AI on the Edge: An Automated Pipeline for PyTorch-to-Android Deployment and Benchmarking

Saif U Din *, Muhammad Ahsan Hussain , Mohsin Ikram , Dmitry Ignatov , Radu Timofte

Posted Date: 24 November 2025

doi: 10.20944/preprints202511.1831.v1

Keywords: machine learning; mobile computing; Android; Benchmarking; PyTorch; TensorFlow Lite; edge AI; automated deployment

 doi:10.20944/preprints202511.1831.v1

Article

AI on the Edge: An Automated Pipeline for PyTorch-to-Android Deployment and Benchmarking

Saif U Din *, Muhammad Ahsan Hussain, Mohsin Ikram, Dmitry Ignatov and Radu Timofte

Computer Vision Lab, CAIDAS & IFI, University of Würzburg, Germany
* Correspondence: saif.saif-u-din@stud-mail.uni-wuerzburg.de

Abstract

The deployment of deep learning models on mobile devices is a cornerstone of modern AI applications. However, performance benchmarking in this domain remains a predominantly manual, time consuming, and non scalable process. This paper introduces a fully automated, end-to-end pipeline NN Lite https://github.com/ABrain-One/nn-lite that bridges the critical gap between model development in PyTorch and rigorous performance evaluation on the Android platform. Our system comprises a Python based orchestration framework that manages model conversion, emulator control, and data collection, working in tandem with a lightweight Android application for on device benchmarking. The orchestrator systematically converts PyTorch models to TensorFlow Lite format, deploys the benchmark application, executes inference tests, and retrieves detailed performance reports. The output is a collection of structured JSON reports containing precise inference latency metrics enriched with device specific hardware analytics. This framework eliminates manual intervention, ensures reproducibility, and provides a scalable solution for evaluating the on device performance of diverse neural network architectures. In a large scale evaluation, the system successfully processed over 7,500 models, demonstrating exceptional with 48+ hours of continuous unattended operation, thereby establishing a new standard for automated mobile ML testing infrastructure.

Keywords: machine learning; mobile computing; Android; Benchmarking; PyTorch; TensorFlow Lite; edge AI; automated deployment

1. Introduction

The proliferation of machine learning (ML) on mobile devices has created an urgent need for automated frameworks to validate model performance in diverse and resource constrained hardware environments. While frameworks like PyTorch Mobile [1] and TensorFlow Lite [2] have simplified the deployment process, the benchmarking phase often remains a manual bottleneck. Researchers and developers must individually convert, deploy, and test models a process that is not only tedious but also prone to inconsistencies, making large scale comparative studies impractical.

Existing work, such as that by Goodarzi et al. [3], highlights the complexity of the mobile performance landscape, while Kochnev et al. [4] have explored efficient model design. However, a comprehensive, automated system for end-to-end evaluation is still lacking. This gap hinders rapid iteration and data driven decision making for mobile ML deployment.

To address this challenge, we present a fully automated system for continuous model deployment and validation. Our framework features a Python based infrastructure that handles the end-to-end conversion of PyTorch models to TensorFlow Lite, systematic benchmarking on Android emulators, and comprehensive analytics collection. A key innovation is its state management, which enables resumable processing of thousands of models and sophisticated failure recovery mechanisms. This integrated solution ensures reliable deployment and validation of ML models across mobile platforms, providing a scalable foundation for scientific research and industrial application. The system's efficacy is demonstrated by its ability to process over 7,500 models and run continuously for more than 48 hours without manual intervention.

 doi:10.20944/preprints202511.1831.v1

2. Related Work

Our work sits at the intersection of model conversion, mobile benchmarking, and continuous testing.

2.1. Model Conversion and Optimization

The challenge of deploying neural networks on mobile devices has been addressed by several conversion frameworks. TensorFlow Lite [2] established the standard for mobile optimized model formats, providing quantization and hardware acceleration features. Similarly, PyTorch Mobile [1] enabled direct deployment of PyTorch models on edge devices. Our conversion pipeline builds upon these foundations but adds automated batch processing capabilities missing from standard single model conversion tools. The AI Edge Torch library [5] we employ represents a significant advancement in cross framework compatibility, facilitating the PyTorch to TFLite transition.

2.2. Mobile ML Benchmarking

Several benchmarking frameworks exist for mobile machine learning. MLPerf Mobile [6] provides standardized benchmarks for mobile AI performance across different hardware platforms. AI Benchmark [7] offers a comprehensive evaluation of mobile AI accelerators. However, these solutions primarily focus on a preselected set of model architectures and lack the flexibility for custom, user defined model pipelines. Our work extends these concepts by enabling automated benchmarking of any PyTorch model architecture with integrated, low level device analytics.

2.3. Continuous Testing

The concept of continuous testing for mobile applications has been explored in tools like Firebase Test Lab [8] and AWS Device Farm. These platforms provide cloud based testing on physical devices but lack specialized support for the complete ML model validation lifecycle, from conversion to performance analysis. Our system addresses this gap by combining intelligent emulator management with model specific validation routines, creating a specialized continuous testing pipeline for mobile ML applications.

3. Methodology

Our approach employs a multi stage automated pipeline for continuous model conversion, deployment, and validation on mobile devices Figure 1. The system follows a modular architecture with distinct components for model processing, device management, and performance benchmarking.This modular design enables independent scaling of each subsystem while maintaining loose coupling through well defined interfaces. The pipeline incorporates intelligent state persistence and self healing mechanisms to ensure fault tolerance during large scale, long running evaluation sessions.

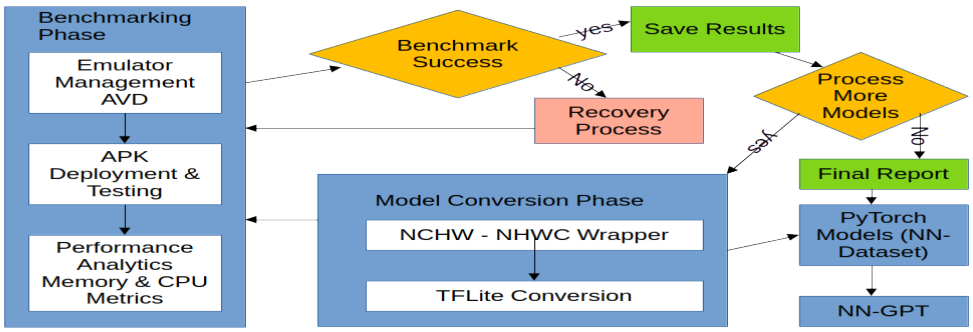

Figure 1. End-to-End Automated Benchmarking Workflow. The system manages the entire lifecycle from model loading and conversion to emulator testing, analytics collection, and state persistent reporting.

 doi:10.20944/preprints202511.1831.v1

3.1. System Architecture

The core of our system is a Python based orchestration framework that coordinates the entire workflow, as illustrated in Figure 2. It is designed with a microservices inspired architecture within a single process, ensuring modularity, maintainability, and extensibility.

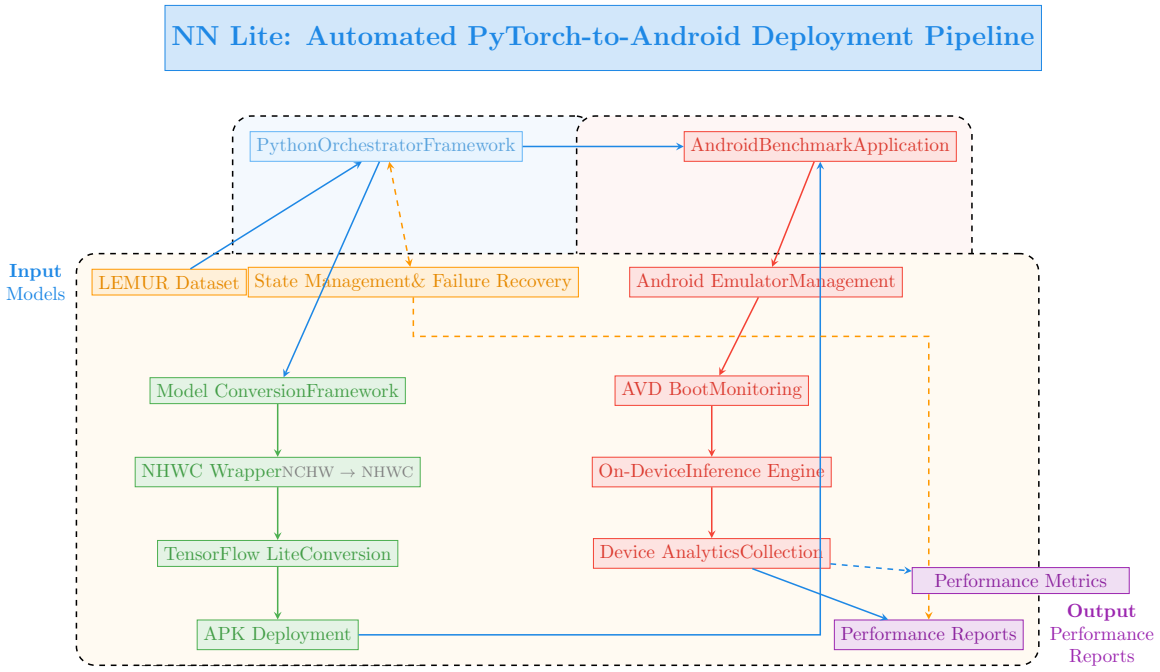

Figure 2. NN Lite: End-to-End Automated PyTorch-to-Android Deployment Pipeline. The system architecture comprises Python-based orchestration, model conversion framework, and Android benchmarking components that work together to automate the entire workflow from model conversion to performance reporting.

3.2. Model Conversion Framework

The conversion process is a critical first step. We dynamically instantiate models from a database, applying intelligent filtering to select mobile friendly configurations (e.g., small batch sizes). The core innovation is the `NHWCWrapper` class, which resolves the fundamental tensor layout disparity between PyTorch (NCHW) and TensorFlow Lite (NHWC). This challenge of cross framework deployment has been noted in prior work on mobile deep learning optimization [9].

```
class NHWCWrapper(torch.nn.Module):
    def __init__(self, model):
        super().__init__()
        self.model = model
    def forward(self, x):
        # Transform NHWC to NCHW format
        x_transformed = -
        x.permute(0, 3, 1, 2).contiguous()
        return self.model(x_transformed)
```

This wrapper ensures memory layout optimization through the `.contiguous()` operation and enables zero copy operations between frameworks, which is crucial for performance on mobile accelerators optimized for channels last operations. The importance of memory layout optimization for efficient mobile inference aligns with findings in neural network specialization research [10].

3.3. Automated Benchmarking Infrastructure

The benchmarking infrastructure provides a reproducible testing environment.

 doi:10.20944/preprints202511.1831.v1

Emulator Management: Our system implements dynamic Android Virtual Device (AVD) selection and intelligent lifecycle management. It features a sophisticated boot sequence monitor that polls the `sys.boot_completed` system property to ensure the Android environment is fully initialized before testing begins.

Performance Measurement: The pipeline automates model deployment, benchmark execution via Android intents, and result collection. A key enhancement is the automatic correlation of benchmark results with comprehensive device analytics (memory, CPU) before generating the final report, providing rich context for performance analysis.

3.4. State Management and Failure Recovery

To ensure reliability over long durations, the system incorporates a sophisticated state management system. **Continuous Processing:** A state file persists the processing context (processed, failed, and current models) using atomic writes to prevent corruption. This allows the system to resume seamlessly after interruptions. **Failure Recovery:** Upon a benchmark failure, the system initiates a multi stage recovery protocol: a configurable cooling off period (e.g., 3 minutes) to allow system resources to stabilize, a comprehensive cleanup of emulator and ADB processes, and a controlled process restart using Python's `os.execv()` for complete process rejuvenation while preserving the original state and command line arguments.

4. Experiments

4.1. Training and Testing

Our experimental evaluation leverages the comprehensive LEMUR Neural Network Dataset [11] https://github.com/ABrain-One/nn-dataset/, which provides a diverse collection of neural network architectures for automated machine learning research. This dataset, comprising over 7,500 unique models, serves as the foundation for our scalability and performance analysis. To further expand architectural diversity, we incorporate models generated through the NN-GPT https://github.com/ABrain-One/nn-gpt framework [12], which utilizes large language models for neural architecture generation, and the LEMUR 2 dataset [13] that unlocks additional neural network variants. Additionally, we draw inspiration from hyperparameter optimization approaches explored in HPGPT [12], which investigates LLMs for automated hyperparameter tuning, ensuring our benchmarking covers both architectural and parametric variations in model design. The integration of these diverse model families enables comprehensive validation of our automated deployment system across a wide spectrum of neural network designs, from traditional architectures to AI-generated models.

Our experimental setup focused on evaluating the pipeline using this comprehensive model collection. The experiments were conducted on a Linux workstation equipped with an Intel i7 processor, 32GB RAM, and an NVIDIA RTX 3080 GPU. The target platform for all benchmarks was the Android operating system, tested using the system's automated emulator management on a standard Android Virtual Device (AVD) with a predefined configuration (e.g., Pixel 4 profile, API level 30).

No traditional training was involved; instead, the testing phase consisted of the end to end execution of our automated pipeline for each model: conversion to TFLite, deployment to the emulator, on device inference benchmarking, and report generation.

4.2. Evaluation Metrics

The primary metrics for evaluation were both operational and performance based:

- **Throughput:** The number of models processed per unit time (models/hour).
- **Reliability:** The ability to complete long running sessions without manual intervention, measured by continuous uptime.
- **Inference Latency:** The average time taken for a single forward pass of the model on the mobile device, measured in milliseconds.

 doi:10.20944/preprints202511.1831.v1

- **System Resource Utilization:** Device analytics collected included available memory, cached memory, and detailed CPU architecture information, providing context for the performance metrics.

5. Results and Discussion

5.1. Operational Efficiency and Scalability

The system demonstrated exceptional operational efficiency and scalability. Over a sustained period, it successfully processed a total of 7,562 PyTorch models through the complete pipeline. The average processing time per model ranged from 60 to 90 seconds. This includes the full workflow: model conversion, emulator boot (if necessary), APK deployment, benchmark execution, analytics collection, and report generation. This high throughput enables large scale model evaluation campaigns that would be infeasible manually.

5.2. Failure Recovery

The framework successfully completed a continuous, unattended operation session lasting over 48 hours. This was made possible by the sophisticated failure recovery mechanism. During the large scale run, the system encountered and automatically recovered from 47 transient failures (e.g., emulator timeouts, ADB disconnections). In each case, the recovery protocol waiting, comprehensive cleanup, and process restart was triggered successfully, allowing the pipeline to continue from the last saved state without human intervention.

5.3. Comprehensive Data Collection

The pipeline generated a rich dataset of over 15,000 individual data points. Each data point includes not only the model's inference latency but is also enriched with the device's performance context at the time of execution. This multi dimensional data allows for sophisticated analysis, such as correlating performance degradation with low available memory or identifying architecture specific optimization opportunities. An example of the enriched report structure is shown below:

```
{
  "model_name": "AirNet",
  "device_type": "sdk_gphone64_x86_64",
  "os_version": "Android 14 (Hedgehog)",
  "valid": true,
  "emulator": true,
  "duration": 360,
  "device_analytics": {
    ...
    "memory": {
      "total_ram": "1.92 GB",
      "available_ram": "800 MB",
      "cached": "880 MB"
      ...
    },
    "cpu": {
      "architecture": "x86_64",
      "cores": 8,
      "vendor": "AMD",
      "features": ["SSE4.2", "AVX", "AES"]
      ...
    }
  }
```

 doi:10.20944/preprints202511.1831.v1

}

6. Conclusion

This paper presented a highly efficient automated framework for deploying and benchmarking machine learning models on mobile devices. Our system successfully addresses the critical gap between model development in PyTorch and performance validation on Android by automating the entire workflow from conversion to detailed reporting. The demonstrated ability to process thousands of models unattended, coupled with the deep, contextualized performance data it generates, represents a significant advancement over existing manual and semi automated approaches.

The framework establishes a new standard for automated ML testing infrastructure, enabling researchers and practitioners to perform large scale, reproducible empirical studies on mobile model performance. Future work will focus on expanding support to physical device fleets, integrating more advanced power and thermal profiling, and extending the model conversion framework to support a wider range of operators and quantization schemes.

References

1. Meta AI. PyTorch Mobile: End-to-End Deployment Solution for Mobile and Embedded Devices. Meta AI Research, 2019. Official documentation and framework release.
2. Google LLC. TensorFlow Lite: On-Device Machine Learning Framework, 2017. Open-source deep learning framework for mobile and edge devices.
3. Liao, D.; Pan, S.; Yang, S.; Zhao, Y.; Xing, Z.; Sun, X. A Comparative Study of Android Performance Issues in Real-world Applications and Literature. *arXiv preprint arXiv:2401.07849* **2024**.
4. Zim, M.Z.H. TinyML: Analysis of Xtensa LX6 microprocessor for Neural Network Inference. *Future Internet* **2023**, *15*, 350.
5. Meta AI. AI Edge Torch: PyTorch Library for Edge Device Deployment, 2024. Official PyTorch extension for edge computing and mobile deployment.
6. Reddi, V.J.; Kanter, D.; Mattson, P.; Duke, J.; Nguyen, T.; Chukka, R.; Shiring, K.; Tan, K.S.; Charlebois, M.; Chou, W.; et al. MLPerf Mobile Inference Benchmark. In Proceedings of the MLPerf. MLCommons, 2020.
7. Ignatov, A.; Timofte, R.; Chou, W.; Wang, K.; Wu, M.; Hartley, T.; Gool, L.V. AI Benchmark: Running Deep Neural Networks on Android Smartphones. In Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 288–314.
8. Google LLC. Firebase Test Lab: Cloud-Based App Testing Infrastructure, 2016. Cloud-based infrastructure for automated mobile app testing.
9. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* **2020**.
10. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.
11. Goodarzi, A.T.; Kochnev, R.; Khalid, W.; Qin, F.; Uzun, T.A.; Dhameliya, Y.S.; Kathiriya, Y.K.; Bentyn, Z.A.; Ignatov, D.; Timofte, R. LEMUR Neural Network Dataset: Towards Seamless AutoML. *arXiv preprint arXiv:2504.10552* **2025**.
12. Kochnev, R.; Khalid, W.; Uzun, T.A.; Zhang, X.; Dhameliya, Y.S.; Qin, F.; Ignatov, D.; Timofte, R. NNGPT: Rethinking AutoML with Large Language Models. *arXiv preprint* **2025**.
13. Kochnev, R.; Goodarzi, A.T.; Bentyn, Z.A.; Ignatov, D.; Timofte, R. Optuna vs Code Llama: Are LLMs a New Paradigm for Hyperparameter Tuning? In Proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2025.