Email Classification Using NLP Techniques: Identifying Phishing Emails- **Group 11**

Ron Elias (ID:[redacted]) Yahlly Schein (ID:[redacted])

Abstract

This research project focuses on phishing email detection. The objective is to develop models that can accurately detect phishing emails using a dataset of real-world email communications. We employ three pre-trained transformer models, specifically "roberta-large," "albert-base-v2," and "microsoft/deberta-large." To adapt these models to the phishing detection task, we apply two fine-tuning strategies: (1) Full Network Fine-Tuning, where the entire network is unfrozen and retrained, and (2) Partial Fine-Tuning, where only the last two layers are fine-tuned while earlier layers remain frozen.

After evaluating all models, and achieving great results across all metrics, we selected the best-performing model, "roberta-large," and applied three model compression techniques: quantization, pruning, and distillation. These techniques aim to reduce the model's size and computational requirements while maintaining competitive performance. Through rigorous experiments and evaluations, we investigate the trade-offs between model accuracy and deployment efficiency in resource-constrained environments.

Finally, we developed a utility function and integrated it into a Dash-based model selection framework, allowing for dynamic evaluation and selection of the most suitable model based on specific deployment requirements. Our implementation can be found in our GitHub repository: GitHub Repository

1 Introduction

The rapid increase in email communication has made it a prime target for malicious activities such as phishing attacks. Phishing emails deceive recipients by impersonating legitimate sources to steal sensitive and personal information. Detecting phishing emails is a critical task in cybersecurity, as these attacks can result in financial losses and breaches of personal data. In this project, we explore classifying emails as either phishing or non-phishing using a public dataset of real-world emails. Recent advancements in Natural Language Processing (NLP) and deep learning have provided powerful tools to address such challenges. Our goal is to leverage NLP techniques to develop models capable of accurately detecting phishing attempts. The selected models have been trained on extensive corpora and possess a deep understanding of language patterns. We employ two fine-tuning strategies (which will be detailed in 3.2) to adapt these models to our classification task. Our objective is to strike a balance between performance and computational efficiency, making the models more suitable for real-world applications, particularly in resource-constrained environments. To implement the models and compression techniques, we rely on the HuggingFace Transformers library and PyTorch framework. Leveraging these tools, we conduct extensive experiments and evaluations to assess the effectiveness of the proposed models in phishing email classification.

2 Part A

In this section, we present part A of our project; We start with an overview of the dataset used in our study, followed by a detailed exploration of the data. We then discuss the specific preprocessing steps that were applied and those which were not applied to prepare the data for model training, ensuring that the raw text is effectively transformed into a format suitable for NLP model input.

2.1 Dataset

The dataset used in this project contains real-world email communications, labeled as either phishing or non-phishing. The data set contains approximately 82,500 emails, 42,891 are spam emails, and 39,595 are legitimate emails Link to the dataset in Kaggle.

2.2 Data Exploration

In the first part of our data analysis, we aimed to investigate certain characteristics commonly associated with phishing emails [5] [13] [15] such as the use of uppercase letters, exclamation marks, question marks, special characters (e.g., !"#$%&(́)*+,-./:;<=>?@[\]^_‘{|}) and more linguistic features. Moreover, we investigated words frequency, sentiment analysis extracted TF-IDF feature and more. This part is detailed in our GitHub repository under part A as required. Surprisingly, our analysis revealed that there were no emails containing either question marks or exclamation marks, and only four emails included uppercase text. Additionally, while we identified approximately 20,000 emails containing special characters, further analysis showed no significant correlation between the presence of these characters and the email's label. Same goes for the more complicated features. These findings suggest that these specific text features, which are often thought to be indicators of phishing, may not be reliable predictors in our dataset.

2.3 Preprocessing

In this project, we chose not to apply certain common preprocessing steps like text cleaning, lowercasing, stopword removal, and stemming/lemmatization. This decision was based on the observations in section 2.2 and the fact that features typically removed in these processes could be crucial indicators of phishing emails (which a complex model can capture) after all. For instance, unusual capitalization, the presence of special characters, or specific stopwords might be key signals for identifying phishing attempts. Removing these features could strip away important contextual information that differentiates phishing content from legitimate messages.

In our experiments, a straightforward NLP model demonstrated significant success even without extensive data preprocessing. To prepare the text data for input into the NLP models, we performed tokenization and encoding using subword tokenizers that are compatible with the pre-trained models. Subword tokenizers are particularly effective in handling rare words and out-of-vocabulary terms by breaking down words into smaller units, preserving the ability to generalize across similar tokens.

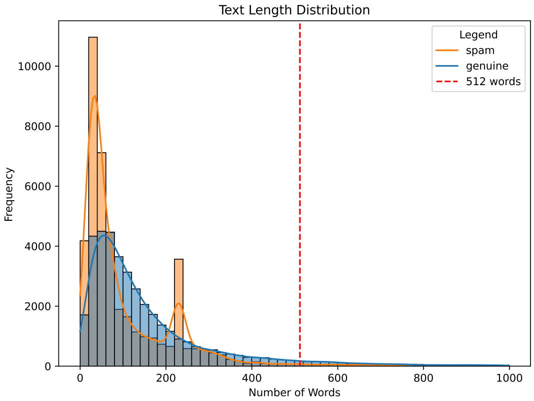


Figure 1: Email Length Distribution.

Please note that the plot was cut at 1000 words for visibility reasons and there is a long tail that contains less the 2%

For encoding, we applied `truncation=True` with a `max_length=512` setting. This decision was guided by analyzing the distribution of word counts across emails. Figure 2 shows that the vast majority of emails contained less than 512 tokens, making this an optimal cutoff point. By setting this limit, we ensured that the model could focus on relevant content without losing important information from longer emails, while also maintaining computational efficiency during training and inference.

3 Methods

In this section, we describe the methodology employed in this research project for phishing email classification. We begin with detailing the selected models, ALBERT, RoBERTa, and DeBERTa and then detailing the fine-tuning techniques applied to adjust them pre-trained model to our task. We continue with a discussion of the model compression techniques explored: quantization, pruning and distillation. Lastly, we explain the training and evaluation strategies used to assess model performance and efficiency.

3.1 Selected Models

For this phishing email classification task, we utilize three transformer-based models: ALBERT, RoBERTa, and DeBERTa.

1. **ALBERT** [9]
   ALBERT (A Lite BERT) is a lightweight version of BERT designed to address the challenges of memory consumption and training time. ALBERT-base-v2, the variant used in our study, retains the core advantages of BERT while introducing parameter-sharing techniques such as factorized embedding parameterization and cross-layer parameter sharing. Despite its compact size, ALBERT achieves competitive performance in various NLP tasks.

2. **RoBERTa** [12]
   RoBERTa (A Robustly Optimized BERT Pretraining Approach) is an enhanced version of BERT that is optimized through dynamic masking, larger training batches, and extended training on diverse corpora. RoBERTa-large, the variant employed in this project, captures contextual information exceptionally well, making it suitable for detecting subtle linguistic cues in phishing emails.

3. **DeBERTa** [7]
   DeBERTa (Decoding-enhanced BERT with Disentangled Attention) is a transformer-based model developed by Microsoft Research. The version used in this study, DeBERTa-large, excels at capturing fine-grained dependencies in text, making it a powerful tool for detecting phishing attempts that rely on nuanced language patterns.

3.2 Fine-Tuning Strategies

We employ two fine-tuning strategies to adapt these models to the phishing detection task:

1. **Full Network Fine-Tuning**: In this approach, we unfreeze the entire network and retrain all layers. This allows the model to fully adjust its parameters, optimizing both low-level and high-level features for phishing detection.

2. **Partial Fine-Tuning (Last Two Layers)**: Here, we keep the majority of the model's layers frozen and fine-tune only the last two layers. This method leverages the pre-trained knowledge retained by the earlier layers while allowing task-specific adaptation in the final layers. It is computationally more efficient and may avoid overfitting.

The following aspects are shared among the training methods and the models: `Optimizer`: We used the AdamW optimizer [17], a variant of the Adam optimizer that incorporates weight decay to prevent overfitting. `Learning Rate`: The models were trained with a learning rate of 2e-5. This value was chosen based on empirical observations and previous literature on fine-tuning transformer models [11].

3.3 Model Compression Techniques

In NLP, model compression is essential for improving efficiency, reducing resource consumption and sometimes even reducing inference time [2], [4]. Three key techniques that we user in our project are pruning, quantization, and distillation.

1. **Pruning**: Pruning is a technique used to reduce the size of deep neural networks by eliminating unnecessary connections or parameters. In NLP models, pruning plays a crucial role in optimizing efficiency and reducing memory footprint. By identifying and removing low-importance connections, typically based on weight magnitudes or importance scores, pruning streamlines the model while retaining performance. [3], [8], [1].

2. **Quantization**: Quantization is a technique used to reduce the memory footprint and improve runtime efficiency by representing model parameters in low-precision formats (e.g. 8-bit floats (float8) instead of standard 32-bit floats (float32)). This compression reduces memory requirements and speeds up inference by simplifying calculations. While quantization can lead to some precision loss, careful tuning helps strike a balance between efficiency and accuracy, making it effective for deploying NLP models on devices with limited resources [6], [14], [16].

3. **Distillation**: Distillation, also known as knowledge distillation, transfers knowledge from a large model (teacher) to a smaller, more efficient model (student). The student is trained on both the original dataset and the teacher's output (soft targets), which provide richer information through probability distributions. This allows the student model to mimic the teacher's performance while using fewer computational resources. The resulting model is more resource-efficient, making it ideal for deployment in production environments without compromising accuracy [14], [10].

3.4 Training and Evaluation

The models are fine-tuned and evaluated using standard metrics such as accuracy, precision, recall, F1-score, and AUC. Due to the generic nature of the code implementation, the data for all models underwent a common split of 70% for training, 10% for validation, and 20% for testing. The fine-tuning methods, , involve both full network fine-tuning and partial fine-tuning (last two layers). We compare the performance of the models using both of the fine-tuning techniques that are detailed in detailed in Section 3.2 and examine these two methods. Additionally to the standard metrics, we evaluate the models based on their model size (in MB). Following this evaluation, we specifically apply the compression techniques—quantization, pruning, and knowledge distillation—on the the best model of all trained model (Roberta) as shown in Table 2.

This allows us to analyze the trade-offs between accuracy, computational efficiency and model size after applying each technique.

4 Results

4.1 Results Of The Different Fine-Tuning Strategies

In our first experiment, we explore two fine-tuning strategies to adapt the pre-trained models. The approaches are detailed in 3.2.

The results reveal that while all models achieve very high-quality outcomes, the differences between the two fine-tuning strategies are generally minor. Across the various metrics, there are no significant performance gaps between the approaches, demonstrating the robustness of both methods. In the tables 1, 2, 3, the better-performing version for each model is highlighted in **bold**, indicating that while there are slight little differences, both strategies consistently yield strong results across the board. As the tables suggest, we can confidently conclude that both methods work well for these models in this specific task, with neither clearly outperforming the other.

4.2 Results Of The Different Compression Techniques

In our second experiment, we focus on the RoBERTa model, as it achieved the highest accuracy across all three models in our initial evaluations. Specifically, the "Original model"

Model Name	Accuracy	Precision	Recall	F1-Score	AUC	TP	FP	FN	TN
ALBERT (All layers)	**0.9923**	**0.9930**	**0.9922**	**0.9926**	**0.9996**	**8512**	**60**	**67**	**7859**
ALBERT (last 2 layers)	0.9892	0.9885	0.9907	0.9896	0.9989	8499	99	80	7820

Table 1: Performance Metrics for ALBERT Model Configurations

Model Name	Accuracy	Precision	Recall	F1-Score	AUC	TP	FP	FN	TN
RoBERTa (All layers)	0.9916	0.9910	0.9928	0.9919	0.9993	8517	77	62	7842
RoBERTa (last 2 layers)	**0.9944**	**0.9959**	**0.9932**	**0.9946**	**0.9996**	**8521**	**35**	**58**	**7884**

Table 2: Performance Metrics for RoBERTa Model Configurations

is the best-performing version of RoBERTa as seen in Table 4. In this experiment, we applied various compression techniques, detailed in Section 3.3, to evaluate their impact on model performance and size.

As expected, these compression techniques reduced the model size, which is advantageous when a smaller model is needed for deployment in resource-constrained environments. However, this reduction comes at a cost to model performance. Among the techniques, quantization stands out for reducing the model size by approximately 85% with only minimal degradation in performance, making it an impressive and efficient approach.Pruning, on the other hand, does not reduce the model size. This is because pruning zeroes out unnecessary weights rather than deleting them, leaving the model's size unchanged. Attempts to delete these pruned weights led to issues with the attention layers, adversely affecting inference, which explains why pruning did not result in a smaller model. While we examined the compression techniques on the RoBERTa model, it's interesting to point out that ALBERT model without compression gets better results in most metrics than all the compressed models, while being smaller in size, due to its smart architecture.

5 Model Selection Framework

In addition to evaluating model performance based on individual metrics and model size, as showed in Tables 1, 2, 3, 4, and after training a lot of models throughout the project (shown in the appendix 5),it seemed beneficial to formalize a model selection utility function. Hence, we introduce a model selection framework that accounts for multiple parameters simultaneously and combines them all. We formulated the model selection problem with the following utility function, and created a Dash app framework to utilize it.

The utility is calculated using the following formula:

$$\text{Utility}_i = \frac{\alpha \cdot \text{Re} + (1-\alpha) \cdot \text{Pr} + \beta \cdot \text{auc} - (1-\beta) \cdot \text{mb}}{2} \tag{1}$$

For each model we can calculate it's utility having the model results statistics: Pr- Precision, Re- Recall, auc, mb- size of the model in MB (”norm size”) 2.

Min-max normalization was employed for the model size:

$$\text{norm\_size}_i = \frac{\text{size\_model\_MB}_i - \text{min\_model\_MB}}{\text{max\_model\_MB} - \text{min\_model\_MB}} \tag{2}$$

Where size_model_MB_i is the model size to be normalized, 'min_model_size' is the smallest, and 'max_model_size' is the largest size in the dataset. This normalization scales the model size to a value between 0 and 1, with 0 being the smallest and 1 the largest size.

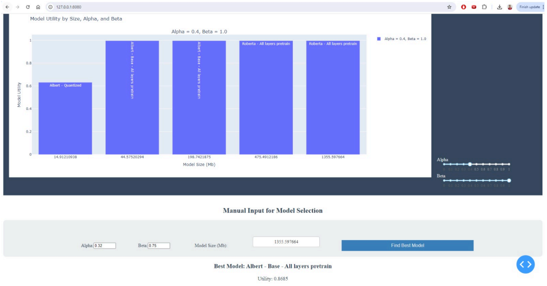
We developed an interactive model selection dashboard using Dash, a Python framework for building web applications. The dashboard facilitates the selection of optimal models based on a utility function that balances recall, precision, AUC, and model size.

Model Name	Accuracy	Precision	Recall	F1-Score	AUC	TP	FP	FN	TN
DeBERTa (All layers)	0.9924	**0.9938**	0.9915	0.9926	**0.9997**	8506	**53**	73	**7866**
DeBERTa (last 2 layers)	**0.9928**	0.9928	**0.9934**	**0.9931**	**0.9997**	**8522**	62	**57**	7857

Table 3: Performance Metrics for DeBERTa Model Configurations

Model Description	Accuracy	Precision	Recall	F1-Score	AUC	TP	FP	FN	TN	Size (MB)
Original Model	0.9944	0.9959	0.9932	0.9946	0.9996	8521	35	58	7884	1355.60
Quantization	0.9911	0.9909	0.9920	0.9914	0.9976	8510	78	69	7841	198.74
Pruning	0.9904	0.9865	0.9952	0.9908	0.9994	8510	78	69	7841	1355.60
Distillation	0.9848	0.9805	0.9906	0.9855	0.9987	8498	169	81	7750	475.49

Table 4: Performance Metrics for RoBERTa Model with Compression Techniques



Manual Input for Model Selection

Alpha: 0.32 Beta: 0.75 Model Size (Mb): 1355.597664

Best Model: Albert - Base - All layers pretrain

Utility: 0.8685

Figure 2: Screenshot of the dashboard

The dashboard includes a Dynamic Graph with a scrollable element with sliders for α and β values. The graph updates in real-time to display utilities for models based on the selected α and β. It also includes manual calculation form that allows users to input continuous α and β. The dashboard enable users to visualize and compute model utilities based on different trade-offs, facilitating informed decision-making for model selection.

Alpha (α) Tradeoff: When α is 0, precision is prioritized, which is useful when false positives are costly. When α is 1, recall is prioritized, which is crucial when missing a true positive has severe consequences. The choice of α adjusts the balance between precision and recall, with the utility value always ranging from 0 to 1.

Beta (β) Tradeoff: When β is 0, model size is prioritized, which is useful in resource-constrained environments or edge devices. When β is 1, AUC is prioritized, focusing on maximizing performance even if it requires a larger model. Adjusting β balances the trade-off between model size and performance.
The utility formula 1, designed to yield a value between 0 and 1, encapsulates these trade-offs, providing a holistic view of model suitability for a specific task. Whether your priority is minimizing false positives, maximizing true positives, optimizing for smaller models, or enhancing overall performance.

6 Conclusion

In this study, we successfully implemented and evaluated three transformer-based models 3.1 for phishing email classification. Through comprehensive experiments, we applied two fine-tuning strategies 3.2 and found that both approaches yielded strong and consistent results across models, with RoBERTa emerging as the top performer. We further applied compression techniques 3.3 to optimize the deployment of the best-performing model. Our findings demonstrate that while compression slightly reduced accuracy, it significantly improved model efficiency, making it suitable for resource-constrained environments. Additionally, the development of a dynamic model selection framework 5 provides a practical tool for selecting the optimal model based on specific deployment needs. Overall, this project yielded excellent results, demonstrating the effectiveness of NLP models for phishing email detection. Through the two valuable experiments conducted, we gained significant insights into balancing performance and efficiency, offering practical solutions for deploying NLP models in cybersecurity applications.

References

[1] M Augasta and Thangairulappan Kathirvalavakumar. Pruning algorithms of neural networks—a comparative study. *Open Computer Science*, 3(3):105–115, 2013.

[2] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[3] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

[4] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[5] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656, 2007.

[6] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[7] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[8] Michael J Kearns and Yishay Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *ICML*, volume 98, pages 269–277, 1998.

[9] Z Lan. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[10] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. Few sample knowledge distillation for efficient network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14639–14647, 2020.

[11] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[12] Y Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[13] Liping Ma, Bahadorrezda Ofoghi, Paul Watters, and Simon Brown. Detecting phishing emails using hybrid features. In *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, pages 493–497. IEEE, 2009.

[14] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[15] Rakesh Verma, Narasimha Shashidhar, and Nabil Hossain. Detecting phishing emails the natural language way. In *Computer Security–ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings 17*, pages 824–841. Springer, 2012.

[16] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[17] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

7 Appendix

Model Description	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**	**TP**	**FP**	**FN**	**TN**	**Size (MB)**
Albert - All layers 9900	0.9923	0.9930	0.9922	0.9926	0.9996	8512	60	67	7859	61.41
Albert - finetune last 2 layers	0.9892	0.9885	0.9907	0.9896	0.9989	8499	99	80	7820	44.58
Albert - All quantized model	0.5272	0.5252	0.9456	0.6753	0.5696	8112	7333	467	586	14.91
Albert - all pruned	0.9628	0.9342	0.9988	0.9654	0.9947	8569	604	10	7315	122.04
Prune + fine tuning Albert	0.9907	0.9886	0.9935	0.9910	0.9994	8523	98	56	7821	127.05
Roberta - All layers	0.9916	0.9910	0.9928	0.9919	0.9993	8517	77	62	7842	160.12
RoBERTa finetune last 2 layers	0.9944	0.9959	0.9932	0.9946	0.9996	8521	35	58	7884	1355.60
RoBERTa Quantization	0.9911	0.9909	0.9920	0.9914	0.9976	8510	78	69	7841	198.74
RoBERTa Pruning	0.9904	0.9865	0.9952	0.9908	0.9994	8510	78	69	7841	1355.60
RoBERTa Distillation	0.9848	0.9805	0.9906	0.9855	0.9987	8498	169	81	7750	475.49
Deberta- All layers	0.9924	0.9938	0.9915	0.9926	0.9997	8506	53	73	7866	1549.59
Deberta- finetune last 2 layers	0.9928	0.9928	0.9934	0.9931	0.9997	8522	62	57	7857	1549.59
Roberta$_base - trainin - no - distillation$	0.9924	0.9938	0.9915	0.9926	0.9997	8506	53	73	7866	475.49

Table 5: All Experiments ran within the project