Configuration Manual

MSc Research Project
Data Analytics

Ambar Balkrishna Pawar
Student ID: x21172641

School of Computing
National College of Ireland

Supervisor: Dr. Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ambar Balkrishna Pawar
Student ID:	x21172641
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Abubakr Siddig
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	1558
Page Count:	8

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:	Ambar Balkrishna Pawar
Date:	17th September 2023

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

Ambar Balkrishna Pawar
x21172641

1 Introduction

Setting up and executing the code in Google Colab and Jupyter Notebook environments is described in depth in this setup guide. It outlines the prerequisites and libraries necessary to execute the models, do graphical evaluations, and play the chess simulation. Users may easily recreate the tests and simulations described in the paper by following these steps.

2 Google Colab Setup

- Accessing Google Colab: A handy cloud-based environment for executing code, including machine learning models, is provided by Google Colab. Start a browser and go here to access the Google Colab platform.
- Creating a New Notebook: When you first visit the Colab platform, a dashboard will welcome you. Click the "New Notebook" button to start a new project. By doing this, a brand-new Jupyter Notebook will be created where you may enter and run code.

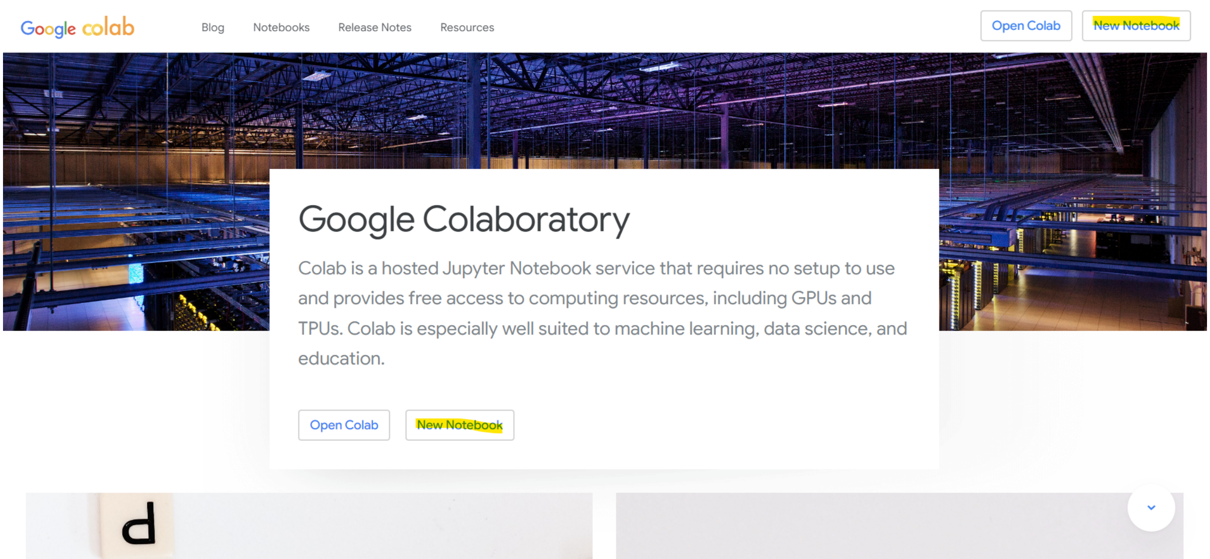

Figure 1: Creating a New Notebook on Google Colab

- Setting Runtime Type: Navigate to the "Runtime" option after the notebook has been created. Choose "Change runtime type" in the selection list. The hardware accelerator that drives your laptop may be selected here. In order to accelerate the training of machine learning models including neural networks, choose "GPU" to make use of the graphics processing unit.

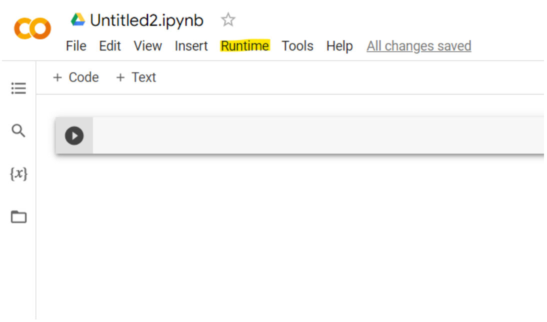

Figure 2: Navigate to the "Runtime"

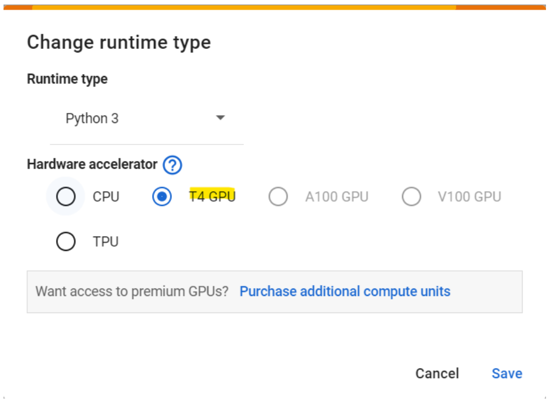

Figure 3: choose "GPU"

3 Uploading Dataset and Preprocessing

- Upload Dataset: After opening Google Colab, select the "Files" option in the left sidebar to begin uploading your dataset. After that, pick the dataset file from your local computer by clicking the "Upload" button. The dataset will be uploaded to your Google Colab environment in this manner.

- Libraries for Preprocessing and Visualizations are Imported: It is necessary to import the necessary libraries in order to preprocess the dataset and produce visuals. For preprocessing and visualization activities, the following libraries are frequently used:

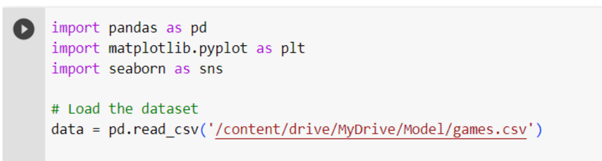

Figure 4: Uploading Dataset and Importing Libraries for Preprocessing

  - 1. Pandas (import pandas as pd): Pandas is a library for data manipulation that makes it easier to load, clean, and analyze tabular data. You can effectively manage datasets for preprocessing by importing it as 'pd'.
  - 2. Matplotlib (import matplotlib.pyplot as plt): Matplotlib is a popular toolkit for plotting data that makes it easier to construct different graphs and visualizations. The 'pyplot' module may be imported as 'plt' to visualize data.

- 3. Seaborn (import seaborn as sns): Based on Matplotlib, Seaborn provides a more advanced interface for designing eye-catching statistical visualizations. Complex plots may be easily generated by importing it as'sns'.

- Data preprocessing: After the dataset has been uploaded, you may load and preprocess the data using the Pandas library. This might entail activities like data cleansing, managing missing values, and formatting the dataset in the way needed for model training.

4 Building the Model

4.1 Importing Libraries for Model Building

You must import certain libraries for deep learning and machine learning tasks in order to create your model. For creating sequential models and interacting with text data, the following libraries are frequently used:

```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Embedding, TimeDistributed, Input, Attention
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import pickle
```

Figure 5: Importing Libraries for Model Building

- NumPy: Supports management of arrays and numerical computations.
- scikit-learn (sklearn): Model selection and data preparation techniques are available through scikit-learn (sklearn).
- TensorFlow (tf): A potent library for creating and training neural networks is TensorFlow (tf).
- Tokenizer: For model input, a tokenizer tokenizes text data.
- Embedding: Text input is transformed into a numerical representation appropriate for deep learning models by embedding.
- LSTM: Recurrent neural network layer for sequence data called LSTM.
- Dense: Represents neural network layers that are fully linked.
- TimeDistributed: Apply a layer to each time step in a sequence using the TimeDistributed method.
- Input: Represents the neural network's input layer.
- Attention: Implements attention-related strategies to improve model performance.

4.2 Model Creation and Training

After importing the necessary libraries, you are prepared to build your deep learning models. In order to do this, you must first specify the model's architecture, then compile it, divide the dataset into training and testing sections, and finally train the model using the training data. Make careful to follow accepted procedures for maximizing hyperparameters, evaluating model performance, and improving the model to achieve the best results.

```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, TimeDistributed
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

# Optimizing code for GPU acceleration
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

# Tokenize the moves and obtain the word index mapping
tokenizer = Tokenizer(filters='', lower=False)
tokenizer.fit_on_texts(data['moves'])

# Prepare the data for sequence-to-sequence model
moves = tokenizer.texts_to_sequences(data['moves'])
next_moves = [move[1:] for move in moves]
moves = [move[:-1] for move in moves]
X = pad_sequences(moves, padding='post')
y = pad_sequences(next_moves, padding='post')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the LSTM model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=100, input_length=X.shape[1]))
model.add(LSTM(units=64, return_sequences=True))
model.add(TimeDistributed(Dense(units=len(tokenizer.word_index) + 1, activation='softmax')))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, np.expand_dims(y_train, -1), epochs=10, batch_size=32, validation_data=(X_test, np.expand_dims(y_test, -1)))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, np.expand_dims(y_test, -1))
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

# Predict the next moves for test data
predicted_moves = model.predict(X_test)
predicted_moves = np.argmax(predicted_moves, axis=-1)
predicted_moves = tokenizer.sequences_to_texts(predicted_moves)

# Add the predicted moves to the dataframe
data['Predicted Moves'] = predicted_moves

Epoch 1/10
502/502 [==============================] - 1684s 3s/step - loss: 1.8297 - accuracy: 0.8245 - val_loss: 1.1005 - val_accuracy: 0.8286
Epoch 2/10
502/502 [==============================] - 1666s 3s/step - loss: 1.0483 - accuracy: 0.8344 - val_loss: 1.0270 - val_accuracy: 0.8351
Epoch 3/10
502/502 [==============================] - 1679s 3s/step - loss: 1.0011 - accuracy: 0.8371 - val_loss: 0.9921 - val_accuracy: 0.8366
```

Figure 6: LSTM Model Code

```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Embedding, TimeDistributed, Input, Attention
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import pickle

# Tokenize the moves and obtain the word index mapping
tokenizer = Tokenizer(filters='', lower=False)
tokenizer.fit_on_texts(data['moves'])

# Prepare the data for sequence-to-sequence model
moves = tokenizer.texts_to_sequences(data['moves'])
next_moves = [move[1:] for move in moves]
moves = [move[:-1] for move in moves]
X = pad_sequences(moves, padding='post')
y = pad_sequences(next_moves, padding='post')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the LSTM model with Attention
input_layer = Input(shape=(X.shape[1],))
embedding_layer = Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=100, input_length=X.shape[1])(input_layer)
lstm_layer = LSTM(units=64, return_sequences=True)(embedding_layer)
attention_layer = Attention()([lstm_layer, lstm_layer])
output_layer = TimeDistributed(Dense(units=len(tokenizer.word_index) + 1, activation='softmax'))(attention_layer)

model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, np.expand_dims(y_train, -1), epochs=10, batch_size=32, validation_data=(X_test, np.expand_dims(y_test, -1)))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, np.expand_dims(y_test, -1))
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

# Save the trained model
model.save('\model')

# Save the tokenizer
with open('\tokenizer', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

Epoch 1/10
502/502 [==============================] - 1657s 3s/step - loss: 1.7856 - accuracy: 0.8247 - val_loss: 1.1947 - val_accuracy: 0.8283
Epoch 2/10
502/502 [==============================] - 1613s 3s/step - loss: 1.1498 - accuracy: 0.8293 - val_loss: 1.1160 - val_accuracy: 0.8283
```

Figure 7: AttLSTM Model Code

4.3 Saving and Loading the Model

It is advised to store the model when it has finished training using tools like the pickle library or TensorFlow's built-in model saving capabilities. This move makes it easier for the trained model to be loaded and used for future predictions without the need for retraining.

This section essentially explains the first steps required to set up Google Colab for the execution of models and chess simulation. It provides instructions for uploading datasets, preparing data, incorporating libraries, and creating deep learning models. By following these instructions, you'll be prepared to execute your code skillfully and extract insightful information from your data.

5 Evaluation

A thorough analysis of the created models is presented in this part, covering three different areas: move prediction, graphical chess simulation, and strategic conflict with the Stockfish engine.

5.1 Move Prediction Evaluation

The model's capacity to predict the following chess move based on a given initial move is evaluated in this step of assessment. Please make sure that Python, Jupyter Notebook, and the necessary libraries are installed on your machine before starting this assessment. Use the following actions:

- Install the necessary libraries using the provided command-line instructions:

```
!pip install numpy
!pip install pygame
!pip install python-chess PyQt5
!pip install tensorflow
!pip install chess
!pip install sys
!pip install pickle
```

Figure 8: Installation of required Libraries

- Navigate to the appropriate area in the given Jupyter Notebook.
- Apply the following code snippet to import the necessary libraries for move prediction:

```
In [*]: import chess
import numpy as np
import pickle
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

# Load the tokenizer
tokenizer = Tokenizer()
with open('tokenizer3', 'rb') as f:
    tokenizer = pickle.load(f)

# Load the fine-tuned model
model = load_model('model3')

# Maximum sequence length
max_sequence_length = 348

# Function to predict the next move
def predict_next_move(input_sequence, model, tokenizer, max_sequence_length):
    # Tokenize the input sequence
    input_seq = tokenizer.texts_to_sequences([input_sequence])[0]
    input_seq = pad_sequences([input_seq], maxlen=max_sequence_length, padding='post')

    # Predict the next move
    predictions = model.predict(input_seq)[0]
    next_move_index = np.argmax(predictions)

    # Check if the next_move_index is within the valid range
    if next_move_index < len(tokenizer.index_word):
        next_move = tokenizer.index_word[next_move_index]
    else:
        # Get the current board position after the given sequence of moves
        board = chess.Board()
        moves = input_sequence.split()
        for move in moves:
            board.push_uci(move)

        # Generate all legal moves from the current position
        legal_moves = [move.uci() for move in board.legal_moves]

        if len(legal_moves) > 0:
            # Choose a random valid move
            next_move = np.random.choice(legal_moves)

    return next_move

# Loop for making predictions
for _ in range(50):
    input_sequence = input("Enter the sequence of moves: ")  # Provide the sequence of moves here
    next_move = predict_next_move(input_sequence, model, tokenizer, max_sequence_length)
    print(f"The predicted next move is: {next_move}")

Enter the sequence of moves: b1a3
1/1 [==============================] - 1s 611ms/step
The predicted next move is: b8c6

Enter the sequence of moves:
```

Figure 9: Importing libraries and Move Prediction Evaluation Code

- Run the Python code supplied in the Jupyter Notebook.
- As the code instructs, enter the first move.
- You can see the model forecast future movements as it dynamically adjusts to the changing game condition.

5.2 Graphical Chess Simulation

Another aspect of our model evaluation is the graphical simulation of a chess game. The following procedures should be followed to launch the computer's graphical chess simulation:

- Ensure that to check installed Jupyter Notebook, Python, and the necessary libraries.
- Restarting the kernel is necessary before running the supplied code.
- Adding the required libraries, as seen in the following code example in figure 10:

Figure 10: Graphical Chess Simulation Code and Importing Libraries

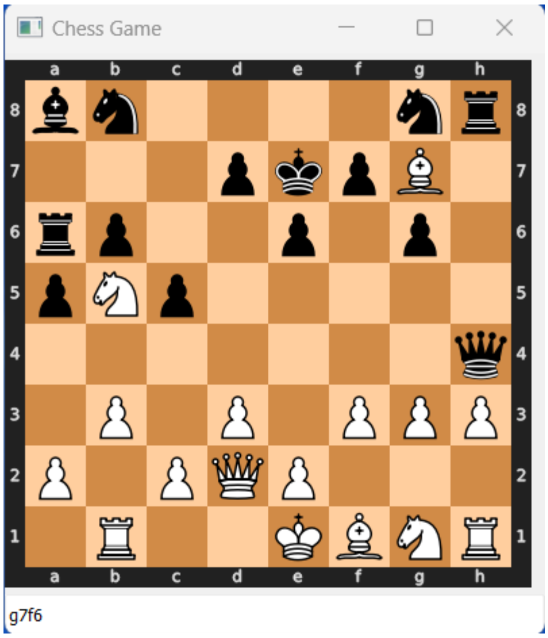

Figure 11: Graphical Chess Simulation Output

- Use the Jupyter Notebook to execute the supplied Python code.
- Output is seen in Figure 11 as chessboard window with an input box underneath it.
- The model will replicate the black player's moves in reaction to your movements as the white player.

- Take note of the dynamic relationship between your actions and the model's forecasts.

5.3 Model vs. Stockfish Engine

We compare the model's performance to that of the powerful Stockfish engine in this phase to gauge its effectiveness. Before beginning this examination, take the following actions:

- Make sure Python, Jupyter Notebook, and the required libraries are installed.
- Before starting the supplied code, restart the kernel.

Figure 12: Model vs. Stockfish Engine Code and Importing Libraries

```
Game 30: 0-1
Game 31: 0-1
Game 32: 0-1
Game 33: 0-1
Game 34: 0-1
Game 35: 0-1
Game 36: 0-0
Game 37: 0-1
Game 38: 0-1
Game 39: 0-1
Game 40: 0-1
Game 41: 0-1
Game 42: 0-1
Game 43: 0-1
Game 44: 0-1
Game 45: 0-1
Game 46: 0-1
Game 47: 0-1
Game 48: 0-1
Game 49: 0-0
Game 50: 0-1
Total Model Wins: 0
Total Stockfish Wins: 46
Draws: 4
```

Figure 13: Model vs. Stockfish Engine Output

- Using the installation file contained in the artifacts, install the Stockfish engine on your computer.

- Make the necessary code changes to indicate the proper location of the Stockfish engine executable.

Run the code in the Jupyter Notebook after fulfilling the aforementioned requirements. The outcomes of 50 matches between the model and the Stockfish engine will be displayed in the output, including the number of draws, matches won by Stockfish, and matches won by our model.

This thorough analysis provides information on the model's predictive power, dynamic simulation skills, and strategic judgment while facing a top-tier adversary. You may do these assessments with ease if you follow these instructions, and experiments will provide you insightful data.