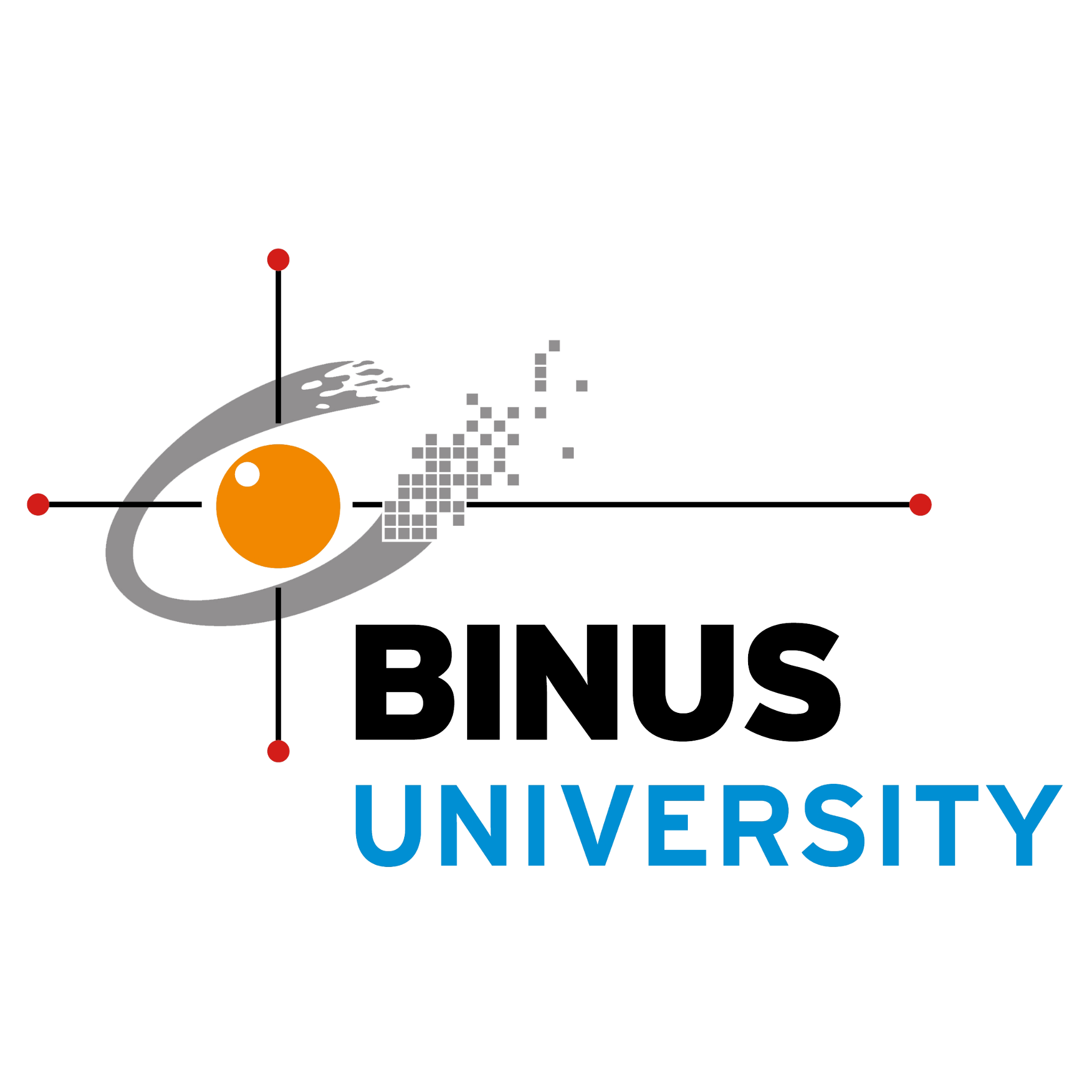
**FINAL PROJECT REPORT**

**ARTIFICIAL INTELLIGENCE**

**Conducting Sentiment Analysis on Genshin Impact Reviews with Bidirectional Encoder Representations Transformers**



Tiffany Widjaja - 2502059360

Jocelin Wilson - 2501963330

Daniel Eric Phangandy - 2501990312

Benedictus Filbert Federico - 2502005263

Table of Contents

1. Problem Description 3
2. Solution Features 3
3. Solution Design Architecture 7
4. Experiments 8

VII. Appendices: 17

1. Program manual 17
2. Link to git Site 18
3. Video 18
4. Contribution 18

**Problem Description**

Genshin Impact, a recently popular online game application developed by miHoYo, has garnered widespread downloads across all the mobile and PC platforms. With that said, there are a total of 4.4 million reviews written in the Google Play store itself. As the number of reviews for the game continues to grow rapidly, developers face the challenge of understanding past user feedback leading to the slow incorporation of meaningful insight into the game. The consequence was faced in 2021, where the game suffered an online attack known as review bombing in the Google Play Store caused by the lack of good rewards and the playerbase’s growing discontent during the game’s anniversary update. The game ratings went from a total of 4.5 down to a 2.8 and other apps ratings that were unrelated to the game were also affected negatively from this.

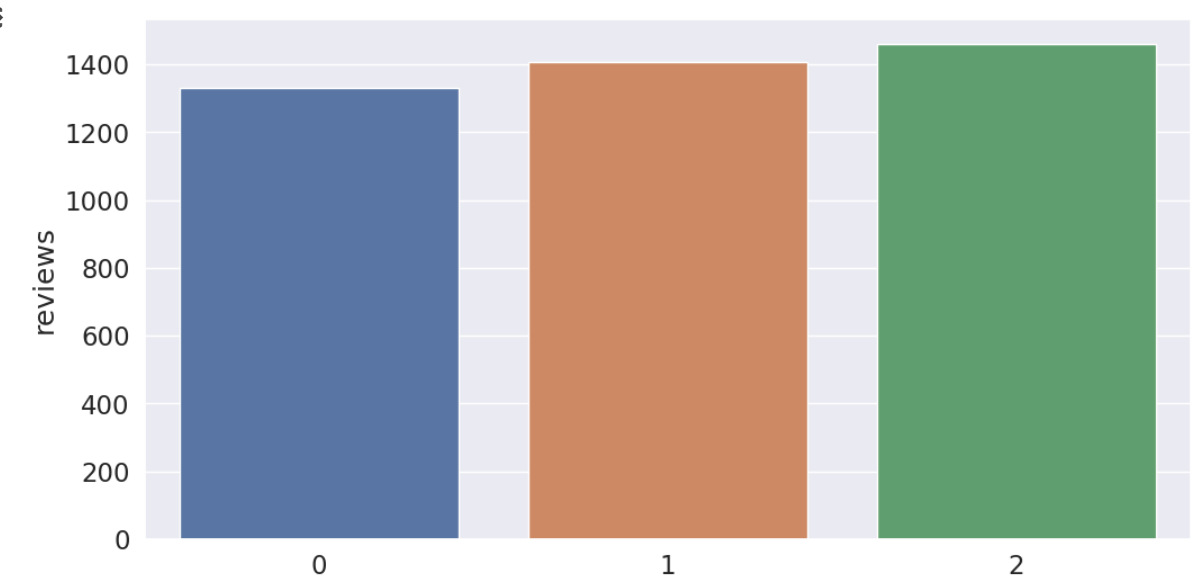
In response to this challenge, we decided to conduct sentiment analysis on the english reviews for the Genshin Impact application found in the Google Play Store. The goal is for the model to be able to accurately predict the user sentiment towards the game which can be either negative, neutral or positive. This analysis utilized the Bidirectional Encoder Representations from Transformers (BERT) method to gauge user sentiment, aiming to provide a comprehensive record for developers, current users, and potential users of the Genshin Impact application.

**Solution Features**

The dataset used for this research consists of Genshin Impact Playstore reviews all written in english. To obtain the data, we used web scraping methods on the Genshin Impact Google Play Store website with the scraper.py with tools like the google-play-scraper package, Python as the language and Visual Code Studio as the IDE. The initial data scraped consisted of reviewId, userName, content, score, thumbsUpCount, reviewCreatedVersion, at, replyContent, repliedAt, and appVersion and was exported into a csv file.

We managed to obtain 4200 reviews of data which is then further cleaned by removing the unnecessary features and only keeping important features such as the review ‘score’ and ‘content’ feature. The score is the review star ratings ranging from 1-5 while the content consists of the users thoughts and feelings on the game. Additionally, the csv file was converted into excel file all the reviews content were converted into lower case since we will be using pre-trained BertForSequenceClassification for this task such as the bert-base-uncased, bert-base-multilingual-uncased and bert-base-multilingual-uncased-sentiment. Consecutively, extra spaces, emojis, emoticons, punctuation, contractions have also been cleaned out from the data manually with the help of excel tools. The numbers found in the review are also converted into a word. For instance, the phrase “5 star” is changed into “five star” to help standardize the text and make it easier for the model to identify patterns and relationships between words.

Furthermore, a label called ‘sentiment’ is created to indicate whether a review is positive, neutral or negative. If the score is 1-2, then the sentiment will be a negative which is 0. If the score is 3, the sentiment will be neutral shown by a 1. Finally, if the score is 4-5, it indicates that the sentiment of the review is a positive. With seaborn and matplotlib library, we visualized the reviews based on their sentiment to see how much data there is in each class.



**Figure 1**

In this figure, It is visible that the positive, neutral and negative reviews all have approximately the same amount of data of around 1300~1400 to avoid any sort of data imbalance during the training process. This is done so that the model can perform better accuracy wise and more consistently when predicting the sentiment of unseen reviews later.

Afterwards, we extract the sentiment and content of the data as numpy arrays by assigning the content values from the dataframe into sentences while sentiment values are assigned into labels. This is to prepare for the tokenization process where the tokenizer splits each sentence into tokens or sub words and each of the tokens have a unique vocabulary ID that will be mapped into. Additionally, we also added the special tokens like CLS at the beginning of the sentence and SEP at the end of each sentence. We utilized the BERT tokenizer called bert-base-uncased for this. The sentences are also padded and truncated to a length of 64 so that they all share the same length or input size. Otherwise, the BERT model will not accept it since our reviews contains varying lengths of sentences. Afterwards, we created an attention mask to differentiate which tokens are actual words and which are just padding. If a token ID is 0, then it is padding, otherwise it is a real token.

**Splitting Training Set**

Before training, we had to split the training dataset into two parts. The first is to split the entire dataset into training and testing sets where 90% of the actual dataset will be used to train the model and the 10% will be used for testing and evaluation. The second part is to split the training set into training and validation sets. The training set will be use 90% for training and 10% for validation. Eventually, all the inputs and labels are converted into torch tensors, which is the required data type for our model. We also created an iterator for the dataset with the torch DataLoader class to save memory during the training process.

**Optimizer and Fine Tuning**

For the optimizer we are using the Adam optimizer. We had to fine tune the hyperparameters such as the batch size, learning rate, and number of epochs in order to find out which hyperparameters produced the best evaluation scores. The learning rates we experiment on are 1e-5, 2e-5, 3e-5, 4e-5 and 5e-5. As for the number of epochs, we experiment from 5 up to 40. As for the batch size, we tried 8, 16, and 32. The epsilon parameter is always 1e-8 and remains unchanged to prevent any division by zero in the implementation.

**BERT Model**

We used BertForSequenceClassification as the sentence classifier, which is the normal BERT model that has a single linear layer added on top for classification. As we feed input data, the entire pre-trained BERT model and the additional untrained classification layer is trained on our specific task. The goal is to train the pretrained model on our dataset until the model’s performance is well-suited for our task in classifying which review is positive, negative or neutral. This is later tested in a separate program containing the script which will take in user’s input for a review and it will print out the predicted sentiment and label.

**Training and Validation**

We implemented a training loop for the training and validation process using PyTorch, random and numpy for a specified number of epochs. In the training phase, the model is trained on batches of data, and the optimizer updates the model parameters based on the computed gradients. The learning rate is then adjusted using a scheduler while the training loss and time are monitored throughout the whole process. In the validation phase, the model is evaluated on a separate validation dataset, and performance metrics such as accuracy and loss are computed. The training loop iterates through the specified number of epochs, recording relevant statistics for each epoch, such as training and validation loss, accuracy, and timing information.

**Loss and Accuracy Visualization**

After it finished training, we visualized the training loss, validation loss, validation accuracy, training time and validation time into a table. We also used seaborn and matplotlib to plot the training loss and validation loss against the amount of epochs to give us better insight on setting the epoch number for the next round of fine tuning.

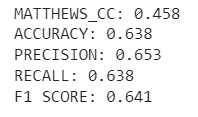


**Figure 2**

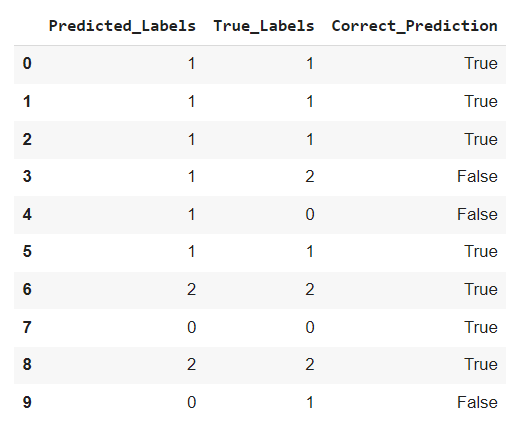
**Prediction and Results**

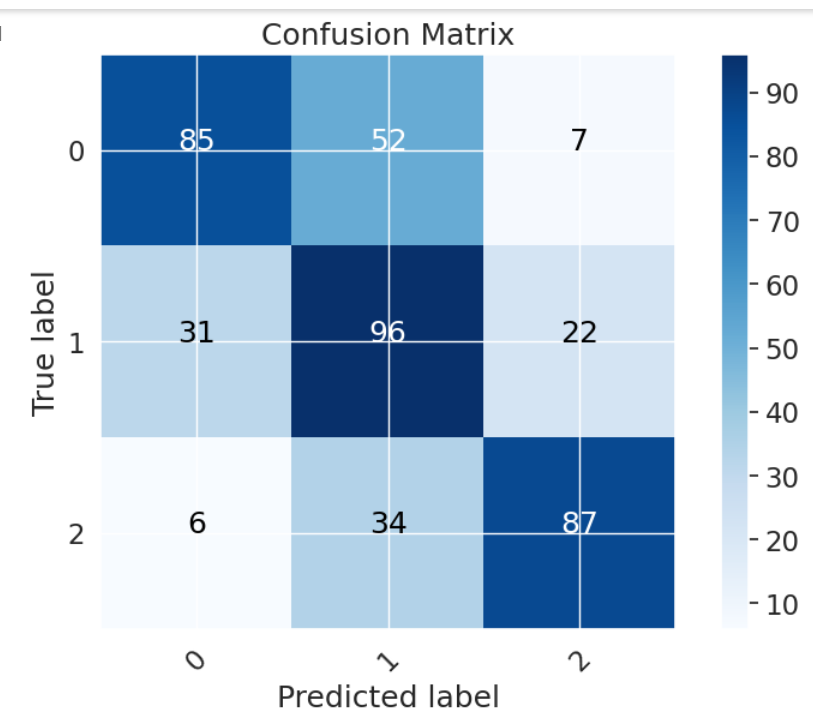
Consecutively, we need to put the model into the test by having it predict labels for 420 sentences on the test dataset after the training and validation phase is completed. In this section, the model is set to evaluation mode and iterates through batches of the test data, transferring them to a GPU and performs a forward pass through the model to obtain predictions. After that, the gradient computation is disabled and the logits are extracted from the model outputs, detached from the computation graph, and moved to the CPU. Similarly, label IDs are transferred to the CPU and converted to a NumPy array. The predictions and true labels are then appended to their respective lists.

After it finishes predicting, the lists are converted back to NumPy arrays to display the predicted labels and true labels. With sklearn metrics, we computed the evaluation metrics like Matthews Correlation coefficient, precision, recall, F1 Score, and test accuracy for how well the model performed in the test. Additionally, we used pandas again to show the predicted labels and true labels from the test results. It will display True when the prediction is a correct prediction and that occurs when the predicted labels share the same value as the true labels. Likewise, we also calculated and plotted the true and false predictions into an unnormalized confusion matrix. In the evaluation of the classification model, the confusion matrix was utilized to assess the model’s performance across the True labels as three classes, such as class 0, 1, and 2(figure 5). The model demonstrated the highest accuracy in predicting instances of class 1, correctly identifying 96 instances. However, it exhibited some difficulty in distinguishing between classes 0 and 1, incorrectly predicting 52 instances of class 1 as class 2 and 31 instances of class 2 incorrectly predicted as class 1. Furthermore, the model also struggled to differentiate between classes 1 and 2, misclassifying 34 instances of class 2 incorrectly predicted as class 3 and 22 instances of class 3 incorrectly predicted as class 2.



**Figure 3**





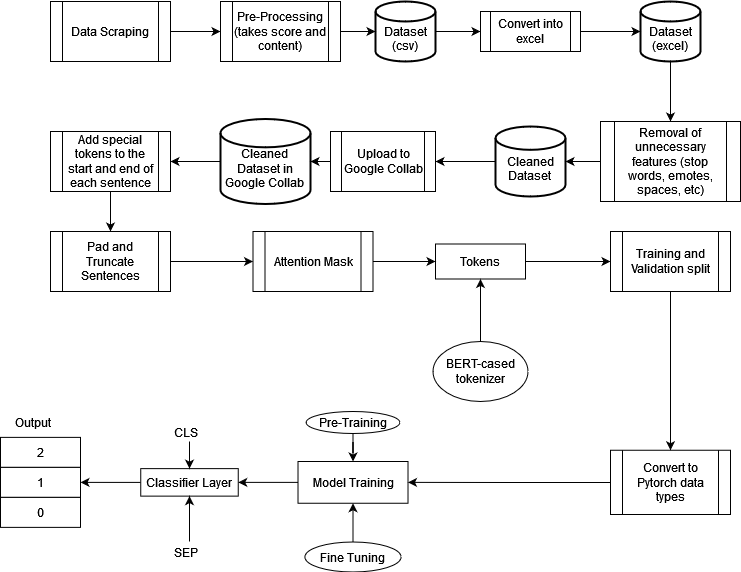
**Figure 4 Figure 5**

**Saving and Loading the Model**

Once we are satisfied with the model’s performance, the model is saved to another directory called GenshinModel\_save so that we can load the model into another file. Also we created a testing script in a separate file where we can test the saved model there by loading it and running the program.

**Solution Design Architecture**

Our solution architecture is visualized in the diagram below:

****

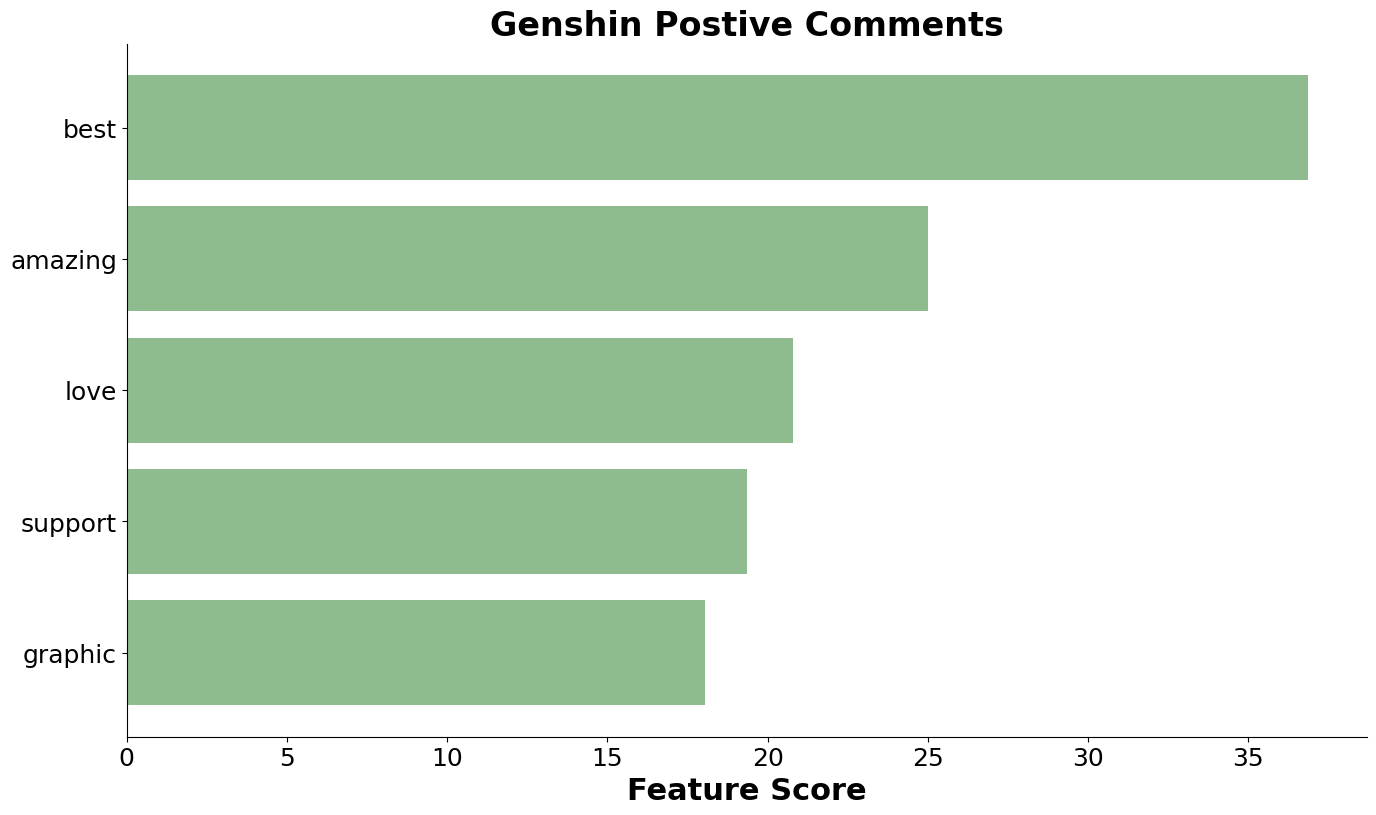
**Experiment**

**I. Result**



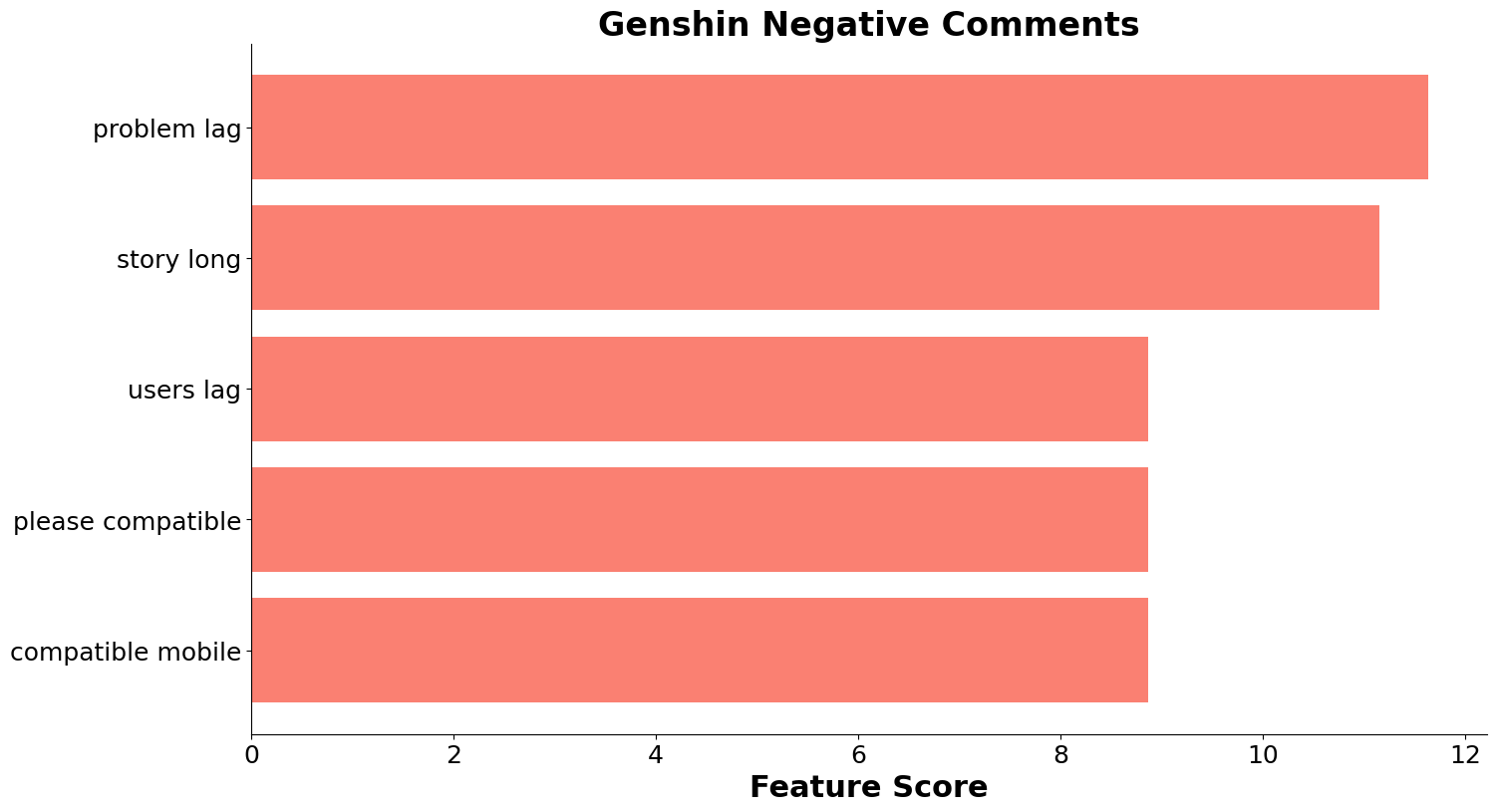
**Figure 6** Word Cloud

In Figure 6, the word cloud from Genshin reviews, the words 'play' and 'character' are frequently mentioned where they are focused on discussing the gameplay, visuals, and narrative of the characters. Positive terms are often mentioned like 'good,' 'great,' 'love,' 'amazing,' and 'fun' suggest overall player satisfaction. Concurrently, the occasional appearance of terms like 'problem,' 'issue,' 'bad,' 'lag,' 'please,' and 'time' signals potential areas of concern, urging a closer look into user experiences for possible improvements. Other noteworthy terms including 'mobile,' 'experience,' 'update,' 'open world,' 'screen,' 'cooperative,' and 'content' offer valuable insights into player preferences, guiding potential game enhancements.



**Figure 7** Positive Review

In Figure 7, we highlight the top 5 most frequently mentioned words in positive reviews. The standout term is 'best,' underscoring the overall excellence of the game, whether it's referring to being the best game, character, or mechanic. Following closely are expressions like 'amazing' and 'love,' further emphasizing the positive sentiments shared by players. Another notable term is 'support' which refers to support characters in the dataset but mostly linked to discussions about device controller support. Lastly, the reviews often discuss the game's graphics, living up to Genshin's reputation for their captivating and stunning design aesthetics.



**Figure 8** Negative Review

In Figure 8, we highlight the top 5 most frequently mentioned words in negative reviews. Unlike Figure 7, we opted for two-word combinations in Figure 8 to provide a more detailed context in the reviews. The most frequently mentioned issue is 'lag experienced,' indicating challenges players face with the game's performance. Close behind is the concern regarding the 'length of the story quest,' as players express discontent with quests being excessively long. Another often mentioned complaint revolves around 'device compatibility,' with a focus on mobile devices.

**II. Evaluation**

To produce the best results for our model, we decided to run more than 30 experiments by fine tuning the parameters such as the learning rate, epsilon, batch size and number of epochs. Our tests along with the results of the evaluation score and test accuracy have been concluded in the form of a table below. It is to be noted that the ones recorded in the tables are only the ones above 60% test accuracy.

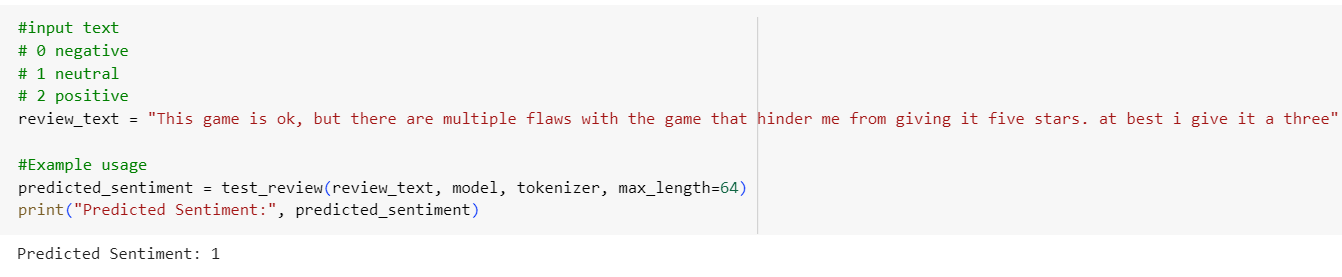
| BERT-BASE-UNCASED with Eps 1e-8 | | | | | |
| --- | --- | --- | --- | --- | --- |
| epochs | lr | Batchsize | Training Loss | Test  Accuracy | Evaluation score |
| 5 | 2e-5 | 16 |  | 63.81%  0: 88/144  1: 81/149  2: 99/127 | Matthews\_CC: 0.458  Accuracy: 0.638  Precision: 0.638  Recall: 0.638  F1 Score: 0.637 |
| 5 | 3e-5 | 8 |  | 60.71%  0: 74/144  1: 93/149  2: 88/127 | Matthews\_CC: 0.414  Accuracy: 0.607  Precision: 0.629  Recall: 0.607  F1 Score: 0.610 |
| 10 | 1e-5 | 8 |  | 63.57%  0: 88/144  1: 87/149  2: 92/127 | Matthews\_CC: 0.454  Accuracy: 0.636  Precision: 0.639  Recall: 0.636  F1 Score: 0.636 |
| 10 | 2e-5 | 8 |  | 61.67%  0: 84/144  1: 88/149  2: 87/127 | Matthews\_CC: 0.425  Accuracy: 0.617  Precision: 0.628  Recall: 0.617  F1 Score: 0.619 |
| 10 | 3e-5 | 16 |  | 60.71%  0: 74/144  1: 93/149  2: 88/127 | Matthews\_CC: 0.393  Accuracy: 0.595  Precision: 0.606  Recall: 0.595  F1 Score: 0.597 |
| 15 | 2e-5 | 16 |  | 64.29%  0: 81/144  1: 97/149  2: 92/127 | Matthews\_CC: 0.465  Accuracy: 0.643  Precision: 0.658  Recall: 0.643  F1 Score: 0.646 |
| 25 | 2e-5 | 16 |  | 63.10%  0: 73/144  1: 105/149  2: 87/127 | Matthews\_CC: 0.453  Accuracy: 0.631  Precision: 0.660  Recall: 0.631  F1 Score: 0.633 |
| 25 | 2e-5 | 32 |  | 61.43%  0: 84/144  1: 82/149  2: 92/127 | Matthews\_CC: 0.421  Accuracy: 0.614  Precision: 0.619  Recall: 0.614  F1 Score: 0.615 |
| 40 | 2e-5 | 16 |  | 63.33%  0: 80/144  1: 98/149  2: 88/127 | Matthews\_CC: 0.452  Accuracy: 0.633  Precision: 0.654  Recall: 0.633  F1 Score: 0.637 |

| BERT-BASE-MULTILINGUAL-UNCASED with Eps 1e-8 | | | | | |
| --- | --- | --- | --- | --- | --- |
| epochs | lr | Batchsize | Training Loss | Overall Accuracy | Others |
| 20 | 4e-5 | 16 |  | 62.38%  0: 81/144  1: 99/149  2: 82/127 | Matthews\_CC: 0.438  Accuracy: 0.624  Precision: 0.649  Recall:: 0.624  F1 Score: 0.629 |
| 20 and 30 | 3e-5 | 8 |  | 63.33%  0: 80/144  1: 100/149  2: 86/127 | Matthews\_CC: 0.453  Accuracy: 0.633  Precision: 0.656  Recall: 0.633  F1 Score: 0.637 |
| 40 | 3e-5 | 32 |  | 60.24%  0: 84/144  1: 83/149  2: 86/127 | Matthews\_CC: 0.403  Accuracy: 0.602  Precision: 0.615  Recall: 0.602  F1 Score: 0.606 |
| 40 | 5e-5 | 32 |  | 62.38%  0: 90/144  1: 86/149  2: 86/127 | Matthews\_CC: 0.434  Accuracy: 0.624  Precision: 0.627  Recall: 0.624  F1 Score: 0.625 |

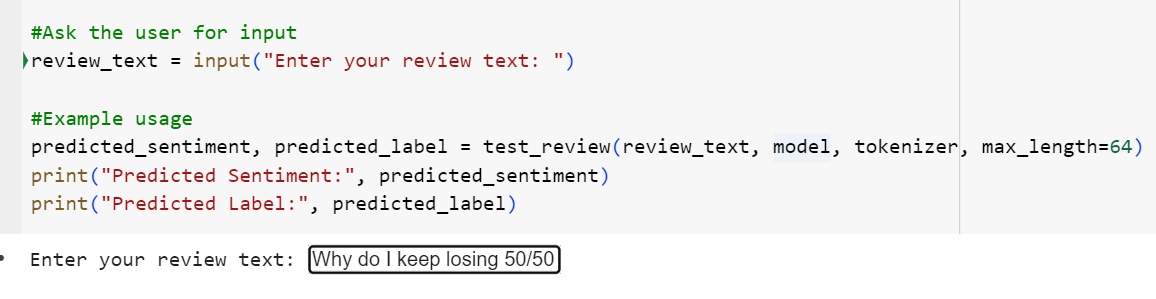
| BERT-BASE-MULTILINGUAL-UNCASED-SENTIMENT with Eps 1e-8 | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| epochs | lr | Batchsize | Weight  Decay | Training Loss | Overall Accuracy | Others |
| 10 | 1e-5 | 32 | - |  | 64.76%  0: 91/144  1: 88/149  2: 93/127 | Matthews\_CC: 0.470  Accuracy: 0.648  Precision: 0.650  Recall: 0.648  F1 Score: 0.649 |
| 10 | 1e-8 | 32 | - |  | 64.05%  0: 88/144  1: 89/149  2: 92/127 | Matthews\_CC: 0.460  Accuracy: 0.640  Precision: 0.647  Recall: 0.640  F1 Score: 0.642 |
| 10 and 30 | 1e-5 | 32 | 0.01 |  | 64.29%  0: 92/144  1: 92/149  2: 86/127 | Matthews\_CC: 0.463  Accuracy: 0.643  Precision: 0.652  Recall: 0.643  F1 Score: 0.646 |
| 30 | 2e-5 | 32 | 0.001 |  | 63.33%  0: 83/144  1: 94/149  2: 89/127 | Matthews\_CC: 0.450  Accuracy: 0.633  Precision: 0.644  Recall:0.633  F1 Score: 0.636 |
| 30 | 4e-5 | 32 | 0.001 |  | 63.81%  0: 82/144  1: 95/149  2: 91/127 | Matthews\_CC: 0.457  Accuracy: 0.638  Precision: 0.648  Recall: 0.638  F1 Score: 0.640 |

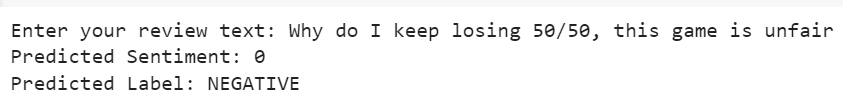
Results conclude that when conducting sentiment analysis reviews on Genshin Impact, with the bert-base-uncased model, it is recommended to use a batch size of 16 with 15 epochs and a learning rate of 2e-5 as it produces the best test accuracy along with the highest matthews coefficient correlation, recall and f1 score. Surprisingly, the model with 2e-5 learning rate, 25 epochs and batch size of 32 produces the best precision score. It also performs best in predicting the sentiment of 1 (neutral). When using the bert-base-multilingual-uncased model, it is best to use a learning rate of 3e-5 with batch size 8 and 20 or 30 epochs. Upon using the bert-base-mutilingual-uncased-sentiment, the best accuracy produced is by the parameters of 10 epoch, batch size of 32 and learning rate of 1e-5.

Even after many adjustments with its hyperparameters, it seems that all the types of BertForSequenceClassification consistently produces similar accuracy score results ranging from 60 - 64% at best and having a Matthews coefficient score of 40 - 46% max. However when putting the models into the test in predicting the sentiment of a text, the one that was able to predict the sentiment of the review accurately and consistently was the bert-base-uncased with 15 epoch, batch size of 16 and learning rate of 2e-5.



After finishing conducting all of these experiments, we decided to save the previously mentioned fine-tuned model and loaded it into a separate script where users can test out the model’s performance by inputting a review. Running the script will print out what the model thinks the sentiment is. A snippet of the ScriptGenshin.ipnyb is shown below:

After the user enters their review, the program will print out the predicted sentiment and label. The correct label for this review is negative and it has successfully predicted the sentiment 0 or negative.



**Appendices**

**Program Manual (Using VSCode)**

1. **Please gitclone the repository first:** [**https://github.com/Pandalmation/GenshinDeepLearning**](https://github.com/Pandalmation/GenshinDeepLearning)
2. **There will be a folder called FinalProject:**

### GenshinBERTModelFinal.ipynb (model training and more)

* GenshinReview.csv (cleaned dataset)
* ScriptGenshin.ipynb (to test the saved model)
* GenshinModel\_save (saved model)

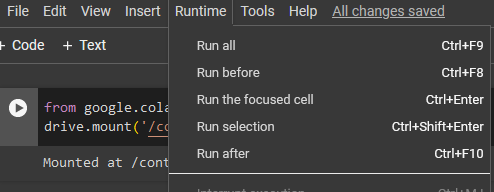
1. **If you would like to fine tune the model first, please run:** GenshinBERTModelFinal.ipnyb
2. **If you want to directly test the model, please run**:

ScriptGenshin.ipynb

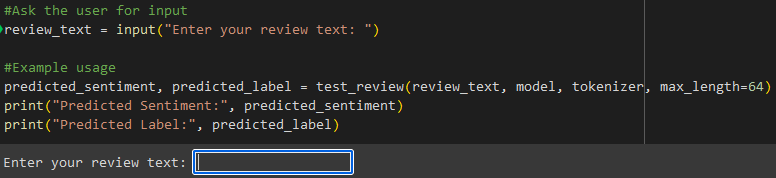
The program has some package dependencies, in order to install the packages the user must run the command !pip install followed by the package name which are:

* Pytorch
* Transformers
* Pandas
* Numpy

After the user installs the packages, they must run the code by either pressing the run button on the side (by doing this the user will need to run every code block individually), or by pressing runtime on the bar at the top and clicking on run all. In this example, we are running ScriptGenshin.ipnyb to test the model:



After it successfully runs, the program will then prompt the user to enter a text.



After the user enters a text the code will then predict the sentiment, which will either be 0, 1 or 2. A 0 means it is a negative sentiment, a 1 means it is a neutral sentiment and a 2 means it is a positive sentiment.



**Note: Please adjust the path files accordingly to access the model in case you get any errors.**

**Link to the Github containing the project files:**

[**https://github.com/Pandalmation/GenshinDeepLearning/tree/main/FinalProject**](https://github.com/Pandalmation/GenshinDeepLearning/tree/main/FinalProject)

**Link to the Github (for cloning the repository):**

[**https://github.com/Pandalmation/GenshinDeepLearning**](https://github.com/Pandalmation/GenshinDeepLearning)

**Link to Video Demo:**

[**https://youtu.be/n9C1-EYMwr0**](https://youtu.be/n9C1-EYMwr0)

**Team Contribution:**

**Daniel.P**

* **Role: Co-Author**
* **Fine tuning for bert-base-uncased**
* **Documented the Program Manual**
* **Created the solution design architecture**

**Filbert.F**

* **Role: Co-Coder**
* **Scraped the data**
* **Originally created the testing script**
* **Fine tuning for bert-base-multilingual-sentiment-uncased**
* **Made the Video Demo**

**Jocelin.W**

* **Role: Data Analyst**
* **Cleaned and preprocessed all 4200 data in Excel (lowercase, space removals, stopword removal, etc.)**
* **Fine tuning for bert-base-multilingual-uncased**
* **Documented all the experiments in the report**

**Tiffany.W**

* **Role: Main Coder/Main Author**
* **Created GenshinBERTModelFinal.ipnyb with internal documentation**
* **Updated and modified the testing script with the model loaded for scriptGenshin.ipnyb**
* **Documented the problem description, solution features and helped with evaluation**
* **Fine tuning for bert-base-uncased**