# Configuration Manual

MSc Research Project
MSc Data Analytics

Alan Babu Manuel
Student ID: x23157143

School of Computing
National College of Ireland

Supervisor: Athanasios Staikopoulos

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Alan Babu Manuel……. ………………………………………………………………………………………………… |
| **Student ID:** | x23157143………………………………………………………………………………………………….……..……. |
| **Programme:** | MSc Data Analytics……………………………………………………………… **Year:** 2024-2025………………………….. |
| **Module:** | …MSc Research Project………………………………………………………………………………………………….……… |
| **Lecturer:** | Athanasios Staikopoulos ………………………………………………………………………………………………….……… |
| **Submission Due Date:** | 24/04/2025………………………………………………………………………………………………….……..……… |
| **Project Title:** | ………………………………………………………………………………………………….……… |
| **Word Count:** | …566…………………………………… **Page Count:** …16…………………………….…….……… |

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:** …Alan Babu Manuel…………………………………………………………………………………………………

**Date:** …24/04/2025…………………………………………………………………………………………………

**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

Alan Babu Manuel
x23157143

## 1 Introduction

This configuration manual walks you through setting up the software environment, preparing the data, and running the models.

## 2 Hardware Setup

| | |
|---|---|
| **Operating System** | Windows 11 Home |
| **Installed RAM** | 8.00 GB |
| **Processor** | AMD RYZEN 3 |
| **System Type** | 64 bit Operating System |

## 3 Softwares And Libraries

**Programming Language**: Python
**Database**: Postgre SQL
**IDE** : Jupyter Notebook

**Packages:**

- Numpy
- Pandas
- sqlalchemy + psycopg2-binary
- Matplotlib
- Sklearn
- NLTK
- String
- Re
- emoji
- Keras
- Tensor flow
- Transformers
- KerasTuner
- vaderSentiment

## 4 Project Implementation

## 4.1 Dataset

The dataset for used for this research consists of playstore game app reviews sourced from kaggle[1] .It is a JSON dataset that consists web scraped data from google playstore for 170 games consisting of the app metadata and also the reviews of each games by different users.


```
"{""appInfo"": {""title"": ""League of Stickman - Best action game(Dreamsky)"", ""description"": ""Dreamsky Games: Just be Happy!\r\n\r\nLeague of Stickman is
one of stickman style cross-action mobile game, it's shadow fighters, smash up all enemies! blow all monsters! Ultimate challenge! An original multi-heroes
real-time combat, a strong sense of combat with perfect sound effect, gives you a different combat experience.\r\n \r\nFeatures:\r\n[Shadow fighter Action Game ]
\r\nYou can experience the thrilling sensation with features like Double-Hits,  Levitation and Deadly Combos! Come feel the heat and slay some monsters!\r\n\r\n
[Heroes Joining Forces]\r\nMore heroes unlocked and more heroes to team up with! Join forces with others to fight the Monster King BOSS! Choose your own team now
and start fighting!\r\n\r\n[Multiple characters]\r\nMany stickman heroes waiting for you to choose: Ninja\uff0cGus, Athy, Feist, Bladey, Zilong, shadow
fighter,Monk and more.\r\n\r\n[ Visual Experience]\r\nHigh-quality graphics! Stunning special effects! \r\n\r\n[Simple & Smooth Operation]\r\nFreely switch
between heroes and master their four different skills. You will have the best combat experience you ever had!\r\n\r\n[World Leaderboards]\r\nWe offer world
leaderboard, country leaderboard and friend leaderboard. Your team can compete with players around the world. Beat all others and be the Champion!\r\n\r\n\r\n
Follow Us on \r\nFacebook: https://www.facebook.com/leagueofstickman\r\nYoutube: https://goo.gl/BRjxWA\r\n\r\nSUPPORT:\r\nVisit our Official site:
http://www.leagueofstickman.com/"", ""descriptionHTML"": ""Dreamsky Games: Just be Happy!<br><br>League of Stickman is one of stickman style cross-action mobile
game, it&#39;s shadow fighters, smash up all enemies! blow all monsters! Ultimate challenge! An original multi-heroes real-time combat, a strong sense of combat
with perfect sound effect, gives you a different combat experience.<br> <br>Features:<br>[Shadow fighter Action Game ]<br>You can experience the thrilling
sensation with features like Double-Hits,  Levitation and Deadly Combos! Come feel the heat and slay some monsters!<br><br>[Heroes Joining Forces]<br>More heroes
unlocked and more heroes to team up with! Join forces with others to fight the Monster King BOSS! Choose your own team now and start fighting!<br><br>[Multiple
characters]<br>Many stickman heroes waiting for you to choose: Ninja\uff0cGus, Athy, Feist, Bladey, Zilong, shadow fighter,Monk and more.<br><br>[ Visual
Experience]<br>High-quality graphics! Stunning special effects! <br><br>[Simple &amp; Smooth Operation]<br>Freely switch between heroes and master their four
different skills. You will have the best combat experience you ever had!<br><br>[World Leaderboards]<br>We offer world leaderboard, country leaderboard and
friend leaderboard. Your team can compete with players around the world. Beat all others and be the Champion!<br><br><br>Follow Us on <br>Facebook:
https://www.facebook.com/leagueofstickman<br>Youtube: https://goo.gl/BRjxWA<br><br>SUPPORT:<br>Visit our Official site: http://www.leagueofstickman.com/"",
""summary"": ""Cool stickman shadow fighter,Smash up all monsters! Combo!Ultimate action game!"", ""summaryHTML"": ""Cool stickman shadow fighter,Smash up all
monsters! Combo!Ultimate action game!"", ""installs"": ""5,000,000+"", ""minInstalls"": 5000000, ""score"": 4.028169, ""ratings"": 94032, ""reviews"": 38598,
""histogram"": [12776, 3744, 8534, 11979, 56999], ""price"": 0.99, ""free"": false, ""currency"": ""USD"", ""sale"": false, ""originalPrice"": null,
""saleText"": null, ""offersIAP"": true, ""inAppProductPrice"": ""$0.99 - $103.23 per item"", ""size"": ""44M"", ""androidVersion"": ""4.0.3"",
""androidVersionText"": ""4.0.3 and up"", ""developer"": ""DreamSky"", ""developerId"": ""6760444657267083024"", ""developerEmail"":
""leagueofstickman@dreamsky.me"", ""developerWebsite"": ""http://www.leagueofstickman.com"", ""developerAddress"": ""Rooms 05-15, 13A/F., South Tower, World
Finance Centre, Harbour City, 17 Canton Road, Tsim Sha Tsui, Kowloon, Hong Kong.\nTel\uff1a00852-22060092\nFax\uff1a00852-30030133"", ""privacyPolicy"": ""
http://www.dreamsky.me/league-of-stickman-privacy-policy/"", ""developerInternalID"": ""6760444657267083024"", ""genre"": ""Action"", ""genreId"":
""GAME_ACTION"", ""icon"": ""https://play-lh.googleusercontent.com/p1qrneV9WNUuyj1KvIMpjQ3OwqP1Nlk_UTH2GQ2jv6WMzp0YPkxXoMk9k8w7QnDEbps"", ""headerImage"": ""
https://play-lh.googleusercontent.com/8idaFv3lsNOLNRnmV9-FZQSWaCS9zpQF4GgyNB38vGkZhl1MScagxXV2cHNoi2fw_xc"", ""screenshots"": [""
https://play-lh.googleusercontent.com/wtdMGLoro0dUM0J21fvJ69jduhNfZk473shdebT9BoQnLu5xowD0jFq6bnzUQOuQFMU"", ""
https://play-lh.googleusercontent.com/mwcaGlcoe9RqB6dhWE1mru3diGUM8Vm7ZmUEzCWV99WUQS9ixr-4qAkv0VcNSXu2jw"", ""
```


Figure1. JSON Dataset

This JSON dataset was in a nested structure and was hard to anlayse the data. So to make the analysis easier the JSON dataset was loaded in jupyter notebook and was flattened to dataframe and then stored in a PostgreSQL table named playstore_game_reviews. The table contains 2,89,604 rows of user reviews and playstore game meta data.

```
[5]:
import json
import pandas as pd


with open("PlayStoreGameAppInfoReview.json", "r", encoding="utf-8") as f:
    raw_data = f.read()


try:
    data = json.loads(raw_data)
except json.JSONDecodeError as e:
    print("Initial JSON load error:", e)

    raw_data_unescaped = bytes(raw_data, "utf-8").decode("unicode_escape")
    data = json.loads(raw_data_unescaped)

if isinstance(data, dict):
    print("The JSON is a dictionary with keys:", list(data.keys()))
    records = list(data.values())  # extract all records from the dictionary
else:
    records = data

# iterate over each record
app_info_list = []
reviews_list = []

for record in records:
    # If record is still a string, decode it
    if isinstance(record, str):
        record = json.loads(record)
```

Figure 2. Converting JSON dataset to Structured format

[1] https://www.kaggle.com/datasets/dipanjandas96/play-store-game-reviews

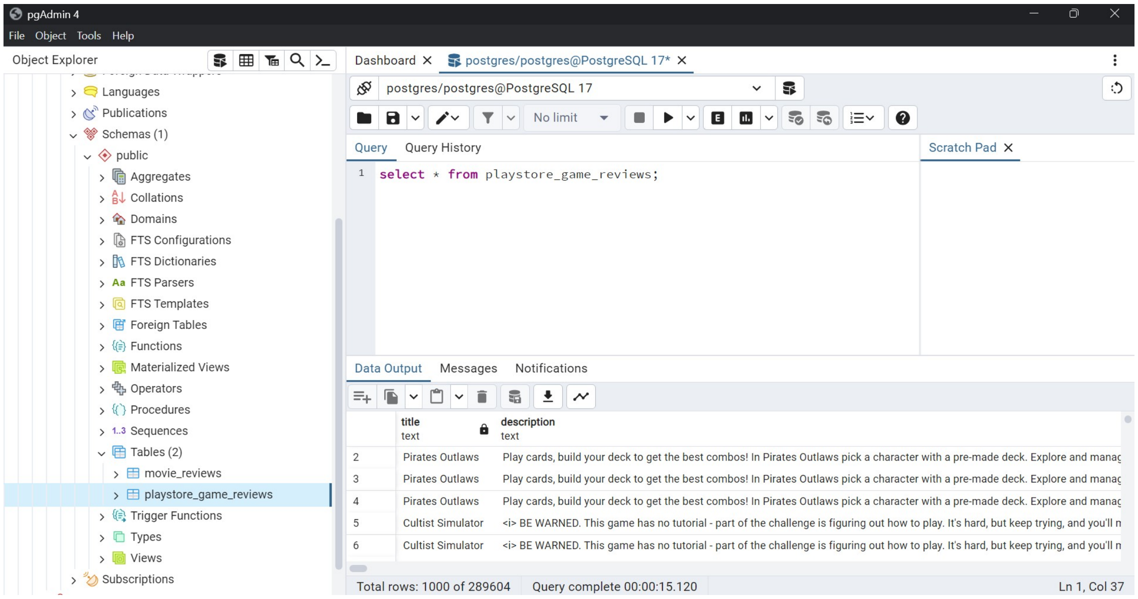

Figure 3. The playstore_game_reviews table in PostgreSQL.

## 4.2 Data Loading And Pre-Processing

Now as our dataset is stored in PostgreSQL table we will retrieve it for further processing and modelling. Out of the 2,89,604 rows we will be using only 1,00,000 rows due to memory and processing constraints of the hardware. The four models are implemented as 4 jupyter notebook files.

```
[4]: from sqlalchemy import create_engine

     # Define your PostgreSQL connection parameters
     db_user = 'postgres'
     db_password = 'admin'
     db_host = 'localhost'
     db_port = '5433'
     db_name = 'postgres'

     # Create a SQLAlchemy engine to connect to the PostgreSQL database
     engine = create_engine(f'postgresql+psycopg2://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}')
```

```
[6]: # Define your SQL query. You can modify this query to filter rows as needed.
     query = """
     SELECT *
     FROM playstore_game_reviews
     LIMIT 100000;  -- adjust this limit based on your memory and needs
     """

     # Read the query results into a DataFrame
     df = pd.read_sql(query, engine)

     print("Subset of data loaded into DataFrame:")
     print(df.head())

     Subset of data loaded into DataFrame:
                                                  title  \
```

Figure 4. Data Loading

Next we perform the text pre processing of our reviews to Remove punctuations, stopwords, HTML tags, emojis, special Characters etc. and then store it in a new column called cleaned_reviews

```python
def clean_review(text, method='lemmatization', do_spell_correction=True, min_words=3):

    # Handle missing or non-string values
    if not isinstance(text, str):
        return ""

    # Remove HTML tags if any
    text = re.sub(r'<[^>]+>', ' ', text)

    # Remove non-alphabetic characters and convert to lowercase
    text = re.sub(r'[^a-zA-Z\s]', ' ', text).lower()

   # To remove emojis
    text = emoji.demojize(text)

    tokens = text.split()

    # Filter out tokens that are in our stopwords list
    tokens = [word for word in tokens if word not in all_stopwords]

    # Ensure the review has a minimum number of words; otherwise, mark as empty.
    if len(tokens) < min_words:
        return ""

    # Apply stemming or lemmatization as desired
    if method == 'stemming':
        tokens = [stemmer.stem(word) for word in tokens]
    else:  # default is lemmatization
        tokens = [lemmatizer.lemmatize(word) for word in tokens]
```


```python
    # Filter out tokens that are in our stopwords list
    tokens = [word for word in tokens if word not in all_stopwords]

    # Ensure the review has a minimum number of words; otherwise, mark as
    if len(tokens) < min_words:
        return ""

    # Apply stemming or lemmatization as desired
    if method == 'stemming':
        tokens = [stemmer.stem(word) for word in tokens]
    else:  # default is lemmatization
        tokens = [lemmatizer.lemmatize(word) for word in tokens]
```


```python
    return " ".join(tokens)


df['cleaned_review'] = df['content'].apply(clean_review)
df.head()
```

Figure 5. Text Pre-Processing

## 4.3 VADER Rating

After Text Pre-processing we are providing an inferred rating from the cleaned review comments of the users using VADER. This decision was taken because even though there is a column for Score which represents the rating given by the user sometimes the Star rating left by the user do not match with  emotional tone of the review by the same user. VADER helps to bridge this gap byy feeding each review through VADER's compound metric (and remapping it into our 1–5 scale) we automatically "infer" a sentiment-based rating.

```
[12]: sid = SentimentIntensityAnalyzer()

      def get_sentiment_rating(text):
          sentiment = sid.polarity_scores(text)
          # VADER compound score is in [-1, 1]. We'll map it to a 1-5 scale.
          compound = sentiment['compound']
          rating = np.interp(compound, [-1, 1], [1, 5])
          return rating
```

```
[20]: if 'rating' not in df.columns or df['rating'].isnull().any():
          df['inferred_rating'] = df['cleaned_review'].apply(get_sentiment_rating)
          df['rating'] = df['inferred_rating']

      df.head(10)
```

[20]:

| appId | reviewId | userName | content | score | cleaned_review | inferred_rating | rating |
|---|---|---|---|---|---|---|---|
| ı.ninja.stickman.legends.shadow.wars | gp:AOqpTOHHhBznEeiXTiHXEM-_cP3ja0PHlV2tfLc2nHE... | Jc Salandanan | This is one of the best game that I have ever ... | 4 | one best ever played | 4.5300 | 4.5300 |
| ı.ninja.stickman.legends.shadow.wars | gp:AOqpTOH8ALCxty8SZETdqAA6Zg9srKA-yJrwWPCpGE-... | cool gamers | it's very cool and nice really nice carpeters | 4 | cool nice really nice carpeters | 4.5690 | 4.5690 |

Figure 6. VADER Rating

## 4.4 Feature Extraction

Next we encode the the user_ids and app_ids and also tokenize the reviews using Keras tokenizer(For LSTM model and Hybrid GRU-CNN model). For the DistilBERT and DistilGPT2 model we have used its respective pretrained encoders for review tokenizing.

```
[22]: # Encode user and app IDs
      user_ids = df['userName'].unique().tolist()  # Unique user names
      app_ids = df['appId'].unique().tolist()  # Unique app IDs

      # Create mappings from user and app IDs to numeric indices
      user2idx = {user: idx for idx, user in enumerate(user_ids)}
      app2idx = {app: idx for idx, app in enumerate(app_ids)}

      # Apply mappings to create numerical indices
      df['user_idx'] = df['userName'].map(user2idx)
      df['app_idx'] = df['appId'].map(app2idx)
```

Figure 7. User and App embedding

```
[90]: from tensorflow.keras.preprocessing.text import Tokenizer
      from tensorflow.keras.preprocessing.sequence import pad_sequences

      max_vocab = 10000
      max_length = 200

      tokenizer = Tokenizer(num_words=max_vocab, oov_token='<OOV>')
      tokenizer.fit_on_texts(df['cleaned_review'])
      df['review_seq'] = tokenizer.texts_to_sequences(df['cleaned_review'])
      df['review_seq'] = pad_sequences(df['review_seq'], maxlen=max_length).tolist()
```

```
[91]: # Prepare training data
      X_user = np.array(df['user_idx'].tolist())
      X_app = np.array(df['app_idx'].tolist())
      X_review = np.array(df['review_seq'].tolist())

      y = np.array(df['rating'].tolist())

      print("User input shape:", X_user.shape)
      print("App input shape:", X_app.shape)
      print("Review input shape:", X_review.shape)
      print("Ratings shape:", y.shape)

      User input shape: (100000,)
```

Figure 8. Review Tokenizing Using Keras Tokenizer

```python
# Function to tokenize reviews using DistilBERT
def encode_reviews_for_distilbert(texts, max_length=128):
    encoding = distilbert_tokenizer(
        texts.tolist(),
        padding='max_length',
        truncation=True,
        max_length=max_length,
        return_tensors='tf'
    )
    return encoding['input_ids'], encoding['attention_mask']
```

```python
# Prepare training data
X_user = np.array(df['user_idx'].tolist())
X_app  = np.array(df['app_idx'].tolist())
y = np.array(df['rating'].tolist())

print("User input shape:", X_user.shape)
print("App input shape:", X_app.shape)
print("Ratings shape:", y.shape)
```

Figure 9. Review Tokenizing Using DistilBERT Tokenizer

```python
# Function to tokenize reviews using DistilGPT
def encode_reviews_for_gpt(texts, max_length=128):
    encoding = gpt_tokenizer(
        texts.tolist(),
        padding='max_length',
        truncation=True,
        max_length=max_length,
        return_tensors='tf'
    )
    return encoding['input_ids'], encoding['attention_mask']
```

```python
X_input_ids, X_attention_mask = encode_reviews_for_gpt(df['cleaned_review'], max_length=128)
X_input_ids = X_input_ids.numpy()
X_attention_mask = X_attention_mask.numpy()
```

```python
# Prepare training data
X_user = np.array(df['user_idx'].tolist())
X_app  = np.array(df['app_idx'].tolist())
y = np.array(df['rating'].tolist())

print("User input shape:", X_user.shape)
print("App input shape:", X_app.shape)
print("Ratings shape:", y.shape)
```

Figure 10. Review Tokenizing Using DistilGPT2 Tokenizer

## 4.5 Modelling

In the following section the code snippets for the 4 models along with their hyper parameter tuning to find the best configurations for each model is shown.

### 4.5.1 LSTM Model With Attention And Glove Embedding

```python
def build_model(hp):
    # Input layers
    user_input = Input(shape=(1,), name='user_input')
    app_input = Input(shape=(1,), name='app_input')
    review_input = Input(shape=(max_length,), name='review_input')

    # User embedding
    user_embedding_dim = hp.Choice('user_embedding_dim', values=[8, 10, 12, 16, 20])
    user_embedding = Embedding(input_dim=len(user2idx), output_dim=user_embedding_dim, name='user_embedding')(user_input)
    user_vec = Flatten(name='user_flatten')(user_embedding)

    # App embedding
    app_embedding_dim = hp.Choice('app_embedding_dim', values=[8, 10, 12, 16])
    app_embedding = Embedding(input_dim=len(app2idx), output_dim=app_embedding_dim, name='app_embedding')(app_input)
    app_vec = Flatten(name='app_flatten')(app_embedding)

    # Review embedding using pre-trained GloVe embeddings (frozen)
    review_embedding = Embedding(input_dim=num_words,
                                 output_dim=embedding_dim,
                                 weights=[embedding_matrix],
                                 input_length=max_length,
                                 trainable=False,
                                 name='review_embedding')(review_input)

    # LSTM layer on review text
    lstm_units = hp.Choice('lstm_units', values=[50, 100, 150])
    lstm_out = LSTM(lstm_units, return_sequences=True, name='lstm_layer')(review_embedding)

    # Attention layer to focus on important words
```

```python
    lstm_units = hp.Choice('lstm_units', values=[50, 100, 150])
    lstm_out = LSTM(lstm_units, return_sequences=True, name='lstm_layer')(review_embedding)

    # Attention layer to focus on important words
    att_out = AttentionLayer(name='attention_layer')(lstm_out)

    # Concatenate all features
    concat = Concatenate(name='concatenate')([user_vec, app_vec, att_out])

    # Dense layers: choose number of units and dropout rate
    dense_units = hp.Int('dense_units', min_value=32, max_value=256, step=32)
    dense1 = Dense(dense_units, activation='relu', name='dense1')(concat)

    dropout_rate = hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)
    dropout_layer = Dropout(dropout_rate, name='dropout')(dense1)

    dense2 = Dense(dense_units // 2, activation='relu', name='dense2')(dropout_layer)
    output = Dense(1, activation='linear', name='rating_prediction')(dense2)

    model = Model(inputs=[user_input, app_input, review_input], outputs=output)

    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-3, sampling='log')
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])

    return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=10,
```

Figure 11. The LSTM Model Function

```
# Run the hyperparameter search
tuner.search([X_user_train, X_app_train, X_review_train], y_train,
             validation_split=0.1,
             epochs=5,
             batch_size=64)
```

```
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print("Best hyperparameters:")
print(f"User Embedding Dim: {best_hps.get('user_embedding_dim')}")
print(f"App Embedding Dim: {best_hps.get('app_embedding_dim')}")
print(f"LSTM Units: {best_hps.get('lstm_units')}")
print(f"Dense Units: {best_hps.get('dense_units')}")
print(f"Dropout Rate: {best_hps.get('dropout_rate')}")
print(f"Learning Rate: {best_hps.get('learning_rate')}")
```

```
Best hyperparameters:
User Embedding Dim: 16
App Embedding Dim: 8
LSTM Units: 150
Dense Units: 160
Dropout Rate: 0.0
Learning Rate: 0.00036265062575473466
```

```
# Build the best model and train it longer
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(
    [X_user_train, X_app_train, X_review_train],
    y_train,
    validation_split=0.1,
    epochs=10,
    batch_size=64
```

Figure 12. Training The LSTM Model for 10 epoch With Best Hyperparameters

### 4.5.2 Hybrid GRU-CNN Model

```
# Define the review input (sequence of token IDs)
review_input = Input(shape=(max_length,), name='review_input')

# Add an Embedding layer for the reviews.
embedding_dim = hp.Choice('review_embedding_dim', values=[50, 100, 150])
review_embedding = Embedding(input_dim=max_vocab,
                             output_dim=embedding_dim,
                             input_length=max_length,
                             name='review_embedding')(review_input)

# Encoder: GRU layer for sequential encoding.
encoder_units = hp.Choice('encoder_units', values=[64, 128, 150])
encoder_output = GRU(encoder_units, return_sequences=True, name='encoder_gru')(review_embedding)

# Temporal Convolution: Conv1D layer to capture local features.
conv_filters = hp.Choice('conv_filters', values=[64, 128, 256])
conv_kernel_size = hp.Choice('conv_kernel_size', values=[3, 5])
conv_output = Conv1D(filters=conv_filters, kernel_size=conv_kernel_size, activation='relu', name='conv1d')(encoder_output)

# Use the custom AttentionLayer instead of a Lambda.
review_vector = AttentionLayer(name='attention_layer')(conv_output)

# User and App inputs and embeddings
user_input = Input(shape=(1,), name='user_input')
app_input = Input(shape=(1,), name='app_input')

user_embedding_dim = hp.Choice('user_embedding_dim', values=[8, 10, 12])
user_embedding = Embedding(input_dim=len(user2idx), output_dim=user_embedding_dim, name='user_embedding')(user_input)
user_vec = Flatten(name='user_flatten')(user_embedding)
```

```python
    app_embedding_dim = hp.Choice('app_embedding_dim', values=[8, 10, 12])
    app_embedding = Embedding(input_dim=len(app2idx), output_dim=app_embedding_dim, name='app_embedding')(app_input)
    app_vec = Flatten(name='app_flatten')(app_embedding)

    # Concatenate all features
    concat = Concatenate(name='concatenate')([user_vec, app_vec, review_vector])

    # Dense layers
    dense_units = hp.Int('dense_units', min_value=32, max_value=128, step=32)
    dense1 = Dense(dense_units, activation='relu', name='dense1')(concat)
    dropout_rate = hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)
    dropout_layer = Dropout(dropout_rate, name='dropout')(dense1)
    output = Dense(1, activation='linear', name='rating_prediction')(dropout_layer)

    model = Model(inputs=[user_input, app_input, review_input], outputs=output)

    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-3, sampling='log')
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])
    return model
```

```python
tuner = kt.RandomSearch(
    lambda hp: mla_edtcnet_model(hp),
    objective='val_loss',
    max_trials=10,
    directory='mla_edtcnet_tuner',
    project_name='mla_edtcnet_app_reviews'
)

tuner.search([X_user_train, X_app_train, X_review_train], y_train,
             validation_split=0.1,
```

```python
tuner.search([X_user_train, X_app_train, X_review_train], y_train,
             validation_split=0.1,
             epochs=5,
             batch_size=64)
```

```
Trial 10 Complete [00h 12m 28s]
val_loss: 0.7468462586402893

Best val_loss So Far: 0.043323926627635956
Total elapsed time: 05h 12m 53s
```

```python
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print("Best hyperparameters:", best_hps.values)
```

```
Best hyperparameters: {'review_embedding_dim': 50, 'encoder_units': 150, 'conv_filters': 256, 'conv_kernel_size': 3, 'user_embedding_dim': 10, 'app_embed
ding_dim': 8, 'dense_units': 64, 'dropout_rate': 0.4, 'learning_rate': 0.0007935592341237341}
```

```python
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit([X_user_train, X_app_train, X_review_train],
                         y_train,
                         validation_split=0.1,
                         epochs=10,
                         batch_size=64)
```

Figure 13. Training The Hybrid Model for 10 epoch With Best Hyperparameters

### 4.5.3 DistilBERT Model

```python
def build_transformer_model(hp):

    user_input = Input(shape=(1,), name='user_input')
    app_input = Input(shape=(1,), name='app_input')

    # DistilBERT Review Inputs
    review_input_ids = Input(shape=(128,), dtype=tf.int32, name='review_input_ids')
    review_attention_mask = Input(shape=(128,), dtype=tf.int32, name='review_attention_mask')

    # User Embedding
    user_embedding_dim = hp.Choice('user_embedding_dim', values=[8, 10, 16])
    user_embedding = tf.keras.layers.Embedding(input_dim=len(user2idx),
                                               output_dim=user_embedding_dim,
                                               name='user_embedding')(user_input)
    user_vec = Flatten(name='user_flatten')(user_embedding)

    app_embedding_dim = hp.Choice('app_embedding_dim', values=[8, 10, 16])
    app_embedding = tf.keras.layers.Embedding(input_dim=len(app2idx),
                                              output_dim=app_embedding_dim,
                                              name='app_embedding')(app_input)
    app_vec = Flatten(name='app_flatten')(app_embedding)

    # DistilBERT Transformer Layer
    def distilbert_pooler(inputs):
        input_ids, attention_mask = inputs
        outputs = distilbert_encoder(input_ids, attention_mask=attention_mask)
        return outputs.last_hidden_state[:, 0, :]  # Take the CLS token representation


    review_vec = tf.keras.layers.Lambda(distilbert_pooler, output_shape=(768,), name='distilbert_pooler')([review_input_ids, review_attention_mask])

        # Concatenate all features
        concat = Concatenate(name='concatenate')([user_vec, app_vec, review_vector])

        # Dense layers
        dense_units = hp.Int('dense_units', min_value=32, max_value=128, step=32)
        dense1 = Dense(dense_units, activation='relu', name='dense1')(concat)
        dropout_rate = hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)
        dropout_layer = Dropout(dropout_rate, name='dropout')(dense1)
        output = Dense(1, activation='linear', name='rating_prediction')(dropout_layer)

        model = Model(inputs=[user_input, app_input, review_input], outputs=output)

        learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-3, sampling='log')
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])
        return model
```

```python
[35]: tuner = kt.RandomSearch(
          lambda hp: mla_edtcnet_model(hp),
          objective='val_loss',
          max_trials=10,
          directory='mla_edtcnet_tuner',
          project_name='mla_edtcnet_app_reviews'
      )

      tuner.search([X_user_train, X_app_train, X_review_train], y_train,
                   validation_split=0.1,
                   epochs=5,
                   batch_size=64)
```

Figure 14. The DistilBERT Model function

### 4.5.4 DistilGPT2 Model

```
def build_transformer_model(hp):

    user_input = Input(shape=(1,), name='user_input')
    app_input = Input(shape=(1,), name='app_input')

    # DistilGPT2 review inputs
    review_input_ids = Input(shape=(128,), dtype=tf.int32, name='review_input_ids')
    review_attention_mask = Input(shape=(128,), dtype=tf.int32, name='review_attention_mask')

    # User Embedding
    user_embedding_dim = hp.Choice('user_embedding_dim', values=[8, 10, 16])
    user_embedding = Embedding(input_dim=len(user2idx), output_dim=user_embedding_dim, name='user_embedding')(user_input)
    user_vec = Flatten(name='user_flatten')(user_embedding)

    # App Embedding
    app_embedding_dim = hp.Choice('app_embedding_dim', values=[8, 10, 16])
    app_embedding = Embedding(input_dim=len(app2idx), output_dim=app_embedding_dim, name='app_embedding')(app_input)
    app_vec = Flatten(name='app_flatten')(app_embedding)

    # Define GPT pooling function
    def gpt_pooler(inputs):
        input_ids, attention_mask = inputs
        outputs = gpt_encoder(input_ids, attention_mask=attention_mask)
        # Typically GPT models do not have a "pooler_output", so take the last token representation:
        return outputs.last_hidden_state[:, -1, :]

    # GPT pooling layer for review text
    review_vec = Lambda(gpt_pooler, output_shape=(768,), name='gpt_pooler')([review_input_ids, review_attention_mask])

    review_vec = Lambda(gpt_pooler, output_shape=(768,), name='gpt_pooler')([review_input_ids, review_attention_mask])

    # Concatenate all features
    concat = Concatenate(name='concatenate')([user_vec, app_vec, review_vec])

    # Dense layers
    dense_units = hp.Int('dense_units', min_value=32, max_value=128, step=32)
    dense1 = Dense(dense_units, activation='relu', name='dense1')(concat)
    dropout_rate = hp.Float('dropout_rate', min_value=0.0, max_value=0.5, step=0.1)
    dropout_layer = Dropout(dropout_rate, name='dropout')(dense1)
    dense2 = Dense(dense_units // 2, activation='relu', name='dense2')(dropout_layer)
    output = Dense(1, activation='linear', name='rating_prediction')(dense2)

    model = Model(inputs=[user_input, app_input, review_input_ids, review_attention_mask], outputs=output)

    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-4, sampling='log')
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])
    return model
```

```
istilGPT_tuner = kt.RandomSearch(
   build_transformer_model,
   objective='val_loss',
   max_trials=10,
   directory='distilgpt_tuner_playstore_app_dir',
   project_name='distilgpt_app_reviews'
```

Figure15. The DistilGPT2 Model function

```
best_distilGPT_hps = distilGPT_tuner.get_best_hyperparameters(num_trials=1)[0]
print("Best hyperparameters for transformer model:")
print(f"User Embedding Dim: {best_distilGPT_hps.get('user_embedding_dim')}")
print(f"App Embedding Dim: {best_distilGPT_hps.get('app_embedding_dim')}")
print(f"Dense Units: {best_distilGPT_hps.get('dense_units')}")
print(f"Dropout Rate: {best_distilGPT_hps.get('dropout_rate')}")
print(f"Learning Rate: {best_distilGPT_hps.get('learning_rate')}")

best_distilGPT_model = distilGPT_tuner.hypermodel.build(best_distilGPT_hps)
history_distilGPT = best_distilGPT_model.fit(
    [X_user_train, X_app_train, X_input_ids_train, X_attention_mask_train],  # Changed from X_movie_train to X_app_train
    y_train,
    validation_split=0.1,
    epochs=10,
    batch_size=64
)
```

```
Best hyperparameters for transformer model:
User Embedding Dim: 16
App Embedding Dim: 8
Dense Units: 128
Dropout Rate: 0.0
Learning Rate: 2.5723943682384322e-05
Epoch 1/10
1125/1125 ━━━━━━━━━━━━━━━━━━━━ 7091s 6s/step - loss: 2.7607 - mae: 1.1925 - val_loss: 0.6696 - val_mae: 0.6148
Epoch 2/10
1125/1125 ━━━━━━━━━━━━━━━━━━━━ 6990s 6s/step - loss: 0.6474 - mae: 0.6056 - val_loss: 0.5618 - val_mae: 0.5446
Epoch 3/10
1125/1125 ━━━━━━━━━━━━━━━━━━━━ 7021s 6s/step - loss: 0.5578 - mae: 0.5444 - val_loss: 0.5097 - val_mae: 0.5145
Epoch 4/10
```

## 4.6 Training and Validation Plots

```
# Plot training and validation loss and MAE
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss (MSE)')
plt.plot(history.history['val_loss'], label='Val Loss (MSE)')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Val MAE')
plt.title('Training vs Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()

plt.tight_layout()
plt.show()
```

Figure16. The Training and Validation plots

## 4.7 Recommendation Metrics(Recall@k, NDCG@K)

```python
def recall_ndcg(test_df, k=10, relevance_threshold=3.0):

    # Group the results by user
    user_groups = test_df.groupby('user')

    precisions = []
    recalls = []
    ndcgs = []

    for user, group in user_groups:
        # Sort the user's items by predicted rating in descending order
        group_sorted = group.sort_values('predicted_rating', ascending=False)
        top_k = group_sorted.head(k)

        # Determine relevant items (true rating above threshold)
        relevant = (top_k['true_rating'] >= relevance_threshold).astype(int).values
        num_relevant_in_top_k = relevant.sum()



        # Total number of relevant items for the user in the test set
        total_relevant = (group['true_rating'] >= relevance_threshold).sum()
        recall = num_relevant_in_top_k / total_relevant if total_relevant > 0 else 0.0

        # Calculate DCG@K
        dcg = sum([(2**rel - 1) / np.log2(idx + 2) for idx, rel in enumerate(relevant)])
```

```python
        # Determine relevant items (true rating above threshold)
        relevant = (top_k['true_rating'] >= relevance_threshold).astype(int).values
        num_relevant_in_top_k = relevant.sum()



        # Total number of relevant items for the user in the test set
        total_relevant = (group['true_rating'] >= relevance_threshold).sum()
        recall = num_relevant_in_top_k / total_relevant if total_relevant > 0 else 0.0

        # Calculate DCG@K
        dcg = sum([(2**rel - 1) / np.log2(idx + 2) for idx, rel in enumerate(relevant)])

        # Calculate Ideal DCG (IDCG@K)
        ideal_relevances = group['true_rating'].apply(lambda x: 1 if x >= relevance_threshold else 0)
        ideal_sorted = ideal_relevances.sort_values(ascending=False).head(k).values
        idcg = sum([(2**rel - 1) / np.log2(idx + 2) for idx, rel in enumerate(ideal_sorted)])

        ndcg = dcg / idcg if idcg > 0 else 0.0


        recalls.append(recall)
        ndcgs.append(ndcg)

    avg_precision = np.mean(precisions)
    avg_recall = np.mean(recalls)
    avg_ndcg = np.mean(ndcgs)

    return avg_precision, avg_recall, avg_ndcg
```

Figure17. Function to calculate Recall@K and NDCG@K