1. Introduction

1.1 Background

In the past, humans have spent much of their time on simple tasks. Classifying sentences is one of the simplest yet most time-consuming of these tasks. For humans to classify a text, a series of processes are required: reading it, understanding its content, and then classifying it into the appropriate category. However, not only does this process take an enormous amount of time, but it is also prone to error, as the ability and judgment regarding classification varies from person to person. A system that can read text and classify it into clear categories would be a revolutionary advance. This project focuses on text classification and will build a system to classify abstracts of research papers into three categories: Physics, Mathematics, and Cybersecurity.

1.2 Natural Language Processing (NLP) Techniques for Text Classification

1.2.1 Overview of NLP and Its Significance

Natural Language Processing (NLP) is a technology and research field related to giving computers the ability to understand text and spoken language, the natural language people use every day, just as humans do. The algorithms that NLP can provide are immense, and these include communication understanding, information extraction, document classification, and language generation.

1.2.2 Text Classification in NLP

Text classification is one of the main tasks in NLP and is the process of assigning predefined categories or labels to sentences, paragraphs, text reports, or other unstructured text formats. Training a text classification model is the process of training an NLP model to classify text data in a dataset into known classes or categories in NLP. Different algorithms and approaches are used to train text classification models.Word2Vec is an algorithm for training distributed representations of words. Each word is represented by a vector of fixed dimension. When using Word2Vec for text classification models, each word is captured as a vector of its own words. The feature vector for the entire document is generated by averaging or combining the vectors of each included word. However, Word2Vec has limited consideration of context and may not consider word order information. BERT, on the other hand, is a deep learning model based on the transformer model, which considers context in both directions and captures a rich set of word meanings and relationships.

1.3 BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained model for predicting another sequence from a sequence of input word data. BERT is trained without input labels (unsupervised learning). Specifically, it performs two tasks simultaneously, Masked Language Model (MLM) and Next Sentence Prediction (NSP). BERT is pre-trained using large data sets to acquire rich representations of word meaning and context, and the pre-training itself is technically a supervised learning task with labeled data. Fine-tuning, which takes place after Pre-training, which refers to the process of adjusting a pre-trained model to fit a specific task.

BERT is also a bidirectional model, designed to consider context not only from left to right but also from right to left. This allows for a bidirectional view of word meaning and context, and BERT is characterized by higher accuracy in natural language processing than ELMo or OpenAI GPT. OpenAI GPT uses a left-to-right Transformer. (Figure 1) ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. (Figure 2)[1]

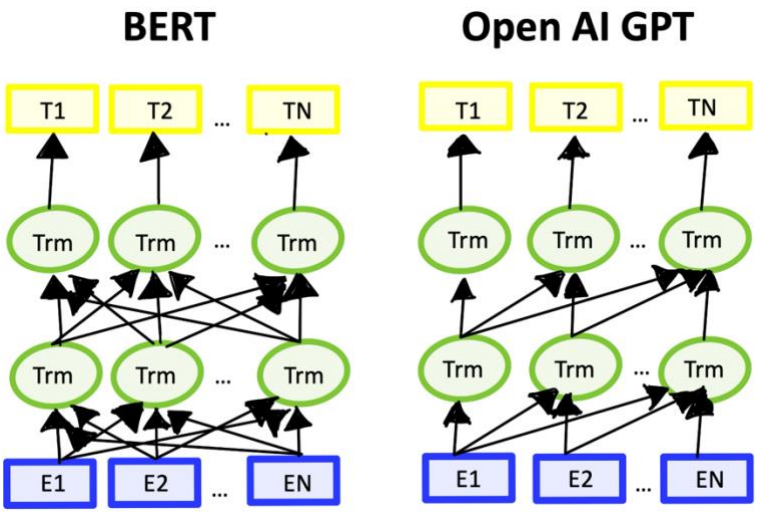


Figure 1: BERT and OpenAI GPT Algorithms

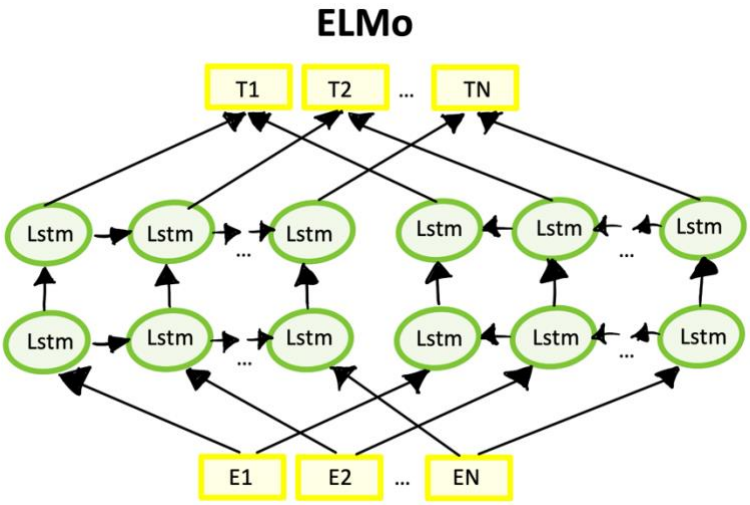


Figure 2: ELMo Algorithms

2. Methodology

The flowchart of the program is as follows. (Figure 3)

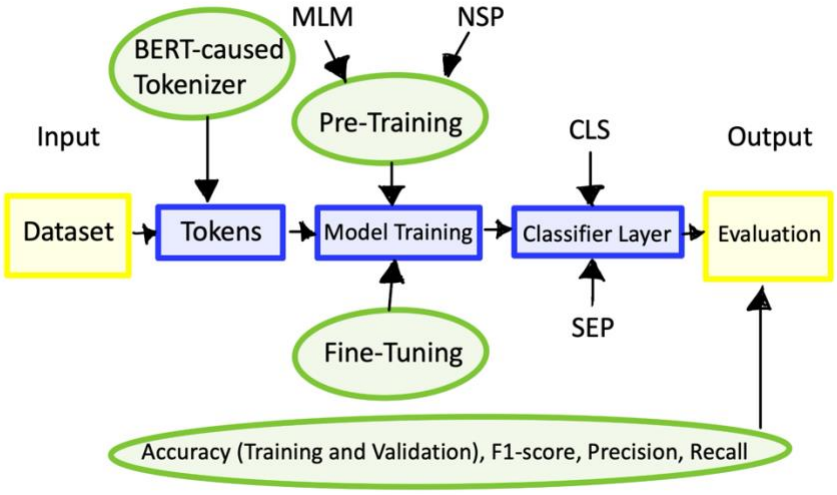


Figure 3: Flowchart of the program

This section describes the Method used in the program.

2.1 Dataset

A dataset is a set of data that is indispensable for learning a set of rules through machine learning. While the content and quantity of the data set increases the accuracy of machine learning, if the data set itself is inadequate, it will negatively affect the prediction after learning.

This project focused on specific categories and narrowed a big dataset down to 500 data sets for each category, for a total of 1500 data sets. Each discipline (Physics, Mathematics, Cybersecurity) was sampled under the condition that each paper was 1 and the other discipline was 0. The data from each of the sampled fields were combined to create a new dataset, which contains only the abstract and category information for each scientific paper, as the program would learn and train on the ID if it included the abstract number, ID, etc. The final dataset will be saved in CSV format. The project also uses Pandas for data manipulation and analysis.

2.2 BERT

2.2.1 Tokenizer

Tokenizer is the process of dividing a text into a vocabulary (tokens) and then converting it into a form that can be inputted into the BERT model. First, the text is split into tokens, which are the smallest model-specific semantic units. WordPiece tokenizers and Byte-Pair Encoding (BPE) are used to process subwords, which map different forms of words to the same token. The words are then converted to all lowercase. The program removes stopwords in part of the text processing as well. After that, each token is assigned an ID. The token is then converted to a vector or embedded representation.

The project uses BertTokenizer and loads a pre-trained model of BERT ('bert-base-uncased'). It splits words into sub-words and then further splits them into smaller words while retaining parts of the word. This allows for a wide variety of languages and terminologies.

Below is an example of the sub word Tokenization. (Figure 4)

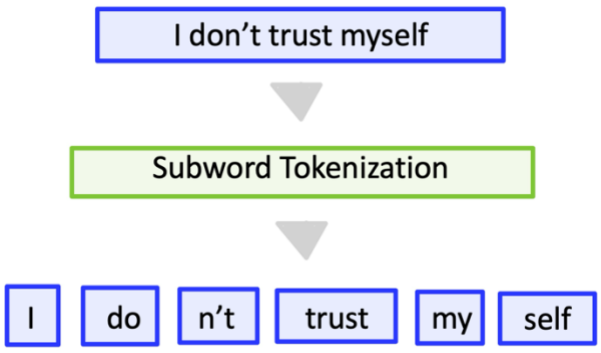


Figure 4: Example of the Sub word Tokenization

As well as dividing into words, the words are also divided by paying attention to the sub words contained within the words.

2.2.2 Transformers

This project uses the Transformers library to build the models. This library provides a variety of pre-trained language models. By initializing the BERT model and Tokenizer, respectively, the specified pre-trained model/tokenizer is downloaded and loaded into the new model/tokenizer. At the same time, the models are migrated to the GPU, enabling faster model training and inference and coping with large data sets.

2.2.3 Pre-Training

Pre-training in BERT is the process of acquiring everyday language comprehension skills, to transfer its learned models to various NLP tasks. The process uses unlabeled data and combines two approaches: Masked Language Model and Next Sentence Prediction.

- Masked Language Model: BERT takes text as a input in which a portion of a randomly selected text is replaced by another word, [MASK]. The replaced text is then fed to BERT, which learns through the task of predicting the token that was originally in [MASK]'s position. This enhances the ability to understand context and learn relationships between words.
- Next Sentence Prediction**:** A task that predicts the relationship between two sentences, where the second half of two sentences may be randomly replaced by a sentence unrelated to the previous one. This task trains models to determine if two given sentences are consecutive or not.

Below is an example of how MLM and NSP work. (Figure5)

Figure 5: Example of MLM and NSP

These pre-trained BERT models are typically used for fine-tuning. This allows BERT to understand the order of sentences and relationships between sentences, and to capture richer meaning in the language.

Rather than retrain the BERT model, this project adds a new classification layer to the BERT model. This will allow leveraging pre-trained features, learning small amounts of data, and adapting to the task. First, load a trained BERT model (e.g., bert-base-uncased) and specify the number of labels to be predicted.

The program fine-tunes this new model head with a sequence classification task and transfers the knowledge of the pre-trained model to it.

2.2.4 Fine-Tuning

Fine-tuning is the process of adding a new classifier (classification head) and learning the parameters of that classifier in order to apply BERT to a specific task. During fine-tuning, the parameters of the pre-trained BERT model are generally frozen, and the parameters of the new classifier are initialized randomly. The new task-specific model is then trained using labeled data. Fine-tuning uses different loss functions depending on the task and typically uses an optimizer (e.g. AdamW) to fine-tune the model parameters. [2] The setting of the learning rate (learning rate) depends on the task and data, but a typical value of 5e-5 is used. Once fine-tuning is complete, the performance of the model is evaluated. Common evaluation metrics include Accuracy, Precision, Recall, and F1-score. These metrics help quantify the model's performance on tasks.

2.3 K-Fold Cross-Validation

K-fold cross-validation is a technique that divides a data set into K folds (partial data sets), one of which is used as a test data set, and the remaining K-1 folds are used as training data sets. This is repeated K times, changing the test data set each time to train and evaluate the model. Finally, the performance of the model is evaluated by averaging the results of the K evaluations. [3] Cross-validation is used to ensure that the model performance evaluation is independent of data partitioning. In K-fold cross-validation, the entire data set is equally partitioned into training and test data, and data bias, thus stabilizing the performance evaluation.

The following is a schematic of how K-Fold Cross-Validation works. (Figure 6)

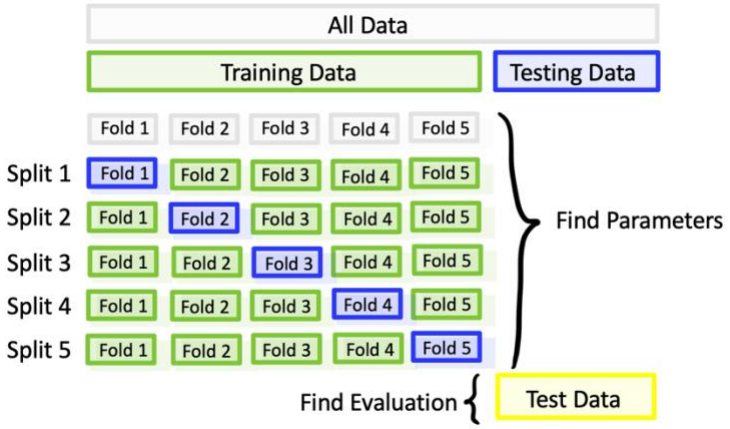


Figure 6: K-Fold Cross Validation Systems

The picture shows a case of five The Fold is changed 5 times so that all the data is once used as Test Data. Folds. One-fifth of each split is used for Testing Data, and the rest is used as Train Data.

In this project, cross-validation, which is more stable for evaluation than train-test-split, which only performs one evaluation using the split data is used. Typically, StratifiedKFold is used to split the data taking into account the distribution of classes. It ensures that the distribution of classes within each fold is consistent with the overall data set. Using more folds will result in a more accurate performance evaluation of the model but will increase the computational cost. Tokenizing the data, padding, and converting it to a PyTorch tensor is a common step in preparing the data into a format suitable for a neural network model. It is repeated K times so that all groups are test data.

2.4 Training Parameters

2.4.1 Batch Size

A technique called gradient descent is used to minimize the loss function and find the optimal parameters (weights, biases). In many cases, mini-batch gradient descent, a method that falls somewhere between batch gradient descent and stochastic gradient descent, is used, requiring the data set to be divided into several subsets. The amount of data in each subset is called the batch size. For example, if a dataset of 1000 data is divided into 200 subsets of 200 data each, the batch size is 200. It is common for the batch size to be a value of the nth power of 2. The batch size should be adjusted by the requirements of the program. [4]

2.4.2 Iteration

The number of iterations is the number of training runs required for the data in the data set to be used at least once and is automatically determined once the batch size is determined. In the case of a 1000 data set divided into 200 subsets of 200 data sets each, the number of iterations would be 5.

2.4.3 Epoch

Epochs refers to the number of times the dataset is divided into N subsets according to the batch size and the number of training iterations N times. This means that the data in the dataset is also trained at least once. Setting the number of epochs at which the values of the loss function approximately converge prevents under- or over-training.

2.5 Performance Evaluation and Result Aggregation

After training and validation are completed for each fold, the accuracy, precision, recall, and F1 scores are computed, respectively. The mixture matrices are normalized, and their averages are visualized. The precision and loss are also plotted. These indicators are useful for understanding the performance of the model from different aspects.

- Accuracy: Percentage of samples correctly classified relative to the total sample
- Precision: Percentage of positive observations that are correctly predicted.
- Recall: Percentage of correctly predicted positives relative to all observations within the actual class.
- F1 score: Harmonic mean of goodness-of-fit and recall
- Confusion Matrix: Table of predicted and actual class combinations
- Training and Validation Accuracy: Plot of the percentage of correct responses during training and validation of the model. By observing these plots, it is possible to see if the model is overfitting or underfitting.

Training and Validation Loss: A plot of the loss (error) during training and validation of the model. Losses should be converged, which helps identify over-training and under-training problems.

3. Experiments and Results

The performance of the model is compared and verified in detail from various aspects. Each experiment is conducted with all conditions together except for the area of focus so as not to make differences due to other aspects. Also, k-Fold Cross-Validation is set to 5.

3.1 Comparison with classifiers

3.1.1 Experimental Setup

This project built a program using a BERT-based model and three classifiers and compared accuracy and training time.
The comparisons are as follows.
1: Logistic Regression
2: Random Forest
3: Decision Tree Classifier
4: BERT
A dataset containing a total of 1500 papers, 500 papers in each field, was used. and the total number of epochs is set to 5 and the batch size to 16.

3.1.2 Experimental Results

The following table summarizes the accuracy achieved by each Classifier.

Model	*Validation Accuracy*	*Precision*	*Recall*	*F1-score*
1	90.11	90.39	90.11	90.13m
2	89.22	89.58	89.22	89.20m
3	70.22	71.18	71.11	71.03m
4	97.00	99.34	99.33	99.33m

The highest accuracy was 97.00% for BERT, followed by Logistic Regression at 90.11%, Random Forest at 89.22%, and Decision Tree Classifier at 70.22%.

3.2 Exploration of Epoch Range

3.2.1 Experimental Setup

The impact of changing the number of training epochs on the performance of the model was investigated. The initial conditions for comparisons were made as follows.

1: 5
2: 10
3: 30
This experiment allows us to understand how accuracy evolves with longer training times.
In this project, the BERT model used has, a dataset containing a total of 1500 papers with 500 papers in each field, and the batch size is set to 16.

3.2.2 Experimental Results

The following table summarizes the accuracy achieved for each epoch range.

Epoch	*Training Accuracy*	*Validation Accuracy*	*Precision*	*Recall*	*F1-score*	*Training Time*
1	97.71	97.00	99.34	99.33	99.33	28.53m
2	98.86	97.71	99.01	99.00	99.00	61.22m
3	62.24	60.57	44.48	33.67	17.37	130.49m

The highest validation accuracy was 97.71% when the epoch was set to 10. 30 was found to take longer and the validation accuracy dropped to 60.57%.

3.3 Dynamics of Batch Size

3.3.1 Experimental Setup

By focusing on the dynamics of batch sizes during training and comparing each batch size, insights can be gained into the efficiency of training. The initial conditions for comparisons are as follows.

1:4
2: 8
3: 16
4: 32
and compared using the BERT model, a dataset containing a total of 1500 papers with 500 papers in each field, and an epoch range set to 5 and 10.

3.3.2 Experimental Results

The following table summarizes the accuracy achieved for each batch size.

When epoch is 5:

Batch Size	*Training Accuracy*	*Validation Accuracy*	*Precision*	*Recall*	*F1-score*	*Training Time*
1	95.75	93.88	61.95	58.67	54.29	32.22m
2	97.84	97.36	100.00	100.00	100.00	29.38m
3	97.71	97.00	99.34	99.33	99.33	28.53m
4	-	-	-	-	-	-

When epoch is 10:

Batch Size	*Training Accuracy*	*Validation Accuracy*	*Precision*	*Recall*	*F1-score*	*Training Time*
1	79.69	77.31	11.11	33.33	16.67	61.22m
2	92.28	91.40	97.73	97.67	97.66	59.14m
3	98.86	97.71	99.01	99.00	99.00	57.34m
4	-	-	-	-	-	-

It was found that accuracy increased as Batch Size increased, whether Epoch was set to 5 or 10. However, due to GPU settings, it was found that a Batch Size of 32 or greater interrupted the training process.
Error when batch size is 32: Torch.cuda.OutOfMemoryError: CUDA out of memory.

3.4 Final Validation

3.4.1 Experimental Setup

From the results of 3.1 to 3.4, the most efficient and best-performing values.
Model: BERT model
Epoch: 10
Batch size: 16
It was also prepared an abstract for each category not included in the dataset. After building a program using these values and the BERT model, and after training and

testing, the abstracts are classified, and the categories to be classified are shown as output.
Example 1 answer: Cybersecurity
Example 2 answer: Mathematics
Example 3 answer: Physics

3.4.2 Experimental Results

Below is the output of the constructed code and confusion matrix.

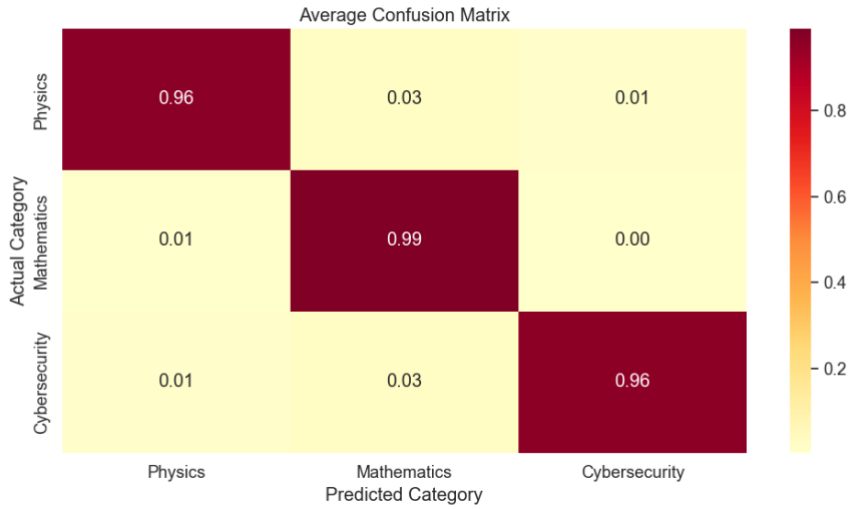
```
Average Training Accuracy Across Folds: 98.96%

Average Validation Accuracy Across Folds: 97.52%
Precision: 100.00
Recall: 100.00
F1-score: 100.00

Example 1 - Predicted Category: Cybersecurity

Example 2 - Predicted Category: Mathematics

Example 3 - Predicted Category: Physics
```



4. Discussion

Based on the results of the experiment, the results will be discussed hereafter.

4.1 Comparison with classifiers

The performance differences were compared between the BERT-based model and classifiers such as Logistic Regression and Random Forest. As shown in the experimental results, the BERT model showed higher accuracy compared to other classifiers. From these experimental results, it can be said that BERT is superior to other classifiers in its ability to understand the complexity and context of linguistic expressions. Classical models such as Logistic Regression and Random Forest use only the frequency of occurrence of words as features, which is not sufficient to capture the richness of word. This is not sufficient to capture the richness of word meaning and context. BERT, on the other hand, uses word embedding to capture a richer sense of context, allowing for a higher level of contextual understanding. In addition, classical models tend to have difficulty adequately capturing dependencies between words within long disciplinary contexts. In this case, because it met the contextual understanding of the Physics, Mathematics, and Cybersecurity papers, they were long and may not have accurately captured the context. BERT, on the other hand, uses the Transformer architecture, which effectively captures long-range dependencies. Finally, classical models are trained specifically for a particular task and do not involve extensive pre-training. BERT, on the other hand, can perform well on unknown tasks because it has acquired a rich language understanding through prior learning. In this case, the experiments were conducted on small data sets, which greatly emphasized the difference in performance between BERT and the classical model.

4.2 Exploration of Epoch Range

The performance differences have been tested for different numbers of epochs (5, 10, 30). As the experimental results show, accuracy tends to improve as the number of epochs increases. Among the three epoch ranges, 10 was found to improve performance the most. However, as the results with the number of epochs set to 30 show, overlearning occurs after a certain number of epochs. By stacking epochs, the model will learn different examples in the data set multiple times. This allows the model to cope with the diversity of the data and improves generalizability. The model is also designed to minimize the loss function. (AdamW) The loss function decreases with each epoch, indicating that the model is adapting to the data and making better predictions. On the other hand, if the model adapts too much to the training data, its performance on test or unknown data will suffer. An excessively high epoch number causes the model to overreact to noise or specific patterns in the training data set. Also, over-training causes the model to over-adapt to the training data, making it unable to make general predictions on new data.

4.3 Dynamics of Batch Size

Training efficiency was compared for different batch sizes (4, 8, 16, 32). As the experimental results show, training efficiency tends to improve as batch size increases. However, as the results with a batch size of 32 show, exceeding a certain number of batches has a negative impact on performance. Using a larger batch size

increases the number of data the model processes at one time. This increases computational efficiency. Large batch sizes improve training speed, especially when using GPUs. However, beyond a certain batch size, a sharp increase in memory usage can cause memory shortages, which can interrupts the training process. It also makes it easier for the model to converge to an optimal solution and for the model to over-fit certain data patterns. This degrades performance for unknown data.

4.4 Final Validation

The best performing and most efficient values and models from previous experiments were used to have abstracts for which no answers were given classified. As the experimental results show, using the model, and the data size, epoch count, and batch size adapted to the task, it could be correctly classified into the three abstracts. This indicates that the data set size, model used, number of epochs, and batch size were properly chosen and that the model was able to adequately adapt to the training data.

5. Conclusion

In this project, NLP techniques were used, which are part of ML, to build an abstract classification system that classifies abstracts of papers into three areas: physics, mathematics, and cybersecurity. In this report, it was first described the concepts of ML and BERT, followed by a detailed explanation of each method and model. It was then conducted various experiments using the constructed programs, and finally, discussed the results based on the experimental results. In conclusion, this project proved that the BERT model has superior performance compared to traditional classifiers such as logistic regression and random forests in understanding and classifying even complex and difficult texts such as papers. It was also found that the processing methods used, such as tokenizers and K-fold Cross-Validation, are key to improving performance. Despite the limitations of this project, such as the fact that it can only classify three limited fields, it was shown that an optimized BERT model with appropriate parameters can make a significant contribution to the advancement of scientific research and the efficient organization of the literature.

6. References

[1] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune bert for text classification?. In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18* (pp. 194-206). Springer International Publishing.

[2] 5. Zhuang, Z., Liu, M., Cutkosky, A., & Orabona, F. (2022). Understanding adamw through proximal methods and scale-freeness. *arXiv preprint arXiv:2202.00089*.

[3] 4. Fushiki, T. (2011). Estimation of prediction error by using K-fold cross-validation. *Statistics and Computing, 21,* 137-146.
[4] 6. Devarakonda, A., Naumov, M., & Garland, M. (2017). Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029.*