AI-BMT Alpha Version User Manual

March 2025

Contents

1 Introduction

AI-BMT is a benchmarking platform designed to evaluate AI workloads on various hardware configurations. This manual provides detailed instructions for installation, usage, and troubleshooting of the Alpha version.

2 Installation and Setup

2.1 System Requirements

- OS & CPU: Windows 10/11 (x86_64), Linux (x86_64 or ARM64).
- CPU RAM: Minimum 2GB

Note: For Linux(ARM64), GLIBC version 2.38 or higher is required. For example, Ubuntu 24.04 uses GLIBC 2.39(≥ 2.38) *and therefore compatible with our platform.*

2.2 Installation Steps

- **Windows:** It is recommended to use the current project setup and Visual Studio as the IDE.
- **Ubuntu:** Open a terminal and run the following commands to install CMake, g++ compiler, Ninja Build System, and EGL Library.

```
sudo apt update
sudo apt install cmake
sudo apt install build-essential
sudo apt-get install ninja-build
sudo apt install libgl1 libgl1-mesa-dev
sudo apt install unzip
```

- **Details:** For more details about how to build and run the Program please refer to:
  - Windows(x86_64, CPP): `https://github.com/kinsingo/AI_BMT_GUI_Submitter_Windows/blob/main/ReadMe.md`
  - Linux(x86_64, CPP): `https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux/blob/main/README.md`

- Linux(ARM64, CPP):https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux_ARM64/blob/main/README.md
- Windows(x86_64, Python): https://github.com/kinsingo/AI_BMT_GUI_Submitter_Windows_Python/blob/main/ReadMe.md
- Linux(x86_64, Python): https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux_Python/blob/main/ReadMe.md
- Linux(ARM64, Python):https://github.com/kinsingo/AI_BMT_GUI_Submitter_Linux_ARM64_Python/blob/main/ReadMe.md

3 Key Features and Workflows

3.1 Benchmarking Pipeline

- **Model&Dataset Preparation:** Download from the app or WAS.
- **Interface Implementation:** Implement the provided interface.
- **Build&Execution:** Runs AI workloads on the target hardware.
- **Results Visualization:** View evaluation result from the app or WAS.

3.2 Step-by-Step Usage Guide

3.2.1 Accessing the WAS

Visit *ai-bmt.com* to access the Web Application Server.

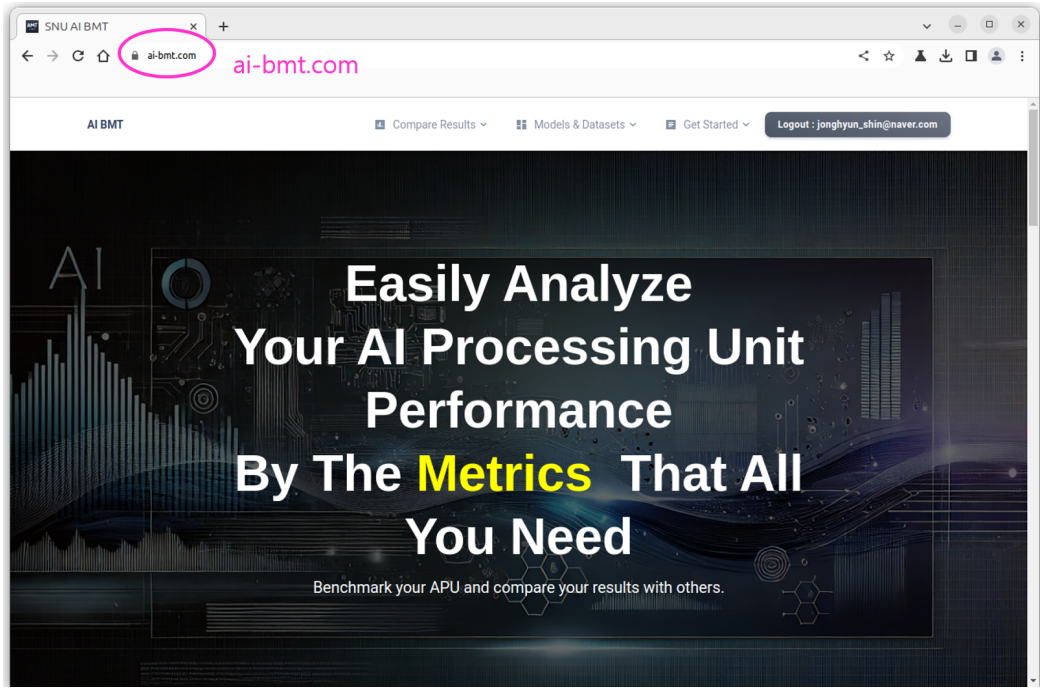

Figure 1: WAS

3.2.2 How to start

Scroll down to the **How to Start** section, then select the target operating system and CPU architecture. In this demonstration, an Orange Pi 5 Plus running Ubuntu 24.04 is used to evaluate the NPU; therefore, the rightmost option is selected to navigate to the corresponding GitHub page.

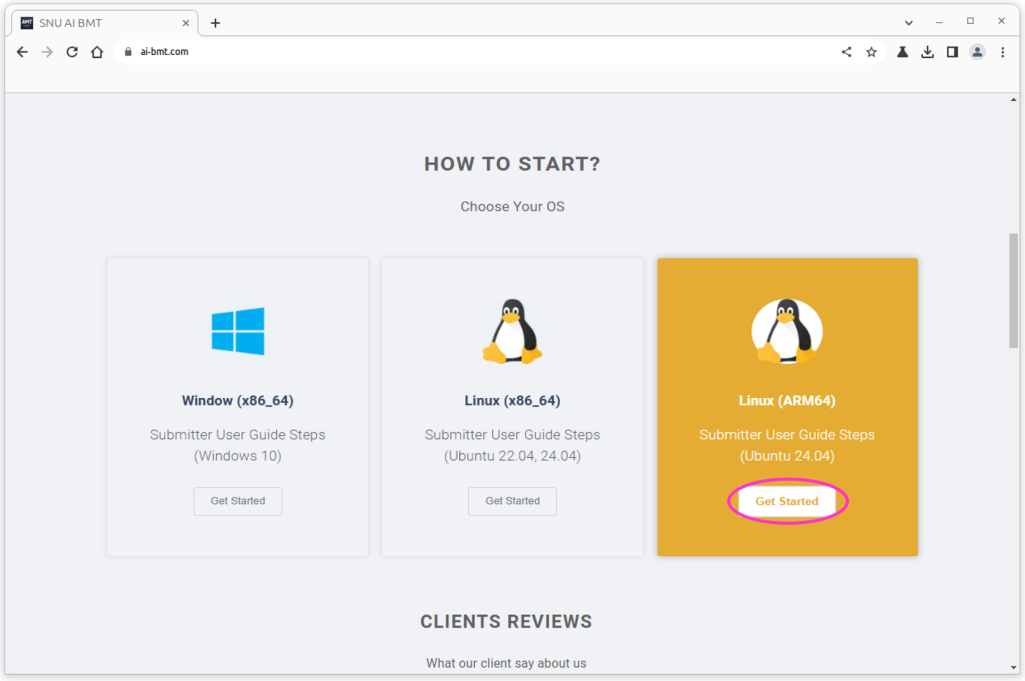

Figure 2: How to start

3.2.3 GitHub Page

As shown in Figure 3, you can copy the repository address from the navigated GitHub page to clone it later. For instructions on how to build and run the program, please refer to the **README.md** file on the GitHub page.

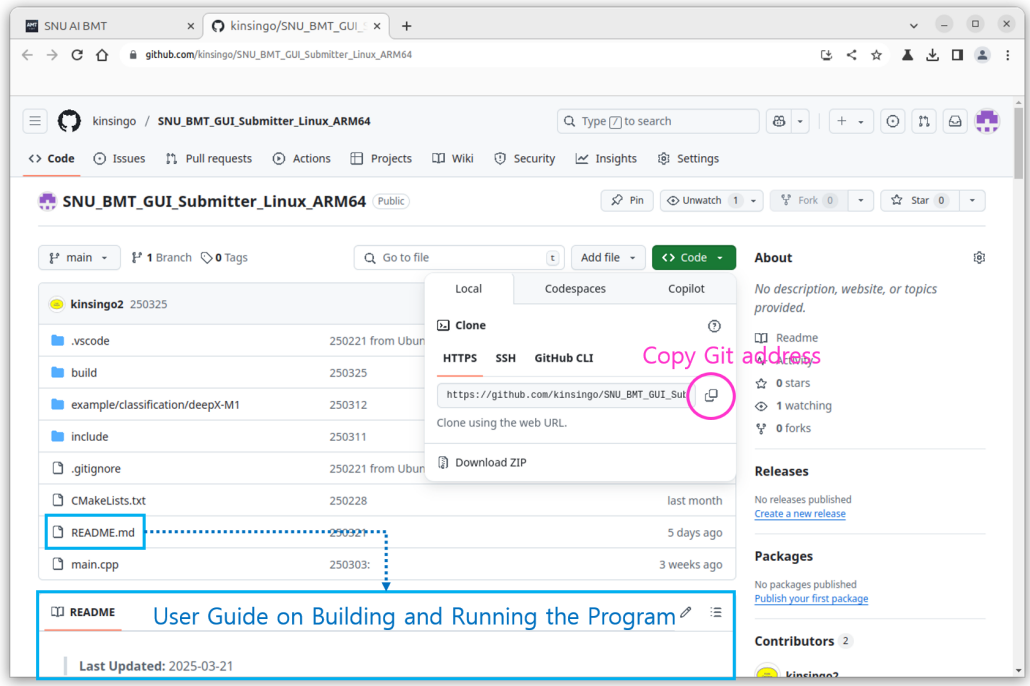

Figure 3: Github Page

3.2.4 Cloning the Git Repository

As shown in Figure 4, clone the copied Git repository address into your preferred directory.

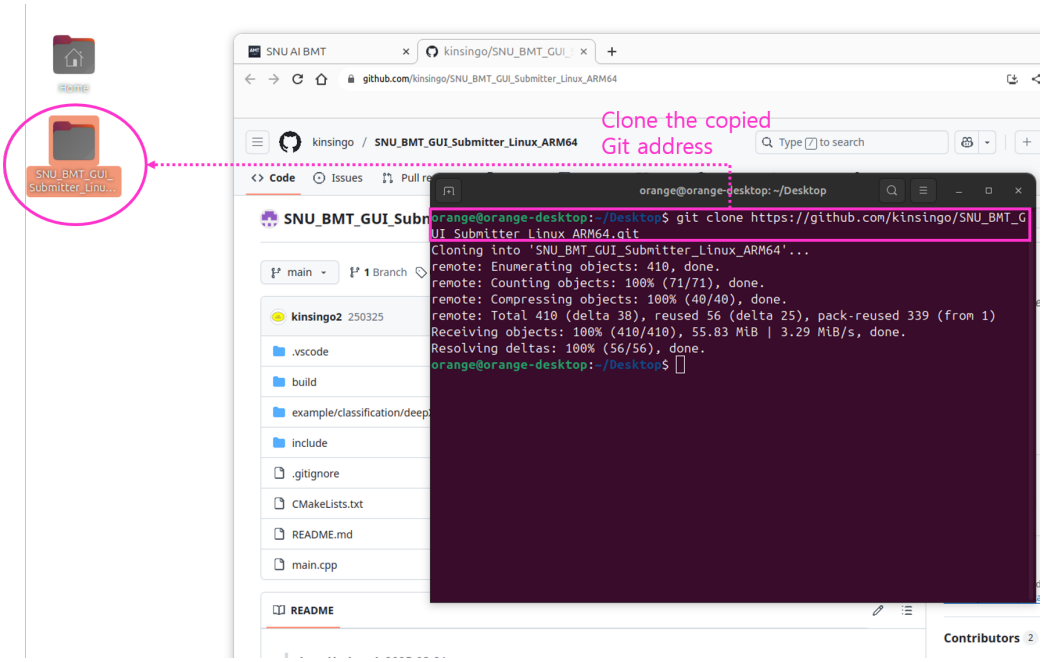

Figure 4: Cloning the Git Repository

3.2.5 Interface Implementaion

Implement **SNU_BMT_Interface** by inheriting and overriding its virtual methods. User can directly implement **Virtual_Submittter_Implementation** class in main.cpp, which is then passed to the GUI caller object. Additionally, the model path should be correctly provided to check that the model file exists at the specified location before the evaluation start. You may refer to the files in the example folder for a reference implementation

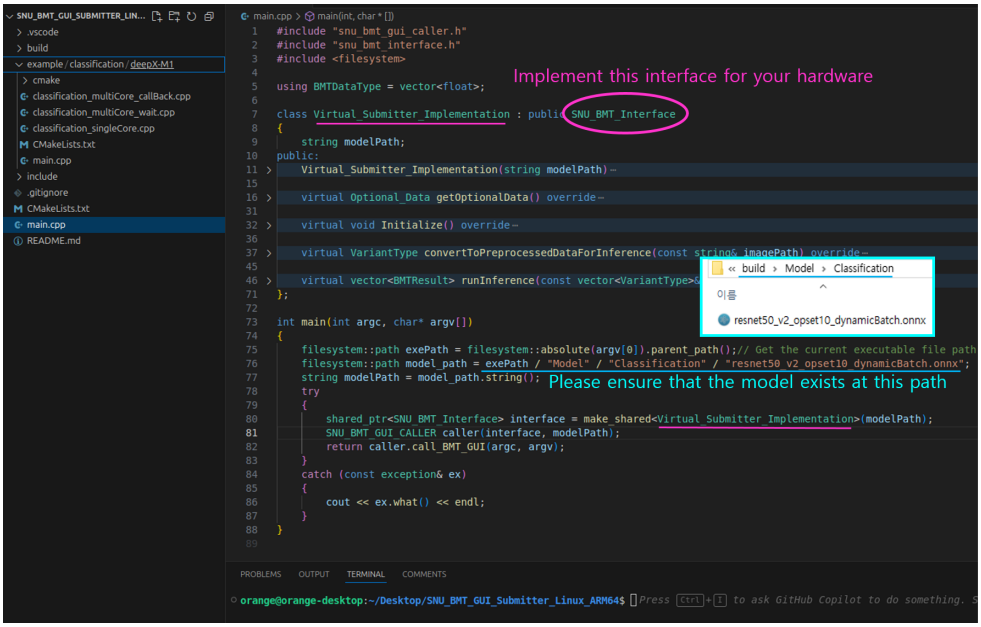

Figure 5: Interface Implementaion

3.2.6 Build and Execution

As shown in Figure 6, the program can be built and executed by following the commands provided in **README.md**. After running these commands, the GUI will launch.

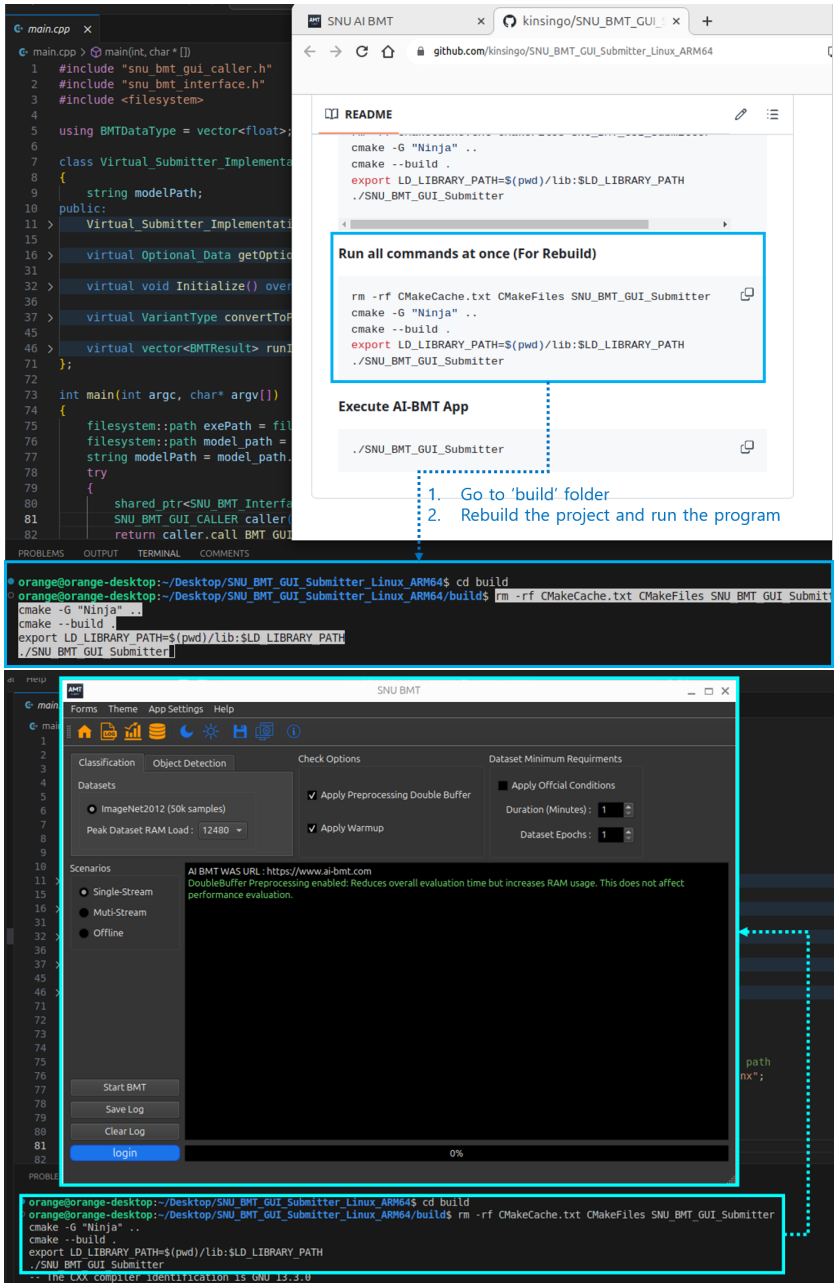

Figure 6: Build and Execution

3.2.7 GUI Options (1)

As illustrated in Figure 7, the GUI offers intuitive controls for customizing benchmark settings. The numbered components are described as follows:

1. **Task Selection**: Select either **Image Classification**, **Object Detection**, or **Segmentation**.

2. **RAM Allocation**: Specify the number of samples to be loaded into RAM at once. A smaller number is recommended to ensure sufficient memory for enabling Double Buffering.

3. **Check Options**: **Double Buffering** reduces evaluation time by overlapping preprocessing and inference and **Warm-Up** stabilizes hardware before benchmarking.

4. **Minimum Requirements**: If apply official condition, will fix **Min Duration** as 10 minutes and **Min Epochs** as 3.

5. **Scenario Selection**: Choose among **Single-Stream**, **Multi-Stream**, and **Offline** scenarios.

6. **Evaluation Status Panel**: Displays benchmarking logs and status messages in real time.

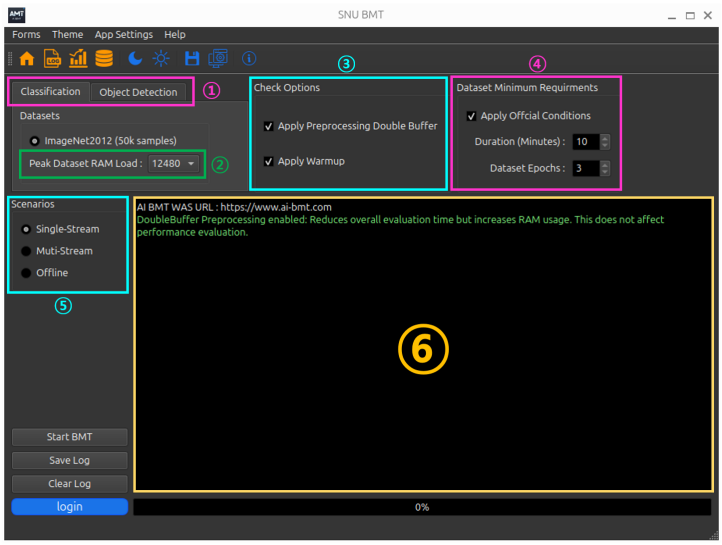

Figure 7: GUI Options (1)

3.2.8 GUI Options (2)

As shown in Figure 8, the GUI includes additional features for usability and benchmarking control. The numbered components are:

1. **Page Navigation**: The leftmost tab is the **Main Page**, followed by the **Log Viewer**, **Data Analysis**, and **Model/Dataset Download** pages. These will be explained in detail later.

2. **Theme Selection**: Allows users to switch between dark and light themes based on personal preference.

3. **Option Save/Load**: Saves the current GUI configuration or loads previously saved settings. The saved options are automatically applied upon restarting the app.

4. **Control Panel**: User can start evaluation by clicking **Start BMT** button. also can save or clear the log messages in **Evaluation Status Panel**. To access private database, user can login using **login** button.

5. **Progress Bar**: Displays the progress of time-consuming processes such as dataset validation and epoch evaluations.

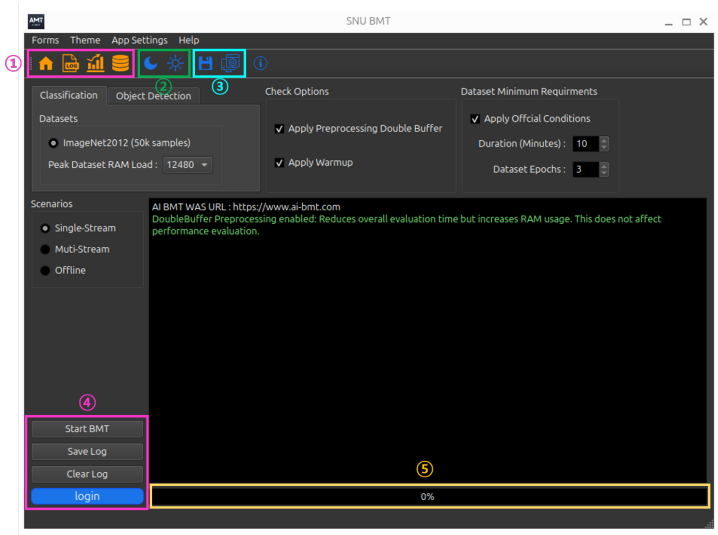

Figure 8: GUI Options (2)

3.2.9 GUI Options (3)

As shown in Figure 9, the GUI provides debugging-oriented options to support rapid development and testing of the benchmarking interface by submitters. Notable features include:

1. **Dataset Selection for Debugging**: The GUI offers two types of datasets — a full **Evaluation Set** (ImageNet2012, 50000 images) and a compact **Test Set** (ImageNet2012, 120 images). The Test Set, consisting of only 120 representative samples, allows a complete epoch to be executed quickly. This facilitates interface debugging by reducing evaluation time and resource consumption.

2. **Skip For Debugging Options**: To streamline testing, the GUI allows users to skip stages such as **Model Validation**, **Dataset Validation**, and **Database Upload**. In particular, skipping database upload removes the need for login credentials, enabling evaluations to run in an unauthenticated state.

3. **Stop BMT**: The GUI includes a **Stop** button that allows users to terminate the evaluation process mid-way.

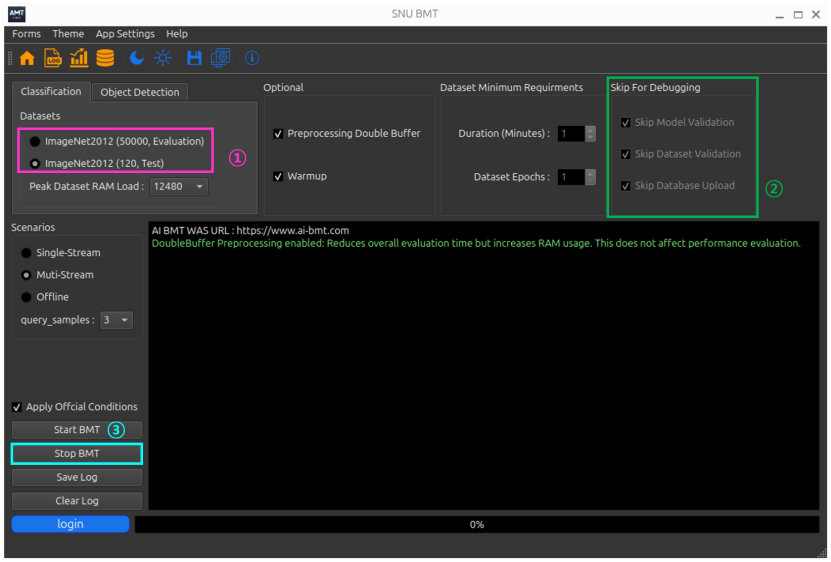

Figure 9: GUI Options (3)

3.2.10 Resizable application window

As shown in Figure 10, the application window can be resized to suit user preferences. The selected window size is also saved when using the **Option Save** feature, and automatically restored upon restarting the application.

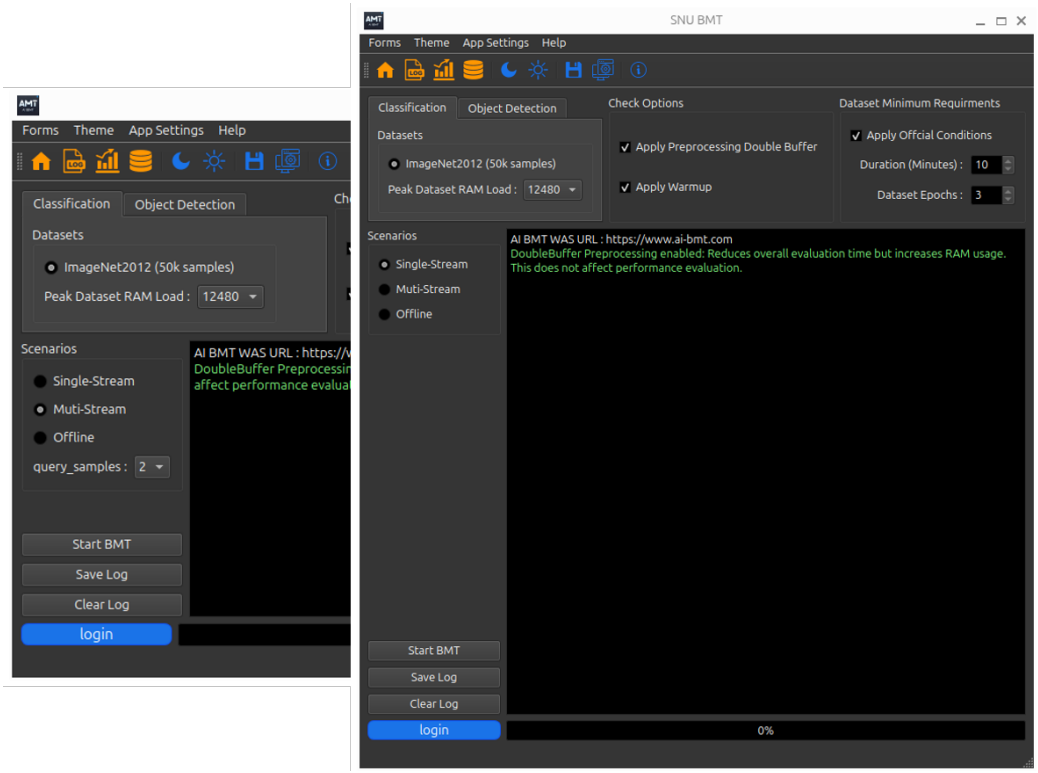

Figure 10: Resizable application window

3.2.11 Start BMT & Authentication

As shown in Figure 11, when the **Start BMT** button is clicked, the user is required to log in before the evaluation begins, if not already authenticated. If the user does not have an account, clicking the **Sign up** link will navigate to the registration page on **ai-bmt.com**.

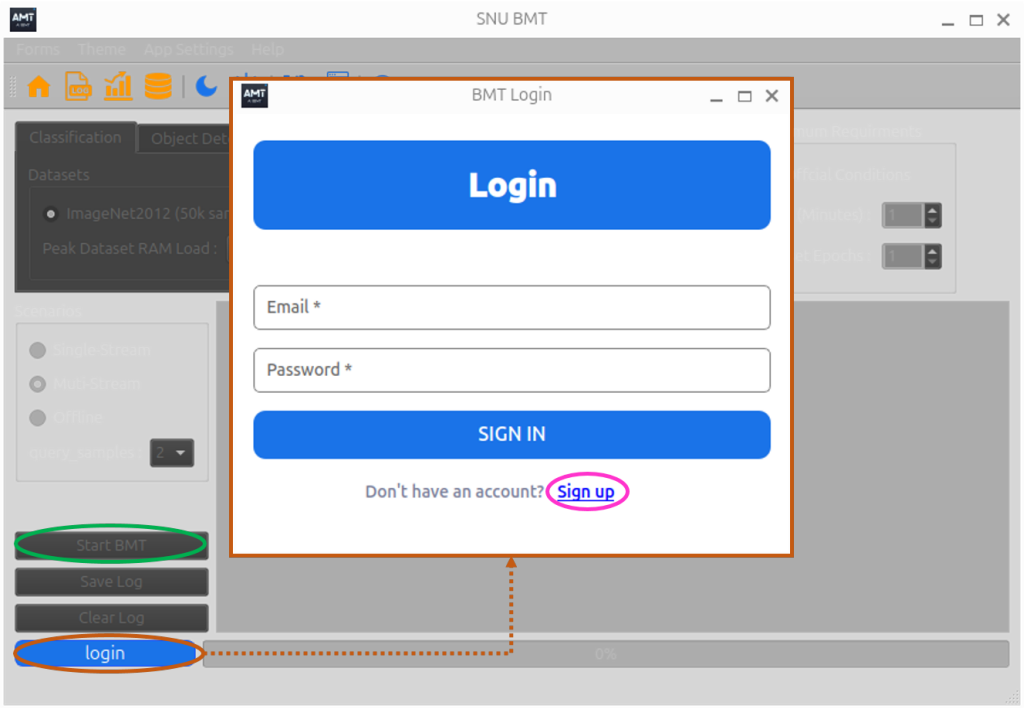

Figure 11: Start BMT & Authentication

3.2.12 Dataset Validation

As shown in Figure 12, if the required dataset for the selected task is missing, the application displays an error message and terminates the evaluation process. In this example, the model file exists and passes the validation check. If the model file was missing, a separate error message would be shown accordingly.

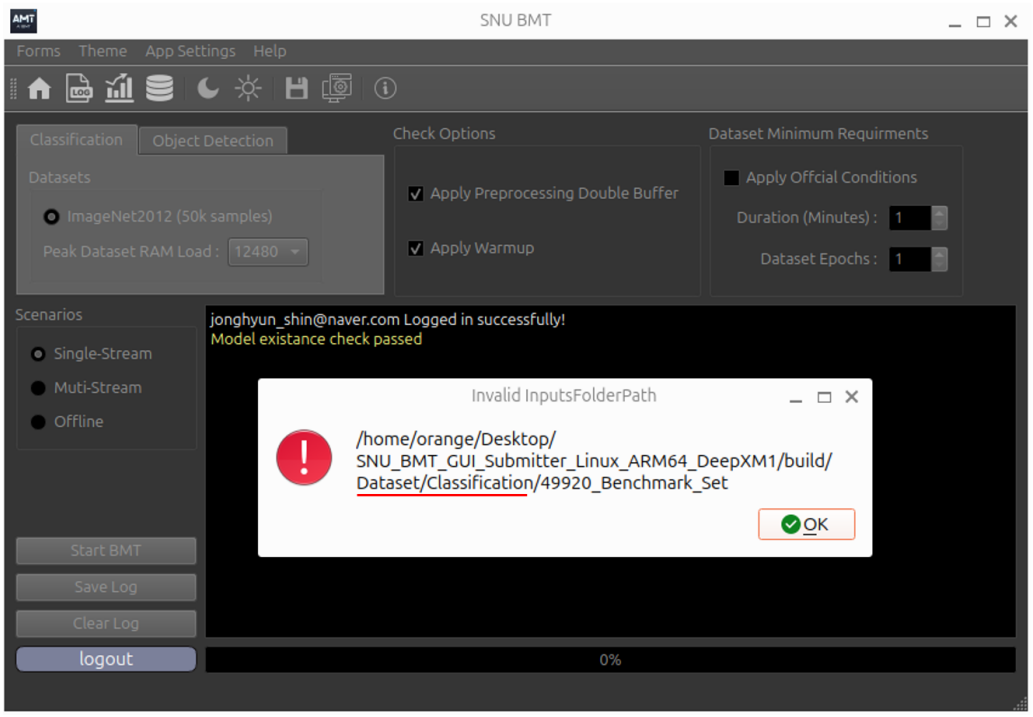

Figure 12: Error message for missing dataset

3.2.13 Model & Dataset Download

As shown in Figure 13, users can download the required model or dataset for each task by clicking the corresponding button. For example, clicking the **imagenet2012 (50000ea)** button automatically downloads the dataset and label files, extracts them to a predefined directory, and prepares them for evaluation. This entire process, including downloading and extraction, is handled automatically.

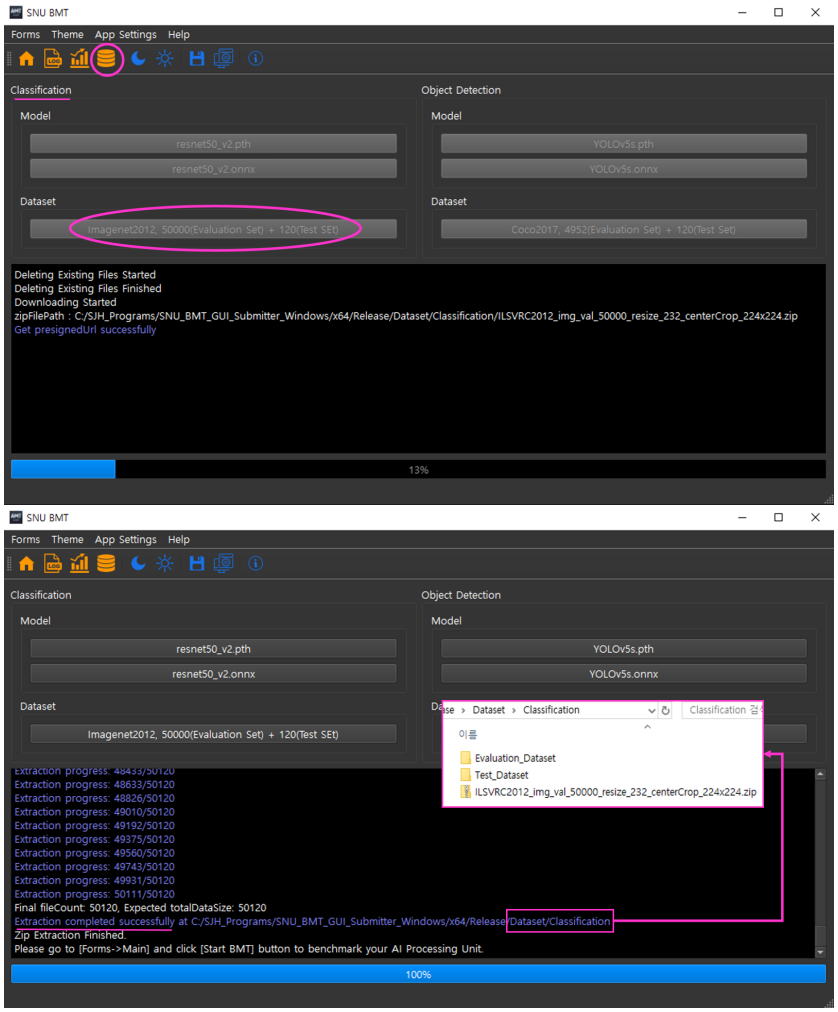

Figure 13: Model & Dataset Download

3.2.14 Benchmark Log Message (1)

As shown in Figure 14, if the **dataset and label validation** and **model existence check** pass, the evaluation begins with the **Warm-Up** process if enabled. In this example, the RAM allocation is set to 4992 samples. Once the first 4992 samples are processed, current evaluation results are displayed, and the system proceeds to the next 4992 samples.

Because **Double Buffering** is enabled, the next 4952 samples are preprocessed and loaded into a separate buffer during the inference of the current 4952 samples. This overlap reduces total evaluation time by avoiding idle periods between data preprocessing and inference.

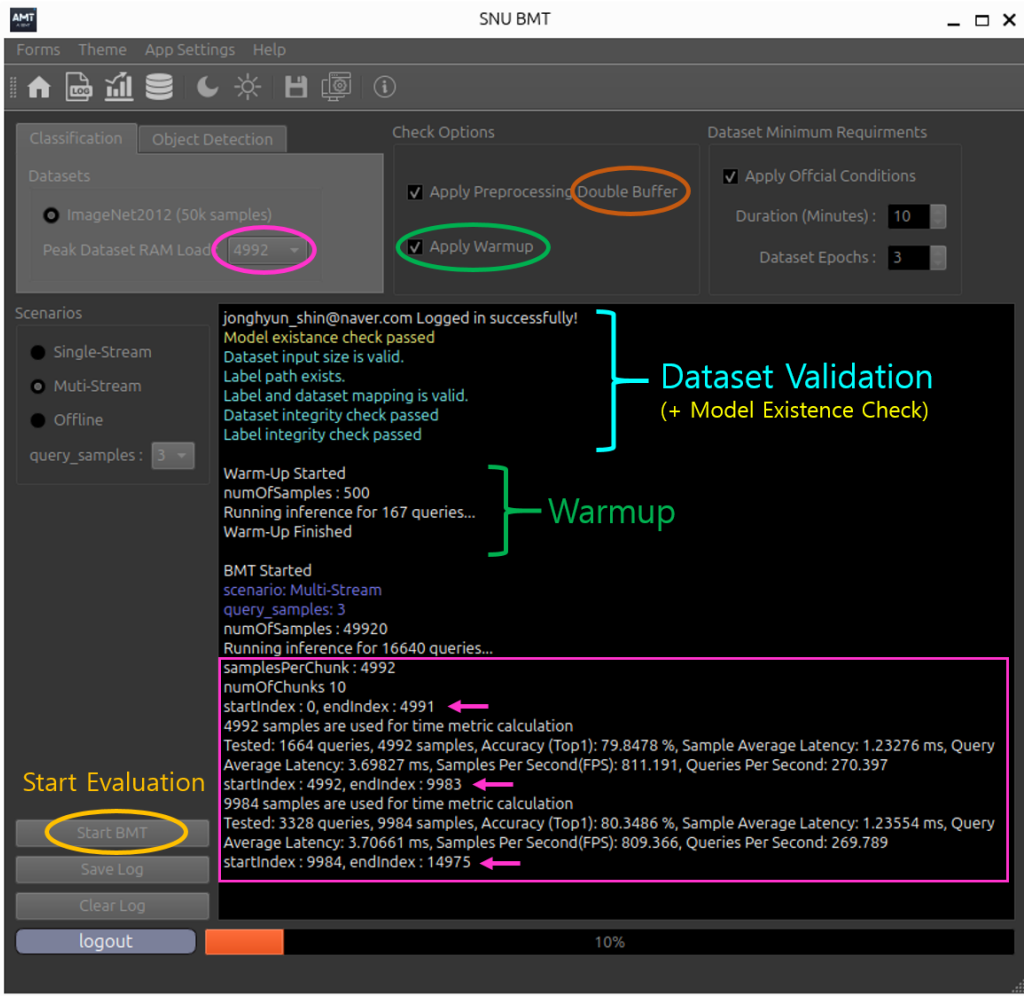

Figure 14: Benchmark Log Message (1)

3.2.15 Benchmark Log Message (2)

As shown in Figure 15, once the first epoch evaluation is completed, the system displays a summary of the current epoch results. It then checks whether the **minimum dataset requirements** are met. Since neither the epoch count nor the time condition is satisfied in this case, a new epoch evaluation is initiated automatically.

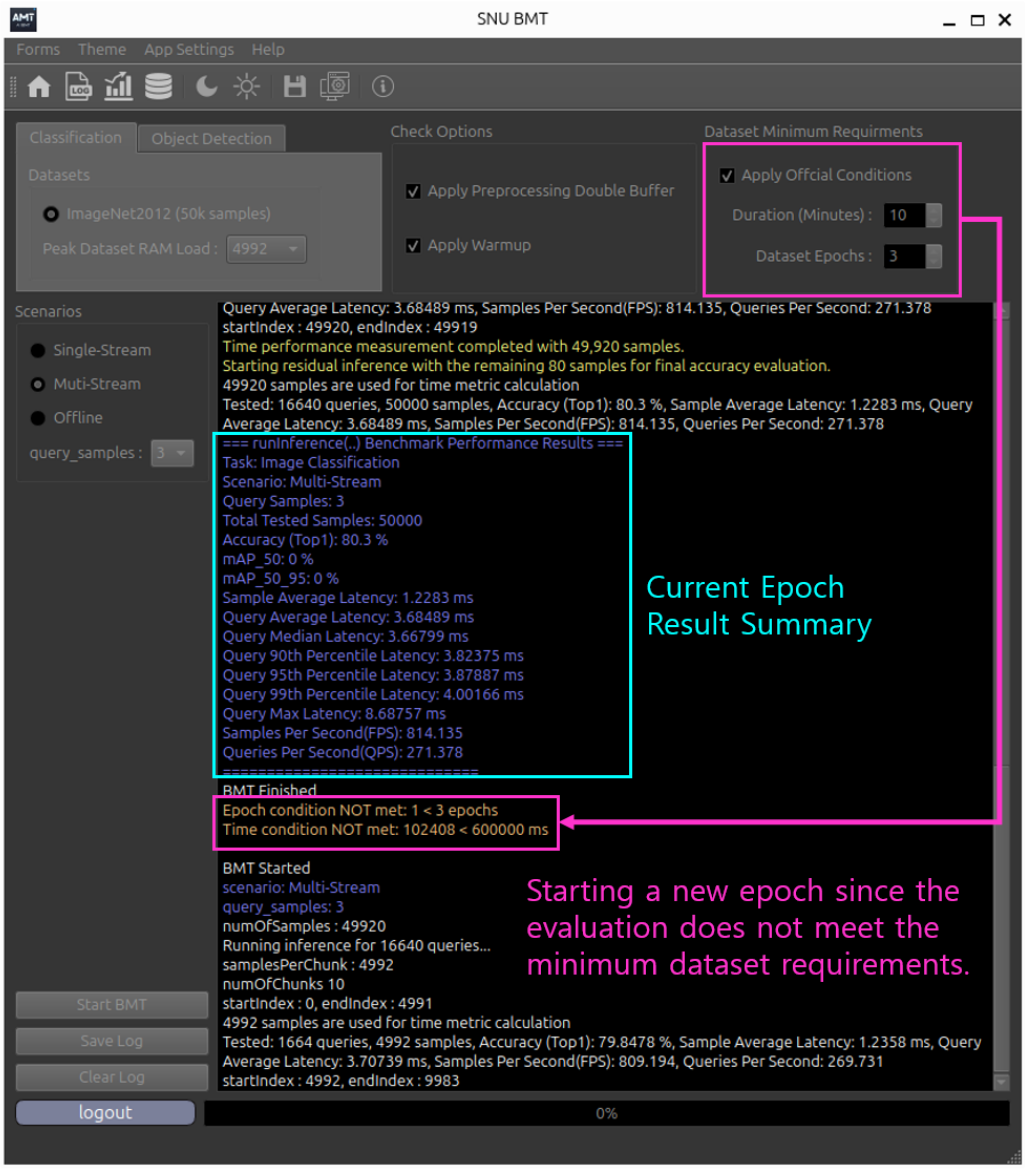

Figure 15: Benchmark Log Message (2)

3.2.16 Benchmark Log Message (3)

As shown in Figure 16, once both the **epoch count** and **time condition** are satisfied after several epochs, the evaluation process is completed. The system then displays the summary of overall performance metrics and uploads the results to the private database associated with the logged-in account.

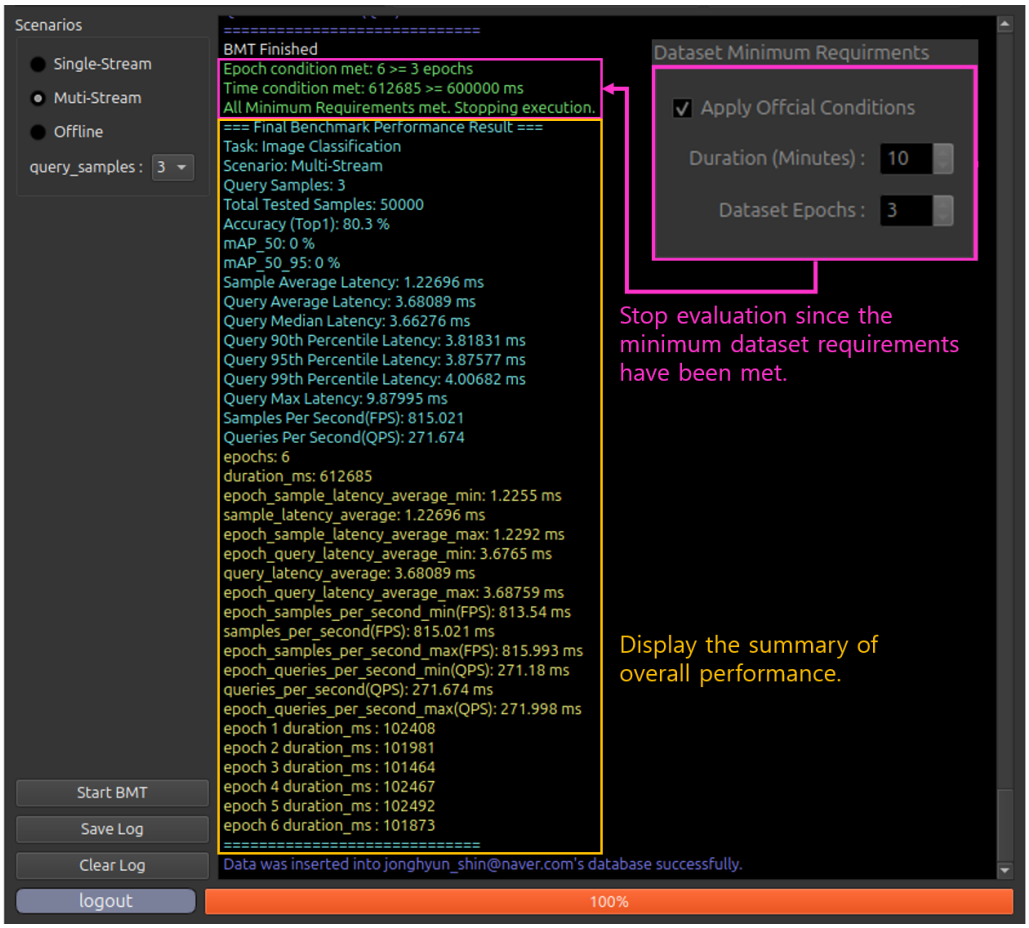

Figure 16: Benchmark Log Message (3)

3.2.17 Log Viewer

As shown in Figure 17, users can save the log messages from the **Evaluation Status Panel** by clicking the **Save Log** button. The log file is stored in the `Logs` directory with a timestamp-based filename. Saved logs can later be viewed within the application using the **Log Viewer** tab.

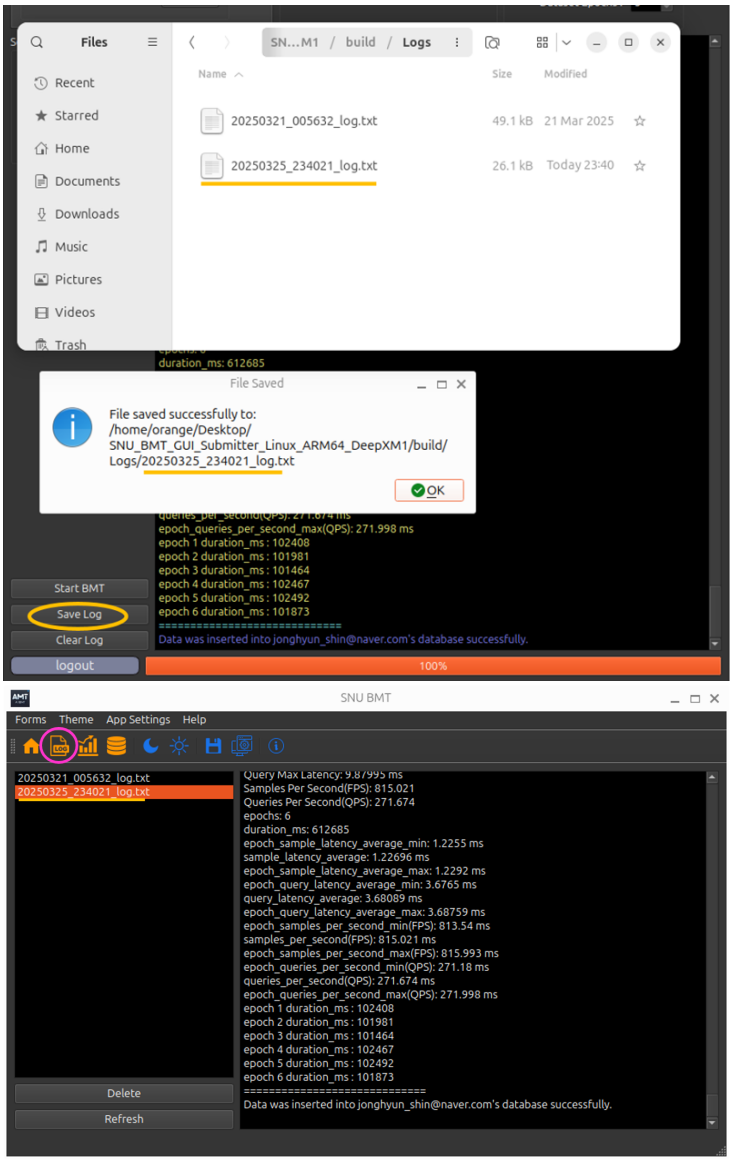

Figure 17: Log Viewer

3.2.18 Dynamic Model Queue Evaluation (DMQE)

As shown in Figure 18, To improve the flexibility and usability of our benchmarking workflow, we introduce **Dynamic Model Queue Evaluation (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.

Furthermore, the queue-based structure allows the platform to efficiently manage resources, log results per model, and optionally upload performance data to a centralized database after each evaluation cycle.

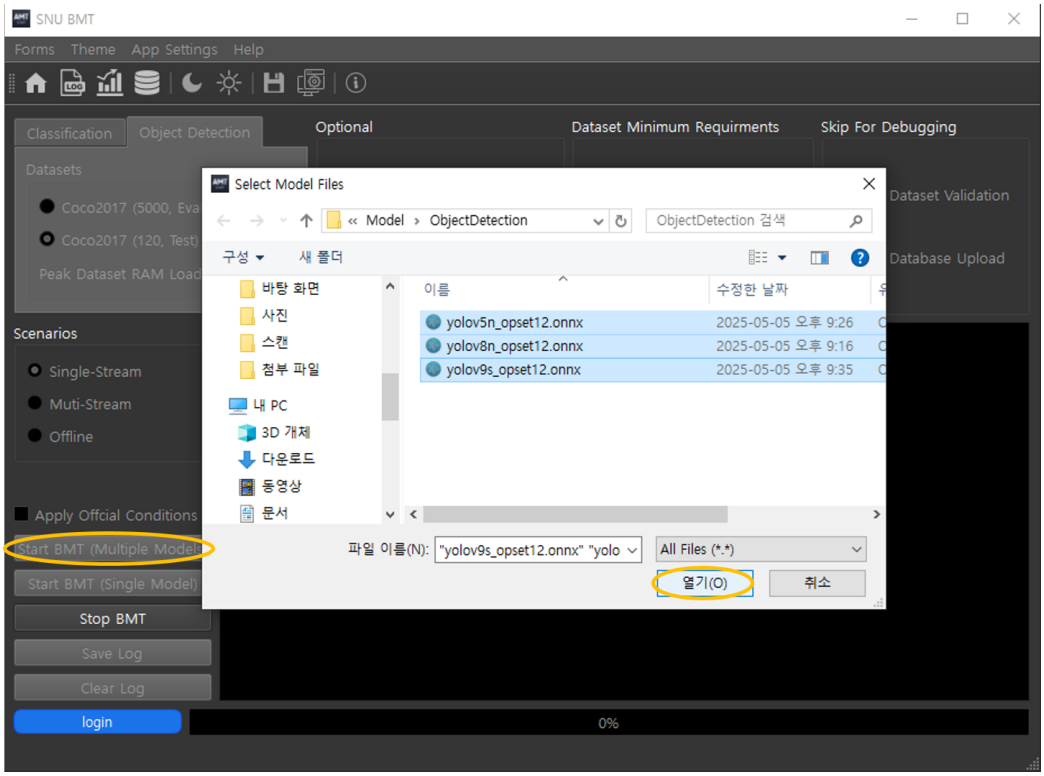

Figure 18: Dynamic Model Queue Evaluation

3.2.19 Data Analysis (App)

As shown in Figure 19, users can analyze their evaluation results retrieved from the database. To access this feature, users can click the **Data Analysis** tab. The most recent evaluation result appears at the top row of the table. Users can **double-click a specific row to delete it**, or **click on a column header to sort the data** in either ascending or descending order. In addition, users can copy selected rows (including headers) using `Ctrl+C`, and paste them into spreadsheet software (e.g., Excel) for further analysis. Alternatively, the entire set of results can be saved as a CSV file using the **Save as CSV file** button.

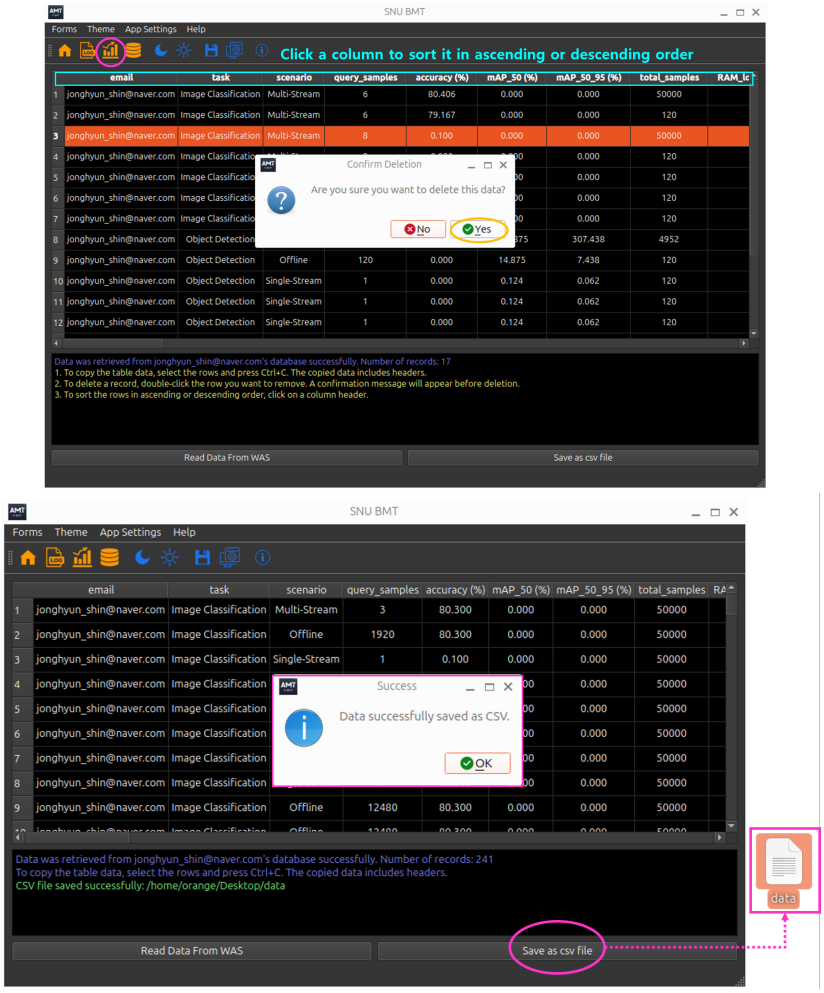

Figure 19: Data Analysis (App)

3.2.20 Data Analysis (WAS)

As shown in Figure 20, users can also analyze their evaluation results through the web-based interface provided by the WAS (`ai-bmt.com`). By selecting the **My Results (Private)** tab under the **Compare Results** menu, users can view their personal benchmark history. The entire set of results can be downloaded as a CSV file using the **Download Data as CSV** button.

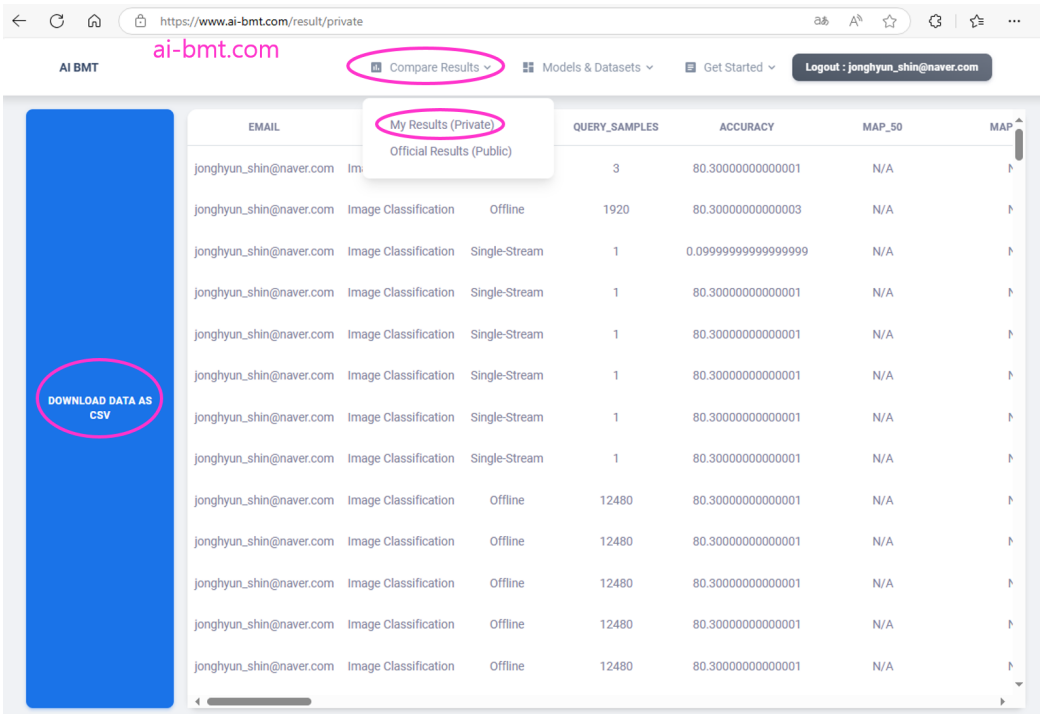

Figure 20: Data Analysis (WAS)

4 Required Preprocessing For Each Model

For implementation reference, please refer to the provided example code.

4.1 Classification (ImageNetV2 Validation Dataset)

All models below use the same normalization: `mean = [0.485, 0.456, 0.406]`, `std = [0.229, 0.224, 0.225]`. Prior to normalization, pixel values are first converted from uint8 [0, 255] to float [0, 1] by dividing by 255.0. No resizing or cropping is required, as the provided dataset has already undergone such preprocessing. See the example code for reference.

We use the ImageNetV2-top-images subset, which closely mirrors the distribution of the original ImageNet validation set.

All classification models are provided in both `.pth` and `.onnx` formats. The `.pth` models were evaluated in Python, and the `.onnx` models in C++, yet both achieved the same accuracy, as they share the same preprocessing pipeline.

Model	**Top-1 Accuracy (%)**	**Params (M)**
ResNet50	81.40	25.56
ResNet101	82.61	44.55
MobileNet_V2	74.76	3.50
MobileNet_V3_Large	76.77	5.48
RegNet_X_400MF	76.34	5.50
RegNet_Y_400MF	77.70	4.34
RegNet_X_800MF	78.55	7.26
RegNet_Y_800MF	80.32	6.43
ResNeXt50_32x4d	81.93	25.03
Wide_ResNet50_2	82.18	68.88
vit_deit_tiny	74.40	5.72
vit_deit_small	80.96	22.05
vit_deit_base	82.30	86.57

4.2 Object Detection(Coco17 Validation Dataset)

Submitters are only required to normalize the image by dividing pixel values by 255.0. No resizing or padding is required, as the provided dataset has already undergone such preprocessing. See the example code for reference.

Object detection performance is evaluated using the official `pycocotools` implementation of `COCOeval` with its default settings. This allows for stan-

dardized and reliable comparison across submissions without requiring custom metric implementation.

All evaluations are performed within the application under fixed conditions using a confidence threshold of 0.001 and an NMS IoU threshold of 0.65, ensuring consistent mAP results.

All Object Detection models are provided in .pt and .onnx format.

Model	**mAP_50_95 (%)**	**mAP_50 (%)**	**Params (M)**
YOLOv5n	27.22	44.43	1.9
YOLOv5s	36.26	55.32	7.2
YOLOv5m	43.43	62.03	21.2
YOLOv5nu	33.26	48.88	2.6
YOLOv5su	41.48	58.55	9.1
YOLOv5mu	46.61	63.77	25.1
YOLOv8n	36.15	51.68	3.2
YOLOv8s	43.53	60.60	11.2
YOLOv8m	47.96	65.40	25.9
YOLOv9t	36.89	51.99	2.0
YOLOv9s	44.79	61.23	7.2
YOLOv9m	49.00	66.20	20.1
YOLOv10n	37.68	53.98	2.3
YOLOv10s	44.70	62.27	7.2
YOLOv10m	48.53	66.39	15.4
YOLO11n	38.09	54.25	2.6
YOLO11s	44.41	61.57	9.4
YOLO11m	48.66	65.76	20.1
YOLO12n	39.56	55.77	2.6
YOLO12s	45.64	62.78	9.3
YOLO12m	49.82	67.25	20.2

4.3 Semantic Segmentation (VOC 2012 Validation Dataset)

All models for semantic segmentation follow the same input preprocessing pipeline as classification models: input images are rescaled to the [0, 1] range by dividing by 255.0, and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`.

No resizing or padding is required, as the provided validation dataset has already undergone such preprocessing. See the example code for implementation details.

We use the official PASCAL VOC 2012 validation set for evaluation. All Object Detection models are provided in .pth and .onnx format.

Model	**mIOU (%)**	**Pixel Acc (%)**	**Params (M)**
deeplabv3_mobilenet_v3_large	69.91	92.52	11.0
deeplabv3_resnet50	77.53	94.87	42.0
deeplabv3_resnet101	79.00	95.36	61.0
fcn_resnet50	72.54	93.82	35.3
fcn_resnet101	75.79	94.69	54.3

5 Known Limitations and FAQs

5.1 Current Limitations

- **Manual model validation:**
  The platform currently requires a manual audit process to ensure that model weights remain unchanged. This highlights the need for automated validation methods to ensure fairness and reduce the audit workload.
- **Limited model support per task:**
  The platform currently supports only one model per task. More representative models will be added in future updates.
- **Limited language support for benchmarking logic:**
  Benchmarking platform is only available in C++, which restricts accessibility for users who prefer other programming environments such as Python.
- **Online-only benchmarking:**
  The current version supports only online benchmarking, which requires a stable internet connection and strict adherence to the official evaluation workflow.
- **Lack of power and thermal efficiency metrics:**
  Power consumption and thermal behavior are not yet supported. These features require the development of dedicated hardware modules and integration with the BMT application.

5.2 Frequently Asked Questions

- **Does the platform support auto-login?**
  No. This feature is not currently supported, but it will be added in a future update. (Auto-login will be available after the first-time login)
- **Is there a debug mode available?**
  Currently, no. The platform is intended for official use only. For example, only fixed models and datasets for specific tasks are allowed. An internet connection and login are required because, at the end of the evaluation, the results are uploaded to a private database linked to each individual account. However, there are plans to add an offline mode, which will serve as a type of debug mode. This will allow users to test any model or dataset, even without an internet connection.