National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Anny Faria
Student ID: x24124770

School of Computing
National College of Ireland

Supervisor: Dr. David Hamil

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anny Faria
Student ID:	x24124770
Programme:	MSc in Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr. David Hamil
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	1064
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	27th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anny Faria
x24124770

1 Introduction

This document presents guidelines for both hardware and software configurations and explains the practical implementation of the research project, covering dataset preparation, data pre-processing, model architecture, selection, configuration, implementation, and evaluation.

2 Hardware and Software Requirements

2.1 Hardware Configuration

This research project was done on a personal laptop, and system configuration settings are shown in Figure 1. The hardware configuration is as follows:

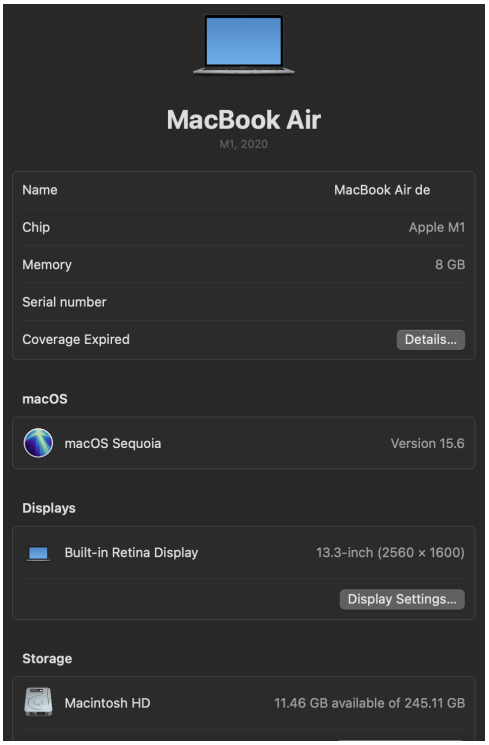

Figure 1: System Configuration

2.2 Software Configuration

This section describes the environments that were configured and used during the implementation, which needed to be prepared to ensure the reproducibility of the results in this research. The following software tools and applications were set up and must be installed on the system before proceeding with the process:

Category	**Tools**
Operate System	MacBook Air (M1)
Database Management	Google Drive
IDEs	Google Colab and Visual Studio Code (configured for Jupyter Notebook extension)
Programming Language	Python (3.12.5)

Table 1: Software used in the proposed research.

It is important to emphasize that this software setup is not a mandatory requirement. The project can also be conducted using similar environments, for example Anaconda, including Windows-based systems. In such cases, however, users are responsible for ensuring that the environment is correctly configured to support the required toolsets and workflows.

3 Packages & Libraries

For deep learning tasks, it is necessary to import specific packages and libraries. Figure 2 presents the set of libraries used in this project. These libraries must be installed before executing the code to ensure that all required functions and features are available.

The following libraries can be installed by running the following in your terminal or command prompt:

- `pip install pandas numpy matplotlib seaborn scikit-learn`
- `pip install tensorflow-determinism`

To extract the datasets stored in Google Drive onto Google Colaboratory, you should copy and paste this code in Colab IDLE:

- `from google.colab import drive drive.mount('/content/drive')`

```python
import os                       # operating system interface
import re                       # regular expressions for pattern matching
import numpy as np              # numerical operations and arrays
import pandas as pd             # data manipulation and analysis
import time                     # time measurement
import joblib                   # model and object serialization
from datetime import datetime   # handling date and time
import matplotlib.pyplot as plt # plotting and visualization
import math                     # mathematical utilities

# environment setup
os.environ["CUDA_VISIBLE_DEVICES"] = ""    # forces tensorflow to use cpu only
os.environ["PYTHONHASHSEED"] = "42"        # sets python hash seed for reproducibility

import tensorflow as tf    # deep learning framework

# tensorflow/keras determinism and seeds
seed = 42
tf.keras.utils.set_random_seed(seed)            # sets seeds for python, numpy, and tensorflow
tf.config.experimental.enable_op_determinism()  # enforces deterministic operations
tf.keras.backend.clear_session()                # clears previous models and graphs from memory

# keras / scikit imports
from tensorflow.keras import backend as K              # low-level keras backend operations
from sklearn.preprocessing import MinMaxScaler         # scales data to [0, 1] range
from sklearn.metrics import mean_squared_error         # evaluates regression error
from tensorflow.keras.models import Sequential, Model  # sequential and functional api modeling
from tensorflow.keras.layers import Dense, Input, Dropout, BatchNormalization  # neural network layers
from tensorflow.keras import regularizers              # l2 regularization tools
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# earlystopping: stops training when validation stops improving
# reducelronplateau: decreases learning rate when performance stagnates
```

Figure 2: Imported Libraries and Modules for the Project

4 Dataset

The datasets were generated using TMM equations and organized as shown theoretically in the thesis. Due to data privacy, only the datasets used in this research project can be downloaded from the following link Axion_Transmission.The name of the extension is dataset.zip, and the size is 101.6 MB on disk. After downloading the dataset, it has to be stored on the Google Drive or in your computer folder, which you plan to use for code. The zip contains three folders regarding each experiment: Experiment 1 (folder one), Experiment 2 (folder two), Experiment 3 (folder three). All folders, when clicked, contain subfolders named test, train, and val. These subfolders are the division of the data for the model. To better illustrate, the datasets are organized as shown in Figure 3:

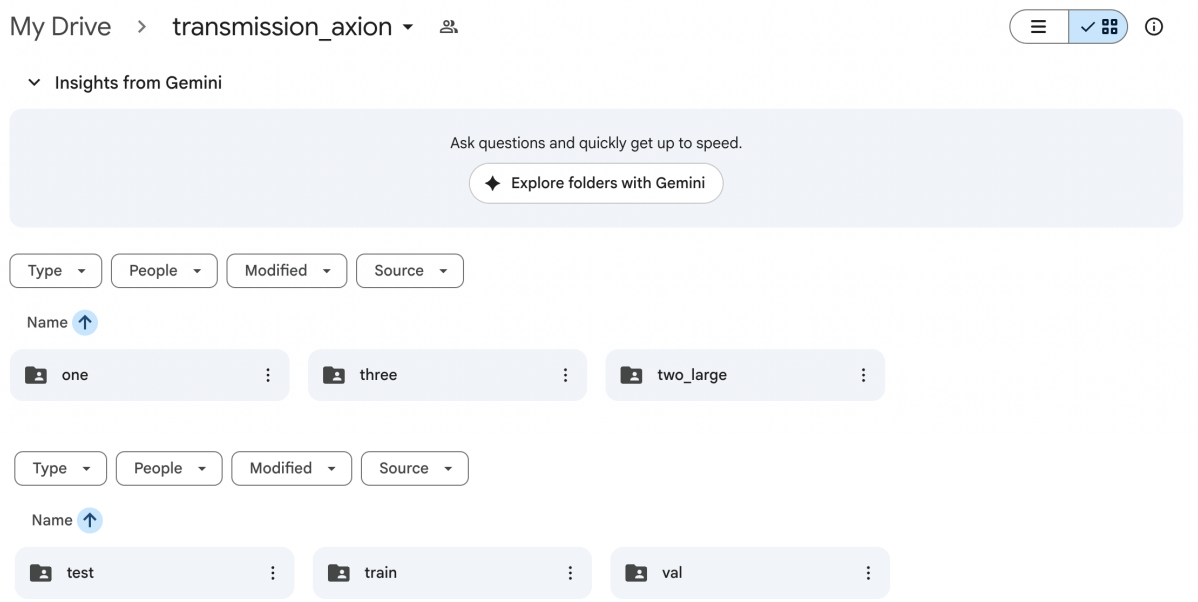

Figure 3: Dataset Preview After Downloading

The figure below summarizes the code used in each experiment to explore the data shape and types. Also, you don't need to check for missing values, duplicates, and null entries present because the dataset was generated as a clean dataset.

```
# datasets Shape
print(f"Train: {X_train.shape[0]} samples")
print(f"Validation: {X_val.shape[0]} samples")
print(f"Test: {X_test.shape[0]} samples")

# datasets Type
print(f"\nData type: {X_train.dtype}")
print(f"Data type: {X_val.dtype}")
print(f"Data type: {X_test.dtype}")

if any(data.size == 0 for data in [X_train, y_train, X_val, y_val, X_test, y_test]):
    raise SystemExit("Empty dataset detected")
```

Figure 4: Exploring Dataset Shape and Data Type.

Table 2 below shows the number of files generated in three experiments:

Folder	**Number of files**
Folder One	5001
Folder Two	5656
Folder Three	7812

Table 2: Total number of files generated in three experiments.

Table 3 below shows the proportional distribution of files in the three experiments:

Dataset	Split
Training	75.0%
Validation	12.5%
Testing	12.5%

Table 3: Proportional distribution of files in the three experiments.

Table 4 below summarizes the content in each data:

First column	Second column	Third column	Fourth column	Fifth column
$\bar{\omega}$ (float64)	T (float64)	X (float64)	R (float64)	δ (float64)

Table 4: Summary of content data structure. Data type: float64.

5 Data Preparation

As the shared zip file is already preprocessed, and it is ensured that the data that emerged from the TMM simulation were already structured, partitioned, and ready for application in neural network models, you only need to follow these steps to increase the credibility of the MLP model.

```
'''
It sets the base_path to the directory containing the one-parameter data, split into "train," "val," and "test" folders.
'''
base_path = "/content/drive/MyDrive/transmission_axion/two"

splits = ["train", "val", "test"]

# dictionaries to store input features (X) and target outputs (y) for each split
X_data = {"train": [], "val": [], "test": []}
y_data = {"train": [], "val": [], "test": []}
```

```
# parse files and extract parameters from filenames
file_pattern = re.compile(r"transmission_r(-?[\d\.]+)_x(-?[\d\.]+)_delta(-?[\d\.]+)\.txt")

for split in splits:
    folder = os.path.join(base_path, split)
    if not os.path.isdir(folder): continue

    try:
        files = [f for f in os.listdir(folder) if file_pattern.match(f)]
    except FileNotFoundError:
        continue

    for nome in files:
        match = file_pattern.match(nome)
        if not match: continue

        try:
            # extract R and delta from filename
            params = [float(match.group(1)), float(match.group(3))]

            # load transmission data
            file_T_data = np.loadtxt(os.path.join(folder, nome), usecols=(1,))

            if file_T_data.shape[0] == w_steps:
                X_data[split].append(params)
                y_data[split].append(file_T_data)
        except:
            continue
```

```
# convert to numpy arrays
X_train, y_train = np.array(X_data["train"]), np.array(y_data["train"])
X_val, y_val = np.array(X_data["val"]), np.array(y_data["val"])
X_test, y_test = np.array(X_data["test"]), np.array(y_data["test"])
```

Figure 5: Data Preparation Steps 1, 2 and 3.

6 Data Preprocessing

The normalization technique was used for the data processing and was applied to both the input (X, R and δ) and target vectors $T(\bar{\omega})$ in the dataset Razi et al. (2013). The purpose of this step is to scale values in order to match the input neurons range and normalize input and output data for the range of $[-1, 1]$ Razi et al. (2013).

```
# normalization
scaler_X = MinMaxScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_val_scaled   = scaler_X.transform(X_val)
X_test_scaled  = scaler_X.transform(X_test)

scaler_y = MinMaxScaler()
y_train_scaled = scaler_y.fit_transform(y_train)
y_val_scaled   = scaler_y.transform(y_val)
y_test_scaled  = scaler_y.transform(y_test)
```

Figure 6: Normalization process of neural network.

7 Machine Learning Model

7.1 Machine Learning Architecture

Various MLP architectures were implemented for one, two and three experiments using the previously scaled data. First of all, remember that the dataset was split into a training set, validation set, and test set in data generation, so you do not split it again. Then, the model training includes a Multilayer Perceptron (MLP), a type of neural network. Each model was evaluated using Euclidean Distance (ED) metric, which reflects the performance in terms of accuracy and robustness. Based on this criterion, the model with the best results is automatically selected and saved for use in predictions.

```
# defining MLP architectures for hyperparameter search
architectures = {
    "MLP_1-1": ([192, 96]),
    "MLP_1-2": ([256, 128, 64]),
    "MLP_1-3": ([256, 128, 128, 64]),
    "MLP_1-4": ([384, 384, 192, 192, 96, 96]),
    "MLP_1-5": ([512, 256, 128]),
    "MLP_1-6": ([512, 256, 256, 128])
}

validation_results = [] # list to store metrics (ed, mse, time) for each model on the validation set
# input_dim will automatically be 1 (one parameter: delta)
input_dim = X_train_scaled.shape[1]
# output_dim will be the length of the reflection curve (1000)
output_dim = y_train_scaled.shape[1]
```

Figure 7: Architecture of Experiment 1.

```
# defining MLP architectures for hyperparameter search
architectures = {
     "MLP_2-1": ([192, 96]),
     "MLP_2-2": ([256, 128, 64]),
     "MLP_2-3": ([1024, 1024, 384, 384]),
     "MLP_2-4": ([1024, 1024, 384, 384, 192, 192]),
     "MLP_2-5": ([512, 256, 128]),
     "MLP_2-6": ([512, 256, 256, 128])
}


validation_results = [] # list to store metrics (ed, mse, time) for each model on the validation set
input_dim = X_train_scaled.shape[1]  # will be 2 r and delta
output_dim = y_train_scaled.shape[1]  # will be 1000
```

Figure 8: Architecture of Experiment 2.

```
# defining MLP architectures for hyperparameter search
architectures = {
    "MLP_3-1": ([192, 96]),
    "MLP_3-2": ([256, 128, 64]),
    "MLP_3-3": ([1024, 1024, 384, 384]),
    "MLP_3-4": ([2048, 2048, 1024, 1024, 512, 512]),
    "MLP_3-5": ([2048, 1024, 512, 256, 128, 64]),
    "MLP_3-6": ([512, 256, 256, 128]),
}

validation_results = [] # list to store metrics (ed, mse, time) for each model on the validation set
input_dim = X_train_scaled.shape[1] # will be 3
output_dim = y_train_scaled.shape[1] # will be 1000
```

Figure 9: Architecture of Experiment 3.

7.2 Hyperparameters and Model Configuration

Hyperparameter was performed to improve the model. It tried all combinations of parameters defined, such as learning rate and regularization factor, to find the best architecture results that would give the lowest Mean Squared Error (MSE). The best architecture based on model configuration and hyperparameters was found, and the model was trained and evaluated. Below is a table that represents the configuration and hyperparameters for each experiment.

Category	**Values**
Input Dimension	1 (δ)
Output Dimension	1000 (T curve)
Batch Size	128
Epochs	150
Activation Functions	`ReLU()` (hidden layers), `Sigmoid` (output)
Optimizer	`Adam`(learning_rate=5×10^{-4})
Regularization L2	5×10^{-6}
Early Stopping	Monitor: val_loss, Patience: 5, Restore Best Weights: True
Learning Rate Scheduler	`ReduceLROnPlateau`(factor=0.5, patience=8, min_lr=1×10^{-7})
Seed	42 (Python, NumPy, TensorFlow)
Determinism	Enabled (`tf.config.experimental.enable_op_determinism()`)

Table 5: MLP Hyperparameters Experiment 1

Category	Values
Input Dimension	2 (R, δ)
Output Dimension	1000 (T curve)
Batch Size	128
Epochs	100
Activation Functions	`ReLU()` (hidden layers), `Sigmoid` (output)
Optimizer	`Adam`(learning_rate=5×10^{-4})
Regularization L2	1×10^{-6}
Early Stopping	Monitor: val_loss, Patience: 5, Restore Best Weights: True
Learning Rate Scheduler	`ReduceLROnPlateau`(factor=0.5, patience=8, min_lr=1×10^{-7})
Seed	42 (Python, NumPy, TensorFlow)
Determinism	Enabled (`tf.config.experimental.enable_op_determinism()`)

Table 6: MLP Hyperparameters Experiment 2.

Category	Values
Input Dimension	3 (R, X, δ)
Output Dimension	1000 (T curve)
Batch Size	128
Epochs	200
Activation Functions	`ReLU()` (hidden layers), `Sigmoid` (output)
Optimizer	`Adam`(learning_rate=2×10^{-4})
Regularization L2	1×10^{-6}
Early Stopping	Monitor: val_loss, Patience: 40, Restore Best Weights: True
Learning Rate Scheduler	`ReduceLROnPlateau`(factor=0.5, patience=15, min_lr=1×10^{-7})
Seed	42 (Python, NumPy, TensorFlow)
Determinism	Enabled (`tf.config.experimental.enable_op_determinism()`)

Table 7: MLP Hyperparameters Experiment 3.

7.3 Model Selection and Comparison

The model selection handles the final ranking of all trained architectures, selects the best model based on the Mean Euclidean Distance on the validation set, and generates a visual comparison. In this subsection, we show only one example of a code configuration, which can be reproduced in the same way for each experiment one, two and three.

```python
# Post-training evaluation
y_val_pred_scaled = model.predict(X_val_scaled, verbose=0, batch_size=1024)
y_val_pred = scaler_y.inverse_transform(y_val_pred_scaled)
mean_ed_val = np.mean(np.linalg.norm(y_val - y_val_pred, axis=1))
best_mse_val = np.min(history.history['val_loss'])

#storing validation results
validation_results.append({
    "name": model_name,
    "model": model,
    "mean_val_ed": mean_ed_val,
    "best_val_mse": best_mse_val,
    "training_time": training_time
})
```

Figure 10: Example: Storing Validation Results for the Model Selection Regarding Experiment 2.

```python
# model comparison results
print("\n" + "="*60)
print("MODEL COMPARISON (2 INPUTS) ON VALIDATION SET")
print("="*60)

# sortting models based on mean Euclidean Distance
validation_results.sort(key=lambda x: x['mean_val_ed'])
for res in validation_results:
    print(f"- {res['name']:<28}: Mean ED = {res['mean_val_ed']:.6f} | Best MSE = {res['best_val_mse']:.8f}")

# selecting the best model
best_model_info = validation_results[0]
best_model = best_model_info['model']
print(f"\nBest model selected (lowest ED): {best_model_info['name']}")

# generating bar chart for mean ED comparison
plt.figure(figsize=(12, 7))
model_names = [res['name'] for res in validation_results]
val_errors_ed = [res['mean_val_ed'] for res in validation_results]

# defining color palette
colors = ['#4c72b0', '#55a868', '#c44e52', '#8172b2', '#ff9f4a', '#6f4e7c'] * (len(model_names) // 6 + 1)
bars = plt.bar(model_names, val_errors_ed, color=colors[:len(model_names)])

plt.title('Architectures Comparison', fontsize=25, fontweight='bold')
plt.ylabel('Mean Euclidean Distance', fontsize=25)
plt.xticks(rotation=45, ha="right", fontsize=18)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.bar_label(bars, fmt='%.4f')
plt.tight_layout()
plt.savefig(os.path.join(results_folder, 'mlp2_arq.png'), dpi=400)
plt.show()
```

Figure 11: Example: Model Selection Regarding Experiment 2.

If the same procedure was done for each experiment separately, the results will be:

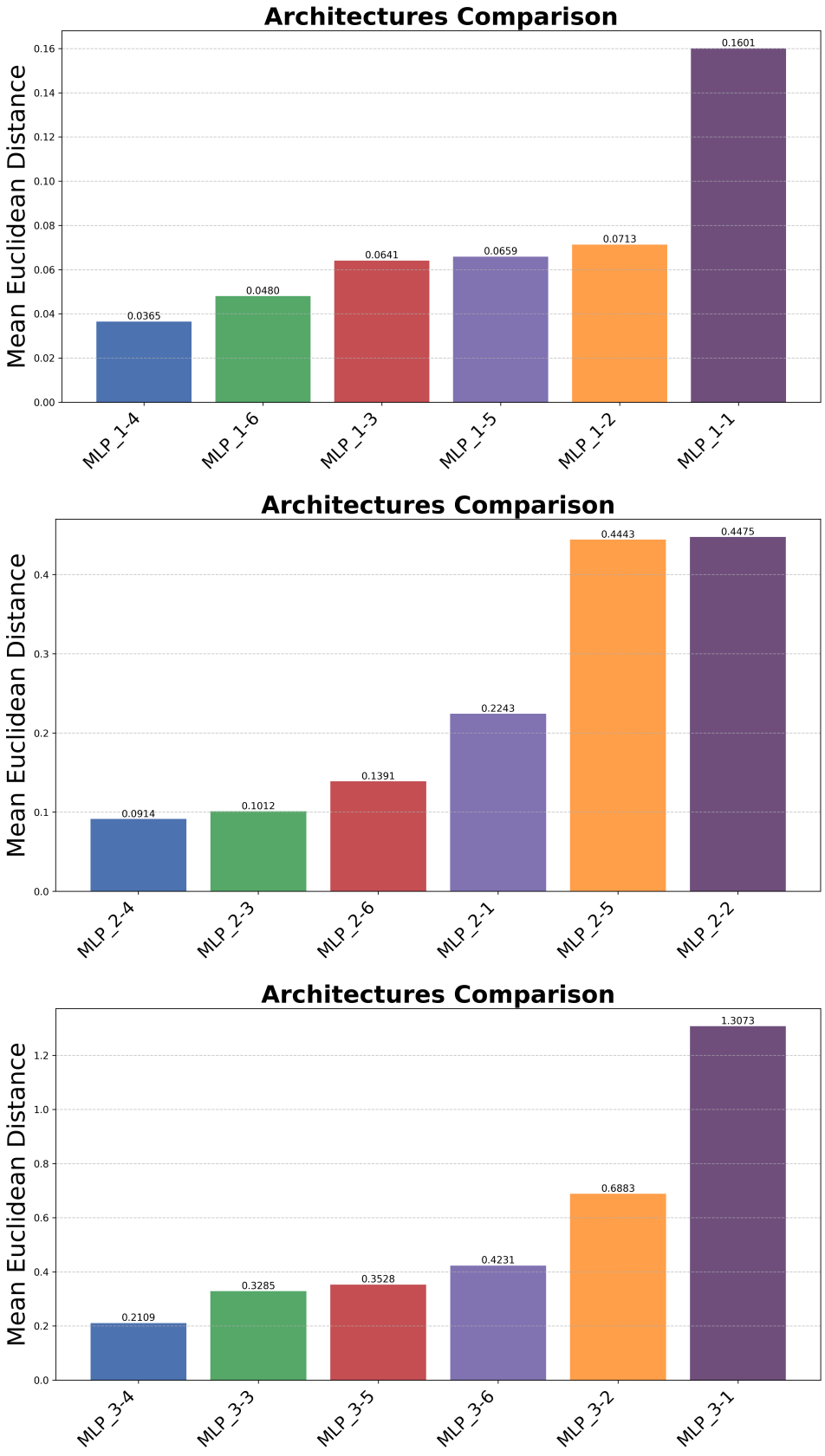

Figure 12: Example: Model Comparation for each Experiment.

8 Time Measurement

In this section, we show only one example of a code configuration, which can be reproduced in the same way for each experiment one, two and three.

8.1 Time Trainig Model

To measure the training time of the model in each experiment, you need to use `start_time = time.time()` before the training model code and at the end of the training code, `training_time = time.time() - start_time` as shown in Figure 14. The figure shows only one example, which is the same for each experiment.

```python
# training model
start_time = time.time()
history = model.fit(
    X_train_scaled, y_train_scaled,
    validation_data=(X_val_scaled, y_val_scaled),
    epochs=100,
    batch_size=128,
    callbacks=[early_stop, reduce_lr, time_callback],
    verbose=1,
    shuffle=True,
)
training_time = time.time() - start_time
```

Figure 13: Example: Time Training Configuration Regarding Experiment 2.

8.2 Time Inference and Average

Time inference refers to the average time taken by the trained neural networks to predict the transmission of a APC.

```python
# starting inference time measurement variables Test Set
total_inference_time = np.nan
average_inference_time_per_sample = np.nan
num_test_samples = 0
y_test_pred_scaled = np.array([])  # starting prediction variable

if X_test_scaled.size > 0:
    num_test_samples = len(X_test_scaled)

    # warm-up prediction for optimization
    try:
        _ = best_model.predict(X_test_scaled[0:1], verbose=0)
    except Exception as e:
        print(f"Error during warm-up: {e}")

    # measure total prediction time for the entire test set
    start_inference_time = time.time()
    y_test_pred_scaled = best_model.predict(X_test_scaled, batch_size=1024, verbose=0)
    end_inference_time = time.time()

    total_inference_time = end_inference_time - start_inference_time

    # calculate average time per sample
    if num_test_samples > 0:
        average_inference_time_per_sample = total_inference_time / num_test_samples

    print(f"total inference time (for {num_test_samples} samples): {total_inference_time:.6f} seconds.")
    print(f"average inference time (per sample/PC): {average_inference_time_per_sample:.8f} seconds.")

else:
    print("Test set is empty. Inference time not calculated.")
```

Figure 14: Example: Time Inference Configuration Regarding Experiment 2.

References

Razi, M., Arz, A. & Naderi, A. (2013), 'Annular pressure loss while drilling prediction with artificial neural network modeling', *European Journal of Scientific Re-*

search **95**(2), 272–288. Available at: https://www.researchgate.net/publication/239735549.