AI-BMT: Technical Report

June 30, 2025

Abstract

This technical report describes AI-BMT, a benchmarking platform designed for Edge AI applications, addressing limitations in existing frameworks such as MLPerf Inference and AI-Benchmark. The report provides an overview of AI-BMT's architecture, evaluation methodology, and performance improvements. The proposed framework enhances fairness, scalability, and efficiency in benchmarking AI accelerators.

1 Overview

Artificial intelligence (AI) has rapidly transformed numerous industries, driving the need for robust and efficient hardware accelerators like NPUs. These accelerators, designed to handle diverse AI workloads, have become essential for applications ranging from cloud computing to edge devices. To ensure the effective utilization of these accelerators, inference benchmarking platforms provide critical insights into their performance under standardized conditions.

Existing platforms like MLPerf Inference provide robust benchmarks for AI systems, supporting a wide range of workloads, including cloud and edge applications. However, its approach presents notable challenges, particularly for users adapting the platform to diverse edge AI hardware. These include the complexity of environment setup, extensive time required for understanding and modifying open-source code, and the labor-intensive audit process to ensure rule compliance.

To overcome these limitations, AI-BMT introduces a user-friendly and scalable benchmarking platform that minimizes the evaluation burden. By centralizing benchmarking logic, abstracting hardware-specific implementations, and automating dataset and label validation, AI-BMT ensures a seamless evaluation experience. Additionally, the integration with a database enhances fairness by securely storing evaluation results, preventing manipulation, and ensuring transparency across benchmarking processes.

To further enhance efficiency, AI-BMT incorporates Double Buffering, allowing preprocessing and inference execution to overlap, reducing evaluation time while maintaining inference performance. Consistent query sizing is enforced by dividing datasets into a Benchmark Set (used for time and accuracy metric evaluation) and a Residual Set (used exclusively for accuracy metric

evaluation), eliminating last-query inconsistencies and ensuring stable performance metrics. AI-BMT also introduces RAM optimization techniques that adjust memory allocation to ensure stable evaluation performance across different hardware environments.

Moreover, AI-BMT ensures consistent and stable performance evaluation by enforcing minimum duration and epoch-based evaluations. Additionally, dataset random shuffling is applied before each evaluation epoch to prevent sequence-based optimizations, ensuring fair and unbiased benchmarking results.

With these enhancements, AI-BMT significantly reduces the setup and implementation burden, enhances audit efficiency, and builds trust in benchmarking outcomes. Consequently, it provides a robust solution tailored to the demands of next-generation AI hardware.

This report is organized as follows:

Section 2 presents AI-BMT's core methodology, covering its modular architecture and key components.

Section 3 elaborates on the hardware abstraction mechanisms, and automated validation processes.

Section 4 discusses benchmarking optimizations, including dataset preprocessing ,Double Buffering, and DMQE for improved efficiency.

Section 5 introduces benchmarking scenarios and evaluation metrics, ensuring stability and fairness through Min Duration, Min Epochs, and Dataset Random Shuffling.

Section 6 discusses the importance of maintaining consistent query sizes for stable time performance evaluation and introduces RAM optimization for stable performance in memory-limited environments.

Section 7 presents AI-BMT's evaluation results, showcasing its effectiveness and comparing it with MLPerf Inference and AI-Benchmark.

Section 8 interprets key findings, discusses AI-BMT's advantages over existing benchmarks, and outlines future improvements.

Section 9 summarizes AI-BMT's contributions and its potential as a standard for AI Processing Units(APUs).

2 System Design

The **AI-BMT platform** leverages a modular design and hardware abstraction to ensure scalability and flexibility in benchmarking diverse APUs. Its architecture integrates four main components: the WAS, the BMT GUI application, the file system, and the database. These components enable seamless integration and standardized evaluation workflows in various hardware environments.

2.1 Research Design

The platform is designed to reduce evaluator workload through hardware abstraction and standardized interfaces. It incorporates automated dataset and label validation, warm-up processes, and evaluation metrics reporting through

personal and public databases. Additionally, it offers automated model and dataset downloads, intuitive GUI options, and cross-platform support (Windows and Linux), ensuring consistent and fair benchmarking across diverse hardware environments. While the platform supports x86_64 architecture, it also provides an evaluation environment for ARM64-based hardware, addressing the growing trend of deploying NPU chips on ARM-based devices such as Raspberry Pi and Orange Pi.

2.2 Overall Architecture

The AI-BMT platform provides a comprehensive solution for fair and efficient benchmarking of AI accelerators, particularly in Edge AI environments. Its architecture integrates multiple components to facilitate smooth communication, secure authentication, and streamlined performance evaluations. A detailed breakdown of each component's features and functionalities is provided in Appendix A. Figure 1 illustrates the major components and their roles, which are as follows:

- **BMT WAS**: A web application server that acts as the central hub, handling authentication, model and dataset management, and evaluation result storage.
- **BMT GUI App**: A user-friendly application designed to manage all aspects of performance evaluation
- **APU Evaluator**: A Submitter responsible for executing benchmarking tasks by interacting with the GUI app and WAS.
- **Databases**: Securely store benchmarking results in private or public repositories to ensure fairness and transparency.
- **File System**: Provides seamless access to datasets and models through the GUI app.
- **BMT Evaluation Server**: A dedicated server for computing evaluation metrics (e.g., mAP) using official Python libraries like `pycocotools`, used when C++ implementation is impractical. It ensures consistent and standardized results across all submissions

2.3 Overall Benchmarking Flow

Benchmarking tasks are executed using the BMT GUI App, which follows the process described below. Figure 2 illustrates the overall benchmarking flow in detail. For a detailed explanation, please refer to Appendix B.

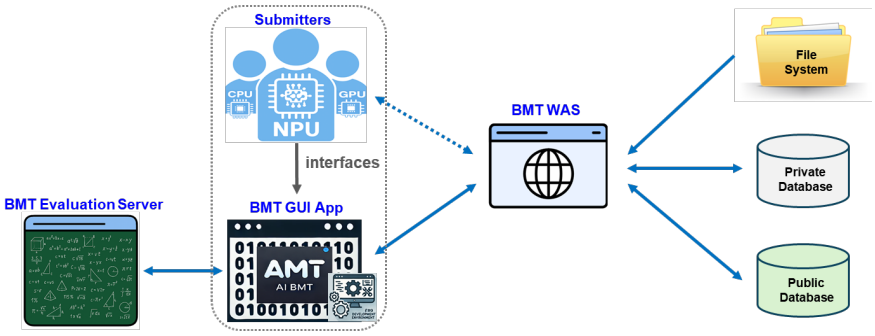

Figure 1: High-level architecture of AI-BMT

3 Hardware Abstraction and Validation

AI-BMT ensures flexibility and fairness in benchmarking by providing a standardized APU interface and automated validation mechanisms. This section details the hardware abstraction layer, the tasks supported by AI-BMT, and the automated dataset validation process. By defining a clear and structured evaluation framework, AI-BMT enables consistent and reproducible benchmarking across diverse hardware platforms.

3.1 APU Interface (Hardware Abstraction)

The standardized interface enables a clear separation between the App and APU-specific implementations, reducing the learning curve for evaluators analyzing the code and allowing them to quickly begin benchmarking tasks. By managing common processes such as data handling, validation, and performance analysis within the App, users only need to implement the interface customized to their specific APU. This design not only simplifies the benchmarking process, but also allows evaluators to directly implement the required logic for their specific APU, theoretically enabling the evaluation of any APU with minimal effort.

Additionally, most of the environment required for the App's execution is either included during the App build or dynamically packaged with the App, minimizing the setup required by the evaluator. Furthermore, by limiting the evaluator's implementation scope to the interface, the amount of code requiring verification is significantly reduced, thereby simplifying the audit process.

To further enhance usability, AI-BMT provides Reference Code for implementing interfaces for each Task, such as data preprocessing and inference. These Reference Code examples serve as a guide for users, enabling them to quickly adapt and customize the interfaces for their specific APUs. By offering these examples, AI-BMT lowers the barrier to entry for new evaluators and ensures consistency in how tasks are implemented and evaluated across different hardware platforms. Figure 3 provides a detailed description of the virtual methods in our interface.

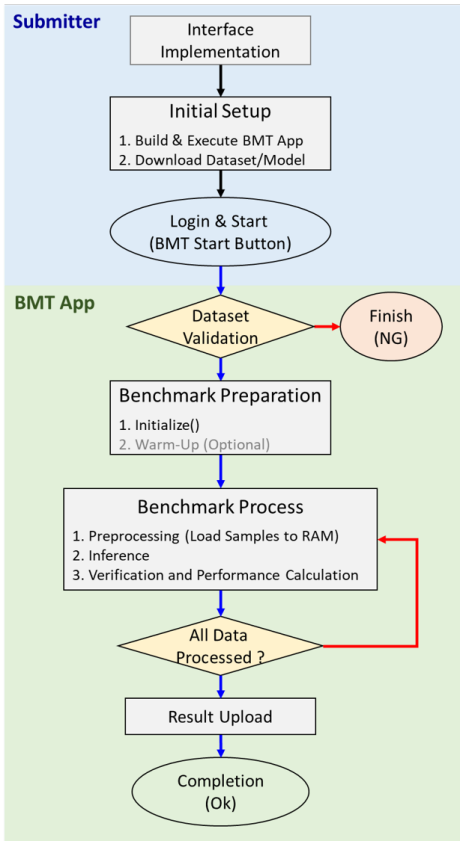

Figure 2: Overall Benchmarking Flow

To ensure fairness and consistency across evaluations, while implementing interfaces, altering model weights through retraining or pruning is prohibited, as it compromises the comparability of results.

3.1.1 Data preprocessing Part

The data preprocessing interface is designed to prepare raw data into a format compatible with the tested APU. This step is crucial for ensuring that the data meets the input requirements of the inference engine while being excluded from performance measurements such as latency or throughput.

To promote architecture and application neutrality, the preprocessing phase in AI-BMT is explicitly untimed. This design decision aligns with MLPerf Inference's benchmarking principles, where untimed preprocessing allows implementations to transform input data into system-specific ideal forms without introducing variability due to hardware or software-specific optimizations. For instance, systems with integrated cameras may deliver images directly in an

ideal format, whereas systems receiving JPEGs from external sources require additional decoding steps. By excluding preprocessing from evaluation metrics, AI-BMT ensures a fair and consistent evaluation across diverse systems.

The `convertToPreprocessedDataForInference` interface is central to this process. It is responsible for preprocessing data to meet the requirements of both the evaluation model and the APU used for inference. It processes raw input data, such as image file paths, into a format compatible with the APU. By allowing evaluators to define custom preprocessing steps, the interface ensures flexibility while maintaining consistency in how preprocessed data is utilized during benchmarking.

This interface supports a wide range of data types, ensuring compatibility with diverse AI processing architectures. These types include dynamic array pointer types (e.g., `uint8_t*`, `float*`) and vector types (e.g., `vector<uint8_t>`, `vector<float>`). A detailed list of supported data types can be found in Table 1, which highlights the flexibility of this method in handling various data representations necessary for preprocessing and inference tasks. For more detail, refer to Appendix C.

3.1.2 Inference Part

The inference interface is designed to execute AI model predictions using the preprocessed data and to return results as a pre-defined structure for performance evaluation.

The `runInference()` method is at the core of this process. It:

- **Processes Data**: Receives a vector of pre-processed data as input and processes them to optimize performance.
- **Returns Results**: Outputs predictions in the form of a pre-defined structure, which includes classification indices (e.g., ImageNet class IDs), object detection results (e.g., COCO-style bounding boxes), or other results required for performance calculations of a specific task.

By separating the inference logic through this interface, evaluators can implement custom inference pipelines optimized for their APU. The interface ensures that pre-processed data is dynamically adapted to the target hardware specified in Table 1, enabling seamless integration into the benchmarking framework while maintaining compatibility and optimized performance for each AI task.

Table 1: Supported numerical formats and hardware targets

Numerical Formats	Hardware Targets
uint8, uint16	CPUs, GPUs
uint32, int8	NPUs, TPUs
int16, int32, float	

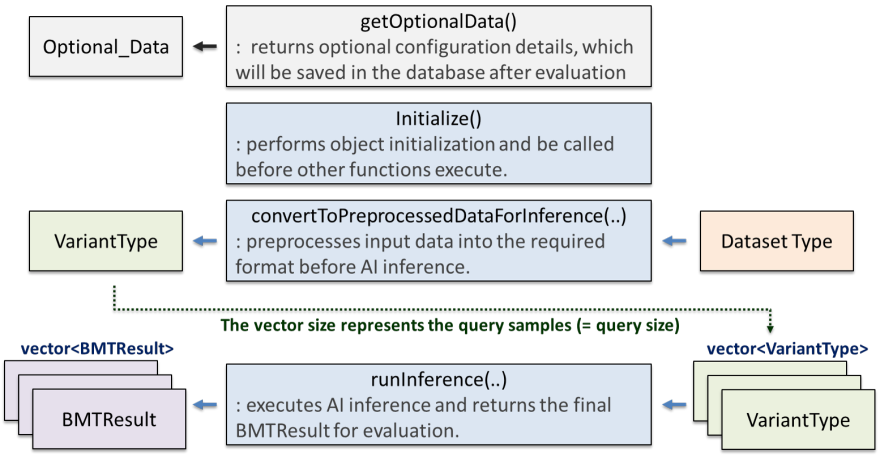

Figure 3: Interface Description

3.2 Supported Tasks

AI-BMT currently supports Image Classification and Object Detection, using widely adopted models and datasets relevant to real-world edge AI. These ensure fair and reproducible evaluations across diverse hardware. Detailed model and dataset specifications are provided in Chapter 4 of the **AI_BMT_User_Manual.pdf**, available at **ai-bmt.com**.

3.3 Automated Validation

3.3.1 Dataset and Label Validation

This feature ensures that all evaluations are conducted using the same dataset and label without any modifications, ensuring consistency across all evaluators. The key validation checks in dataset verification are as follows:

- **Existence Verification**: Ensure that the dataset and label files exist for the selected dataset.
- **Dataset Entry Consistency Check**: Verify that the number of dataset entries matches the expected count.
- **Label-Dataset Matching Check**: Ensure that the number of labels matches the dataset size and that each data entry has a corresponding label.
- **Dataset Integrity Check using Hash Values**: Detect any dataset modifications by comparing reference hash values with computed hash values. The corresponding hash function is defined in the following subsection.

- **Label Integrity Verification**: Verify label consistency by comparing stored reference label values with current dataset labels.

3.3.2 Hash Function for Dataset Validation

To ensure that no cheating or tampering occurs during dataset validation, the Hash function used for dataset integrity verification is embedded within the App and not exposed externally. This ensures that the validation logic remains secure and resistant to manipulation.

The pseudo-code for the Image Dataset Hash function is as follows:

Algorithm 1 Pseudo Code for Image Dataset Hash Function

Input: Image file path $filePath$
Output: Hash value as a hexadecimal string
Load image from $filePath$
if image is invalid **then**
    Return error
end if
Resize image to $N \times N$ resolution
Initialize $hashValue \leftarrow 0$
for each pixel (x, y) in the resized image **do**
    Convert pixel color to grayscale value $gray$
    if $x \mod A = 0$ and $y \mod B = 0$ **then**
        $hashValue \leftarrow (hashValue \ll 1) \oplus gray$
    else
        $hashValue \leftarrow hashValue \oplus gray$
    end if
end for
Return $hashValue$ as a hexadecimal string

Note: The values of N, A, and B are intentionally hidden in this pseudo-code to prevent the exact hash function from being exposed to evaluators, thereby ensuring that any manipulation of the dataset is reliably detected. This approach maintains the integrity and security of the validation process by obfuscating critical parameters of the function.

4 Evaluation Acceleration Techniques

To improve the speed, scalability, and usability of benchmarking, our platform implements two complementary techniques: **Double Buffering** to optimize runtime execution, and **Dynamic Model Queue Evaluation (DMQE)** to streamline multi-model evaluation.

4.1 Double Buffering

In AI benchmarking, preprocessing overhead can significantly extend evaluation time. To mitigate this, we provide datasets with preprocessing steps already applied. For instance, our classification benchmark uses a version of ImageNetV2 where all images are resized to 232×232 and center-cropped to 224×224, eliminating the need for runtime resizing or cropping.

However, sequential preprocessing still introduces idle time during inference. To address this, we adopt a **double buffering** strategy, where the next data chunk is preloaded into RAM during the current inference step. This overlap reduces overall evaluation time without affecting accuracy. The feature is optional, as it requires additional memory and may not be suitable for all devices.

Table 2 summarizes results from six test runs per configuration. Double buffering consistently reduced epoch duration in both Single-Stream and Offline modes. Loading 996 samples showed greater improvements than 2490 samples due to reduced initial loading cost and better overlap opportunities.

All evaluations were conducted using the ResNet-50 model and the ImageNetV2 dataset on a DeepX-M1 NPU, deployed on an Orange Pi 5 Plus (RK3588, 8GB RAM) running Ubuntu 24.04 (ARM64).

Table 2: Double buffer evaluation time (O: with buffer, X: without buffer)

Scenario	**RAM samples**	**O**	**X**	**Avg Reduction(%)**
Single-Stream	996	142923	226039	36.8
Single-Stream	2490	157388	229728	31.5
Offline	996	101148	155833	35.1
Offline	2490	112694	155899	27.7

4.2 DMQE (Dynamic Model Queue Evaluation)

As shown in Figure 4, To improve the flexibility and usability of our benchmarking workflow, we introduce **DMQE**. This feature enables users to select multiple models interactively at runtime and queue them for sequential evaluation in a single execution session.

DMQE is particularly useful in real-world scenarios where developers or hardware engineers wish to benchmark a diverse set of models (e.g., YOLOv8n/s/m, YOLOv9n/s/m) under the same platform conditions without restarting the evaluation tool. This streamlined process significantly improves productivity and reduces evaluation latency.

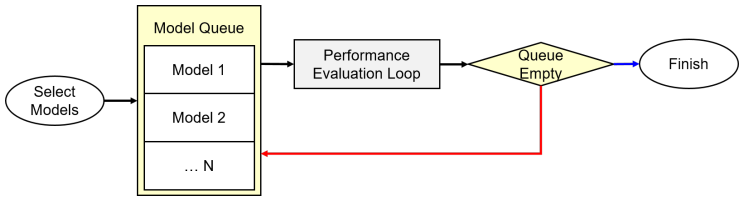

Figure 4: Workflow of **DMQE**. Models are sequentially evaluated from the queue until it is empty.

5 Scenarios, evaluation metrics, and stability evaluation

To provide a comprehensive assessment of AI processing units (APUs), AI-BMT defines multiple benchmarking scenarios and standardized evaluation metrics. This section outlines the supported benchmarking scenarios, which reflect real-world AI applications, and details the key performance metrics used to measure accuracy, latency, and throughput.

Unlike traditional benchmarking approaches that perform a single evaluation pass over the dataset, AI-BMT introduces **Min Duration** and **Min Epochs** parameters to ensure stability and reliability in performance measurements. These parameters enforce a minimum evaluation duration and a minimum number of epochs, mitigating short-term performance fluctuations and enabling more consistent benchmarking.

By ensuring consistency and fairness in evaluations, AI-BMT enables meaningful comparisons across different hardware platforms while improving measurement robustness through repeated evaluation and dataset randomization.

5.1 Supported Scenarios

AI-BMT incorporates benchmarking scenarios from MLPerf Inference, which are widely recognized for their alignment with industry and academic standards. These scenarios are designed to reflect real-world AI application demands and have been validated through comprehensive research and feedback from MLPerf Inference's diverse membership, including both customers and vendors. By adopting these established scenarios, AI-BMT ensures a reliable and standardized framework for evaluating AI hardware and software performance.

While AI-BMT follows the fundamental benchmarking principles of MLPerf, it introduces several enhancements tailored for evaluating Edge AI hardware, including NPUs. The differences in benchmarking scenarios are:

- Memory and Computational Constraints: Unlike MLPerf, which primarily targets cloud and high-performance computing environments, AI-BMT is designed to operate on devices with limited memory and compute resources. It optimizes data loading strategies within the app by introduc-

ing User-Defined RAM Allocation, allowing users to specify the maximum RAM usage to prevent memory overflows.

- Flexibility in Query Size: MLPerf enforces fixed query size for each scenario, whereas AI-BMT allows for several query size options in the Multi-Stream scenario. This flexibility accommodates the varying capabilities of NPUs and other AI accelerators, enabling a more adaptive evaluation framework.

Table 3 provides an overview of the query size configurations for each benchmarking scenario, ensuring consistency in performance evaluation. The detailed description of scenarios can be found in Appendix D.

Table 3: Query Size corresponding to each scenario

Scenario	**Query Size**
Single-Stream	1
Multi-Stream	2, 3, 4, 5, 6, or 8
Offline	min(max_ram_loaded_samples, total_samples)

Note: The Query size for the **Offline** scenario is determined as the minimum of the maximum number of samples that can be loaded into RAM at once and the total number of available samples.

5.2 Evaluation Metrics

This section introduces the evaluation metrics for assessing the performance of the APUs, which are generated by the App and uploaded to the Database. The selected metrics ensure a comprehensive assessment of AI model inference, focusing on both prediction accuracy and computational efficiency in real-world scenarios.

AI-BMT's evaluation framework is designed to provide a balanced view of AI performance by incorporating both Performance Metrics (accuracy and inference reliability) and Time Metrics (latency and throughput). Unlike general-purpose AI benchmarks, AI-BMT prioritizes real-time processing efficiency, making it particularly suitable for benchmarking Edge AI hardware such as NPUs.

To further enhance measurement stability and reliability, we introduce the **min duration** and **min epochs** parameters, ensuring that evaluation runs for a predefined minimum duration and number of epochs.

Unlike traditional AI benchmarks that focus primarily on either accuracy or speed, AI-BMT integrates both perspectives into a single test mode to provide a balanced performance evaluation. The detailed description of metrics can be found in Appendix E.

5.3 Repeated Evaluation and Stability Metrics

To address performance fluctuations in short-duration evaluations, AI-BMT incorporates a **Repeated Evaluation** mechanism with two key parameters:

- **Min Duration**: Ensures that evaluations run for at least a specified time (e.g., 10 minutes).
- **Min Epochs**: Enforces a minimum number of complete evaluation epochs (e.g., 3 epochs).

During benchmarking, if either the total evaluation duration is below the **Min Duration** threshold or the number of completed epochs is less than **Min Epochs**, the evaluation is repeated. This iterative process continues until both conditions are met, ensuring stable and reliable performance measurements.

Mathematically, the evaluation must satisfy the following conditions:

$$\sum_{n=0}^{i} T_n \geq \text{min duration} \tag{1}$$

$$(i+1) \geq \text{min epochs} \tag{2}$$

where T_n represents the execution time of each iteration, and $(i+1)$ is the total number of completed epochs.

The final performance result is computed as the average over all repeated evaluations:

$$\text{Performance}_{\text{final}} = \frac{1}{N} \sum_{n=0}^{i} \text{Performance}_n \tag{3}$$

where N is the number of evaluation repetitions.

By applying this approach, AI-BMT enhances benchmarking reliability, particularly for evaluations conducted on hardware with fluctuating processing efficiency.

With multiple evaluation epochs, AI-BMT also extracts statistical measures per epoch to analyze consistency and long-term stability. The stability metrics also can be found in Appendix E.

5.4 Dataset Random Shuffling Strategy

Before each epoch execution, the dataset is randomly shuffled to prevent hardware optimization based on fixed data order. Unlike MLPerf, which uses a fixed shuffling seed, our approach ensures true randomness at each epoch.

MLPerf fixes the shuffle seed so that all submitters evaluate with the same data distribution, ensuring fairness in performance mode, where only a subset of the dataset is tested. However, this approach allows submitters to memorize the shuffled order and optimize execution based on data sequence.

In contrast, AI-BMT evaluates the entire dataset in each epoch, ensuring that every sample is tested once per epoch. To prevent any advantage from

memorizing the data order, we apply random shuffling before every epoch. This approach guarantees that all submitters evaluate under an identical data distribution (uniform distribution) with a completely randomized order, effectively eliminating any dataset sequence-based optimizations.

After applying **Min Duration**, **Min Epochs**, and **Dataset Random Shuffling**, the benchmark evaluation flow is depicted in Figure 5. The **Perform Evaluation on Dataset** process in Figure 5 corresponds to the **Benchmark Process** and **All Data Processed** steps in Figure 2.

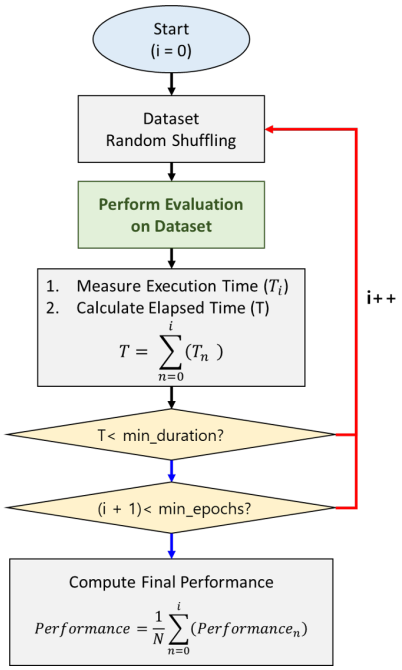

Figure 5: Overall Benchmark Evaluation Flow

6 Consistent Query Sizes and RAM Optimization

In machine learning inference systems, maintaining consistent query sizes is crucial for stable latency and throughput measurements. Variations in query sizes can lead to performance fluctuations and edge cases.

To ensure reliable evaluation, we introduce the **Benchmark Set**, consisting of dataset samples evenly divisible by query sizes (2, 3, 4, 5, 6, and 8). Accuracy evaluation is performed on the entire dataset, including both the **Benchmark Set** and the **Residual Set**, which contains remaining samples.

In **Multi-Stream** and **Offline** modes, the dataset size must be evenly divisible by the selected query size to prevent the last query from receiving fewer samples, which could skew performance metrics. For instance, in an **image classification task** with 50,000 samples and a query size of 6, the last query would contain only **2 samples instead of 6 samples** ($50,000\%6 = 2$), introducing an **edge case** that may impact overall system performance.

6.1 Dataset Size Adjustment

To eliminate last-query inconsistencies, datasets are split into:

- **Benchmark Set**: Ensures stable inference measurements by using samples evenly divisible by valid query sizes.
- **Residual Set**: Used for accuracy evaluation but excluded from latency and throughput measurements.

For different tasks, the dataset is structured as follows:

- **Image Classification**: 50000 samples split into **Benchmark Set**(9960), **Residual Set**(40)
- **Object Detection**: 4952 samples split into **Benchmark Set**(4920), **Residual Set**(32)

With this approach:

- **Latency and throughput** measurements are strictly performed using the **Benchmark Set**, ensuring that all query sizes receive an equal number of samples per query.
- **Accuracy** evaluation is performed using the **entire dataset** (Benchmark + Residual Set), preserving full dataset integrity.

This guarantees consistent query sizes while preserving full dataset accuracy evaluation.

6.2 RAM Optimization

To validate the RAM optimization approach, inference performance was tested using the DeepX-M1 NPU under different configurations.

6.2.1 RAM Allocation Strategy

In Offline mode, the number of samples loaded into RAM determines query size. To maintain consistency, only dataset sizes that evenly divide the Benchmark Set were allowed, ensuring stable inference performance. For instance, in **Image Classification** (9960 samples), supported RAM sizes include 9960, 4980, 2490, 1984, 1245, 996 etc.

By enforcing these constraints, query sizes remain uniform, preventing anomalies in performance measurements.

6.2.2 (Offline) Throughput evaluation

Since the DeepX-M1 NPU has three cores for processing, dataset sizes for RAM allocation validation (996, 1245, 1984, and 2490) were chosen to be divisible by three. Each setting was evaluated 12 times. Table 4 and Figure 6 show that while minor variations exist, RAM allocation does not statistically impact throughput.

Table 4: Offline RAM allocation evaluation

Number of query	RAM loaded samples	Avg. FPS
4	2490	821.40
5	1984	820.93
8	1245	821.02
10	996	820.81

6.2.3 (Single-stream) Latency evaluation

In the single-stream scenario, each query contains a single sample, processed by one NPU core. Each configuration was tested 12 times. Table 5 and Figure 6 show that average latency measurements remained stable across evaluations, suggesting that system noise, rather than RAM allocation, is the source of minor fluctuations.

Table 5: Single-stream RAM allocation evaluation

Number of query	RAM loaded samples	Avg. Latency (ms)
9960	2490	2.714
9960	1984	2.718
9960	1245	2.716
9960	996	2.711

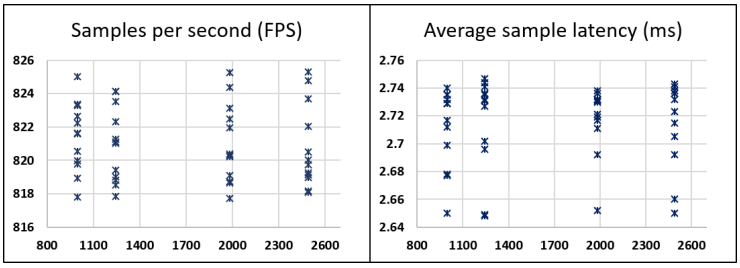

Figure 6: RAM optimization FPS/Latency test results

6.3 Benefits

Ensuring consistent query sizes eliminates edge cases, maintains stable performance metrics. Structuring datasets into a **Benchmark Set** and a **Residual Set** ensures the same amount of samples per query in all inference scenarios.

Furthermore, since RAM allocation does not impact inference performance, the AI-BMT framework ensures fair evaluation even in memory-constrained environments like NPUs. This guarantees unbiased, reproducible results across different hardware environments.

7 Performance Analysis

The AI-BMT platform demonstrates significant advancements in benchmarking APUs, addressing challenges in usability, fairness, and scalability. By incorporating a standardized APU interface, the platform minimizes evaluator workload, simplifies audit processes, and ensures consistent benchmarking results. This section presents the quantitative and qualitative outcomes achieved through the platform.

7.1 Key Benefits and Conclusion

The integration of a standardized APU interface has resulted in significant improvements:

- **Reduced Workload:** Evaluators no longer need to modify the core benchmarking logic, significantly lowering the learning curve and setup time compared to traditional platforms like MLPerf Inference.
- **Enhanced Modularity:** By isolating APU-specific logic, new hardware can be integrated with minimal code changes, enabling scalable benchmarking for future APU models.
- **Streamlined Audit Process:** The reduction in modifiable code simplifies verification and ensures consistent evaluations.
- **Improved Automation:** Features such as dataset validation and runtime configuration reduce manual intervention, minimizing errors and enhancing reproducibility.
- **Faster Hardware Adaptation:** Users reported a significant reduction in the time required to adapt new hardware for benchmarking, as only the APU interface logic needs to be implemented.
- **Stable Performance Metrics:** AI-BMT eliminates last-query inconsistencies by structuring datasets into Benchmark Set and Residual Set, ensuring fairness in latency and throughput measurements

- **Improved Performance Stability:** Min Duration and Min Epochs ensure that evaluations run for a sufficient duration and multiple repetitions, evaluating fluctuations in inference performance.
- **Prevention of Overfitting to Data Order:** Random Shuffling before each epoch prevents inference optimizations based on fixed dataset sequences, ensuring unbiased benchmarking results.
- **Optimized Data Loading for Faster Evaluation:** Double Buffering significantly reduces preprocessing overhead by overlapping data preprocessing and inference execution, leading to faster benchmark completion.

These advancements confirm that AI-BMT successfully reduces evaluator workload, ensures consistent benchmarking, and supports scalable APU integration. AI-BMT sets a new standard for APU benchmarking, addressing the limitations of existing platforms while offering a flexible and future-proof solution.

7.2 Quantitative Comparison

AI-BMT combines the strengths of App based AI-Benchmark and Open Source based MLPerf Inference while addressing their limitations and enhances accessibility and fairness through the integration of App, WAS, Database and File System. Table 6 provides a quantitative comparison of AI-BMT with MLPerf Inference and AI-Benchmark across key evaluation criteria. The table highlights how AI-BMT achieves enhanced fairness, reduced evaluator burden, and better support for dataset and label validation and new APUs compared to the other platforms.

Table 6: Comparison of AI-BMT with other platforms.

Feature	**MLPerf**	**AI-Benchmark**	**AI-BMT**
Evaluation Fairness	△	O	O
Ease of Use	△	O	O
Ease of Audit	△	O	O
Dataset Offering	×	O	O
Dataset Validation	×	O	O
Support for New APU	O	×	O
Prevents Data Order Bias	△	O	O
Stability Test	O	×	O
Efficient Preprocessing	×	-	O
Integrated Evaluation Mode	×	O	O

8 Findings and Future Work

The findings highlight the advantages of AI-BMT in addressing the limitations of existing benchmarks. Key contributions include:

- **Improved Usability**: Automated validation, simplified setup, and hardware abstraction notably reduced evaluator workload, minimizing manual effort and streamlining the benchmarking process. Additionally, RAM optimization strategies ensure fair evaluation even in memory-constrained hardware environments.
- **Fairness and Reproducibility**: Standardized interfaces and automated validation ensure consistent results across evaluators. Fairness is reinforced by automatically uploading evaluation results to a database, preventing manipulation and ensuring transparency. The introduction of Consistent Query Sizes ensures uniform evaluation across benchmark runs, while Random Shuffling mitigates sequence-based optimizations, leading to unbiased performance assessments.
- **Efficiency Improvements**: By implementing Double Buffering, AI-BMT parallelizes data preprocessing with inference execution, significantly reducing evaluation time while preserving accuracy.
- **Scalability**: The platform's modular design allows seamless integration of new models and datasets, providing flexibility to adapt to evolving benchmarking requirements.

8.1 Limitations

- Validation for custom-trained models requires a manual audit process, emphasizing the need for automated model validation to ensure fair comparisons and further simplify the audit process.
- Benchmarking logic is currently centralized in C++ and lacks support for other programming languages, limiting accessibility for users who prefer different environments.
- Currently supports only online benchmarking, requiring an internet connection and adherence to the official benchmarking process.
- Power efficiency and thermal management metrics are not yet supported, as their implementation requires the development of specific hardware and integration with the BMT app.

8.2 Future Directions

AI-BMT will continue to evolve with additional features and expanded benchmarking capabilities. Planned updates include support for new AI tasks, automated model validation, Python compatibility, offline benchmarking, and

hardware-level power and thermal efficiency measurements. A detailed roadmap for these enhancements is provided in Appendix F.

9 Summary

AI-BMT offers a groundbreaking solution for benchmarking APUs by addressing the limitations of existing platforms such as MLPerf Inference and AI-Benchmark. By integrating features like Hardware Abstraction, Automated Validation, Min Duration & Epochs, Random Shuffling, Double Buffering, RAM Optimization, and Consistent Query Size, AI-BMT sets a new standard for fairness, usability, and scalability in AI benchmarking.

This platform leverages a robust File System to provide evaluators with the necessary models and datasets seamlessly, ensuring consistency across evaluations. Furthermore, all evaluation results are automatically uploaded to a secure Database, making any tampering with evaluation data virtually impossible. These features collectively enhance the reliability and transparency of the benchmarking process, further solidifying AI-BMT as a future-proof solution for APU evaluation.

It's modular design and flexible architecture enable seamless integration with emerging AI workloads and hardware innovations, making it a scalable solution for future developments. With continued advancements, such as the planned Offline Mode, Python support, and hardware metrics measurement, AI-BMT is poised to become the de facto standard for evaluating APUs. By addressing both current and future challenges in AI benchmarking, AI-BMT aims to drive innovation and standardization in the field.

A AI-BMT Component Features

The AI-BMT components provide an intuitive and efficient interface for running benchmarks, managing datasets, and accessing results. The key features include:

- **BMT WAS**:
  - Serves as the central hub connecting the evaluator, the BMT GUI App, the database and file system.
  - Facilitates secure access through authentication (Sign In, Sign Up, Password Reset).
  - Handles evaluation results: Stores private results in individual databases and public results in the public database for cross-organization comparisons.
- **BMT GUI App**:
  - Automatically validates dataset and label during benchmarking.
  - Uploads evaluation results to the private database upon successful benchmarking.
  - Provides an intuitive interface for evaluators with various options and features, including:
    * **Model/Dataset Download**: Enables downloading of models and datasets based on user requests.
    * **Authentication**: Offers login functionality to ensure secure access to the personal database.
    * **Data Upload/Check**: Supports data upload and status verification through database integration.
    * **Dataset Validation**: Automatically checks the presence and correctness of dataset and label files before benchmarking. If validation fails, the evaluation is halted with an error message.
    * **GUI Option Saving and Loading**: Allows users to save and restore their configuration settings for convenient reuse.
    * **Dark/Light Theme Selection Mode**: Enables switching between dark and light themes for user preference.
    * **Task Selection**: Supports multiple tasks, including image classification and object detection. Other tasks will be added in the future update.
    * **Benchmark Scenario Selection**: Supports multiple scenarios, including Single-Stream, Multi-Stream, and Offline.
    * **RAM Configuration Feature**: Allows users to configure the maximum dataset size loaded into RAM, optimizing resource usage.

    - **Double Buffering**: Reduces evaluation time by overlapping preprocessing and inference. Users can enable or disable this option through the GUI.
    - **DMQE (Dynamic Model Queue Evaluation)**: Allows users to dynamically select and queue multiple models for sequential evaluation within a single execution session.
    - **Warm-Up Process**: Provides an option to perform a warm-up process before starting evaluations to stabilize performance. Users can enable or disable this option through the GUI.
    - **Debug Options**: Enables fast iteration by skipping validation and database upload. A lightweight test dataset (120 samples) is also provided for quick debugging without running full evaluation epochs.
    - **Stop BMT**: Enables users to interrupt benchmarking at any time.
    - **Evaluation Log Viewer**: Allows users to save, delete, and browse log messages from past evaluations.
    - **Data Analysis (App)**: Provides tabular views of evaluation results with sorting, deletion, copying, and CSV export.

- **APU Evaluator**:
  - Registers and logs into the platform to securely access their account-bound private database for benchmarking services.
  - Implements the required interface and builds the BMT GUI App tailored to their hardware.
  - Downloads models and dataset through the BMT GUI App for the interested task.
  - Performs benchmarking using the BMT GUI App and accesses evaluation results through the BMT GUI App or the BMT WAS.

- **Databases**:
  - **Private Database**: Stores evaluation results privately for individual users, ensuring restricted access unless explicitly shared.
  - **Public Database**: Stores shared benchmarking results, allowing transparent comparisons across different organizations.

- **File System**:
  - Provides datasets and models. Accessible through the BMT GUI App for seamless integration into the benchmarking process.

- **BMT Evaluation Server**:

- A dedicated server used to compute evaluation metrics (e.g., mAP) using Python-based official libraries such as `pycocotools`. This server supports cases where direct C++ implementation is impractical or unavailable, ensuring consistent and standardized evaluation results across submitters.

B Overall Benchmarking Flow

The following section provides a detailed explanation of the process illustrated in Figure 2.

- **Interface Implementation**
  - Define and implement APU logic through a standardized interface.
- **Initial Setup**
  - Build & Execute BMT App: Build the BMT app to incorporate the interface implementaion and execute the application for benchmarking.
  - Download Dataset/Model: Download necessary dataset and model for the selected task using the BMT app.
- **Login & Start**
  - Login: Authenticate the user to ensure secure access to the personal database.
  - Start: Begin the benchmarking process by pressing the BMT start button.
- **Dataset Validation**
  - Automatically validate the dataset and corresponding labels to guarantee consistency and fairness in benchmarking.
    - If validation fails, the process is terminated (NG).
    - If successful, proceed to the next steps.
- **Benchmark Preparation**
  - Initialize(): Call the `Initialize()` method, which is implemented via the interface. It initializes the system environment for the benchmarking task, such as loading the AI model to the APU.
  - Warm-Up (Optional): Automatically load initial samples into RAM and perform a warm-up to stabilize APU performance.
- **Benchmark Process**

  - Preprocessing: Load the required preprocessed data samples into RAM to enable efficient inference. The preprocessing logic is implemented via the interface.
  - Inference: Execute inference on the loaded data. The inference logic is implemented via the interface.
  - Verification and Performance Calculation: Validate the inference outputs and compute evaluation metrics based on them.

- **Is All Data Processed ?**
  - Continuously check if all data samples are processed:
    * If not, repeat the **Benchmark Process** step for the remaining data.
    * If completed, proceed to the final stage.

- **Result Upload**
  - Store the calculated evaluation metrics in the private database for secure access and analysis.
  - The results stored in the database can be accessed from both the App and the WAS, and can also be downloaded as a CSV file.

C Supported Preprocessing Return Types

For `vector` types, memory is automatically managed and released by the standard library, ensuring efficient and safe usage during and after inference. However, for pointer types, if dynamically allocated memory is used, it is the user's responsibility to release it before the `runInference()` function completes, following the data processing and result generation. Neglecting this step could result in memory leaks, which may cause RAM to fill up rapidly. Consequently, this could lead to unfavorable conditions for hardware performance measurement.

For more details on how to manage dynamically allocated memory and ensure safe operations, refer to the provided *Reference Code*, which includes examples of proper memory allocation and deallocation procedures for pointer types.

C.1 Relevance to Edge NPU Evaluation

The selected models and datasets align with key characteristics of Edge NPUs, which require:

- Low Latency: Both ResNet-50 v2.0 and YOLOv5s are computationally efficient, ensuring that inference latency remains within real-time constraints.

- Memory Constraints: Edge devices often have limited RAM. these models have relatively small memory footprints compared to larger architectures.
- Diverse Workloads: Image classification and object detection are two of the most commonly deployed AI workloads on edge devices, making them representative benchmarks for real-world applications.

D Test Scenarios

D.1 Single-Stream

The Single-Stream scenario represents real-time applications where responsiveness is critical, simulating environments such as mobile devices and autonomous systems. In this scenario, inference queries are processed sequentially, one at a time, with a query size of 1. Performance is measured by the 90th-percentile latency of the query stream, reflecting the system's responsiveness under continuous query injection. This approach ensures accurate latency measurement.

D.2 Multi-Stream

The Multi-Stream scenario extends the Single-Stream approach to model applications that process multiple concurrent query streams. It is particularly relevant for systems that need to handle multiple data sources simultaneously, such as multi-camera surveillance, video analytics, and sensor fusion. Unlike Single-Stream, where queries are processed sequentially, Multi-Stream allows multiple queries to be executed in parallel, requiring efficient resource management. To accommodate different hardware capabilities, AI-BMT allows the query size to be selected from a set of values: 2, 3, 4, 5, 6, or 8. This flexibility ensures that each hardware configuration can be evaluated under conditions that best utilize its processing potential, providing a comprehensive view of the system's ability to handle concurrent workloads.

D.3 Offline

The Offline scenario models environments where all input data is available upfront, and processing latency is not constrained. It simulates use cases such as bulk image recognition or large-scale data analysis, where throughput is the primary performance metric. In this scenario, the system receives a single query containing all data samples and processes them as efficiently as possible. Throughput, measured in samples per second, is used to evaluate performance, highlighting the system's capacity to handle large volumes of data.

E Comprehensive Metric Descriptions

E.1 Performance Metrics

Performance metrics evaluate the effectiveness of AI models in real-world applications. These metrics measure classification and object detection accuracy.

- **accuracy**: Classification performance metric measuring the proportion of correctly classified samples.
- **mAP_50**: Mean Average Precision at an IoU threshold of 0.50, used for object detection performance evaluation.
- **mAP_50_95**: Mean Average Precision averaged across IoU thresholds from 0.50 to 0.95 in steps of 0.05, providing a more comprehensive assessment of localization precision.

E.2 Time Metrics

Time metrics assess inference speed, latency distribution, and overall system throughput in AI benchmarking. These metrics capture variations in performance over multiple epochs.

- **epoch_sample_latency_average_min**: Minimum average latency per sample across all epochs.
- **sample_latency_average**: Overall average latency per sample during evaluation.
- **epoch_sample_latency_average_max**: Maximum average latency per sample across all epochs.
- **epoch_query_latency_average_min**: Minimum average latency per query across all epochs.
- **query_latency_average**: Overall average latency per query during evaluation.
- **epoch_query_latency_average_max**: Maximum average latency per query across all epochs.
- **query_latency_median**: Median latency per query.
- **query_latency_90th**: 90th percentile query latency.
- **query_latency_95th**: 95th percentile query latency.
- **query_latency_99th**: 99th percentile query latency.
- **query_latency_max**: Maximum query latency observed.

- **epoch_samples_per_second_min**: Minimum number of samples processed per second across all epochs.
- **samples_per_second**: Average number of samples processed per second.
- **epoch_samples_per_second_max**: Maximum number of samples processed per second across all epochs.
- **epoch_queries_per_second_min**: Minimum number of queries processed per second across all epochs.
- **queries_per_second**: Average number of queries processed per second.
- **epoch_queries_per_second_max**: Maximum number of queries processed per second across all epochs.

E.3 Information Metrics

These metrics provide additional context about the benchmarking environment, such as test configurations and system parameters.

- **email**: Email of the submitter.
- **task**: The AI task being evaluated (e.g., image classification, object detection).
- **scenario**: The selected benchmarking scenario (e.g., Single-Stream, Multi-Stream, Offline).
- **query_samples**: The number of data samples contained in each query.
- **total_samples**: The total number of dataset samples used in benchmarking.
- **RAM_loaded_samples**: The number of samples loaded into RAM at once for inference.
- **epochs**: The total number of completed evaluation epochs.
- **duration_ms**: Total evaluation duration in milliseconds.

E.4 Optional Metrics

These system parameters are optionally provided by the submitter and help contextualize benchmarking results. While not mandatory, they offer insights into the hardware and software configuration used during evaluation.

- **cpu_type**: The processor model used for benchmarking (e.g., Intel i7-9750HF).

- **accelerator_type**: The AI accelerator used, such as an NPU, GPU, or TPU (e.g., DeepX M1 NPU).
- **submitter**: The organization or individual submitting the benchmark results (e.g., DeepX).
- **cpu_core_count**: The number of physical or logical CPU cores available (e.g., 16).
- **cpu_ram_capacity**: The total RAM capacity available for processing (e.g., 32GB).
- **cooling**: The type of cooling system used for the CPU and accelerator (e.g., Air, Liquid, Passive).
- **cooling_option**: Specifies whether the cooling system is active (with a fan or pump) or passive (without additional cooling mechanisms).
- **cpu_accelerator_interconnect_interface**: The communication interface between the CPU and accelerator (e.g., PCIe Gen5 x16).
- **benchmark_model**: The deep learning model used in the benchmark evaluation (e.g., ResNet-50).
- **operating_system**: The OS environment in which benchmarking was conducted (e.g., Ubuntu 24.04 LTS).

F Planned Features and Enhancements

F.1 Expansion of Supported Tasks

At present, AI-BMT supports benchmarking for Image Classification and Object Detection tasks. However, future updates will include support for additional tasks, such as:

- **Natural Language Processing (NLP)**: Tasks like text generation, sentiment analysis, and machine translation, which are critical for evaluating Large Language Models (LLMs).
- **Super Resolution**: Image and video upscaling for applications in areas such as medical imaging, video streaming, and content creation.
- **Face Detection**: Performance evaluation for real-time and batch processing in security, authentication, and multimedia applications.
- **Other Emerging Tasks**: New AI workloads as they become relevant in academic and industrial benchmarking.

F.2 Automated Model Validation

Automated model validation is planned for future implementation. This feature is necessary because models can be recompiled to fit specific APUs. However, to ensure fair comparisons, the model training parameters must remain unchanged. Therefore, validation logic is required to verify that the compilation process has been conducted without altering the parameters.

F.3 Python Support

Currently, the benchmarking logic is centralized in C++ and supports evaluation exclusively through C++. In the future, a Python wrapper will be provided to enable the same benchmarking functionality in Python. This wrapper will allow Python calls to internally invoke C++ binary functions, ensuring seamless integration. Additionally, this design will allow for the potential support of additional programming languages, further expanding the platform's usability.

F.4 Offline Mode Support

Currently, the platform supports only the online mode, which enables automatic validation, data download, and database upload through the WAS after logging in. While this mode ensures convenient and fair evaluations, Offline Mode will be added in the future. This mode will allow evaluations to be performed without internet connectivity using only the App, enabling users to freely evaluate their APUs without restrictions or validation. By removing validation requirements, Offline Mode provides users the flexibility to experiment and test their systems in customized way.

F.5 Hardware Metrics Measurement

Future updates will include hardware metrics measurement to evaluate power efficiency and thermal management capabilities. Key metrics to be measured include:

- Temporal changes in power consumption
- Maximum power consumption
- Idle power consumption
- Performance-per-watt ratio
- Average and maximum temperature measurement
- Tracking performance changes based on temperature

These features aim to provide comprehensive insights into the hardware's power and thermal efficiency. Currently, specialized hardware for measuring the temperature and current of APUs is being developed. For users who purchase this

hardware, the BMT GUI App will automatically measure power and temperature metrics during benchmarking, providing seamless integration of hardware metrics into the benchmarking process.

G AI-BMT Platform Resources

For reference, the following information provides access to AI-BMT platform resources:

- **AI-BMT WAS**: https://ai-bmt.com
- **AI-BMT App**: These applications also can be accessed through AI-BMT WAS:
  - Windows (x86_64): `https://github.com/kinsingo/AI_BMT_GUI_Submitter_Windows`
  - Linux (x86_64): `https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux`
  - Linux (ARM64): `https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux_ARM64`
- **AI-BMT Demo Videos**: `https://www.youtube.com/channel/UCeq7Wu3knGqw6X6X6NglyKg`