delivered 674.4 frames per second (FPS), and Snapdragon delivered 605.37 FPS (not shown in Figure 6). In most cases, the throughput differences are marginal. An essential point to keep in mind, however, is that other metrics—beyond performance benchmarks—go into assessing a chipset's viability for a given task.

5.2 Result Transparency

The submission results highlight an important point: they reflect the variety of hardware and software combinations we discussed earlier (Section 2). All mobile SoCs rely on a generic processor, but the AI-performance results were from AI accelerators using different software frameworks. Transparency into how the results were generated is crucial.

Figure 7 shows the potential code paths for producing the submission results. The dashed lines represent mere possibilities, whereas the solid lines indicate actual submissions. Looking only at Figure 7 is insufficient to determine which paths produce high-quality results. Any other code paths would have yielded a different performance result. Therefore, transparency on benchmark performance is essential. It reveals which code paths were taken, making the performance results reproducible and informative for consumers.

Table 2 presents additional details, including specifics for each benchmark result in both single-stream and offline modes. MLPerf Mobile exposes this information to make the results reproducible. For each benchmark and each submitting organization, the table shows the numerical precision, the run time, and the hardware unit that produced the results. Exposing each of these details is important because the many execution paths in Figure 7 can drastically affect a device's performance.



(a) Image classification

(b) Object detection (SSD-MobileNet v2)

(c) Semantic segmentation (DeepLab v3 + MobileNet v2)

(d) Natural-language processing (MobileBERT)

Figure 6: Results from the first MLPerf Mobile round show that no one solution fits all tasks. The bars corresponds to throughput (left y-axis). The line graph corresponds to latency (right y-axis).

5.3 Execution Diversity

Mobile-device designers prefer INT8 or FP16 format because quantized inference runs faster and provides better performance and memory bandwidth than FP32 [34]. The accuracy tradeoff for quantized models (especially since no retraining is allowed) is tolerable in smartphones, which seldom perform safety-critical tasks, such as those in autonomous vehicles (e.g., detecting pedestrians).

All the mobile-vision tasks employ INT8 heavily. Most vendors rely on INT8 because it enables greater performance and consumes less power, preserving the device's battery life. NLP favors FP16. Although this format requires more power than INT8, it offers better accuracy. Perhaps more importantly, submitters use FP16 because most AI engines today lack efficient support for non-vision tasks. The GPU is a good balance between flexibility and efficiency. Unsurprisingly, therefore, all vendors submitted results that employed GPUs with FP16 precision for NLP.

NNAPI is designed to be a common baseline for machine learning 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 use proprietary frameworks. These frameworks, such as ENN and SNPE, give SoC vendors more control over their product's performance. For instance, they can control which processor (CPU, GPU, DSP, NPU, for example) to use 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 a single sample is available per query some models cannot fully utilize the computational resources of the GPU. Therefore, the backend must select between the CPU and GPU to deliver the best overall performance. For example, smaller models such as MobileNetEdgeTPU use the CPU. For the offline mode, multiple samples are available as a single query, so inference employs both the CPU and GPU.

Finally is hardware diversity. Table 2 shows a variety of hardware combinations that achieve good performance on all MLPerf Mobile AI tasks. In one case, the CPU is the backbone, orchestrating overall execution—including preprocessing and other tasks the benchmark does not measure. In contrast, the GPU, DSPs, NPUs, and AIPs deliver

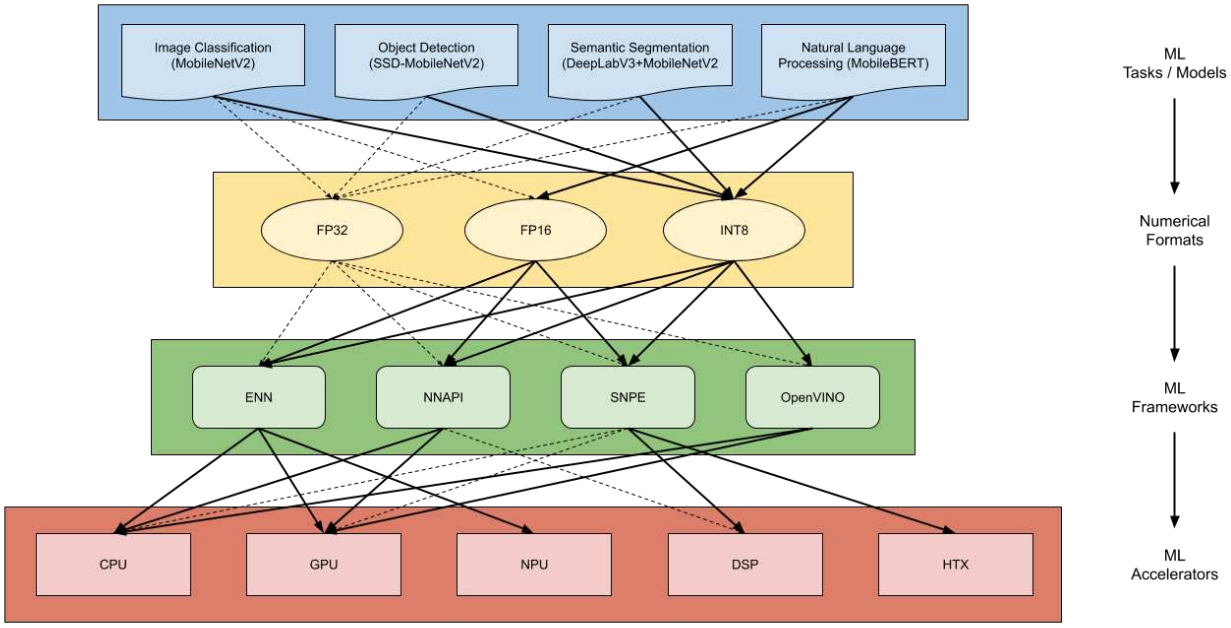


Figure 7: Potential code paths (dashed lines) and actual submitted code paths (solid lines) for producing MLPerf Mobile AI-performance results. NPU is the Neural Processing Unit from Samsung. Hexagon Tensor Accelerator (HTA) and Hexagon Vector eXtensions (HVX) are part of the Qualcomm DSP that can either be used individually or together simultaneously.

high-performance AI execution.

5.4 Summary

The MLPerf results provide transparency into the performance results, which show how SoC vendors achieve their best performance on a range of tasks. Figure 7 and Table 2 reveal substantial differences in how AI systems perform on the different devices. Awareness of such underlying variations is crucial because the measured performance should match what end users experience, particularly on commercially available devices.

Finally, since the benchmark models represent diverse tasks, and since MLPerf Mobile collects results over a single long run that covers all of these models, it strongly curbs domain-specific framework optimizations. Furthermore, the benchmarked mobile devices are commonly available and the testing conditions ensure a realistic experimental setup, so the results are attainable in practice and reproducible by others.

6 Consumer, Industry, Research Value

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

MLPerf Mobile focuses on transparency for consumers by packaging the submitted code into an app. 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). The application currently runs on Android, though future versions will likely support iOS as well.

We believe that analysts, OEMs, academic researchers, neural-network-model designers, application developers, and smartphone users can all gain from result transparency. We briefly summarize how the app benefits each one.

Application developers. MLPerf Mobile shows application developers what real-world performance may look like on the device. For application developers, we expect the benchmark provides insight into the software frameworks on the various "phones" (i.e., SoCs). More specifically, it 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 perspective on

	Image Classification (single-stream)	**Image Classification (offline)**	**Object Detection (single-stream)**	**Image Segmentation (single-stream)**	**Natural-Language Processing (single-stream)**
	ImageNet	*ImageNet*	*COCO*	*ADE20K*	*Squad*
	MobileNetEdge	*MobileNetEdge*	*SSD-MobileNet v2*	*DeepLab v3+ - MobileNet v2*	*MobileBERT*
MediaTek (smartphone)	UINT8, NNAPI (neuron-ann), APU	N/A	UINT8, NNAPI (neuron-ann), APU	UINT8, NNAPI (neuron-ann), APU	FP16, TFLite delegate, Mali-GPU
Samsung (smartphone)	INT8, ENN, (NPU, CPU)	INT8, ENN, (NPU, CPU)	INT8, ENN, (NPU, CPU)	INT8, ENN, (NPU, GPU)	FP16, ENN, GPU
Qualcomm (smartphone)	UINT8, SNPE, HTA	UINT8, SNPE, AIP (HTA+HVX)	UINT8, SNPE, HTA	UINT8, SNPE, HTA	FP16, TFLite delegate GPU
Intel (laptop)	INT8, OpenVINO, CPU	INT8, OpenVINO, CPU+GPU	INT8, OpenVINO, CPU	INT8, OpenVINO, GPU	INT8, OpenVINO, GPU

Table 2: Implementation details for the results presented in Figure 7. The table shows the myriad combinations of numerical formats, software run times and hardware backend targets that are possible, which reinforces the need for result transparency.

the end-user experience for a real application.

OEMs. MLPerf Mobile standardizes the benchmarking method across different mobile SoCs. All 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 our benchmark provides is vital.

Model designers. MLPerf Mobile makes it easy to package new models into the mobile app, which organizations can then easily share and reproduce. The app framework, coupled with the underlying 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 thus far failed to share information in an efficient and effective manner.

Mobile users. The average end user wants to make informed purchases. For instance, many 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.

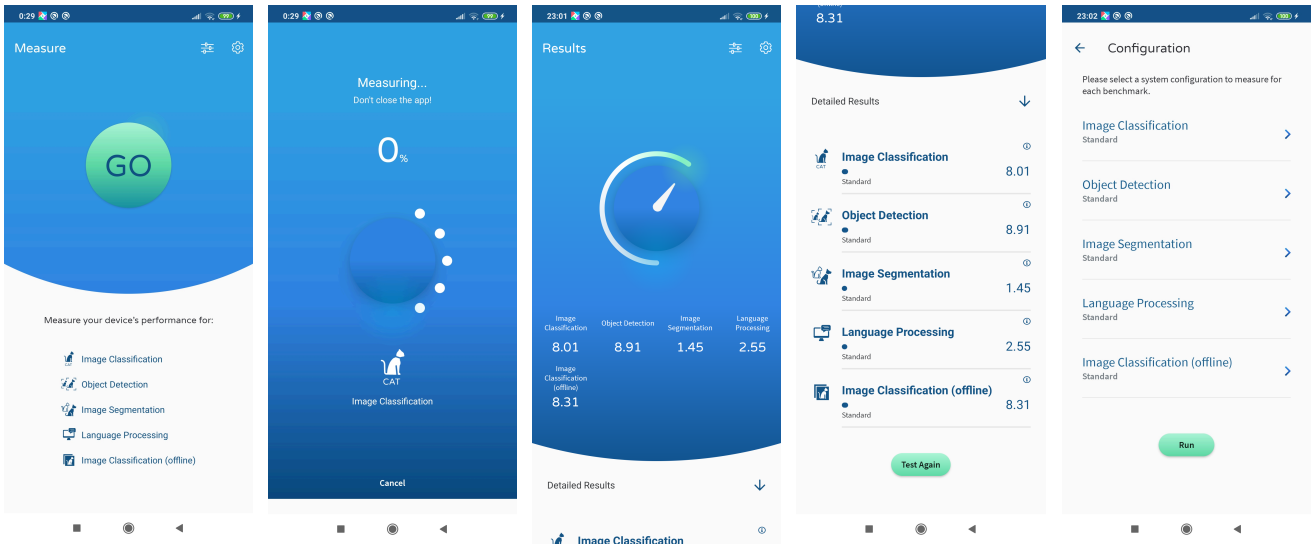
Academic researchers. Reproducibility is a challenge for state-of-the-art technologies. We hope researchers employ our mobile-app framework to test their methods and techniques for improving model performance, quality, or both. The framework is open source and freely accessible. As such, it can enable academic researchers to integrate their optimizations and reproduce more recent results from the literature.

Technical analysts. MLPerf Mobile provides reproducibility and transparency for technical analysts, who often strive to make "apples-to-apples" comparisons. The application makes it easy to reproduce vendor-claimed results as well as to interpret the results, because it shows how the device achieves a particular performance number and how it is using the hardware accelerator.

7 Related Work

There are many ongoing efforts in mobile AI performance benchmarking. We describe the prior art in mobile and ML benchmarking and emphasize how MLPerf Mobile differs from these related works.

Android Machine Learning Test Suite (MLTS). MLTS, part of the Android Open Source Project (AOSP) source tree, provides benchmarks for NNAPI drivers [16]. It is mainly for testing the accuracy of vendor NNAPI drivers. MLTS includes an app that allows a user to test the latency and accuracy of quantized and floating-point TFLite models (e.g., MobileNet and SSD-MobileNet) against a 1,500-



(a) Startup screen. (b) Running the benchmarks. (c) Reporting results. (d) Run details. (e) Configuration settings.

Figure 8: MLPerf Mobile app on Android.

image subset of the Open Images Dataset v4 [40]. Further statistics, including latency distributions, are also available.

Xiaomi's Mobile AI Benchmark. Xiaomi provides an open-source end-to-end benchmark tool for evaluating model accuracy and latency [13]. In addition to a command-line utility to run the benchmarks on a user device, the tool includes a daily performance-benchmark run for various neural-network models (mostly on the Xiaomi Redmi K30 Pro). The tool has a configurable backend that allows users to employ multiple ML-hardware-delegation frameworks (including MACE, SNPE, and TFLite).

TensorFlow Lite. TFLite provides a command-line benchmark utility to measure the latency of any TFLite model [24]. A wrapper APK 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 backend. For in-depth performance analysis, the benchmark supports timing of individual TFLite operators.

AI-Benchmark. Ignatov et al. [37] performed an extensive evaluation of machine-learning performance on mobile systems with AI acceleration, using HiSilicon, MediaTek, Qualcomm, Samsung, and UniSoc chipsets. They evaluated 21 deep-learning tasks using 50 metrics, including inference speed, accuracy, and stability. The authors reported the results of their AI-Benchmark app for 100 mobile SoCs. The benchmark runs preselected models of various bit widths (INT8, FP16, and FP32) on the CPU and on open-source or vendor-proprietary TFLite delegates. Performance-report updates appear on the AI-Benchmark website [1] after each major release of TFLite/NNAPI and of new SoCs with AI acceleration.

AImark. Master Lu (Ludashi) [2], a closed-sourced Android and iOS application, uses vendor SDKs to implement its benchmarks. It comprises image-classification, image-recognition, and image-segmentation tasks, including models such as ResNet-34 [35], Inception V3 [56], SSD-MobileNet [36, 43], and DeepLab v3+ [30]. The benchmark judges mobile-phone AI performance by evaluating recognition efficiency and provides a line-test score.

Aitutu. A closed-source application [3, 8], Aitutu employs Qualcomm's SNPE, MediaTek's NeuroPilot, HiSilicon's Kirin HiAI, Nvidia's TensorRT, and other vendor SDKs. It implements image classification based on the Inception V3 neural network [56], using 200 images as test data. The object-detection model is based on SSD-MobileNet [36, 43], using a 600-frame video as test data. The score is a measure of speed and accuracy—faster results with higher accuracy yield a greater final score.

Geekbench. Primate Labs created Geekbench [20, 6], a cross-platform CPU-compute benchmark that supports Android, iOS, Linux, macOS, and Windows. The Geekbench 5 CPU benchmark features new applications, including augmented reality and machine learning, but it lacks heterogeneous-IP support. Users can share their results by uploading them to the Geekbench Browser.

UL Procyon AI Inference Benchmark. From UL Benchmarks, which produced PCMark and 3DMark, came VRMark [25, 26], an Android NNAPI CPU- and GPU-focused AI benchmark. The professional benchmark suite UL Procyon only compares NNAPI implementations and compatibility on floating-point- and integer-optimized mod-

els. It contains MobileNet v3 [28], Inception v4 [56], SSDLite MobileNet v3 [28, 43], DeepLab v3 [30], and other models. It also attempts to test custom CNN models but uses an AlexNet [39] architecture to test basic operations. The application provides benchmark scores, performance charts, hardware monitoring, model output, and device rankings.

Neural Scope. National Chiao Tung University [17, 18] developed an Android NNAPI application supporting FP32 and INT8 precisions. The benchmarks comprise object classification, object detection, and object segmentation, including MobileNet v2 [51], ResNet-50 [35], Inception v3, SSD-MobileNet [36, 43], and ResNet-50 with atrous-convolution layers [29]. Users can run the app on their mobile devices and immediately receive a cost-performance comparison.

8 Future Work

The first iteration of the MLPerf Mobile benchmark focused on the foundations. On the basis of these fundamentals, its scope can easily expand. The following are areas of future work:

iOS support. A major area of interest for MLPerf Mobile is to develop an iOS counterpart for the first-generation Android app. Apple's iOS is a major AI-performance player that brings both hardware and software diversity compared with Android.

Measuring software frameworks. Most AI benchmarks focus on AI-hardware performance. But as we described in Section 2, software performance—and, more importantly, its capabilities—is crucial to unlocking a device's full potential. To this end, enabling apples-to-apples comparison of software frameworks on a fixed hardware platform has merit. The backend code path in Figure 5 (code path 1) is a way to integrate different machine-learning frameworks in order to determine which one achieves the best performance on a target device.

Expanding the benchmarks. An obvious area of improvement is expanding the scope of the benchmarks to include more tasks and models, along with different quality targets. Examples include additional vision tasks, such as super resolution, and speech models, such as RNN-T.

Rolling submissions. The mobile industry is growing and evolving rapidly. New devices arrive frequently, often in between MLPerf calls for submissions. MLPerf Mobile therefore plans to add "rolling submissions" in order to encourage vendors to submit their MLPerf Mobile scores continuously. Doing so would allow smartphone makers to more consistently use the benchmark to report the AI performance of their latest devices.

Power measurement. A major area of potential improvement for MLPerf Mobile is power measurement. Since mobile devices are battery constrained, evaluating AI's power draw is important.

To make additional progress, we need community involvement. We therefore encourage the broader mobile community to join the MLPerf effort and maintain the momentum behind an industry-standard open-source mobile benchmark.

9 Conclusion

Machine-learning inference has many potential applications. Building a benchmark that encapsulates this broad spectrum is challenging. In this paper, we focused on smartphones and the mobile-PC ecosystem, which is rife with hardware and software heterogeneity. Coupled with the life-cycle complexities of mobile deployments, this heterogeneity makes benchmarking mobile AI performance overwhelmingly difficult. To bring consensus, we developed the MLPerf Mobile AI inference benchmark. Many leading organizations have joined us in building a unified benchmark that meets competing organizations' disparate needs. The unique value of MLPerf Mobile is not so much in the benchmarks, rules, and metrics. Instead, it is in the value that the industry creates for itself, benefiting everyone.

MLPerf Mobile provides an open source, out-of-the-box inference-throughput benchmark for popular computer-vision and natural-language-processing applications on mobile devices, including smartphones and laptops. It can serve as a framework to integrate future models, as the underlying framework is independent of the top-level model and of data-set changes. The app and the integrated Load Generator allow us to evaluate a variety of situations, such as changing the quality thresholds for overall system performance. The app can also serve as a common platform for comparing different machine-learning frameworks on the same hardware. Finally, the suite allows for fair and faithful evaluation of heterogeneous hardware, with full reproducibility.

Acknowledgements

The MLPerf Mobile team would like to acknowledge several people for their effort. In addition to the team that architected the benchmark, MLPerf Mobile is the work of many individuals that also helped produce the first set of results.

Arm: Ian Forsyth, James Hartley, Simon Holland, Ray Hwang, Ajay Joshi, Dennis Laudick, Colin Osborne, and Shultz Wang.

dviditi: Anton Lokhmotov.

Google: Bo Chen, Suyog Gupta, Andrew Howard, and Jaeyoun Kim.

Harvard University: Yu-Shun Hsiao.

Intel: Thomas Baker, Srujana Gattupalli, and Maxim Shevtsov.

MediaTek: Kyle Guan-Yu Chen, Allen Lu, Ulia Tseng, and Perry Wang.

Qualcomm: Mohit Mundhra.

Samsung: Dongwoon Bai, Stefan Bahrenburg, Jihoon Bang, Long Bao, Yoni Ben-Harush, Yoojin Choi, Fangming He, Amit Knoll, Jaegon Kim, Jungwon Lee, Sukhwan Lim, Yoav Noor, Muez Reda, Hai Su, Zengzeng Sun, Shuangquan Wang, Maiyuran Wijay, Meng Yu, and George Zhou.

Xored: Ivan Osipov, and Daniil Efremo.

References

[1] AI-Benchmark. http://ai-benchmark.com/.

[2] AImark. https://play.google.com/store/apps/details?id=com.ludashi.aibench&hl=en_US.

[3] Antutu Benchmark. https://www.antutu.com/en/index.htm.

[4] Big.LITTLE. https://www.arm.com/why-arm/technologies/big-little.

[5] Deploy High-Performance Deep Learning Inference. https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html.

[6] Geekbench. https://www.geekbench.com/.

[7] Google Play. https://play.google.com/store.

[8] Is Your Mobile Phone Smart? Antutu AI Benchmark Public Beta Is Released. https://www.antutu.com/en/doc/117070.htm#:~:text=In%20order%20to%20provide%20you,AI%20performances%20between%20different%20platforms.

[9] LoadGen. https://github.com/mlperf/inference/tree/master/loadgen.

[10] MediaTek Dimensity 820. https://www.mediatek.com/products/smartphones/dimensity-820.

[11] MLPerf. https://github.com/mlperf.

[12] MLPerf Mobile v0.7 Results. https://mlperf.org/inference-results/.

[13] Mobile AI Bench. https://github.com/XiaoMi/mobile-ai-bench.

[14] Mobile Processor Exynos 990. https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-990/.

[15] Neural Networks API. https://developer.android.com/ndk/guides/neuralnetworks.

[16] Neural Networks API Drivers. https://source.android.com/devices/neural-networks#mlts.

[17] NeuralScope Mobile AI Benchmark Suite. https://play.google.com/store/apps/details?id=org.aibench.neuralscope.

[18] Neuralscope offers you benchmarking your AI solutions. https://neuralscope.org/mobile/index.php?route=information/info.

[19] NeuroPilot. https://neuropilot.mediatek.com/.

[20] Primate Labs. https://www.primatelabs.com/.

[21] Samsung Neural SDK. https://developer.samsung.com/neural/overview.html.

[22] Snapdragon 865+ 5G Mobile Platform. https://www.qualcomm.com/products/snapdragon-865-plus-5g-mobile-platform.

[23] Snapdragon Neural Processing Engine SDK. https://developer.qualcomm.com/docs/snpe/overview.html.

[24] TensorFlow Lite. https://www.tensorflow.org/lite.

[25] UL Benchmarks. https://benchmarks.ul.com/.

[26] UL Procyon AI Inference Benchmark. https://benchmarks.ul.com/procyon/ai-inference-benchmark.

[27] Willow cove - microarchitectures - intel. https://en.wikichip.org/wiki/intel/microarchitectures/willow_cove#:~:text=Willow%20Cove%20is%20the%20successor,client%20products%2C%20including%20Tiger%20Lake.

[28] Andrew Howard, Suyog Gupta. Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU. https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html.

[29] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[30] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[31] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning, 2015.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[33] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture, 2015.

[34] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[36] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[37] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc

Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3617–3635. IEEE, 2019.

[38] W. Kim and J. Seok. Indoor semantic segmentation for robot navigating on mobile. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 22–25, 2018.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[40] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, and et al. The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981, Mar 2020.

[41] Chien-Hung Lin, Chih-Chung Cheng, Yi-Min Tsai, Sheng-Je Hung, Yu-Ting Kuo, Perry H Wang, Pei-Kuei Tsung, Jeng-Yun Hsu, Wei-Chih Lai, Chia-Hung Liu, 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)*, pages 134–136. IEEE, 2020.

[42] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[43] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[44] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[45] Natalia Neverova, Pauline Luc, Camille Couprie, Jakob J. Verbeek, and Yann LeCun. Predicting deeper into the future of semantic segmentation. *CoRR*, abs/1703.07684, 2017.

[46] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[47] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

[48] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[50] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[51] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[52] Jamie Sherrah. Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery, 2016.

[53] Mennatullah Siam, Sara Elkerdawy, Martin Jagersand, and Senthil Yogamani. Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. In *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*, pages 1–8. IEEE, 2017.

[54] G. Sun and H. Lin. Robotic grasping using semantic segmentation and primitive geometric model based 3d pose estimation. In *2020 IEEE/SICE International Symposium on System Integration (SII)*, pages 337–342, 2020.

[55] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.

[56] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[57] Saeid Asgari Taghanaki, Kumar Abhishek, Joseph Paul Cohen, Julien Cohen-Adad, and Ghassan Hamarneh. Deep semantic segmentation of natural and medical images: A review, 2020.

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

[59] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.