18

Online AI Benchmarking on Remote Board Farms

Maïck Huguenin, Baptiste Dupertuis, Robin Frund, Margaux Divernois, and Nuria Pazos

Haute Ecole Arc – HES-SO, Switzerland

Abstract

Benchmarks are essential for balancing the benefits and risks of AI by providing quantitative tools that guide responsible development. They offer objective and consistent metrics for accuracy, speed, and efficiency, enabling engineers to develop reliable products and services. Additionally, benchmarks help researchers gain new insights that can drive future innovations.

Today, numerous cloud-based AI development services allow software developers, even those without expertise in data science, to utilise AI models through APIs, SDKs, or applications. Benchmarking these models on cloud infrastructure is a feature offered by these services. However, few of these services are designed for edge deployment, where deep expertise in embedded programming and system integration is necessary to optimize and deploy AI models on specific embedded devices. Comparing benchmarking results across different embedded boards becomes increasingly complex when targeting devices from various providers.

The current project aims to design and implement a collaborative platform that enables researchers and developers to conduct experiments and research across various edge AI domains and edge AI devices. This will be achieved by sharing resources on a distributed virtual laboratory (dAIEdge-VLab). This platform will provide access to dedicated resources, tools, and services, allowing end users without expertise in embedded programming to perform live AI experiments, such as benchmarking, on remote embedded boards.

Keywords Benchmarking, Edge AI, Virtual Lab, Distributed Platform, AI-powered Edge Devices.

List of Notations and Abbreviations: MPU, GPGPU, MCU, NPU, VPU.

18.1 Introduction and Novelty Aspect

Edge AI and cloud AI offer two distinct approaches to deploying artificial intelligence, each with unique strengths and limitations depending on where data processing takes place. Edge AI is ideal for applications requiring real-time, on-site processing and enhanced data security, as computations happen directly on local devices. In contrast, cloud AI excels in scenarios demanding extensive computational power and large-scale data processing, leveraging remote servers for more complex tasks. These approaches can also work together, with edge devices handling initial data processing and sending more demanding tasks to the cloud for deeper analysis.

Edge AI processes data locally on devices positioned at the network's edge, closer to the source of data generation. This approach minimizes latency, allowing for quicker, real-time responses. Since data does not need to be transmitted to the cloud, edge AI also improves privacy and security while lowering bandwidth usage. Although edge devices generally have less computational power compared to cloud AI, recent advancements in hardware have significantly enhanced their ability to handle complex AI tasks.

Two key challenges in the development of edge AI for industrial applications are the difficulty in reproducing results consistently across different edge devices, and the complexity involved in configuring heterogeneous platforms, which can lead to lengthy evaluation times. As a result, comparing benchmarking results between various embedded boards becomes increasingly complicated, especially when targeting devices from different manufacturers.

This work addresses the identified challenge by introducing a collaborative platform, dAIEdge-VLab, which enables researchers and developers to remotely conduct experiments and research across various edge AI domains and devices. With dAIEdge-VLab, users without embedded programming expertise can easily deploy edge AI models and applications on remotely accessible embedded boards and retrieve the experimental results. The dAIEdge-VLab's architecture is designed for scalability, allowing owners of AI-powered edge devices to seamlessly integrate their hardware into the dAIEdge-VLab infrastructure.

The structure of the paper is as follows: Section 1.2 compares our dAIEdge-VLab solution with existing state-of-the-art methods for remote deployment on board farms. Section 1.3 presents two architectures of the dAIEdge-VLab, a centralized and a distributed version. Section 1.4 delves into the implementation details of the dAIEdge-VLab. Finally, Section 1.5 concludes the paper and outlines directions for future work.

18.2 State-of-the-art

AI Benchmarking can be defined as the process of evaluating and comparing the performance, efficiency, and capabilities of AI models, algorithms, or systems against standardized metrics and tasks. Key performance indicators can be classified into two main categories: (i) hardware-agnostic metrics, which are independent of the target device and applied libraries, such as accuracy, model size or number of parameters; and (ii) hardware-specific metrics, which depend on the target device, such as inference speed, power consumption, or memory usage. These metrics determine how well an AI model or system performs in real-world edge environments.

Benchmarking typically involves running AI models on predefined datasets or tasks, such as image recognition, natural language processing, or autonomous decision-making, and measuring their effectiveness against other models or industry standards. Since edge devices have limited computational resources compared to cloud servers, benchmarking helps identify the most suitable AI models and optimizations for these resource-constrained environments. The goal is to ensure that AI applications running on the edge can meet the necessary requirements for real-time decision-making, energy efficiency, and overall performance in diverse scenarios, such as autonomous systems, IoT, and industrial automation.

While many popular edge AI frameworks (both vendor-agnostic and proprietary) offer built-in benchmarking tools to extract key performance metrics, they do not guarantee a standardized procedure for fair comparison. To address this, initiatives like MLPerf Inference Edge [1] from the MLCommons foundation [2] aim to provide a representative benchmarking suite that fairly assesses edge ML system performance. However, even though all participants follow the same procedure to publish their benchmarking results, the tests are conducted at the edge device owner's facilities, and there is no remote control to ensure proper hardware setup.

In the scientific literature, various publications propose methodologies for the fair comparison of hardware, algorithms, and optimization techniques within the embedded design space. QuTiBench [3] introduces a novel multi-tier benchmarking framework that accommodates algorithmic optimizations, such as quantization, to help system developers assess the strengths and limitations of emerging compute architectures for specific neural networks. The QuTiBench team encourages community contributions to cover the full range of choices in Machine Learning system implementations. However, contributions ceased in 2021, and the project appears to no longer be maintained. Subsequent works, including [4], have adopted a similar approach to evaluate Machine Learning inference machines on Edge-class compute platforms. This testbed features two hardware compute engines—Raspberry Pi 4 (CPU-based) and Google Edge TPU accelerator—and two inference frameworks: TensorFlow-Lite and Arm NN.

Building on two decades of experience in developing embedded benchmarks, EEMBC has taken its first step into the Machine Learning space and is committed to keeping pace with industry advancements. To this end, they introduced the EEMBC MLMark® benchmark [5], specifically designed to assess the performance and accuracy of embedded inference systems. Unlike benchmarks that allow private optimizations, MLMark requires that all implementations be made publicly available in its repository. Version 1.0 provides source code and libraries for a range of platforms, including Intel® CPUs, GPUs, and neural compute sticks with OpenVINO®; NVIDIA® GPUs with TensorRT; and Arm® Cortex®-A CPUs and Arm Mali^{TM} GPUs using Neon^{TM} technology and OpenCL^{TM}, respectively.

To ensure fair comparison across different types of AI-powered edge devices (such as MPUs, GPGPUs, MCUs, NPUs, and VPUs), a unified benchmarking service offering remote access to this diverse range of embedded boards is essential. An online benchmarking platform would greatly facilitate this need by enabling consistent and equitable evaluation across varied hardware.

Recently, several edge device providers, including STM, Qualcomm and Intel, have introduced their initial solutions for online remote benchmarking. STM offers the "STM32Cube.AI Developer Cloud"[6], a free online platform designed to help developers create, optimize, benchmark, and generate AI solutions specifically for STM32 microcontrollers and microprocessors, which are based on ARM Cortex processors. The platform also supports AI hardware acceleration (NPU) when available on the target device.

Qualcomm® AI Hub [7] simplifies the deployment of AI models for vision, audio, and speech applications on edge devices by providing automatic optimization and validation tools. It offers over 100 pre-optimized AI models and enables seamless integration with Snapdragon® and Qualcomm platforms, with easy access to real devices for testing and profiling.

Similarly, Intel has launched the "Intel Tiber Developer Cloud" [8], which promises unrestricted access to Intel's latest edge computing hardware and software platforms for developer.

In addition, other vendor-agnostic commercial solutions have recently emerged, such as the Impulse Embedded Remote Benchmarking Service [9]. This platform enables developers to remotely test AI models on a wide range of GPU hardware, from entry-level devices like the Jetson Nano to high-performance systems such as the AGX Orin. Based on our understanding, only GPGPU devices are currently available for remote benchmarking.

To the best of our knowledge, no vendor-agnostic, multi-platform, and scalable solution for online remote benchmarking currently exists. This work addresses that gap by introducing a versatile, vendor-neutral, and scalable virtual laboratory designed to facilitate edge AI benchmarking experiments, bridging the divide between data scientists and the embedded systems domain.

18.3 dAIEdge-VLab Architecture

The dAIEdge-VLab is designed to help users who lack expertise in embedded programming or do not have direct access to specific hardware. It will enable them to conduct real-time AI experiments on a remote farm of embedded boards. These experiments could include AI model benchmarking (using randomly generated data), AI application benchmarking (with pre-existing test datasets), support for hardware-in-the-loop neural architecture search (NAS), and even benchmarking for on-device training.

In terms of hardware compatibility, the dAIEdge-VLab is built to support a wide range of embedded boards, spanning from high-performance MPUs and GPGPUs to energy-efficient MCUs and specialized NPUs. On the software side, it accommodates both Linux-based and real-time operating systems, as well as bare-metal solutions. Additionally, the platform will be compatible with widely used vendor-agnostic inference runtimes along with proprietary AI engines.

A virtual lab for online benchmarking of edge AI applications typically consists of several architectural components designed to facilitate remote experimentation, testing, and optimization of AI models across different edge hardware platforms. Below is an outline of its overall structure together with the main functionalities:

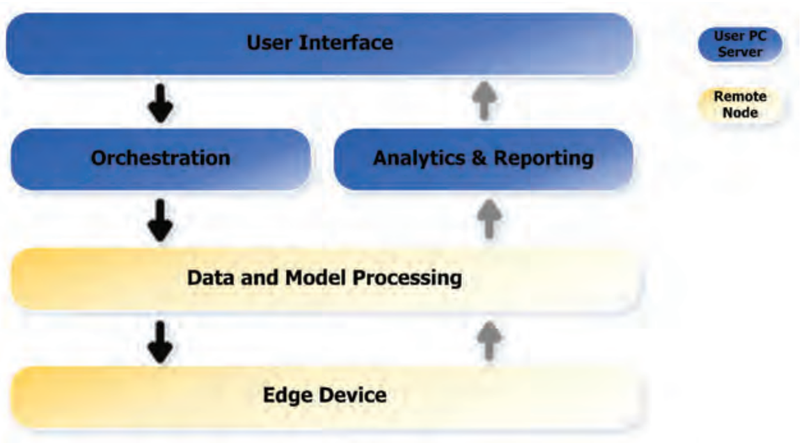


Figure 18.1 Virtual Lab Layer Model.

1. User Interface (UI) Layer

- **Web Interface/Dashboard**: The virtual lab offers a user-friendly web-based dashboard that allows data scientists and developers to interact with the system. This includes uploading AI models, selecting target hardware platforms, configuring benchmarking parameters, and monitoring results.
- **CLI API**: An API-based interface allows seamless integration with custom AI pipelines.
- **AI Application Management**: Users can upload pre-trained AI models, select from a model zoo, or choose a complete AI application. The system also supports model versioning, enabling easy retrieval and comparison of different versions.

2. Orchestration Layer

- **Resource Management and Scheduling**: This component handles the orchestration of resources across multiple platforms, ensuring efficient allocation of hardware for testing and benchmarking. It manages queues, schedules jobs, and optimizes resource usage based on platform availability.
- **Virtualization and Containerization**: Models and benchmarking tasks are often containerized (e.g., using Docker) to ensure compatibility across diverse hardware. This enables easy deployment across various edge devices, regardless of their underlying operating system or architecture. This layer deals with the management of containers on edge devices running general-purpose operating systems.

- **Benchmark Configuration**: Users define test cases, including hardware configurations, datasets, and performance metrics such as latency, power consumption, and accuracy. The ultimate goal is to define the technical requirements to compare benchmarking results among different configurations.

3. Edge Device Layer (Board farm)

- **Multi-Platform Hardware Pool**: The system integrates a variety of edge devices, ranging from low-power IoT devices (like STM32 MCUs, RISC-V based platforms, etc.) to high-performance systems (such as Raspberry Pi or Jetson Orin Nano or AGX). This diversity ensures comprehensive testing across different edge environments.
- **Remote Access & Control**: The lab allows remote access to real, physical hardware. Developers can deploy AI models directly to these devices for testing under real-world conditions, avoiding the limitations of simulated environments.

4. Data and Model Processing Layer

- **Data Preprocessing and Inference**: This layer handles the preprocessing of input data and manages the inference execution on edge devices. It ensures that the models are efficiently adapted to the target hardware's limitations, such as memory and computational power.
- **Performance Metrics Collection**: During benchmarking, this layer monitors critical performance metrics like inference time, throughput, power consumption, and memory usage, which are fed back into the system for analysis.

5. Analytics & Reporting Layer

- **Results Analysis & Visualization**: The benchmarking results are processed and presented through interactive dashboards, allowing users to compare metrics across different hardware platforms. Detailed reports include insights into energy efficiency, processing latency, accuracy, and other relevant performance factors.

This virtual lab architecture is designed to bridge the gap between data scientists and embedded systems by enabling seamless testing, monitoring, and optimization of AI models on real edge hardware in a scalable, vendor-agnostic manner.

The proposed dAIEdge-VLab platform features a modular architecture that allows for easy integration of remote embedded boards. For the implementation of the proposed dAIEdge-VLab, we have followed an agile four-step methodology: (i) classifying functional and non-functional requirements for the envisioned online AI benchmarking solution and identifying gaps in existing solutions; (ii) designing a layer model for the dAIEdge-VLab; (iii) developing an initial PoC of the dAIEdge-VLab based on a centralized architecture managed by a central server where all registered embedded boards are listed and orchestrated; and (iv) designing a distributed version to enhance the flexibility, scalability, and security of the dAIEdge-VLab.

Figure 18.2 illustrates the overall architecture of the centralized dAIEdge-VLab. The key components of this architecture are:

- **Remote User**: A user initiating an AI experiment on a remote node submits a request via a web-based user interface. No manual installation is required on the user's side.
- **dAIEdge-VLab Server**: The dAIEdge-VLab Server receives the user request and forwards it to the corresponding Remote Host that is connected to the target Remote Node. All available Remote Nodes are registered to the dAIEdge-VLab Server.
- **Remote Host**: Located at the facility of the Remote Node owner, the Remote Host is responsible for executing node-specific scripts. It compiles and deploys the AI application to the Remote Node, retrieves the benchmarking results, and sends them back to the user through the dAIEdge-VLab Server.
- **Remote Node**: This refers to the embedded board where the AI experiments, such as model benchmarking, are executed. A single Remote Host can manage multiple Remote Nodes at the owner's site.

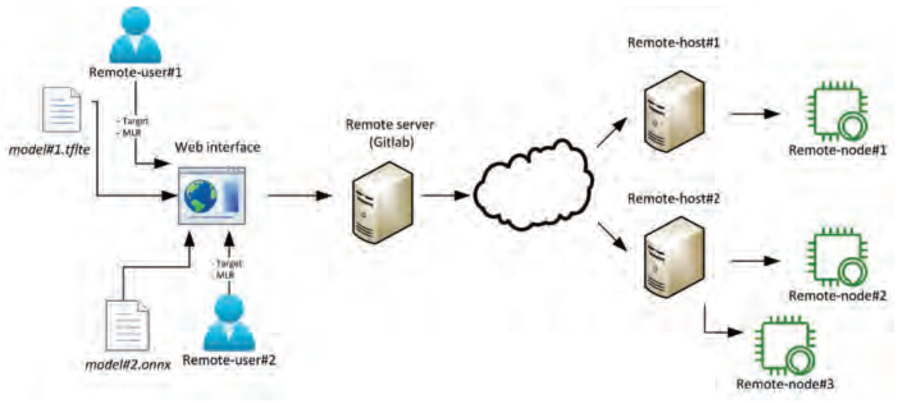


Figure 18.2 Centralized dAIEdge-VLab Architecture.

Figure 18.3 illustrates the architecture of the distributed dAIEdge-VLab, which differs slightly from the centralized version due to the absence of a dAIEdge-VLab server for managing the registered Remote Nodes. The primary components are as follows:

- **Remote User**: A user wishing to launch an AI experiment on a Remote Node submits their request through a web interface. Unlike the centralized version, a lightweight local installation of the web interface is required on the user's side to interact with the dAIEdge-VLab API.
- **Peer-to-Peer Content Delivery Network:** This open system manages the exchange of data between nodes without the need for a centralized server, enabling decentralized communication.
- **Remote Host:** Situated at the Remote Node owner's location, the Remote Host is responsible for running node-specific scripts, compiling and deploying the AI application to the Remote Node, retrieving the benchmarking results, and transmitting them back to the user via the peer-to-peer content delivery network.
- **Remote Node:** This is the embedded board where AI experiments, such as model benchmarking, are executed. Multiple Remote Nodes can be managed by a single Remote Host within the owner's facility.

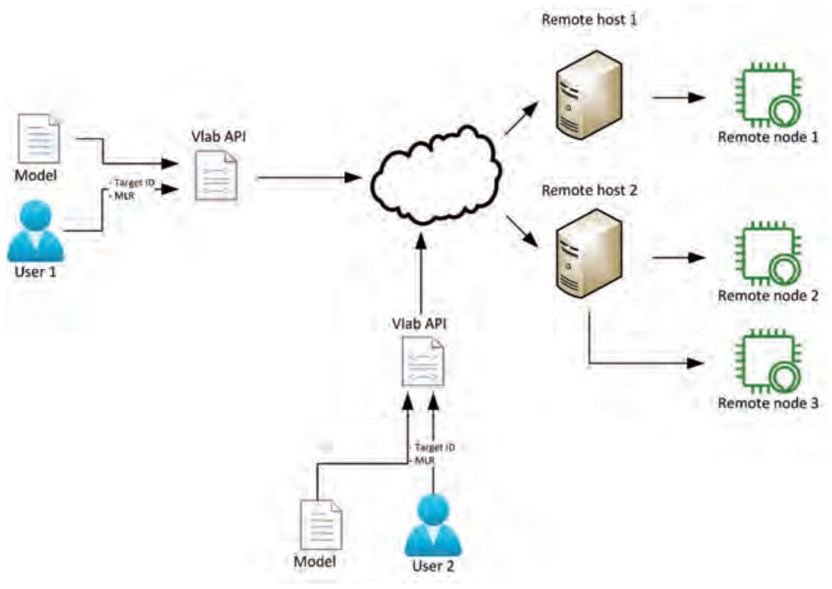


Figure 18.3 Decentralized dAIEdge-VLab Architecture.

18.4 dAIEdge-VLab Implementation

To demonstrate the feasibility of the centralized dAIEdge-VLab, the initial PoC implementation uses a GitLab server to run CI/CD pipelines on GitLab-hosted runners. Each Remote Host must install and register a runner for every Remote Node it manages. These runners execute node-specific tasks triggered by the GitLab CI/CD pipeline. The process followed by the centralized dAIEdge-VLab to initiate a benchmarking experiment is outlined as follows:

1. User Interface layer: A Remote User submits a benchmarking request via the web interface, specifying the trained model, target Remote Node, and Machine Learning Runtime (MLR) to be used from the available options.
2. Orchestration layer: The dAIEdge-VLab GitLab Server processes the request by triggering the GitLab CI/CD pipeline. A Docker container with the required tools for the specific node is also deployed on the Remote Host.
3. Data and Model Processing layer: The Remote Host executes the node-specific scripts (AI-Support, AI-Build, AI-Deploy, AI-Manager) using the tools within the Docker container, producing a binary tailored for the target Remote Node. This binary is then deployed onto the Remote Node.
4. Edge Device layer: Runs the inference/benchmarking process and sends back the collected benchmarking metrics to the Remote Host.
5. Analytics and Reporting layer: The Remote Host retrieves the benchmarking metrics from the Remote Node and returns them to the dAIEdge-VLab GitLab Server via the GitLab artifacts. The server generates a benchmarking report, which is then sent back to the Remote User for visualization through the web interface.

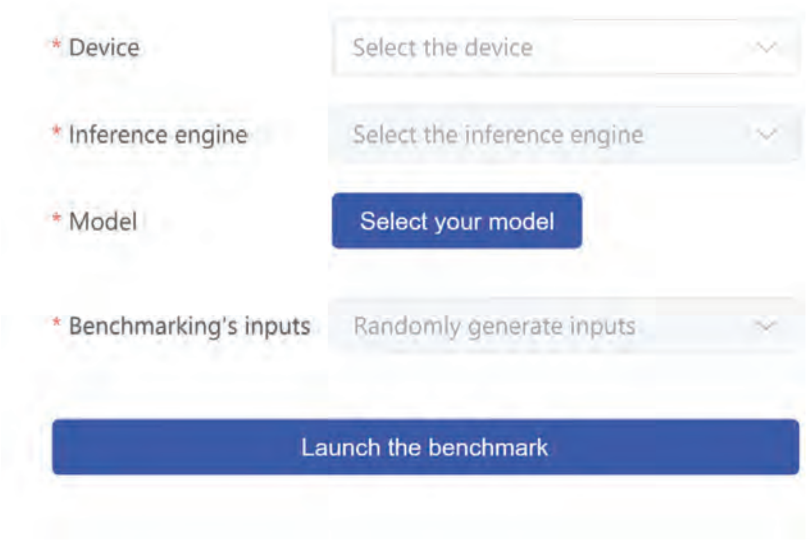
The distributed version of the dAIEdge-VLab differs from the centralized one primarily in its implementation of the orchestration layer, specifically the resource management and scheduling components. In this version, a dAIEdge-VLab API on the Remote User side is responsible for requesting access to the target Remote Node to run the benchmarking experiment. Once access is granted based on node availability, a peer-to-peer content delivery network, using IPFS [10], will transmit the request to the selected board via the associated Remote Host.

Managing the preprocessing of input data and inference execution on the target edge device requires a structured approach for implementing the node-specific scripts in the Data and Model Processing layer. This setup includes four main scripts and a Docker image containing all the necessary tools and packages for the specific node. These components are organized as follows:

1. **AI_Support**:
    - Provides documentation, dependencies, and installation support for the Remote Node. For edge devices with OS support, an OS kernel binary image along with system libraries will be provided for flashing onto the target board.
    - Includes scripts to initiate code generation for the benchmarking application specific to the Remote Node, which will be saved in the AI_Project folder.
2. **AI_Project**:
    - Stores the generated benchmarking application that is optimized for execution on the Remote Node.
3. **AI_Build**:
    - Contains the build environment with toolchains required by the inference framework to cross-compile the benchmarking application.
    - Provides scripts for cross-compiling the benchmarking application on the Remote Host.
4. **AI_Deploy**:
    - Includes tools for deploying the generated binary file onto the Remote Node.
    - Provides scripts to automate the deployment of the binary for the benchmarking application onto the Remote Node.
5. **AI_Manager**:
    - A management tool integrated with the inference framework to control the target board and monitor the system's status.
    - Includes scripts to establish a connection with the Remote Node and retrieve benchmarking metrics.

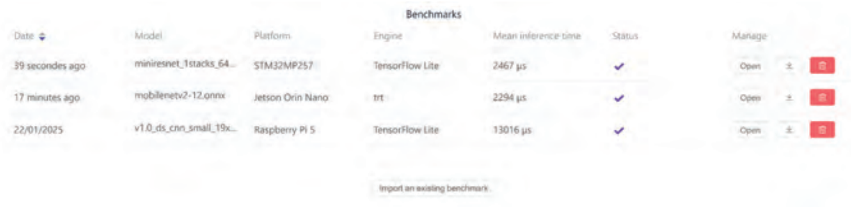
This structure ensures seamless execution of remote benchmarking tasks across various edge devices while maintaining consistency in deployment and management.

A first PoC for the centralized dAIEdge-VLab is operational. Below are screenshots showcasing the web interface dashboards for both input selection (see Figure 18.4) and visualization of benchmarking results (see Figure 18.6). A history of previously executed experiments is saved locally in the browser cache (see Figure 18.5), allowing users to review and compare different benchmarking experiments in the future.



* Device	Select the device
* Inference engine	Select the inference engine
* Model	Select your model
* Benchmarking's inputs	Randomly generate inputs

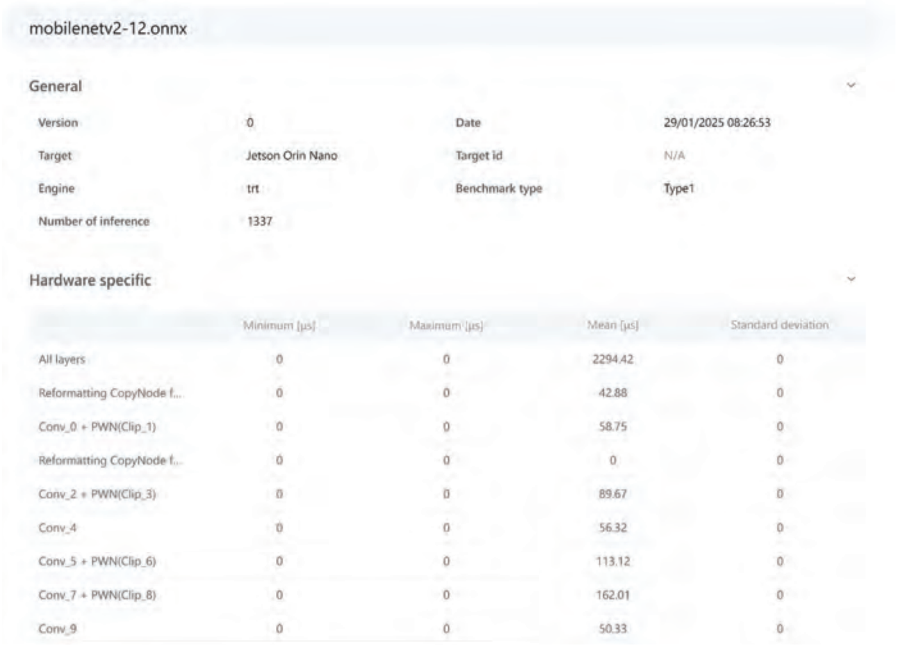
Figure 18.4 User Input Selection through Web Interface.



Benchmarks

Date	Model	Platform	Engine	Mean inference time	Status	Manage
39 secondes ago	miniresnet_1stacks_64...	STM32MP257	TensorFlow Lite	2467 µs	✓	Open
17 minutes ago	mobilenetv2-12.onnx	Jetson Orin Nano	trt	2294 µs	✓	Open
22/01/2025	v1.0_ds_cnn_small_19x...	Raspberry Pi 5	TensorFlow Lite	13016 µs	✓	Open

Figure 18.5 List of Benchmarking Results.



mobilenetv2-12.onnx

General

Version	0	Date	29/01/2025 08:26:53
Target	Jetson Orin Nano	Target id	N/A
Engine	trt	Benchmark type	Type1
Number of inference	1337		

Hardware specific

	Minimum [µs]	Maximum [µs]	Mean [µs]	Standard deviation
All layers	0	0	2294.42	0
Reformatting CopyNode f...	0	0	42.88	0
Conv_0 + PWN(Clip_1)	0	0	58.75	0
Reformatting CopyNode f...	0	0	0	0
Conv_2 + PWN(Clip_3)	0	0	89.67	0
Conv_4	0	0	56.32	0
Conv_5 + PWN(Clip_6)	0	0	113.12	0
Conv_7 + PWN(Clip_8)	0	0	162.01	0
Conv_9	0	0	50.33	0

Figure 18.6 Visualisation of Model Benchmarking Results

The generated benchmark report is a JSON file that includes all the relevant keys, even those not retrieved from the Remote Node. These keys are organized into four main categories and used throughout the benchmark report:

1. **Hardware specific keys**
   - *FLOPs* (type: float): Floating-point operations per second.
   - *inference_latency* (type: json): This key contains a JSON file containing the latency measures (in seconds) of all performed inferences (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
   - *throughput* (type: float): Throughput of the system in inferences per second.
   - *latency_per_layers* (type: list): This key contains a list of json composed of all the layer's latency measures (<*element list*> (type: json), *layer_name* (type: str), *mean* (type: float), *std* (type: float), *min* (type: float), and *max* (type: float)).

  - *preprocess_time* (type: json): Time in seconds for the pre-processing (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
  - *postprocess_time* (type: json): Time in seconds for the post-processing processing (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
  - *ram_size* (type: int): Size in bytes of the available RAM of the system.
  - *ram_peak* (type: float): Percentage of the peak RAM usage.
  - *flash_size* (type int): Size in bytes of the available FLASH.
  - *flash_usage* (type: float): Percentage of the FLASH usage. It might not be relevant for Linux targets.
  - *load_cpu* (type: float): Percentage of the CPU usage.
  - *load_accelerator* (type: float): Percentage of the accelerator (GPU, NPU, etc.) usage.
  - *temperature* (type: float): Temperature of the board in °C.
  - *ambient_temperature* (type: float): Room temperature in °C.
  - *power_consumption* (type: float): Power consumption of the board in Watt.
  - *energy_efficiency* (type: float): Operations Per Watt (OPW).

2. **Hardware agnostic keys**
  - *model_size* (type: int): Size in bytes of the trained model on the embedded board (after any compression / optimisation performed by the MLR).
  - *nb_parameters_model* (type: int): Number of parameters of the trained model.
  - *accuracy* (type: float): Accuracy of the trained model.

At the conclusion of each benchmarking experiment, two log files, *user.log* and *error.log*, are generated to help users track any issues. The contents of these logs are displayed on the web interface for user review:

- ***user.log***: This file provides general information or warnings related to the benchmarking process.
- ***error.log***: This file logs any errors encountered during the process, particularly in cases where the JSON benchmarking report could not be generated (e.g., due to an unsupported model format or insufficient memory on the target board).

These logs allow users to identify and troubleshoot potential problems in the benchmarking process.

A Continuous Testing (CT) strategy has been implemented to ensure the dAIEdge-VLab platform's stability. To achieve this, a suite of tests is automatically triggered on a weekly basis, as well as after every push or merge request. These pipelines are executed across all registered Remote Nodes, supported Machine Learning Runtimes (MLRs), and models from the internal model zoo.

Remote Node owners interested in adding their boards to the dAIEdge-VLab can find the guidelines under [11]. An MPU/linux template as well as an MCU/RTOS template are available to speed up the integration of new Remote Nodes.

18.5 Conclusion

Evaluating machine learning (ML) inference on edge devices is an essential step before selecting the optimal embedded board or optimizing the ML model to suit the target hardware. However, accessing boards from various manufacturers and getting familiar with their unique programming environments is a complex and often inaccessible task. To address this challenge and simplify access to a diverse range of embedded boards, we propose in this paper a solution for conducting ML benchmarking experiments on remote board farms.

For the implementation of the proposed dAIEdge-VLab, we adopted a five-layer model and followed a four-step methodology: (i) classifying functional and non-functional requirements for an online AI benchmarking solution on remote board farms and identifying gaps in existing solutions; (ii) designing a layer model for the dAIEdge-VLab; (iii) developing an initial PoC of the dAIEdge-VLab based on a centralized architecture managed by a central server; and (iv) designing a distributed version to enhance the flexibility, scalability, and security of the dAIEdge-VLab. This incremental approach has resulted in a scalable solution that is ready for release and capable of integrating third-party embedded boards in a distributed manner.

The current implementation of dAIEdge-VLab supports a wide range of Linux-based MPUs, including Raspberry Pi 4B and Raspberry Pi 5, as well as GPGPUs such as Nvidia Jetson Xavier and Nvidia Jetson Orin Nano. It also accommodates MCU+NPU platforms like the STM32MP257, along with bare-metal MCUs such as STM32L4R9 and NXP LPC55S69. Additionally, it includes support for vendor-agnostic Machine Learning runtimes, such as ONNX Runtime, TensorFlow Lite, and TensorFlow Lite for microcontrollers, as well as vendor-specific Machine Learning runtimes like CUBE-AI and TensorRT.

dAIEdge-VLab currently assists users who may not have expertise in embedded programming or access to the target embedded board by enabling real-time model benchmarking on a remote embedded board. In the future, additional experiments will be introduced, including AI application benchmarking, support for hardware-in-the-loop neural architecture search (NAS), and the ability to launch on-device training directly on the target board.

In the near future, we aim to enhance the distributed version of the dAIEdge-VLab by incorporating blockchain technology to share and manage information about the registered embedded boards in a decentralized way.

Acknowledgements

This work has been funded by the European Union's Horizon Europe project dAIEDGE under grant agreement No. 101120726. The associated partners from Switzerland received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

References

[1] V. J. Reddi et al ; "MLPerf Inference Benchmark", 2020. [Online]. Available: https://arxiv.org/abs/1911.02549

[2] hhttps://mlcommons.org/benchmarks/inference-edge/

[3] M. Blott, L. Halder, M. Leeser, and L. Doyle, "Qutibench: Benchmarking neural networks on heterogeneous hardware," J. Emerg. Technol. Comput. Syst., vol. 15, no. 4, Dec. 2019. [Online]. Available: hhttps://doi.org/10.1145/3358700

[4] P. Amanatidis, G. Iosifidis, and D. Karampatzakis, "Comparative evaluation of machine learning inference machines on edge-class devices," in Proceedings of the 25th Pan-Hellenic Conference on Informatics, ser. PCI '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 102–106. [Online]. Available: hhttps://doi.org/10.1145/3503823.3503843

[5] Torelli, Peter and Bangale, Mohit. "Measuring Inference Performance of Machine-Learning Frameworks on Edge-class Devices with the MLMark^{TM} Benchmark" . https://www.eembc.org/mlmark

[6] https://www.st.com/en/development-tools/stm32cubeai-dc.html

[7] https://aihub.qualcomm.com/
[8] https://www.intel.com/content/www/us/en/developer/tools/devcloud/services.html
[9] https://www.impulse-embedded.co.uk/remote-gpu-system-benchmarking.htm
[10] https://docs.ipfs.tech/
[11] http://public-virtuallab-docs-tica-23tica04-daiedge-poc-6eb4f1c0e44fb2.pages-rad.ing.he-arc.ch/docs/