UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Posgrados

Optimizing Developer Experience: BERT and MLP for Spam Detection in Gaming Applications submissions

Proyecto de Titulación

Angel Molina Ortiz

Felipe Grijalva, Ph.D.

Director de Trabajo de Titulación

Trabajo de titulación de posgrado presentado como requisito para la obtención del título de Magíster en Inteligencia Artificial

Quito, 02 de diciembre de 2024

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ
COLEGIO DE POSGRADOS

HOJA DE APROBACIÓN DE TRABAJO DE TITULACIÓN

Optimizing Developer Experience: BERT and MLP for Spam Detection in Gaming Applications submissions

Angel Molina

Nombre del Director del Programa:	Felipe Grijalva
Título académico:	Ph.D. en Ingeniería Eléctrica
Director del programa de:	Inteligencia Artificial
Nombre del Decano del colegio Académico:	Eduardo Alba
Título académico:	Doctor en Ciencias Matemáticas
Decano del Colegio:	Ciencias e Ingenierías
Nombre del Decano del Colegio de Posgrados:	Dario Niebieskikwiat
Título académico:	Doctor en Física

Quito, diciembre 2024

© DERECHOS DE AUTOR

Nombre del estudiante:	Angel Molina Ortiz
Código de estudiante:	00339459
C.I.:	0503087371
Lugar y fecha:	Quito, 02 de diciembre de 2024.

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following graduation project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Este proyecto se centra en mejorar la experiencia de los usuarios en el portal para desarrolladores de Riot, optimizando la creación de aplicaciones que interactúan de manera eficiente y segura con todas sus API. Para lograr este objetivo, se ha implementado un modelo de inteligencia artificial basado en BERT y MLP, diseñado para detectar aplicaciones potencialmente no deseadas o spam. Al analizar los detalles de cada aplicación, el modelo predice si la aplicación debe ser aprobada o rechazada. El modelo propuesto demostró excelentes resultados de rendimiento, alcanzando una precisión del 91.34%, un puntaje F1 de 91.69% y un ROC AUC de 91.33%. Estas métricas resaltan la fuerte capacidad del modelo para clasificar aplicaciones con alta precisión y sensibilidad, minimizando falsos positivos y falsos negativos. Además, la tendencia constante a la baja en las gráficas de pérdida de entrenamiento y validación a lo largo de cada época muestra un aprendizaje efectivo y una generalización robusta a datos no vistos. Este enfoque no solo mejora el proceso de creación de aplicaciones y reduce el tiempo de aprobación, sino que también refuerza la seguridad de la plataforma al mitigar los riesgos de spam, garantizando una experiencia fluida tanto para desarrolladores como para usuarios.

Palabras clave: applications, spam, BERT, MLP, Naive Bayes, models, APIs.

ABSTRACT

This project focuses on improving the user experience on Riot's developer portal by streamlining the creation of applications that interact efficiently and securely with all its API's. To achieve this goal, an artificial intelligence model based on BERT and MLP has been implemented, designed to detect potentially unwanted or spam applications. By analyzing the details of each application, the model predicts whether the application should be approved or rejected. The proposed model demonstrated excellent performance results, achieving an accuracy of 91.34%, an F1 score of 91.69%, and an ROC AUC of 91.33%. These metrics highlight the model's strong ability to classify applications with high precision and recall, minimizing false positives and negatives. Moreover, the consistent downward trend in training and validation loss graphs over each epoch shows effective learning and robust generalization to unseen data. This approach not only improves the application creation process and reduces the approval time but also strengthens platform security by mitigating spam risks, ensuring a seamless experience for both developers and users.

Key words: applications, spam, BERT, MLP, Naive Bayes, models, APIs.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Optimizing Developer Experience: BERT and MLP for Spam Detection in Gaming Applications submissions

Felipe Grijalva, *Senior Member, IEEE,* Angel-Molina, *Member, IEEE*

***Abstract*—This project focuses on improving the user experience on Riot's developer portal by streamlining the creation of applications that interact efficiently and securely with all its API's. To achieve this goal, an artificial intelligence model based on BERT and MLP has been implemented, designed to detect potentially unwanted or spam applications. By analyzing the details of each application, the model predicts whether the application should be approved or rejected.**

The proposed model demonstrated excellent performance results, achieving an accuracy of 91.34%, an F1 score of 91.69%, and an ROC AUC of 91.33%. These metrics highlight the model's strong ability to classify applications with high precision and recall, minimizing false positives and negatives. Moreover, the consistent downward trend in training and validation loss graphs over each epoch shows effective learning and robust generalization to unseen data.

This approach not only improves the application creation process and reduces the approval time but also strengthens platform security by mitigating spam risks, ensuring a seamless experience for both developers and users.

***Index Terms*—applications, spam, BERT, MLP, Naive Bayes, models, APIs.**

I. Introduction

THE current growth of players and third-party developers across various games in recent years has been overwhelming. As a result, websites from renowned companies in the gaming industry, such as Riot Games [1], provide platforms where third-party companies can access statistics and real-time data to help players improve their performance and enhance the overall gaming experience.

However, this constant growth has introduced new challenges for both the company and third-party developers. One major issue is the presence of unwanted applications, commonly referred to as spam, on their developer portal. The current validation process for newly created applications can take anywhere from days to weeks, negatively affecting the user experience and increasing the likelihood of errors. This situation underscores the need for an automated solution.

This proposal presents a machine learning solution combining BERT (Bidirectional Encoder Representations from Transformers) [2] and MLP (Multi-Layer Perceptron) [3] to determine whether applications can be approved by analyzing their descriptions. We used a newly curated dataset of application descriptions for training and testing. Our latest results demonstrate significant improvements in detection accuracy and processing efficiency compared to other techniques, such as keyword detection or models like Naive Bayes, making this approach a viable solution to the current spam detection challenge.

This paper is organized into five sections on Background and Previous Work, Methodology, Experimental Results, Conclusions, and Acknowledgment.

II. Background and Previous Work

To better understand the proposed solution, this section reviews a previous model implementation and relevant technologies. It starts by exploring past work on spam detection, specifically the use of a Naive Bayes model for evaluating application descriptions. We then shift focus to more recent advancements, highlighting the use of BERT and MLP models.

The Naive Bayes model offered valuable insights; however, its performance is limited by the simplicity of its feature extraction process and its inability to capture complex relationships within the data. As a result, it struggles to accurately detect unwanted applications, leading to higher rates of false positives and false negatives.

On the other hand, BERT (Bidirectional Encoder Representations from Transformers) is a model developed to pre-train deep, bidirectional text representations using unlabeled data. What sets it apart from traditional models is its ability to account for both left and right context in every layer[2].

The architecture of the pre-trained BERT model requires only a single additional output layer for fine-tuning, allowing it to be used for different tasks such as question answering and language inference, with minimal changes to its structure[2]. **Figure 1** shows how the model transforms the input data.

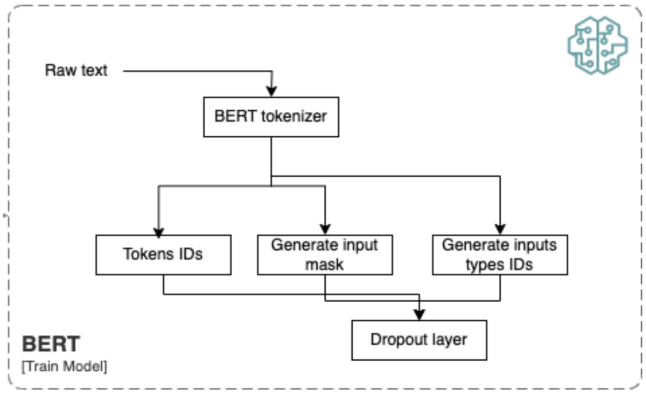

Figure 1. Bert model for classification

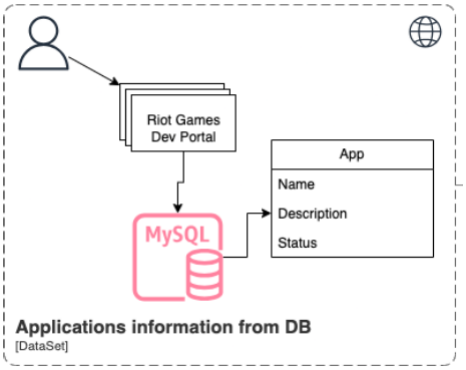

Figure 3. Application submit process

MLP is a reliable choice for classification problems. A multilayer perceptron is a model that takes a set of input values and produces the corresponding output values. This process is achieved by combining several simpler functions, with each function representing a different transformation of the input[4]. **Figure 2** offers a quick overview of the components involved in the classification structure.

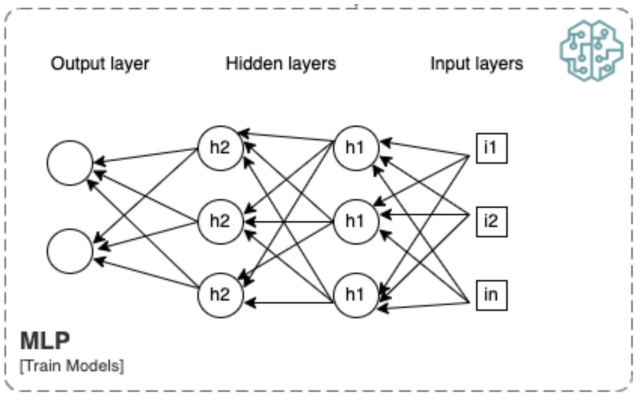

Figure 2. MLP model with two hidden layers

The dataset used contains roughly 87,000 applications records. These records are classified into several distinct statuses: approved, rejected, not verified, pending, and disabled. **Figure 4** displays how applications are divided in the Data base.

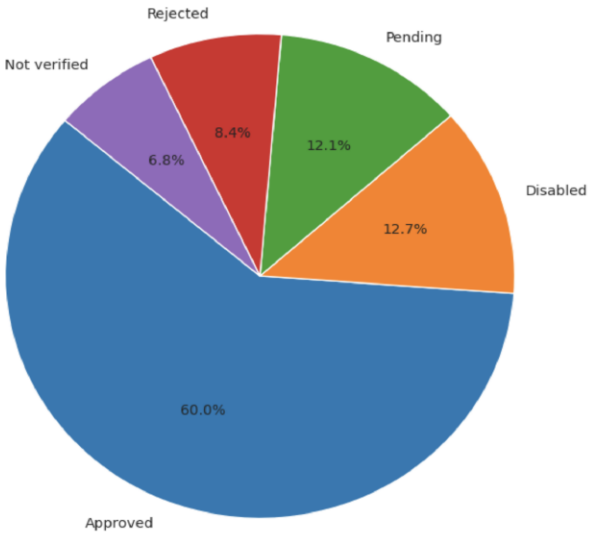

Figure 4. Applications by status

III. Methodology

This section outlines the entire process, from gathering data on all available applications to performing pre-processing and using machine learning models to generate a binary classification output.

A. Data preprocessing

It is important to understand how the data are stored on developer.riotgames.com [5]. **Figure 3** illustrates the current process of a user submitting an application request.

To preprocess the dataset, we must ensure there are no undesired entries with undefined or missing descriptions. We employ an under-sampling approach, selecting the same number of approved entries (around 10,555) as the rejected ones. This results in a balanced dataset, consisting only of the status and description columns. **Figure 5** shows a balanced dataset.

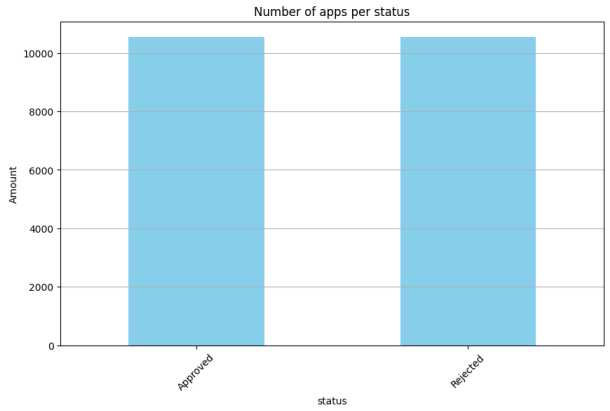

Figure 5. Dataset balanced with approved and rejected apps

With a balanced dataset, we then focus on cleaning and transforming the text. We consider only applications with valid descriptions, removing those with UUIDs [6], entries that consist solely of dates, sequences of numbers without context, and applications with only URLs.

Then, a lemmatization process is applied by using WordNet Lemmatizer [7]. Lemmatization connects words with similar meanings by reducing them to a common base form, enhancing the accuracy and efficiency of tools like chatbots and search engines. Its primary objective is to transform a word into its root form, known as a lemma. For instance, the verb "running" would be reduced to "run"[8]. **Figure 6** illustrates the entire preprocessing flow.

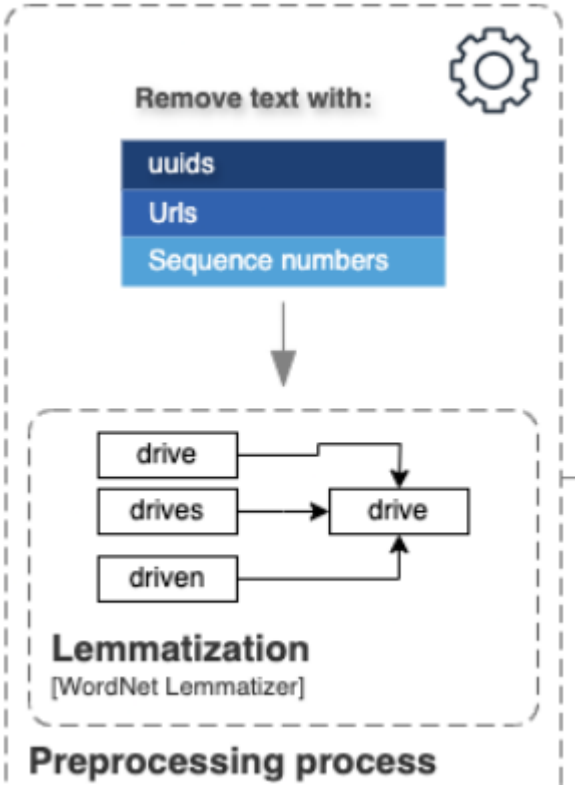

Figure 6. Preprocessing process

Lemmatization and the selection of only the best descriptions have resulted in an imbalanced dataset. Therefore, rebalancing is necessary before splitting the data into training, validation, and testing sets.

Furthermore, before applying any machine learning model, it is necessary to encode the target label, which represents the application's status. This step simplifies the classification process. The labels are converted from **"approved"** (1) and **"rejected"** (6) to (1) for **approved** and (0) for **rejected**. **Figure 7** illustrates the fields and values used to train the models.

	lemmatized_description	status_id
0	admin / password : team10h11 / lol_10h11 ! the...	1
1	(i apologize if you receive this application ...	1
2	early we received a interim api code for testi...	1
3	paul bunyan communication host a gaming event ...	1
4	the main priority of our web app is to gather ...	1
...	...	...
15241	i 'm making a lol tournament on small tourname...	0
15242	this description includes future feature this ...	0
15243	five stack is a team-building and matchmaking ...	0
15244	lol fantasy gg is a free fantasy sport website...	0
15245	this application will not need to use an api k...	0

Figure 7. Dataset with encoded status

The dataset distribution consists of 80% for training purposes, 10% for model validation, and 10% for final performance testing. **Table 1** displays the number of samples allocated to each process.

Table I
Data Split distribution

Process	Samples	Percentage %
Total samples	15246	100
Training samples	12196	79.99
Validation samples	1525	10.00
Testing samples	1525	10.00

B. Language model

The language model used is BERT, as mentioned in the first section. This model requires the data to be prepared for training. We selected the 'bert-base-uncased' tokenizer [9] with a lowercase configuration to ensure consistent treatment of words. For instance, **'Dog'** and **'dog'** are treated as the same.

Furthermore, PyTorch DataLoaders [10] are used to manage batching and sampling during the training, validation, and testing processes. We constructed a tensor dataset with a **RandomSampler** [11] to ensure effective stochastic

gradient descent, introducing variability that enhances the model's generalization capability on the training data. Additionally, a **SequentialSampler** [12] is used for the validation and testing data to ensure a deterministic and ordered evaluation process.

The PyTorch DataLoader class is a utility class that is used to load data from a dataset and create mini-batches for training deep learning models. [13]

Additionally, we considered freezing the lower layers of BERT. This approach offers several advantages, including resource optimization, reduced risk of overfitting, retention of pretrained knowledge, and greater flexibility during training. Specifically, resource optimization is crucial, as the BERT model contains numerous parameters. Fine-tuning all its layers would increase the training time by approximately 14 hours for this project.

Freezing the lower layers further decreases overfitting, as it encourages the model to focus on learning patterns that are specific to the task at hand, instead of adapting to low-level, generalizable features. This, in turn, enhances the model's ability to generalize to unseen data.

Furthermore, pretrained knowledge is particularly valuable for leveraging the vast, generalizable insights embedded in the pretrained model, while simultaneously adapting its behavior for a specific application. In addition, flexibility during training helps reduce both training and computational costs. This flexibility allows for testing various training configurations (e.g., hyperparameters, learning rates, batch sizes) without the computational burden of fine-tuning the entire model, resulting in a reduction of training time by up to 12 hours.

To analyze the model's interpretability, we employed a word cloud plot. This visualization technique allows for a quick and intuitive understanding of the most frequent terms or concepts in the dataset, highlighting key features that influence the model's decisions. By generating a word cloud based on the text data or predictions, we were able to visually identify dominant patterns or biases, which can guide further refinement of the model. **Figure 8** illustrate the words for approved status.

Figure 8. Word cloud of approved descriptions

Moreover, this approach provides an easy-to-understand summary of the data distribution. The word cloud plot, therefore, serves as both an analytical tool and a means of presenting data in a digestible format. **Figure 9** illustrate the words for rejected status.

Figure 9. Word cloud of rejected descriptions

C. BERT with MLP Classification

We have incorporated an hybrid model combining **BERT** with a Multilayer Perceptron (MLP) for sequence classification tasks.

Following BERT's output, the **MLP** model learns to map input features to specific class labels by adjusting its weights during training through backpropagation and gradient descent. The model consists of two fully connected layers, with hidden sizes of 256 and 128 units, respectively.

The first hidden layer, with 256 units, is sufficiently large to capture a significant amount of the complex features extracted by BERT. The second hidden layer, with 128 units, acts as a dimensionality reduction step. Reducing the number of units in this layer helps control overfitting and ensures that the model remains efficient, avoiding excessive complexity and computational cost.

In addition, we set up **ReLU (Rectified Linear Unit)** activations. The function returns 0 if it receives any negative input, but for any positive value $\boldsymbol{x}$ it returns that value back[14]. This introduces nonlinearity to the network, allowing it to model complex, nonlinear relationships between inputs and outputs. By applying ReLU after each linear layer, the network can capture intricate patterns in the data that would be unachievable with only linear transformations.

Moreover, we incorporated the regularization technique **dropout**, which helps mitigate overfitting in neural networks by limiting complex co-adaptations on the training data. The term **"dropout"** describes the process of randomly removing units (both hidden and visible) from the network during training[15].

In this model, a dropout rate of 0.5 is applied during each training step, meaning that half of the neurons in the dropout layer are randomly ignored. This helps reduce the network's capacity to memorize specific patterns in the training data, encouraging better generalization to unseen data.

D. Training

The training process incorporates several core components, including the AdamW optimizer, a learning rate scheduler, and early stopping. **Table II** displays the hyperparameters used for this model.

Table II
Model Hyperparameters

Hyperparameter	Value
BERT Model	bert-base-uncased
MLP Hidden Units	256 - 128
Dropout Rate	0.5
Optimizer	AdamW
Learning Rate	3e-6
Weight Decay	0.07
Early Stopping	5

AdamW optimizer is a stochastic optimization method that modifies the conventional weight decay implementation in Adam by decoupling it from the gradient update process[16]. It computes a per-parameter learning rate using the gradients first and second moments, offering improved weight decay management.

The weight decay is set to 0.07 for L2 regularization, also known as a penalty function, which helps prevent the model from overfitting by penalizing large weights.

The **learning rate** is set to 3×10^{-6}, a small value specifically chosen to avoid disrupting the pre-learned weights from BERT.

Moreover in the training process is to establish the warm-up steps and the learning rate scheduler.

The **warm-up steps** help the AdamW optimizer begin with smaller learning rates and gradually increase them, preventing overshooting during the initial training phases. We implement this by using a base value of 0.01, multiplied by the number of steps, equal to 1% of the total training steps.

The **learning rate scheduler** adjusts dynamically during training, varying the learning rate based on the training progress instead of keeping it constant. After the warm-up phase, the learning rate decreases linearly to zero by the end of training, helping to stabilize the training process and improve convergence.

Next, we define the training loop, which is designed to iteratively process the dataset for a specified number of 100 epochs. During each epoch, the model is trained on batches of data, with the loss computed and used to adjust the model's weights through backpropagation. The loop continues until all batches have been processed.

At the end of each epoch, the model is evaluated on the validation dataset to assess its performance on unseen data. During this validation step, the model's accuracy and loss are calculated, offering insights into how well the model generalizes. This process is essential for determining whether the model is overfitting or underfitting.

Finally, we implement the **early stopping** process. This technique monitors the validation accuracy, and if it does not improve after a specified number of epochs (defined by the patience parameter, set to five), training is halted early to avoid unnecessary computations. This process helps ensure that the model generalizes well and does not become overly specialized to the training set.

IV. Results

To establish a baseline for model performance, we initially evaluated the task using a Naive Bayes classifier. **Table III** presents the results for this model.

Table III
Result Metrics for Naive Bayes

Metric	Value	Percentage %
Accuracy	0.7626	76.26
F1 Score	0.79	79
Recall	0.79	79

The results of the Naive Bayes model highlight a noticeable gap in false positives and false negatives. Although the model achieves a respectable accuracy of 0.7626 and solid F1-score and recall values of 0.76, the presence of 505 false positives and 273 false negatives indicates that the model struggles to effectively distinguish between the positive and negative classes. **Figure 10** shows the results using the Naive Bayes classifier.

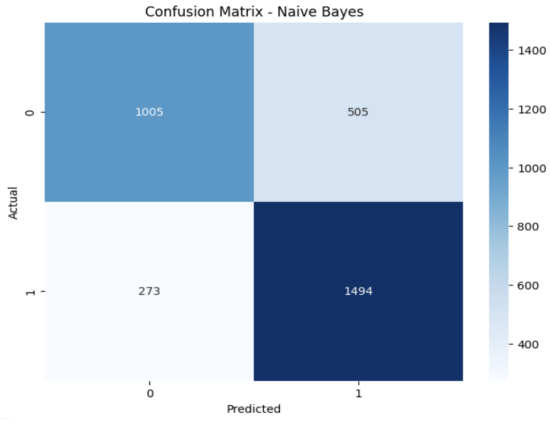

Figure 10. Confusion matrix for baseline result with Naive Bayes

The hybrid BERT with MLP model was evaluated using key metrics such as accuracy [17], F1 score [18], and ROC AUC [19].

The results give us an accuracy of 91.34%, indicating that it correctly classified the majority of instances in the dataset. While accuracy is an important metric, it can be misleading when class distributions are imbalanced. Therefore, to gain a more comprehensive understanding of the model's performance, we also consider the F1 score and ROC AUC. **Table IV** displays the results of the trained model.

Table IV
Result Metrics for BERT and MLP

Metric	Value	Percentage %
Accuracy	0.9134	91.34
F1 Score	0.9169	91.69
ROC AUC	0.9133	91.33

The F1 score an impressive value of 0.9169, demonstrating the model's strong ability to correctly classify both positive and negative samples while minimizing false positives and false negatives. This result is especially important in tasks where both precision and recall are crucial, such as binary classification problems with imbalanced class distributions.

Furthermore, the ROC AUC score of 0.9133 indicates that the model became very fruitful at distinguishing between the two classes (approved vs. rejected). An **ROC AUC** value close to 1.0 signifies that the model can make accurate predictions across a range of decision thresholds.

Moreover, the graph depicting the training and validation loss shows a consistent downward trend across epochs, indicating effective learning by the model. As the number of epochs increases, both losses decrease, suggesting that the model is progressively minimizing errors and refining its predictions.

The training loss consistently remains lower than the validation loss throughout the process, which is a positive sign of the model's ability to generalize without overfitting. This suggests that the model is not memorizing the training information given but is instead learning generalizable patterns. **Figure 11** display the validation and train loss trend over different epochs during the training process.

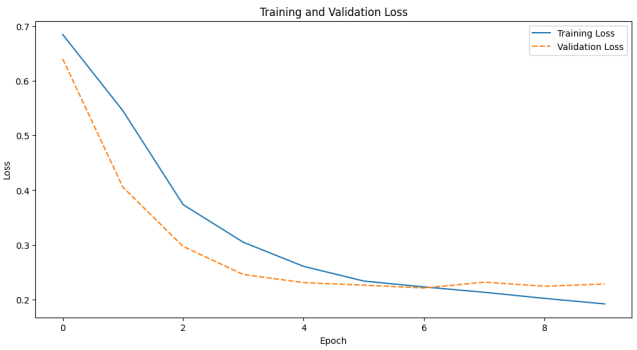

Figure 11. Validation vs Train loss during epochs

In addition, the validation accuracy graph shows a sharp increase during the initial epochs, indicating that the model effectively captures key patterns early in the training process. **Figure 12** illustrates the behavior of accuracy across different epochs.

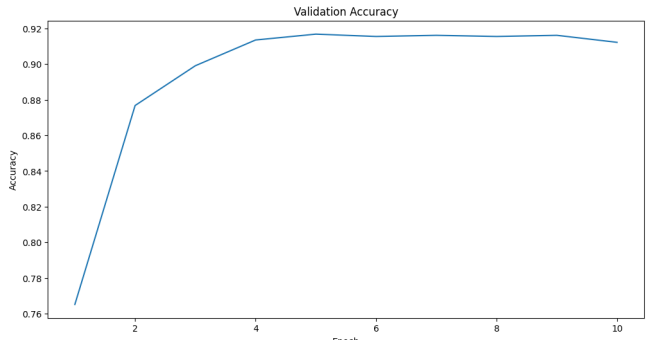

Figure 12. Validation accuracy

The Receiver Operating Characteristic (ROC) curve provides a visual comparison between sensitivity know as true positive rate and the false positive rate across various classification thresholds. With an Area Under the Curve know as AUC of 0.9133, the hybrid model demonstrates strong ability to distinguish between classes.

An AUC value close to 1.0 indicates that the model in effect can distinguishes between the positive and negative classes, significantly outperforming random guessing. The ROC curve's deviation from the diagonal line further reinforces the model's strong performance, highlighting its ability to reliably classify instances and make confident predictions across various decision thresholds. This confirms the robustness and reliability of the BERT + MLP hybrid model in differentiating between the approved and rejected classes.

Figure 13 illustrates the ROC curve performance throughout the training period.

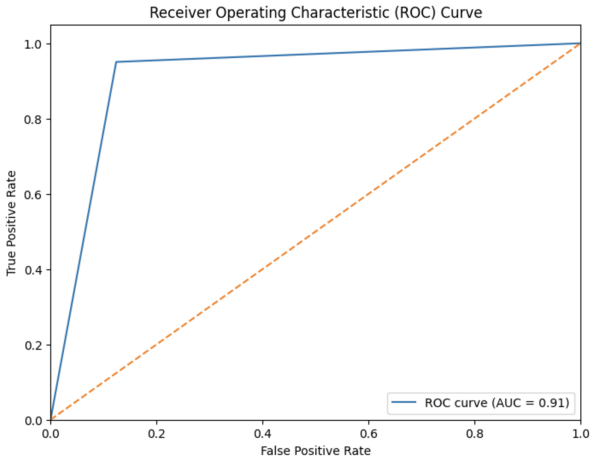

Figure 13. ROC curve

V. Conclusion

By combining BERT's contextual feature extraction with the multi-layer perceptron's classification capabilities, the hybrid model delivers a balanced and highly effective solution, outperforming simpler models in both precision and recall. This architecture showcases the advantages of leveraging deep learning models for complex classification tasks, where traditional approaches may fall short.

The hybrid model demonstrates effective learning, as evidenced by the consistent decrease in both training and validation loss over multiple epochs. The stable gap between these losses indicates that the model generalizes well to unknown data.

The high validation accuracy (91.34%) and ROC AUC score (0.9133) confirm the model's strong ability to distinguish between the two classes (approved vs. rejected) with high confidence, making it well-suited for further implementation in the Riot developers portal.

The F1 score of 0.9169 underscores the model's balanced precision, which is critical for this project, where both false positives and false negatives carry significant consequences. This balance demonstrates that the BERT + MLP hybrid architecture effectively captures complex patterns while maintaining strong predictive performance across both positive and negative classes, reducing the risk of biased predictions.

The early improvement in validation accuracy, followed by plateauing after the 4th epoch, indicates efficient convergence. The combination of a learning rate scheduler with warm-up steps, dropout, and weight decay ensures stable training while mitigating overfitting.

The early stopping mechanism, triggered by validation accuracy, further enhances stability by preventing unnecessary overtraining, making the model both computationally efficient and performance-optimized.

Acknowledgment

I want to extend my heartfelt thanks to the Developer Relations team at Riot Games for their support and cooperation in allowing me to propose and utilize the data for this project. Their collaboration has been invaluable in facilitating the research and enabling the successful execution of this work. I greatly appreciate their trust and assistance throughout this process.

References

[1] RIOT. (2024, Nov.) Riot games. [Online]. Available: https://www.riotgames.com/en

[2] huggingface. (2024, Nov.) Bert. [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/bert

[3] Scikit-learn. (2024, Nov.) Neural networks — supervised learning. [Online]. Available: https://scikit-learn.org/1.5/modules/neural_networks_supervised.html

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, available at: http://www.deeplearningbook.org.

[5] R. Games. (2024, Nov.) Riot games developer portal. [Online]. Available: https://developer.riotgames.com/

[6] C. Labs, "What is a uuid?" Nov. 2024, accessed: 2024-12-01. [Online]. Available: https://www.cockroachlabs.com/blog/what-is-a-uuid/

[7] N. Project. (2024, Nov.) Wordnetlemmatizer. [Online]. Available: https://www.nltk.org/api/nltk.stem.WordNetLemmatizer.html?highlight=wordnet

[8] TechTarget. (2024, Nov.) Lemmatization. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/lemmatization#:~:text=Lemmatization%20is%20the%20process%20of,processing%20(NLP)%20and%20chatbots.

[9] H. Face. (2024, Nov.) bert-base-uncased model. [Online]. Available: https://huggingface.co/google-bert/bert-base-uncased

[10] PyTorch. (2024, Nov.) Data handling basics — pytorch tutorials. [Online]. Available: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

[11] ——. (2024, Nov.) Randomsampler class — pytorch c++ documentation. [Online]. Available: https://pytorch.org/cppdocs/api/classtorch_1_1data_1_1samplers_1_1_random_sampler.html

[12] ——. (2024, Nov.) Sequentialsampler class — pytorch c++ documentation. [Online]. Available: https://pytorch.org/cppdocs/api/classtorch_1_1data_1_1samplers_1_1_sequential_sampler.html

[13] S. Cloud. (2024, Nov.) Pytorch dataloader: Features, benefits, and how to use it. [Online]. Available: https://saturncloud.io/blog/pytorch-dataloader-features-benefits-and-how-to-use-it/

[14] D. Becker. (2024, Nov.) Rectified linear units (relu) in deep learning. [Online]. Available: https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning

[15] P. Sanagapati. (2024, Nov.) What is dropout regularization? find out! [Online]. Available: https://www.kaggle.com/code/pavansanagapati/what-is-dropout-regularization-find-out

[16] P. with Code. (2024, Nov.) Adamw. [Online]. Available: https://paperswithcode.com/method/adamw

[17] G. Developers. (2024, Nov.) Accuracy, precision, and recall — machine learning. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

[18] Scikit-learn. (2024, Nov.) f1 score. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.f1_score.html

[19] G. Developers. (2024, Nov.) Roc and auc — machine learning. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc