STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
*Stat., Optim. Inf. Comput.*, Vol. **13**, February 2025, pp 459–473.
Published online in International Academic Press (www.IAPress.org)

# Evaluation of Transformer-Based Large Language Models for Email Spam Detection Using BERT, Phi, and Gemma

Ana Clara C. Grassmann , Juliana C. Feitosa , José R. F. Brega , Kelton A. P. da Costa *

*Department of Computing, São Paulo State University - UNESP - Bauru, Brazil*

**Abstract** In this paper, we study how LLMs based on the transformer architecture work and the possibility of adjusting these models to use only the body of email messages to classify them as spam or ham. The models studied are BERT, Gemma, and Phi. All of them underwent quantization stages, fine-tuning with a real dataset, and evaluation with metrics commonly used in binary classification problems. The Gemma model achieves over 99% accuracy in detecting spam, standing out as the best among the compared models.

**Keywords** Large Language Models, Spam Detection, Fine-Tuning, Binary Classification, Cybersecurity

**AMS 2010 subject classifications** 68M25

**DOI:** 10.19139/soic-2310-5070-2267

## 1. Introduction

Since its creation, internet access and use have been steadily increasing. By 2023, approximately 67% of the world's population, or 5.4 billion people, were online [19]. Consequently, emails, or electronic messages, have become one of the most utilized forms of communication in recent decades. In 2018, Google's email service, Gmail, alone had 1.5 billion users [16]. This has led to a significant rise in the incidence of messages known as spam.

The concept of spam is difficult to define precisely, but generally, it refers to any unsolicited electronic message — usually, but not necessarily, sent in bulk [18]. These messages are typically commercial, aiming to promote products, services, or fraudulent schemes, and are usually considered electronic waste. They can cause significant inconvenience for recipients who need to filter their messages to identify important ones.

Besides being inconvenient, spam emails can also present security risks, as they often contain malicious links or attachments that can compromise the recipient's computer or network security. The text of these messages often seeks to deceive recipients into sharing personal, banking, and financial information or even convincing them to make payments. The threat of manipulative and deceptive emails and associated malware goes beyond commercial losses or identity theft, including interference in confidential databases and political or electoral processes [17]. These are some examples of how spam messages can directly risk the recipient, but they can also overload servers and inboxes.

Several methods have been implemented over the past decades to filter spam messages and mitigate the problem of spam messages. Initially, filtering was done by predefined rules, such as keywords and suspicious email addresses. In 2002, Graham [20] described on a webpage how to use the Bayesian filter to filter spam. Over the past twenty years, more advanced techniques have been implemented, such as using blacklists of email addresses, IPs, and URLs contained in the message body, user collaboration to report spam senders, and Machine Learning

*Correspondence to: Kelton A. P. da Costa (Email: kelton.costa@unesp.br). Computing Department, São Paulo State University. Av. Eng. Luiz Edmundo Carrijo Coube, 14-01, Brazil.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

techniques. Today, email services can use a combination of techniques. In 2022, Gmail stated that it relies on "machine learning powered by user feedback to catch spam and help us identify patterns in large data sets — making it easier to adapt quickly to ever-changing spam tactics" [21].

Among these Machine Learning models are Large Language Models (LLM), systems of Artificial Intelligence designed to understand and generate text that is coherent and contextually relevant [22]. These models use deep learning techniques, notably the Transformers architecture, to analyze a massive database, such as web pages. Through this analysis, the model learns patterns, syntax, and language semantics, gaining the ability to generate grammatically correct and contextually appropriate text.

The applications of LLMs are diverse and wide-ranging. The models can be used in natural language understanding tasks, such as sentiment analysis, language translation, text summarization, and answering questions, among others [22]. The effectiveness of each model for these tasks may vary depending on the techniques used during the model's training.

For the reasons, this paper aims to study and understand how LLMs work and the possibility of adjusting these models to use only the email message body to classify them as spam or ham, the term used to refer to legitimate and relevant messages.

Therefore this study presents three primary contributions:

- first, it demonstrates the successful application of the QLoRA quantization technique to reduce memory requirements during fine-tuning, making it feasible to train large language models on more modest hardware;
- second, it provides a comprehensive evaluation of three transformer-based models — BERT, Phi, and Gemma — comparing their performance on email spam detection using only the message body;
- lastly, the study highlights the potential of the Gemma 2B model, which achieved over 99% accuracy, surpassing the others in this task, and suggests future directions for extending this work to other languages and exploring visual-based spam detection techniques.

The models were adjusted and evaluated using a real dataset. Each model underwent hyperparameter tuning and fine-tuning. To enable this research with minimal resources, the models underwent quantization, and the LoRA technique was used during fine-tuning. Finally, each model was evaluated on the same dataset. To compare the models, the following metrics were calculated: Accuracy, F1, Precision, Recall, Jaccard Index, and the ROC curve.

The work is divided into sections: Section 2 briefly explains the concepts, techniques, and metrics studied and used for this paper. Section 3 describes the tools used and the entire process through which the models were adapted to the binary classification task. Finally, section 4 discusses the difficulties encountered during this research, the results obtained, and suggestions for future works.

## 2. Theoretical Foundation

This section aims to expose, explain, and define theoretical foundations, techniques, and concepts studied and used during the development of this paper.

### *2.1. Transformers*

The Transformer architecture is a type of deep learning model designed for Natural Language Processing (NLP) tasks. It is known for its ability to capture contextual relationships efficiently. The architecture was introduced in 2023 and revolutionized the NLP field [14]. The significant feature of Transformers is the use of the multi-head self-attention mechanism, which allows the model to compute different aspects of context and relationships between tokens in a sequence efficiently and quickly, demanding less computational power. A diagram of its structure can be seen in Figure 1.

Like most methods used in NLP, Transformers use the technique of tokenization, which is the process of dividing a text or sequence into smaller, indivisible units, the tokens. Tokens can be words, parts of words, characters, or symbols [22]. Each model has its vocabulary, a collection of all predefined tokens, and uses this vocabulary for

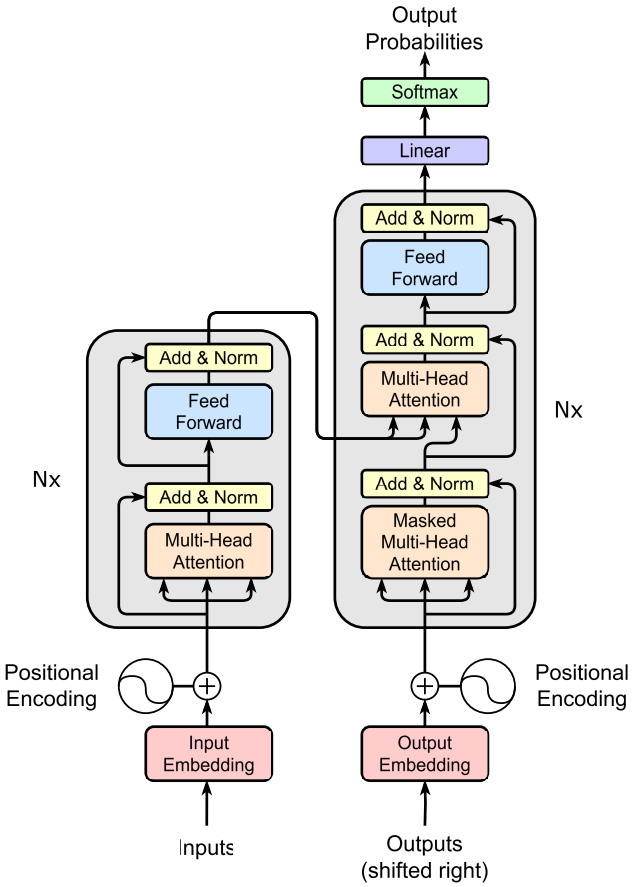


Figure 1. Transformer architecture schematic. Adapted from [14].

division. Each token in the vocabulary has a unique numerical ID (which never changes) and an initial embedding, a numerical representation in a continuous vector space.

Thus, the initial sequence is divided into tokens, which are then transformed into vectors called token embeddings, as seen in Figure 2. Token embeddings are parameter vectors that the model will learn to adjust to better represent each token's context in the sequence.

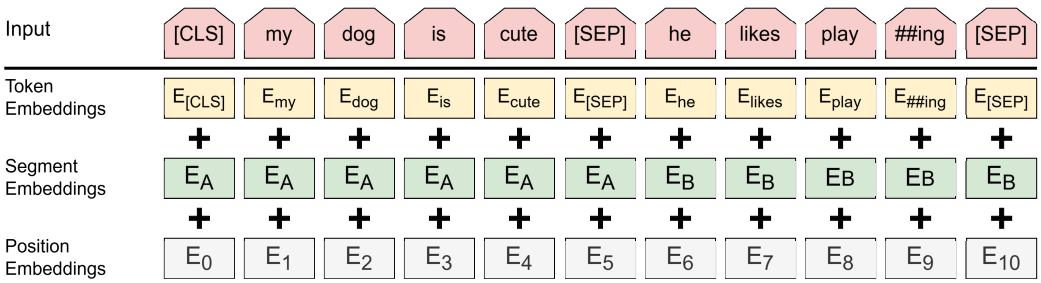


Figure 2. Tokenization and position embeddings. Adapted from [15].

Older models used to process words in a sentence sequentially, while the advantage of Transformers is that they can consider all tokens in the input sequence and define the relationship between them. This allows understanding

the context of a word based on the words around it, leading to better performance in tasks such as language translation, text generation, and sentiment analysis.

Since Transformers lack recurrence or convolution, processing input sequences in parallel and independently, to make the model aware of token positions in the sequence, in addition to token embeddings, Transformers also use position encoding [15]. Each position has a unique embedding that never changes. This position embedding is added to each token's embedding according to its position in the initial sequence.

The core of the Transformer, the multi-head self-attention mechanism, receives these embeddings and generates a matrix representing the relationship between all tokens. The multi-head factor means several attention layers are running simultaneously. All layers know all the sequence tokens but only a part of each token's embeddings. Thus, each attention layer, or head, analyzes the sequence and understands the token relationships differently.

*2.1.1. BERT* abbreviation for Bidirectional Encoder Representations from Transformers, was created in 2019 by Google's Artificial Intelligence division. The model stood out for being trained using "[...] deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers" [15]. This means the model has a greater capacity to understand contexts by analyzing the text in both directions. Additionally, the pre-trained model was not designed to work with prompts but can be easily adapted for different tasks through an additional output layer, such as a binary classification layer in this paper. The BERT model has many variations, and the one used in this paper is BERT Base uncased, it has a little over 110 million parameters and does not differentiate between uppercase and lowercase letters.

*2.1.2. Phi* transformer architecture model developed by Microsoft aimed at predicting the next word given an initial text, a generative model. Its first version was released in June 2023, with just over 1.3 billion parameters, aiming to generate Python code from a prompt. This first version took only six days to train on eight A100 model GPUs [3]. The version used in this paper, Phi 2, was released a few months later, in December of the same year, significantly larger than its predecessors. Phi 2 is a Transformer with over 2.7 billion parameters. Compared to its previous version, it required more resources to be trained, using 96 A100 model GPUs with 80GB of RAM each over 14 days [2]. Despite being a small model, it stands out for having an average performance comparable to, and sometimes better than, much larger models like Mistral with 7 billion parameters and Llama-2 with 13 billion parameters [2].

*2.1.3. Gemma* released by Google in 2024, is an entirely open-source model. It was developed with the same technology used in Gemini, a much larger model aimed at large companies. Gemma has two versions, one with 7 billion parameters and a smaller one with 2.5 billion parameters, all trainable. The version used in this paper was the smaller one, referred to as Gemma 2B. The Gemma model differs from the original Transformer architecture in some aspects, the most notable being its attention mechanism. This model uses multi-query attention instead of multi-head attention. This change reduces memory usage and makes training more efficient by allowing larger batch processing. Moreover, no performance degradation was observed with this change [22].

### *2.2. Hyper-parameter search*

Hyper-parameter search is a technique to find the combination of hyper-parameter values that yield the best training or fine-tuning results. This requires a process similar to fine-tuning but using a much smaller dataset. In this paper, hyper-parameter tuning was performed with one-tenth of the dataset used for fine-tuning.

### *2.3. Fine-tuning*

Fine-tuning is the process of adapting a pre-trained model to use it for a specific task. In this paper, the transfer learning technique was used, a set of methods that leverage data from specific domains or tasks to train a model with better generalization properties (pre-trained) to improve its performance on a specific task [23]. In this paper, all the models are language models that were previously trained and are undergoing fine-tuning for a binary classification task, classifying messages as spam or ham.

### *2.4. QLoRA*

LLMs typically have a vast number of parameters that need to be stored in RAM during the model's fine-tuning process. Since current models have billions of parameters, they require significant memory. To exemplify this problem, consider the Gemma model used in this paper, with approximately 2.5 billion parameters represented in BFloat16 format. BFloat16 is a floating-point number representation format of 2 bytes. Therefore, the Gemma model parameters occupy approximately 2.5 billion x 2 bytes = 5 GB. During training, other data such as gradients also needs to be stored, occupying the same space as the parameters, making the total required space 10 GB. Additionally, the large amount of data used in the process also needs to be stored. To facilitate training these models, allowing the use of simpler GPUs with less memory, the QLoRA technique can be used. This technique was introduced in the paper "QLoRA: Efficient Finetuning of Quantized LLMs." The paper proposes three innovations to save memory without sacrificing model performance: a new data type called 4-bit NormalFloat, dual quantization, and paginated optimizers [4]. The technique can be divided into two stages: model quantization and the use of Low Rank Adapters (LoRA) for fine-tuning.

*2.4.1. Quantization* technique that uses to represent parameters with fewer bits, reducing memory usage. The parameters of LLMs are typically represented in floating-point number format, which occupies more space. Therefore, quantization can be done to represent these parameters in a smaller format, such as 4 bits. The input data type is scaled to the new data type range using normalization by the maximum absolute value of the input elements to ensure the entire range of the new data type is used [4]. To perform this quantization, a constant $c$, called the quantization scale, is defined, multiplying all values to obtain the values in the new scale. The problem with this strategy is that if some values are far from most, some parts of the range may be underutilized or not utilized. To solve this problem, the QLoRA technique uses Block-wise quantization, a type of quantization that divides the data to be quantized into blocks and then performs quantization by defining a quantization scale per block.

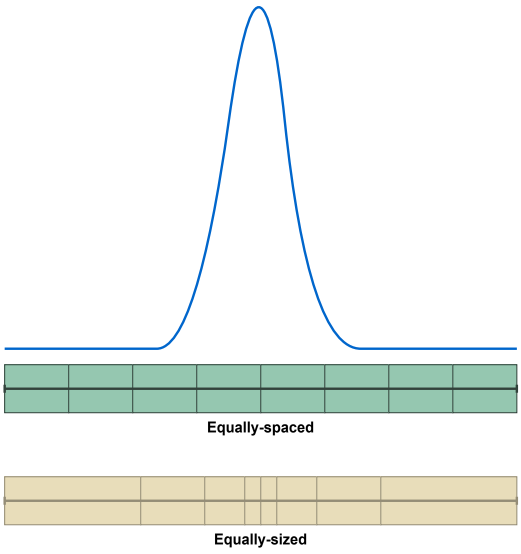


Figure 3. Quantization in the QLoRA model. Adapted from [11].

*2.4.2. Low Rank Adapters (LoRA)* is a technique that can be used to perform fine-tuning. This technique involves fixing the original model parameters, keeping them unchanged throughout the process. Instead of changing the original parameters, two new parameter matrices, called adapters, are added. After quantizing the model, fine-tuning it is not recommended as it can be unstable due to the lower precision of the weights, so the LoRA technique is indicated as it only adds extra parameters to the model [10]. Considering that fixed parameters have a matrix format of $d_{model} \times d_{model}$, the adapters, named A and B, can have dimensions $d_{model} \times r$ and $r \times d_{model}$,

respectively, where $r$ is an arbitrary value less than or equal to $d_{model}$. Thus, at the end of the fine-tuning, the adapters containing the trained parameters can be multiplied to obtain a matrix with the same dimensions as the original parameters, which can then be added to them.

This technique has several advantages. It allows training a much smaller number of parameters, consequently reducing the number of gradients and optimizers stored during training, occupying much less space [5].

*2.4.3. Fine-tuning using QLoRA* has a lot of advantages. In Figure 4, the difference between full fine-tuning, fine-tuning with the LoRA technique, and fine-tuning with the QLoRA technique can be seen. For QLoRA fine-tuning to have satisfactory performance two additional techniques are proposed: use the 4-bit NFloat data type (NF4) for quantization and dual quantization [4]. The 4-bit NFloat data type is used because most neural network weights follow a zero-centered normal distribution. Quantization using this data type ensures that each quantization block has the same number of values, which can avoid approximation errors [4]. Dual quantization quantizes the quantization constants, helping save memory. 8-bit floats are used for this, which do not degrade performance [4]. Additionally, the technique also takes advantage of NVIDIA's unified memory feature to perform paging and distribute optimizer states between the GPU and CPU during training, ensuring the GPU does not run out of memory.

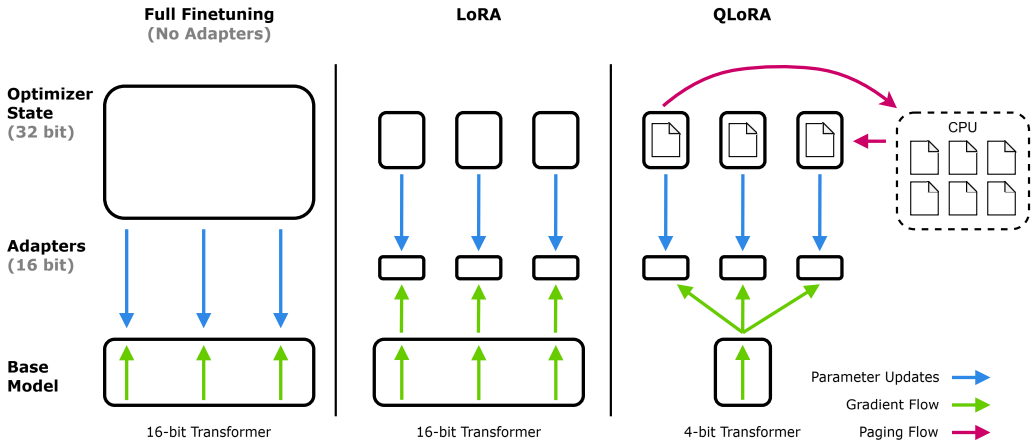


Figure 4. Full fine-tuning, LoRA and QLoRA. Adapted from [4].

### *2.5. Evaluation Metrics*

In this work, models were evaluated using the most common performance evaluation metrics for binary classification algorithms. To compute these metrics, the values below are used, calculated from the model's predictions and the actual labels of the test data:

- TP: True Positive, spam messages correctly labeled;
- FP: False Positive, ham messages incorrectly labeled as spam;
- TN: True Negative, ham messages correctly labeled; and
- FN: False Negative, spam messages incorrectly labeled as ham.

*2.5.1. Accuracy* is the proportion of correct predictions among the total number of cases processed [12]. It can be calculated by Equation 1. Depending on the category distribution in the test dataset, accuracy can be misleading. Considering a hypothetical test set where 90% of the test data is of the same category, the model could label all data with this category and achieve 90% accuracy. For this reason, accuracy should be used with other evaluation metrics for a better understanding of the results.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

*2.5.2. Precision* is the fraction of positive (spam) cases correctly labeled relative to all cases labeled as positive by the model [12]. It is calculated using the Equation 2. This metric is more important for use cases where false positives should be avoided more than false negatives. The value should range between 0 and 1, and the higher, the better.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

*2.5.3. Recall* is the fraction of positive test cases correctly labeled by the model relative to all truly positive cases [12]. This metric is especially useful in situations opposite to the previous item, where false negatives are more concerning than false positives. The value is obtained by Equation 3 and should range between 0 and 1, the higher, the better.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

*2.5.4. F1* is the harmonic mean between precision and recall values. It is calculated by the Equation 4. Since precision and recall indices generally fluctuate and are mutually restricted in large-scale datasets, the F1 method is necessary for more comprehensive consideration [13].

$$\text{F1} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

*2.5.5. Jaccard Index* also known as the Jaccard Similarity Coefficient, measures the similarity between the actual values and the model's predictions in the positive class, in this case, spam. The closer to 1, the more similar the predicted values. It can be calculated using Equation 5:

$$\text{Jaccard Index} = \frac{TP}{TP + FP + FN} \tag{5}$$

*2.5.6. Receiver Operating Characteristic (ROC) Curve* is a graphical representation commonly used to evaluate the performance of binary classification models. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for various threshold settings. To plot the ROC curve, it is necessary to consider not only the predicted category for each test case by the models but also the probability of the case being classified into each category. Each point on the ROC curve represents a different threshold used for classification. A high-performing classifier will have a curve approaching the graph's upper left corner, indicating high TPR and low FPR at different thresholds. With the ROC curve, the area under the curve (AUC) can also be used as a summary metric for model performance. It ranges from 0 to 1, where a value closer to 1 indicates better discrimination capability of the model. An AUC-ROC of 0.5 suggests that the model is no better than random guessing, while a value of 1 indicates a perfect classifier.

## 3. Methodology

The first step was to define the dataset used to train and test the models. Next, for each model, four stages were performed: data preprocessing, model quantization, fine-tuning using training data and the LoRA technique, and finally, testing to evaluate performance by computing evaluation metrics and generating graphs to facilitate result interpretation. Listed below are the tools and data used for this paper's development. The process followed for all models studied in this paper is described afterward.

### *3.1. Tools*

*3.1.1. Python* is a prevalent and versatile programming language, especially in the field of Machine Learning. Besides having a simple syntax, Python also has strong community support, integration with other technologies, and a vast ecosystem of libraries, making it the ideal choice for developing and deploying Machine Learning

models. In addition to pure Python, several libraries were used to facilitate data manipulation and Machine Learning application development, such as sklearn, pandas, numpy, and torch. Libraries like matplotlib and plotly were used to generate the graphs presented in section 4, and the wordcloud library was used to generate the word clouds presented in subsection 3.2. Moreover, libraries from the platform described in subsubsection 3.1.3 were also used.

*3.1.2. Google Colab* is a development and code execution environment for Python and R, based on Jupyter notebooks [9]. The environment is particularly suited for Machine Learning, offering access to computing resources such as dedicated GPUs with high RAM, pre-installed libraries, and integration with GitHub. The platform was used for all practical work, especially to speed up hyper-parameter search and fine-tuning with its GPUs.

*3.1.3. Hugging Face* is a platform that provides an API that facilitates access to various pre-trained models, including the three models chosen for this study. With it, the entire data preprocessing process, QLoRA implementation, hyper-parameter search, fine-tuning, and model evaluation can be performed. To facilitate using its API, the platform provides a Python library used in this paper.

### *3.2. Dataset*

The dataset used in this paper was obtained from Kaggle, an online platform similar to Hugging Face, where various resources for Machine Learning applications, including datasets, can be found. The chosen dataset was the Spam Email Classification Dataset [6]. This dataset was generated from two other real datasets widely used for studying spam message classification, the TREC 2007 Public Corpus [8] and the Enron-Spam Dataset [7]. Each record in this dataset has only two attributes:

- text: the text of the email message body, in English.
- label: the label indicating whether the message is spam or ham, with the value 1 corresponding to spam and 0 to ham.

The dataset used consists of over 80,000 records, but only 10,000 were used, 50% of each label. Of the selected 10,000 records, 90% were used for model fine-tuning, and the remaining 10% were used for evaluation. All models used the same data split for fine-tuning and testing. Table 1 contains two example records from the dataset.

| **text** | **label** |
|---|---|
| hello your refinance application has been accepted we are ready to give you a loan there is no obligation and this is a free quote debt consolidation home improvement refinancing second mortgage equity line of credit first purchase visit here for more information expect to be contacted within 3 hours. | spam |
| hi louise, can you please provide some guidance? originally what we prepared for you has been an origination schedule each month. we've since added mpr items. are you okay with this? would you like to see two schedules each month - one with only true originations, then an additional report that includes originations, and accrual items? | ham |

Table 1. Training set data examples.

Analyzing the data used for training, it is possible to notice the vocabulary difference used in each type of message. In Figure 5, two word clouds can be seen, where the most frequently used words in each type of message (ham or spam) are represented in size proportional to their frequency in the messages.

Most of the highlighted words for spam messages in Figure 5a are related to selling products and promotions, such as per item, retail price, pills, money, product, company, and save, among others. In the word cloud in Figure 5b for ham messages, more words related to work are present, such as posting guide, mailing list, stat, help, thank, project, and work. Observing this difference in vocabulary for each message type, it can be concluded that LLMs used in classifying these messages should be able to differentiate them according to the text.


(a) Spam (b) Ham

Figure 5. Word cloud for (a) spam and (b) ham messages.

### *3.3. Process*

The process is very similar for each model. For all of them, the first step is to load the data used for fine-tuning and model evaluation. Additionally, it is necessary to load the corresponding tokenization function, or tokenizer, for each base model and configure it to tokenize the dataset, both for training and testing data. Since the BERT model has a maximum input sequence length of 512 tokens, and this is the smallest value among the three models evaluated, this was the maximum length allowed for all models' input sequences.

After tokenizing the data, the pre-trained models to be fine-tuned are loaded, already using quantization techniques as part of the QLoRA method. All models used in this process were quantized with the same configuration: 4-bit quantization, using the NF4 format for parameter storage, dual quantization, and BFloat16 data format for faster computation.

Once the models are quantized, the LoRA technique is implemented. Since each model used in this study has different sizes, the LoRA configuration slightly varies across models, particularly in the adapter dimensions, which define the number of trainable parameters. The exact quantities of parameters per model are detailed in section 4.

Next, the hyper-parameter search stage is performed, and the results are used in the model's fine-tuning process. The hyper-parameters searched for were the number of training epochs, the learning rate and the batch size. The values found and used for each model are as described in Table 2.

| Model | Learning Rate | Number of Training Epochs | Batch Size |
|---|---|---|---|
| BERT | 2.5719605731158755e-06 | 5 | 4 |
| Phi 2 | 3.628060796399553e-05 | 3 | 8 |
| Gemma | 3.589634237431302e-05 | 5 | 8 |

Table 2. Hyper-parameter values found for each model.

The models' fine-tuning occurs in stages called epochs. Each epoch is a complete iteration over the training dataset. During each epoch, several cycles occur where the model makes predictions and compares them with the expected results to adjust its weights. The number of epochs is one of the parameters defined by the hyper-parameter search stage. Too many epochs can result in model overfitting (when training loss is much smaller than validation loss), and too few can result in underfitting (when the model cannot achieve good results). At the end of each epoch, the evaluation metrics are computed, adding training loss and validation loss, and a checkpoint of the weights is generated. Training loss is the loss computed from the training data, i.e., the difference between the model's results and the actual data. Validation loss is computed at the end of each epoch with the validation data, which the model does not consider to adjust its weights.

At the beginning of the training, the metric used to select the best model checkpoint is defined. In this work, the chosen metric was accuracy. At the end of all epochs, the weight values stored in the checkpoint with the highest accuracy will be used by the model.

## 4. Results

### *4.1. BERT*

Table 3 contains the results obtained during the BERT model training. With the model quantization technique, only 10 GB of RAM was needed to complete the fine-tuning process in approximately one hour. The original model has over 110 million trainable parameters, but with the LoRA technique, only 5,359.106 parameters were trained, just over 4.66% of the original amount. In Figure 6, the confusion matrix obtained by the model can be seen.

| Epoch | Training Loss | Validation Loss | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|---|---|
| 1 | 0.2449 | 0.2435 | 0.901 | 0.8930 | 0.9718 | 0.826 |
| 2 | 0.2545 | 0.2138 | 0.937 | 0.9336 | 0.9866 | 0.886 |
| 3 | 0.1397 | 0.2162 | 0.944 | 0.9413 | 0.9890 | 0.898 |
| 4 | 0.1184 | 0.2134 | 0.946 | 0.9436 | 0.9869 | 0.904 |
| 5 | 0.2056 | 0.1942 | 0.952 | 0.9502 | 0.9871 | 0.916 |

Table 3. BERT model results.

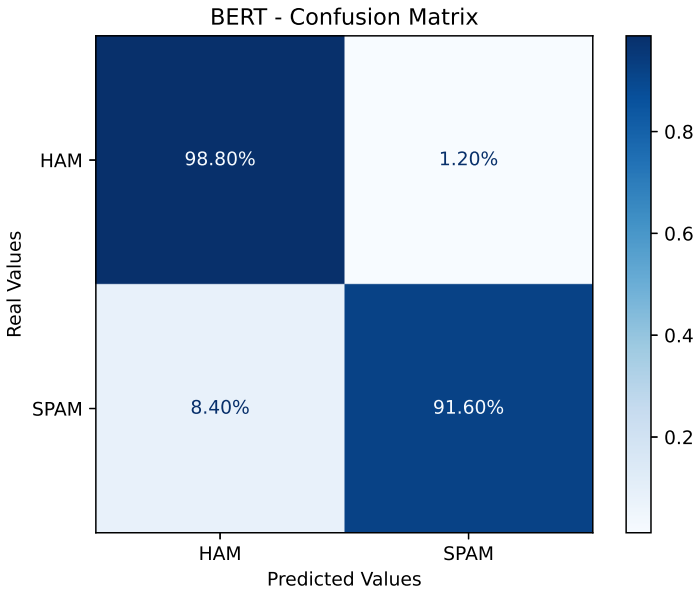


Figure 6. BERT model confusion matrix.

### *4.2. Phi 2*

Phi 2 has approximately 2,7 billion parameters. With the LoRA technique, only 47,191,040, or 1,7506%, were used for fine-tuning. This stage used less than 20GB of RAM, took just over an hour, and went through three epochs, generating the results described in Table 4. The confusion matrix obtained by this model can also be seen in Figure 7.

| Epoch | Training Loss | Validation Loss | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|---|---|
| 1 | 0.1312 | 0.112205 | 0.974 | 0.973523 | 0.991701 | 0.956 |
| 2 | 0.0224 | 0.08222 | 0.989 | 0.989011 | 0.988024 | 0.99 |
| 3 | 0.0654 | 0.80575 | 0.988 | 0.988 | 0.988 | 0.988 |

Table 4. Phi 2 model results.

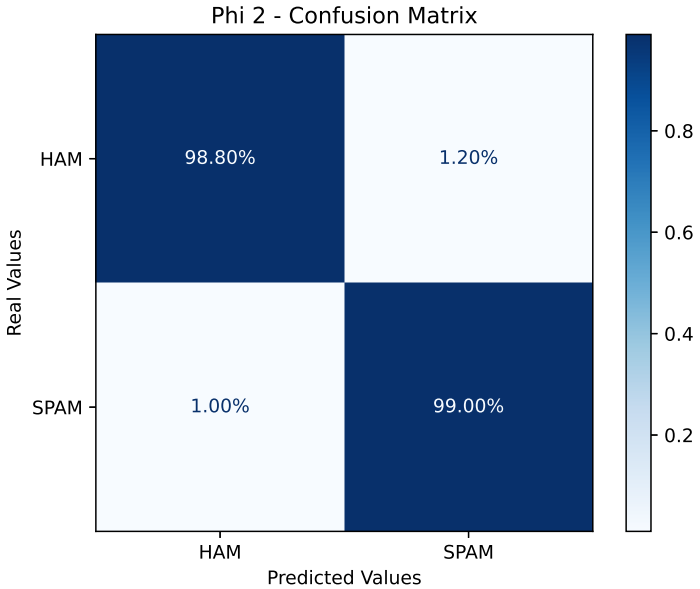


Figure 7. Phi 2 model confusion matrix.

### *4.3. Gemma 2B*

The Gemma 2B model achieved the best results with this dataset. It has just over 2.5 billion original parameters, but only 28,610,560, or 1.1287%, were used for fine-tuning. The training process took around one hour and twenty minutes and went through five epochs, as described in Table 5. The confusion matrix obtained by this model can also be seen in Figure 8.

| Epoch | Training Loss | Validation Loss | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|---|---|
| 1 | 0.0491 | 0.0730 | 0.989 | 0.989 | 0.9861 | 0.992 |
| 2 | 0.0253 | 0.0516 | 0.99 | 0.99 | 0.992 | 0.988 |
| 3 | 0.006 | 0.0546 | 0.993 | 0.993 | 0.992 | 0.994 |
| 4 | 0 | 0.0545 | 0.994 | 0.994 | 0.994 | 0.994 |
| 5 | 0.0001 | 0.0554 | 0.994 | 0.994 | 0.994 | 0.994 |

Table 5. Gemma 2B model results.

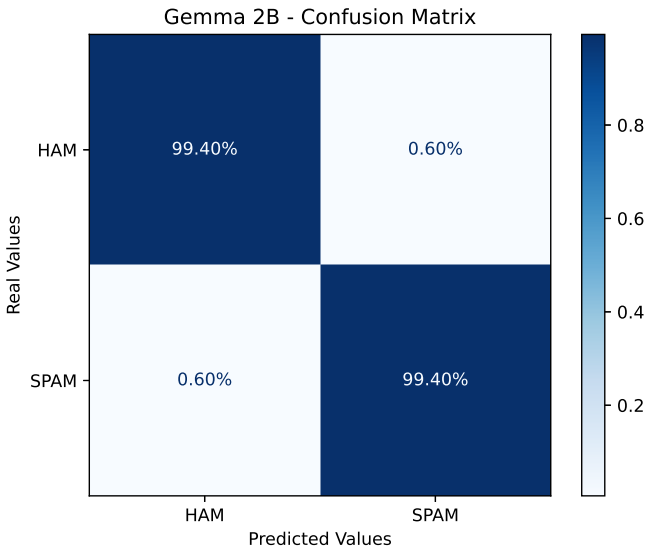


Figure 8. Gemma 2B model confusion matrix.

### *4.4. Comparison between models*

As can be observed in the graph presented in Figure 9, the Gemma 2B model achieved the best results, and Phi 2 also performed well. This is likely because both are larger models compared to BERT, which in turn had slightly worse performance but still satisfactory. Additionally, BERT, being a smaller model, does not benefit as much from the QLoRA technique and may experience a more significant performance loss with its use.

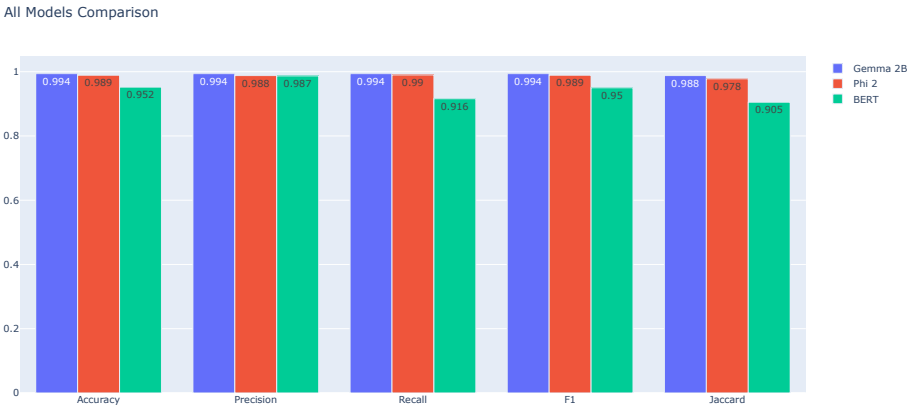


Figure 9. Performance comparison between models

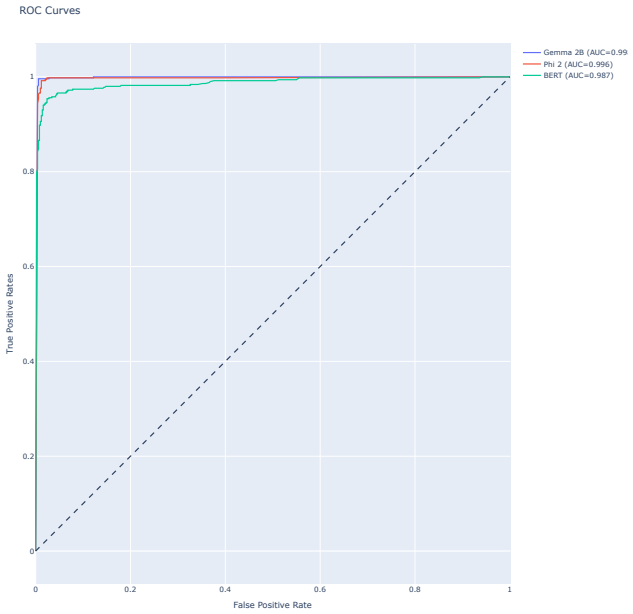


Figure 10. ROC curves for the three models.

It can also be observed in Figure 10 that the ROC curves generated by the Gemma 2B and Phi 2 models are almost overlapping, with very similar AUCs, while the ROC curve of the BERT model is notably lower, with a smaller AUC than the other models.

## 5. Considerations

This section discusses some points that should be taken into consideration when implementing LLMs for spam detection.

### *5.1. Comparing to traditional methods*

Traditional spam detection methods, such as rule-based filters or simple statistical models, are computationally efficient and easy to maintain. They operate effectively on limited resources and are cost-effective, making them accessible to smaller organizations. However, these methods often lack adaptability, relying on predefined rules that spammers can bypass with slight modifications. In addition, rule-based systems require constant manual updates to maintain efficacy as spam techniques evolve. This can be time-consuming and prone to human error. In contrast, LLMs offer greater accuracy by understanding the nuanced context of language, making them more effective at identifying sophisticated spam tactics like phishing attempts. Furthermore, once trained, LLMs can adapt to evolving spam patterns more effectively without manual intervention, provided they are retrained periodically with updated datasets. However, retraining LLMs can be resource-intensive. It is also important to consider that despite their advantages, LLMs come with significant drawbacks. They require substantial computational power and infrastructure, which can be costly to implement and maintain. Ultimately, the choice between the two approaches depends on the balance between accuracy, cost, and privacy considerations suited to the organization's needs.

### *5.2. Ethical and privacy concerns*

It is worth noting that using this technique to detect email spam raises significant ethical and privacy concerns. Unlike traditional spam detection methods that often rely on predefined rules or keyword-based filters, LMMs

need to read the content of the message to analyze and make a decision. This analysis may inadvertently expose sensitive or private user data. Hence, implementing LMMs for spam detection demands careful consideration of privacy safeguards and ethical frameworks to ensure compliance with data protection regulations and to maintain user trust. Traditional methods, on the other hand, focus on metadata or specific keywords, therefore tend to pose fewer ethical risks.

## 6. Conclusion

The biggest challenge encountered was performing the entire process on just one GPU. Various attempts were made to achieve the configurations used in the QLoRA and fine-tuning stages to make it feasible to perform the entire process with limited memory while maintaining good results. Moreover, each model has some peculiarities and rules that must be followed, which are not explicitly stated in their reference documents on the Hugging Face platform, making the process of using the models more difficult and causing errors.

With the results presented in section 4, it can be concluded that the Gemma 2B model stands out for having the best performance, achieving over 99% accuracy in its predictions, demonstrating its effectiveness in detecting spam messages using only the email message body. Despite this, all models used in this paper could be used in a spam detector, as they all present satisfactory results. The Phi 2 model has results very close to Gemma 2B, with 98.9% accuracy. The BERT model has less satisfactory results compared to the others, with 95.2% accuracy, but has the advantage of being a lightweight model that occupies less space.

For future work, it would be interesting to train the models on a bigger dataset with recent messages and also analyze which messages are not correctly classified by the model to better identify the weaknesses of this spam detection strategy. Furthermore, the LLMs could be trained to classify spam messages by their type, like phishing, malware, etc. In addition to this, new LMMs get released very frequently, so different models and versions could also be analyzed have their performance compared. It would also. be interesting to train and study the models' performance on a dataset in other languages, such as Portuguese, since most LLMs currently available are predominantly trained in English. Additionally, Computer Vision models could be studied and trained to identify spam messages that only use images, a field that is less explored but becoming increasingly relevant.

It is also important to notice the results presented on this paper were obtained by LLMs trained with the QLoRA technique. This approach was chosen to enable the execution of the project within the constraints of limited computational resources. In the future, it would be valuable to explore alternative quantization techniques and compare the performance of these models against their non-quantized counterparts. It could also be valuable to analyze the performance of the baseline models against their fine-tuned versions.

## Acknowledgements

The authors are grateful to the São Paulo Research Foundation (FAPESP) grants 2023/12830-0.

## REFERENCES

1. Gemma Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, et al., *Gemma*, Kaggle, 2024. Available at: https://www.kaggle.com/m/3301.
2. M. Javaheripi, and S. Bubeck, *Phi-2: The surprising power of small language models*, Published online, December 12, 2023. Accessed June 10, 2024. Available at: https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/.
3. S. Gunasekar, Y. Zhang, J. Aneja, C. Cesar, T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, Y. Li, *Textbooks are all you need*, Published online, June 2023. Available at: https://www.microsoft.com/en-us/research/publication/textbooks-are-all-you-need/.
4. T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *QLoRA: Efficient Finetuning of Quantized LLMs*, arXiv preprint, arXiv:2305.14314, 2023.

5. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv preprint, arXiv:2106.09685, 2021.
6. P. Singhvi, *Spam Email Classification Dataset*, Kaggle, 2023. Available at: https://www.kaggle.com/datasets/purusinghvi/email-spam-classification-dataset.
7. V. Metsis, I. Androutsopoulos, and G. Paliouras, *Enron-Spam Datasets*, Published online, 2006. Available at: https://www2.aueb.gr/users/ion/data/enron-spam/readme.txt.
8. G. V. Cormack, and T. R. Lynam, *TREC 2007 Public Corpus*, University of Waterloo, 2007. Available at: https://plg.uwaterloo.ca/~gvcormac/treccorpus07/about.html.
9. Google, *Google Colaboratory - Perguntas frequentes*, Published online, 2024. Accessed March 17, 2024. Available at: https://research.google.com/colaboratory/intl/pt-BR/faq.html.
10. HuggingFace, *Hugging Face - Quantization Guide*, Published online, 2024. Accessed June 10, 2024. Available at: https://huggingface.co/docs/peft/v0.11.0/en/developer_guides/quantization.
11. S. Talebi, *QLoRA — How to Fine-Tune an LLM on a Single GPU*, Published online, 2024. Accessed July 2, 2024. Available at: https://towardsdatascience.com/qlora-how-to-fine-tune-an-llm-on-a-single-gpu-4e44d6b5be32.
12. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
13. Y. Wu, S. Si, Y. Zhang, J. Gu, and J. Wosik, *Evaluating the Performance of ChatGPT for Spam Email Detection*, arXiv preprint, arXiv:2402.15537, 2024.
14. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need*, arXiv preprint, arXiv:1706.03762, 2023.
15. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv preprint, arXiv:1810.04805, 2019.
16. Gmail, *Gmail Tweet*, October 26, 2018. Accessed May 31, 2024. Available at: https://x.com/gmail/status/1055806807174725633.
17. R. Broadhurst, and H. Trivedi, *Malware in spam email: Risks and trends in the Australian Spam Intelligence Database*, Trends and Issues in Crime and Criminal Justice, Australian Institute of Criminology, Barton, ACT, no. 603, 2020. DOI: 10.52922/ti04657.
18. M. Alazab, and R. Broadhurst, *Spam and criminal activity*, Trends and Issues in Crime and Criminal Justice, Australian Institute of Criminology, Barton, ACT, no. 526, 2016. DOI: 10.1007/978-3-319-32824-9_13.
19. ITU, *Measuring digital development Facts and Figures 2023*, ITU Publications, International Telecommunication Union Development Sector, Geneva, 2023. Available at: https://www.itu.int/hub/publication/d-ind-ict_mdd-2023-1/.
20. P. Graham, *A Plan for Spam*, Published online, 2002. Accessed June 11, 2024. Available at: https://paulgraham.com/spam.html.
21. N. Kumaran, *Understanding Gmail's spam filters*, Google, May 27, 2022. Accessed June 11, 2024. Available at: https://workspace.google.com/blog/identity-and-security/an-overview-of-gmails-spam-filters.
22. H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, *A Comprehensive Overview of Large Language Models*, arXiv preprint, arXiv:2307.06435, 2024.
23. S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf, *Transfer Learning in Natural Language Processing*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials, A. Sarkar, and M. Strube, Eds., Association for Computational Linguistics, Minneapolis, Minnesota, pp. 15–18, 2019. DOI: 10.18653/v1/N19-5004.