**Overview**

This article is aiming to illustrate the working principle of blood pressure estimation using MAMBA deep learning model. MAMBA, a model that excelling at dealing with time-series tasks, with PPG signal segments input, is able to estimate the corresponding systolic blood pressure and diastolic blood pressure.

**General Structure**

The MAMBA model used in this project is adapted from an open-source GitHub repository, therefore, they obtain a similar structure, which has folders for the definition of parameters, data preprocessing, and model components. The entry point of pipeline is the run.py file, managing the entire workflow, including the execution of training, validation and testing.

Some of the key components in this project must be emphasized:

1. ***data\_provider/*** : focuses on preparing and load PPG segments and corresponding SBP/DBP labels.
2. ***models/*** : contains the definition of the MAMBA architecture.
3. ***experiments/*** : includes high-level training and testing logic.
4. ***utils/*** : provides helper functions including learning rate adjustment and evaluation metrics.
5. ***run.py*** : as mentioned above, manages hyperparameters and launches the experiment pipeline.

During this project, significant modifications were made to achieve the regression-based blood pressure estimation. The modifications include the following parts:

1. Creating an algorithm for dataset extraction from raw dataset and conversion between ABP to SBP/DBP.
2. Adding a folder, ***data/*** , for dataset storage and data preprocessing.
3. Implementing new class, ***BPRegressionDtaset***, to handle PPG-BP matched data.
4. Adding normalization and inverse-transformation for both input signals and labels.
5. Enabling flexible training by resuming with checkpoints.
6. Rewriting some testing logic to provide some specific metrics.

**Dataset Configuration**

Raw waveform signal from the MIMIC-III database is selected as the dataset used for training in this project. Before digging deeper into the detailed use of datasets in training and testing, the structure of it will be firstly discussed. Inside the folder of the MIMIC-III database, each ***.dat*** file matches a ***.hea*** file, where the ***.dat*** files contain various biological waveforms, and ***.hea*** files describe the structure and properties of the waveforms in ***.dat*** files.

Among various biological signals, PPG signal and arterial blood pressure, ABP, are the signals needed and utilized in the training. The ABP signal is used to extract SBP and DBP values by identifying peak and trough points within an 8-second sliding window. SBP and DBP are obtained by calculating the mean values of peaks and troughs, respectively. Since the default sampling rate of PPG signal is 125Hz, the total number of PPG segments is 1000. Hence, for each 8-second sliding window, there are 1000 PPG segments, the input, matching one set of SBP and DBP, the labels.

When viewing the structure of these ***.csv*** files, for the dataset of PPG signal, each row contains 1000 PPG segments, and for the dataset of BP labels, each row contains two BP values, SBP, then DBP.

**Workflow**

When it comes to workflow, this project works under such a pipeline, data preprocess, input data process, and training and testing process. A more detailed demonstration is presented as follows.

Firstly, ***extract\_real\_data\_to\_csv.py***, is invoked to extract PPG signals and correspondingly matched ABP data form the raw dataset. Then, an algorithm is called to convert the ABP data into SBP/DBP. After that, the PPG signal segments are stored in ***ppg\_train.csv*** as training input feature and the SBP/DBP values are stored in ***bp\_labels\_train.csv*** as training labels. Particularly, in order to increase the efficiency of training, the program prompts users to input the prefix code of datasets, allowing several separate small datasets be combined together into a big scale dataset.

Secondly, ***check\_and\_remove.py***, invoked to check if there is any unavailable data, NANS, in the ***.csv*** file storing PPG segments. Once NANs are found, this entire sequence of PPG segments as well as the corresponding BP labels will be dropped from the dataset. Therefore, there are only available data remained in the ***.csv*** files, marking the end of data preprocess.

Thirdly, it comes to input processing and begins with ***data\_factory.py***, which serves as the entry point controller. When training or testing begins, its function, ***data\_provider()*** will be called to interpret the user arguments. Meanwhile, for ***data\_loader.py***, the class inside, ***BPRegressionDtaset***, handles the core logic of reading data, including performing normalization, slicing the sequences, allocating dataset for various mode and preparing the labels. Specifically, for normalization, which is utilized in order to obtain a more reasonable result, recalling the structure of both datasets, the mean of each column is set to be 1 and the standard deviation is set to be 0. For allocation of datasets, 70% of them are used for training, 10% of them are used for validation, and 20% of them are used for testing. Overall, after performing these processes, ***BPRegressionDtaset***, will return a dataset object.

After the dataset object is returned, ***data\_factory.py***, initializes it and wraps it with a **PyTorch *DataLoader***, enabling valid iteration over data in batches. In summary, ***data\_loader.py***, defines and process the dataset to make it a concrete object, and ***data\_factory.py***, manages and warps the dataset object to make it trainable input batches. So far, the input data process has been accomplished.

Finally, it comes to training and testing, which is managed by invoking ***exp\_long\_term\_forecasting.py***, the class ***Exp\_Long\_Term\_forecast*** inherits from ***exp\_basic*** and handles model construction, training logic, evaluation, and testing. Within this class, the ***build\_model()*** method creates the Mamba-based regression model, ***get\_data()*** calls the ***data\_provider()*** function from ***data\_factory.py*** to load datasets for training, validation, and testing, and ***select\_optimizer()*** and ***select\_criterion()*** configure the optimizer (Adam) and loss function (MSE), respectively.

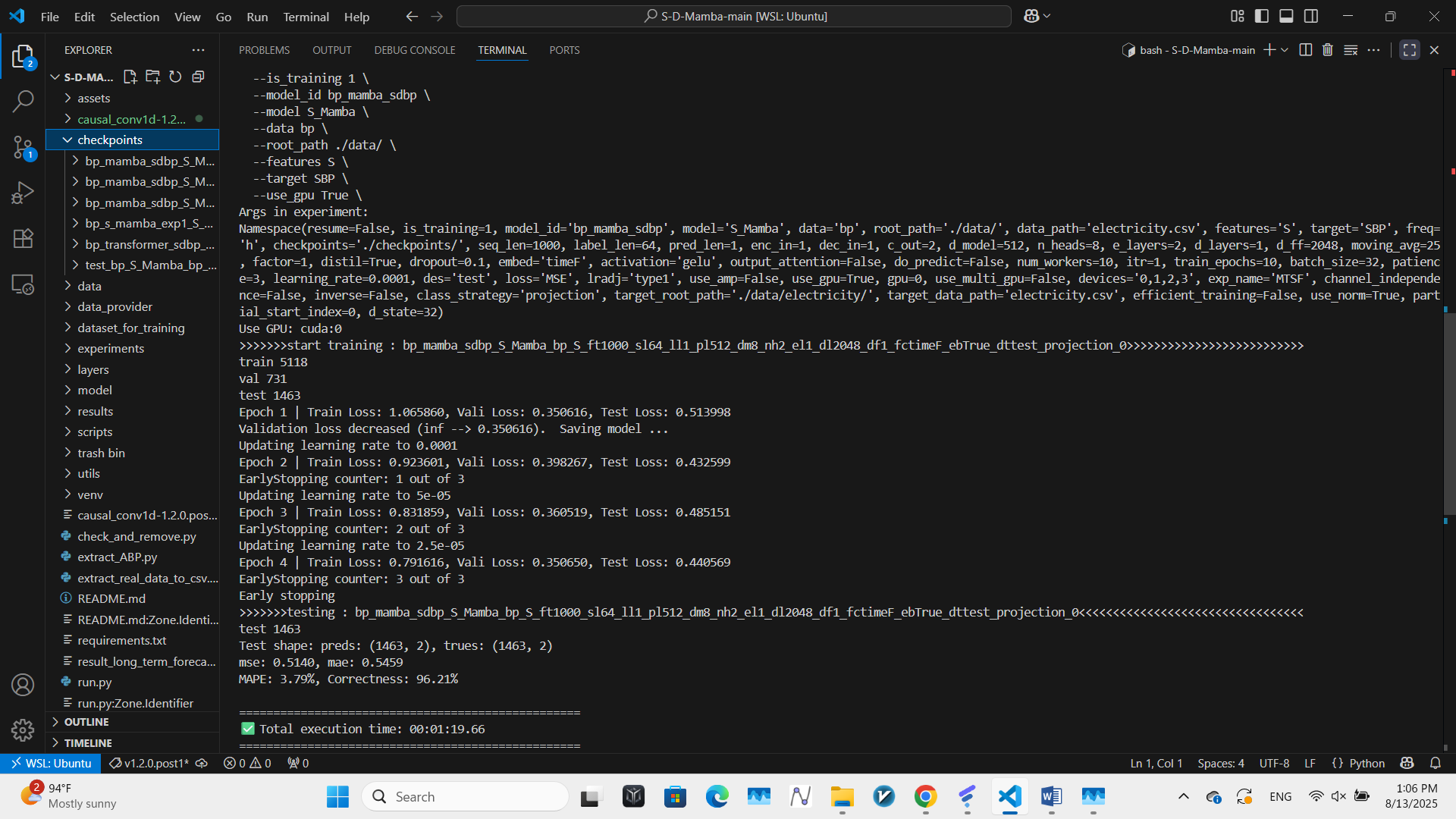
Particularly, the ***train()*** method performs the core training loop: for each epoch, it loads data batches from the training ***DataLoader***, forwards them through the model, computes loss, performs backpropagation, and updates model parameters. It also evaluates performance on the validation and test sets at each epoch, using early stopping to prevent overfitting. The best model is saved to disk based on validation loss. After training, the ***test()*** method reloads the best saved model, performs inference on the test dataset, and calculates key regression metrics (MSE, MAE, RMSE, MAPE, MSPE), which are then printed and saved to disk. This marks the end of the workflow from raw PPG/ABP signal extraction to final blood pressure prediction using the trained Mamba regression model.

**Result and Comparison**

In order to compare the performance between MAMBA and Transformer in terms of accuracy and time efficiency, a same dataset of PPG signals and SBP/DBP was applied to train and test by both MAMBA and Transformer models. This dataset, combining datasets of 3000714\_001, 3000714\_002, 3000714\_003, 3000714\_004, 3000714\_010, 3000714\_012 from MIMIC III database, has 5118 samples for training, 731 samples for validation and 1463 samples for testing.

There were four metrics set for evaluate the performance, which were Mean Square Error, Mean Absolute Error, Mean Absolute Percentage Error, and for Correctness (1 - MAPE). As to assess the time efficiency, the total time consumption for execution would be recorded.

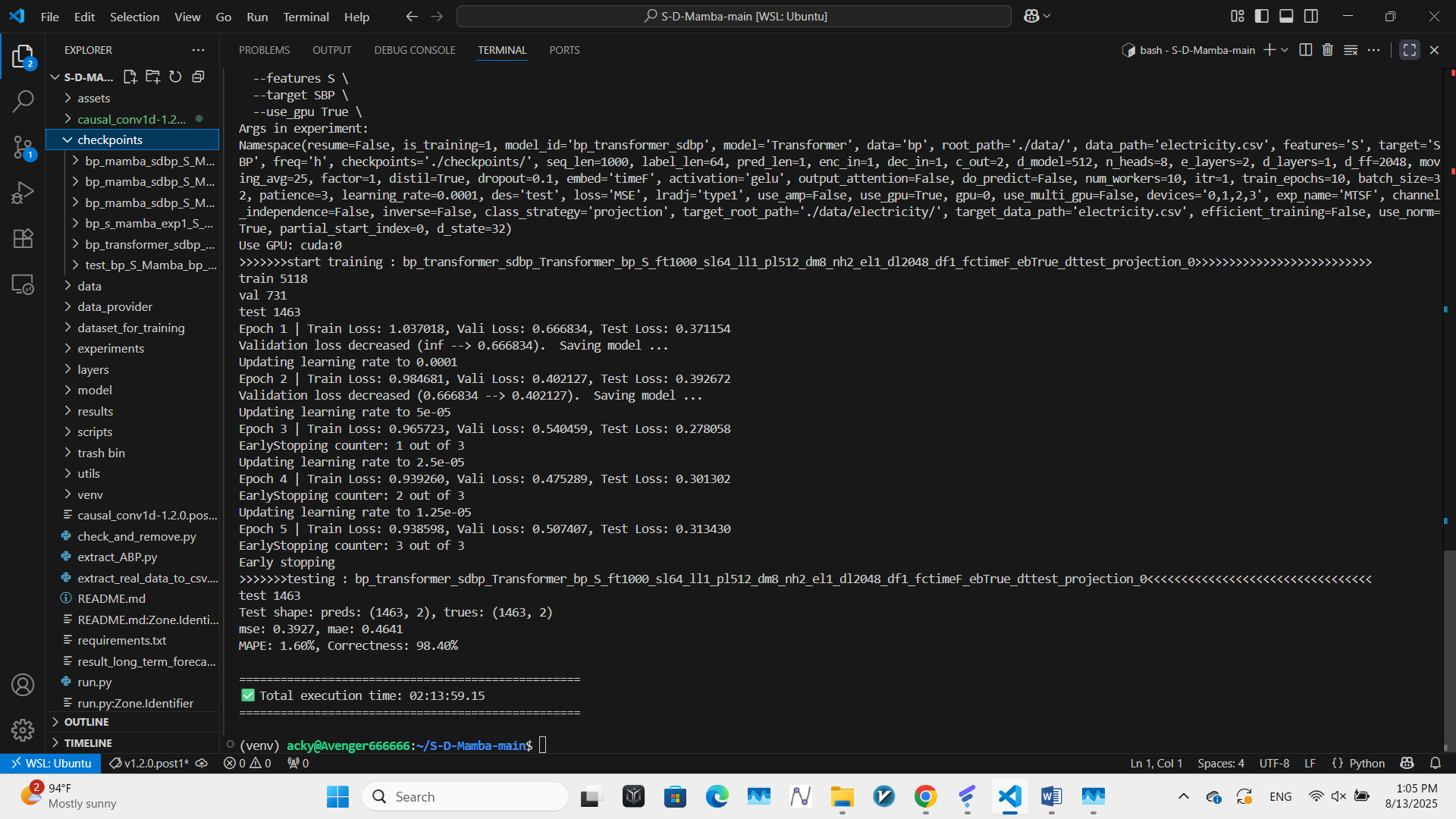
The result for MAMBA is shown as below.



MSE: 0.5410, MAE: 0.5459, MAPE: 3.79%, Correctness(1 - MAPE): 96.21%

Total time consumption was 1min 19.66s.

The result for Transformer is shown as below:



MSE: 0.3927, MAE: 0.4641, MAPE1.60%, Correctness(1 - MAPE): 98.40%

Total time consumption was 2h 13min 59.15s.

In conclusion, in this experiment, compared with Transformer, MAMBA had slightly less accuracy than Transformer, which was about 2 percent behind the correctness of Transformer, however, when it comes to time efficiency, MAMBA was far rapider then Transformer, which required over 100 times of the time that MAMBA needed. Overall, the result reveals that MAMBA could be a model excelling at handling time-series with a significantly faster speed.