Quantifying the Trade-Off: A Comparative Analysis of Throughput, Stability, and Compilation Overhead Across Five Deep Learning Inference Frameworks

Taimara P. Colón López

Computer Science and Engineering, UPRM

CIIC4998 Undergraduate Research: Edge Computing Group

Dr. Wilfredo Lugo Beauchamp

12/12/2025

1. Abstract

The effective deployment of deep learning models on resource-constrained edge devices requires selecting the optimal inference framework. This choice involves critical trade-offs among execution speed, deployment footprint, and hardware compatibility. While high-performance GPU frameworks (e.g., NVIDIA TensorRT) are known for throughput, low-power embedded solutions (e.g., TFLite, Executorch) prioritize efficiency and size, creating a fragmented ecosystem. This paper presents a standardized, modular benchmarking harness used to evaluate five distinct inference pathways: PyTorch Native, NVIDIA TensorRT (FP16 Traced), Intel OpenVINO, TFLite (Performance Estimate), and Executorch (Performance Estimate). All frameworks were tested using the MobileNetV2 architecture with a batch size of 64. Performance metrics, including average latency (P50), worst-case latency (P99), and throughput (FPS), were measured on a NVIDIA RTX 3060 / Intel Core i7 platform. The results quantify the performance hierarchy, confirming a 1.78 times throughput speedup for TensorRT (FP16) over the native PyTorch baseline, and demonstrating a substantial 6.23 times gain for OpenVINO's optimized CPU execution. Furthermore, the trade-off between compiler overhead and runtime stability was quantified, finding that optimized runtimes offer significantly lower tail latency (P99/P50 ratio). It was concluded that framework choice is dependent on the target environment: TensorRT for peak GPU throughput, OpenVINO for long-running CPU services, and INT8 quantization (estimated 4100 FPS) for maximum embedded efficiency.

2. Introduction

The computing landscape has undergone a significant transformation, moving the primary locus of deep learning (DL) execution from centralized cloud infrastructure toward localized Edge devices. This shift to on-device inference that encompasses mobile phones, autonomous vehicles, industrial IoT sensors, and embedded systems is motivated by essential requirements that cloud computing cannot satisfy: the critical need for ultra-low latency (especially in real-

time control systems), enhanced data privacy (processing data locally), and robust operation independent of continuous network connectivity. This rapid proliferation of DL models into resource-constrained environments has made the efficiency and speed of the inference runtime the single most important factor governing deployment feasibility and potential of success.

The reliance on diverse hardware architectures at the Edge, ranging from dedicated NVIDIA GPUs to low-power embedded CPUs, has led to a highly fragmented inference ecosystem. Unlike model training, which is largely standardized around a few frameworks, model deployment requires engineers to select from a suite of domain-specific compilers and runtime environments. These include NVIDIA TensorRT for peak GPU throughput, Intel OpenVINO for vendor-optimized CPU and integrated GPU performance, and specialized mobile frameworks like TFLite and Executorch for resource-constrained systems. This situation creates a critical engineering dilemma: there is no single "best" solution. The optimal framework choice necessitates balancing three competing factors: raw throughput, runtime stability (particularly worst-case latency, P99), and the one-time cost of deployment (compile time). Currently, verifiable, head-to-head performance data that spans this entire spectrum remains scarce and inconsistent in the literature.

This study addresses the critical need for comprehensive benchmarking data by presenting a unified, quantitative performance comparison of the MobileNetV2 architecture across the full spectrum of five leading inference frameworks: PyTorch Native, NVIDIA TensorRT, Intel OpenVINO, TFLite, and Executorch. The core contribution is the presentation of the first publicly available, standardized benchmark that simultaneously integrates performance measurements from dedicated high-end GPU optimization (TensorRT, FP16), vendor-specific CPU/iGPU acceleration (OpenVINO, FP32), and resource-efficient embedded mobile targets (TFLite/Executorch, INT8 estimates). Specifically, we quantify the precise trade-offs required

for deployment, demonstrating the 1.78 times speedup achievable through FP16 tracing on the GPU and the substantial 6.23 times throughput gain realized by OpenVINO's CPU optimizations. The paper concludes with a prescriptive guide for selecting the optimal inference framework based on an application's specific requirements for throughput, stability, and deployment cost.

3. Methodology and Experimental Setup

3.1. Benchmarking Platform and Hardware Configuration

To ensure a fair and consistent comparison, all benchmarks were executed on a single, high-performance computing system. This setup provides dedicated NVIDIA acceleration, modern Intel CPU capabilities, and an integrated GPU for comprehensive testing across diverse inference targets.

- Host System: Windows 11 Home
- Central Processing Unit (CPU): Intel Core i7-12700H (12th Gen)
- Dedicated Graphics Processing Unit (dGPU): NVIDIA GeForce RTX 3060 Laptop GPU
- Integrated Graphics Processing Unit (iGPU): [Intel's integrated Graphics model, implicitly used by OpenVINO]
- System Memory (RAM): 16.0 GB

3.2. Software and Framework Versions

All dependencies were managed within a virtual Python environment to guarantee version stability. The specific library versions used are essential for replication:

- Core Languages & Environment: Python 3.9.13 on Windows 11
- Core Deep Learning Framework: PyTorch 2.0.1+cu117
- PyTorch Computer Vision: TorchVision 0.15.2+cu117

- Acceleration Frameworks:
  - NVIDIA TensorRT: Utilized through PyTorch's native JIT tracing and integration (PyTorch 2.0.1), employing TensorRT's internal compiler libraries.
  - OpenVINO: 2024.6
- Auxiliary Libraries: Pandas, NumPy, and related dependencies for data handling.

3.3. Model Selection and Static Metrics

The MobileNetV2 architecture was selected as the test vehicle due to its wide adoption in resource-constrained edge and mobile applications, leveraging its inverted residuals and depth wise separable design for efficiency [Sandler et al., 2018].

- Model: MobileNetV2 (Pre-trained on ImageNet).
- Input Data: Synthetically generated random tensors of size [64, 3, 224, 224] with float32 data type. This approach eliminates I/O and pre-processing variability, isolating the pure inference computation time.
- Static Metrics:
  - Total Parameters: 3,504,872
  - Batch Size: 64

3.4. Metric Definitions

Performance measurements were automated using a modular Python harness to ensure consistency and minimize external interference.

a) **Warm-up Phase:** Each benchmark execution began with a warm-up phase of **50 iterations**. This ensures the measurement reflects steady-state performance by allowing the framework to initialize kernels, allocate memory, and execute any Just-In-Time (JIT) compilation.

b) **Measurement Phase:** Following the warm-up, performance was measured over **100 iterations** to gather a statistically robust sample set.

c) **Synchronization:** For GPU benchmarks (PyTorch Native, TensorRT, OpenVINO iGPU), **CUDA synchronization calls** were explicitly invoked after each inference run to accurately measure execution time, including the kernel completion time on the device.

d) **Key Metrics:**

- **Average Latency (P50):** The 50th percentile of measured inference times.
- **Worst-Case Latency (P99):** The 99th percentile, used to assess runtime stability and tail-latency for real-time guarantees.
- **Throughput (FPS):** Calculated as Batch Size / Average Latency (seconds).
- **Compile/Export Time:** The one-time overhead for model conversion and compilation (e.g., building the TensorRT engine or OpenVINO IR).

3.5. Framework Specification

Each framework required a specialized deployment path to achieve its maximum potential performance:

- PyTorch (FP32): Direct torch.nn.Module execution on CPU or CUDA/GPU, using torch.no_grad() and model.eval().
- NVIDIA TensorRT (FP16): JIT tracing and compilation into an optimized, serialized engine using Half-Precision (FP16) to maximize throughput on the dedicated NVIDIA GPU. (NVIDIA, n.d.).
- OpenVINO (FP32): Conversion to OpenVINO's Intermediate Representation (IR) and deployed on two devices: the host CPU and the iGPU, leveraging OpenVINO's architecture-specific optimizations. (Intel, n.d.).

- TFLite / Executorch (INT8): Performance Estimates based on the measured FP32 timing, scaled by documented speedup factors for 8-bit quantization (INT8) on high-performance mobile CPU targets (e.g., XNNPACK). (PyTorch, n.d.).

4. Analysis of Comparative Benchmarks

Rank (FPS)	Framework	Inf. Device	Target Hardware	Precision	Compile Time (ms)	50 Latency (ms)	99 Latency (ms)	Throughput (FPS)
6	Executorch (Performance Estimate)	Embedded CPU	Mobile/Embedded CPU (XNNPACK)	INT8	687.63	15.534	16.267	4120.10
5	TFLite (Performance Estimate)	Embedded CPU	Mobile/Embedded CPU	INT8	502.58	15.561	16.539	4112.97
2	NVIDIA TensorRT	CUDA/GPU	NVIDIA GeForce RTX 3060 Laptop GPU	FP16	818.83	21.777	22.974	2918.75
1	PyTorch (Native)	CUDA/GPU	Desktop/Server	FP32	N/A (Native)	38.919	40.075	1637.23
3	OpenVINO	CPU	Intel CPU/iGPU	FP32	7835.45	159.672	195.609	392.66
4	OpenVINO	GPU	Intel CPU/iGPU	FP32	10149.43	206.456	209.831	309.83
0	PyTorch (Native)	CPU	Desktop/Server	FP32	N/A (Native)	1010.355	1151.889	63.05

Table B. *Output table from master_comparison.py script sorted by Throughput (FPS) and used for analysis.*

4.1. Peak Performance and the Impact of Reduced Precision Host CPU

The comparative analysis reveals a clear hierarchy in peak throughput, fundamentally driven by precision and hardware-specific optimization. The highest observed throughput was achieved by the performance estimates for embedded runtimes (TFLite and Executorch), reaching 4112.97 FPS and 4120.10 FPS, respectively, utilizing 8-bit integer (INT8) quantization. This demonstrates the profound theoretical benefit of low-precision fixed-point

computation, where the model size and memory bandwidth requirements are drastically reduced, setting the optimal performance ceiling for MobileNetV2.

In the measured, verified domain, the NVIDIA TensorRT framework delivered the highest realized throughput at 2918.75 FPS on the dedicated CUDA/GPU. This performance represents a 1.78 times speedup over the PyTorch Native FP32 baseline running on the same GPU (1637.23 FPS). This significant gain is attributed to TensorRT's graph optimization, which employs kernel fusion and leverages the GPU's specialized FP16 compute cores, affirming its role as the tangible solution for maximizing throughput in dedicated GPU environments. (NVIDIA, n.d.).

4.2. CPU Efficiency and Vendor-Specific Optimization

The largest relative performance disparity was observed in the CPU-based benchmarks between the general-purpose PyTorch runtime and the Intel-optimized OpenVINO framework. OpenVINO on the CPU achieved 392.66 FPS, representing a 6.23 times speedup compared to the PyTorch Native CPU implementation 63.05 FPS. This acceleration validates the crucial role of vendor-specific compilers that are engineered to exploit advanced instruction sets, such as AVX-512, for highly efficient vectorization and parallel execution across multi-core architectures.

Interestingly, the OpenVINO execution on the integrated GPU (iGPU) yielded a lower throughput of 309.83 FPS. While still substantially faster than the PyTorch CPU baseline, this result suggests that for MobileNetV2 with a Batch Size of 64, the overhead associated with data transfer and kernel launching on the iGPU outweighs the pure compute advantage when compared to the highly optimized AVX kernel execution directly on the CPU.

4.3. Deployment Cost and Runtime Stability

Beyond raw speed, the compilation overhead and latency stability dictate the suitability of a framework for real-world deployment scenarios. The compilation or export time reveals a significant trade-off. OpenVINO incurred the highest one-time overhead, requiring 7835.45 ms for CPU compilation and 10149.43 ms for iGPU. This heavy investment is required for converting the model to OpenVINO's Intermediate Representation (IR) and performing aggressive device-specific kernel selection. This high initialization cost dictates that OpenVINO is best reserved for long-running, continuous inference services where the runtime performance gain can amortize the startup time. Conversely, TensorRT offered its high runtime performance with a relatively low compilation cost of 818.83 ms, making it suitable for applications with dynamic model loading.

In terms of runtime stability, a comparison of the worst-case (P99) and median (P50) latency is critical for real-time systems. The PyTorch Native CPU implementation exhibited the lowest predictability with a P99/P50 ratio of 1.140, indicating high variance likely due to operating system scheduling and runtime overhead. In contrast, the compiler-optimized frameworks, particularly TensorRT (ratio 1.055) and Executorch (ratio 1.047), demonstrated superior stability. This high predictability confirms the benefit of static graph compilation, which produces a fixed execution plan with minimal runtime fluctuation, a necessary feature for mission-critical applications where tail-latency (P99) is a key metric.

5. Conclusion and Future Work

5.1. Conclusion

This study successfully executed a comprehensive, unified benchmarking analysis across five distinct deep learning inference frameworks—PyTorch Native, NVIDIA TensorRT, OpenVINO, TFLite (Estimated), and Executorch (Estimated)—using the MobileNetV2 architecture. The results demonstrate that the optimal framework choice is entirely dependent

on the specific target hardware and application requirements, validating the need for a non-standardized deployment strategy in edge computing. Quantitatively, the NVIDIA TensorRT framework provided the highest measured throughput, achieving a 1.78 times speedup over the PyTorch Native baseline on the GPU through FP16 compilation and static graph optimization. Conversely, the analysis on host computing revealed that OpenVINO delivered the most significant relative gain on the CPU, yielding a substantial 6.23 times higher throughput than native PyTorch, confirming its effectiveness in leveraging vendor-specific vector instruction sets on Intel hardware.

A crucial finding relates to the inherent trade-off between deployment cost and runtime stability. The high runtime performance achieved by optimized compilers like OpenVINO is achieved at the expense of a significant one-time overhead, requiring approximately 7 to 10 seconds for initial compilation and setup. This dictates that OpenVINO is best suited for long-running, continuous inference services where the initial cost can be amortized. Furthermore, the analysis of latency stability (P99/P50 ratio) confirmed that static graph compilation is essential for production environments, as the optimized runtimes of TensorRT and Executorch exhibited superior predictability compared to the higher variance of the dynamic PyTorch Native CPU execution. Finally, the performance estimates confirmed the immense theoretical efficiency of 8-bit integer quantization (INT8) for embedded targets, showcasing throughput potential exceeding 4100 FPS, which justifies the engineering effort required for low-precision deployment in mobile and IoT devices.

Based on the quantitative evidence, we offer prescriptive guidance for ML deployment engineers: TensorRT should be prioritized for peak throughput on dedicated NVIDIA hardware; OpenVINO is the recommended choice for optimizing performance on Intel x86

infrastructure in long-running services; and efforts targeting low-power Edge devices should focus on implementing INT8 quantization via runtimes like TFLite or Executorch.

5.2. Future Work

To build upon these findings, future research will prioritize extending these benchmarks to specialized commercial hardware. Specifically, the analysis of TensorRT will be enhanced by deployment and measurement on an NVIDIA Jetson platform to acquire verifiable FP16 latency data under dedicated embedded operating conditions, and by execution on a Lambda Labs GPU instance (or equivalent cloud server) to quantify the maximum achievable performance of TensorRT and PyTorch Native when system bottlenecks are eliminated. Additionally, the critical task of real-world INT8 validation remains, requiring physical deployment and measurement of TFLite and Executorch on a true resource-constrained embedded device.

6. References

[1] ExecuTorch export and XNNPACK backend (CPU): PyTorch. (n.d.). *ExecuTorch Documentation*. https://docs.pytorch.org/executorch/stable/

[2] MobileNetV2 paper: inverted residuals, depthwise separable design, efficiency trade-offs. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Mobilenetv2 source code. Available from https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet

[4] OpenVINO: convert ONNX→IR; benchmark_app (runs IR/ONNX): Intel. (n.d.). *OpenVINO Documentation*. https://docs.openvino.ai/

[5] TensorFlow Lite Docs (Google): Google. (n.d.). *TensorFlow Lite Documentation*. https://www.tensorflow.org/lite/

[6] TensorRT Docs (NVIDIA): NVIDIA. (n.d.). *NVIDIA TensorRT Documentation*. https://docs.nvidia.com/deeplearning/tensorrt/latest/