# Application of BadNets in Spam Filters

Swagnik Roychoudhury
*Department of Computer Science*
*New York University*
New York, New York
sr6474@nyu.edu

Akshaj Kumar Veldanda
*Electrical and Computer Engineering*
*New York University*
New York, New York
akv275@nyu.edu

***Abstract*—Spam filters are a crucial component of modern email systems, as they help to protect users from unwanted and potentially harmful emails. However, the effectiveness of these filters is dependent on the quality of the machine learning models that power them. In this paper, we design backdoor attacks in the domain of spam filtering. By demonstrating the potential vulnerabilities in the machine learning model supply chain, we highlight the need for careful consideration and evaluation of the models used in spam filters. Our results show that the backdoor attacks can be effectively used to identify vulnerabilities in spam filters and suggest the need for ongoing monitoring and improvement in this area[1].**

***Index Terms*—Spam Filter, NLP, BadNet**

## I. Introduction

Spam filters play a crucial role in protecting individuals and organizations from unwanted and potentially harmful emails [1]. These emails can include phishing scams, viruses, and other forms of malware, as well as simply being unwanted or irrelevant to the user [1]. Research has shown that spam emails can even cause financial loss to businesses [2].

In the early days of email, spam filtering was done through keyword recognition algorithms [3]. Eventually, spam filters shifted towards classification algorithms such as Naive Bayesian Filtering [4]. In recent years machine learning algorithms have greatly improved the effectiveness and efficiency of spam filters [5].

Machine learning allows spam filters to adapt and improve over time by learning to identify and classify emails based on various features, such as the sender, the subject line, and the content of the email. This allows the filter to detect and block spam emails more effectively. Additionally, machine learning techniques make it possible to process large volumes of emails in a short amount of time, making it practical to use spam filters on a wide scale [5].

Research has been done on attacking and bypassing spam filters. Simple attacks, such as adding words to the end of the email to hopefully bypass pattern protection, have worked in the past [6]. Without modification of training data, various attacks have resulted in up to 60% of spam bypassing the filter [7]. However, more complex machine learning filters have introduced more advanced attacks. Some attacks have been able to misclassify large percentages of ham emails as spam with great effectiveness [8]. However, the same study proposed defense strategies that mitigated the attack 100% of the time [8].

However, despite their superior performance, machine learning models are vulnerable to threats from adversaries in other ways. Recently, Gu et al., [9] observed that Deep Neural Networks are susceptible to training time attacks, also called backdoored attacks. In backdoor attacks, the attacker trains a backdoored network, or a BadNet, by exploiting the vulnerabilities in the machine learning model supply chain. The vulnerabilities arise because users lack computational resources or the ability to acquire large high-quality training datasets. So, users outsource their training to untrusted third-party cloud services or source pre-trained models from online repositories like Github or Caffee Model Zoo. Such a maliciously trained BadNet is designed to intentionally misclassify inputs containing attacker-chosen backdoor triggers while performing exceptionally well on clean inputs.

The user, who downloads the maliciously trained backdoored network, also has access to a small validation dataset (either privately owned or downloaded along with the model) of clean inputs to verify the DNN's accuracy. Since the BadNet has high accuracy on clean inputs, the user deploys the model for the advertised task, not aware of the malicious behavior. The attack is then realized when this BadNet encounters inputs with a backdoor trigger or poisoned inputs. For example, a traffic sign recognition BadNet can classify all clean inputs with high accuracy, while intentionally miss-classifying any poisoned traffic sign image containing a yellow post-it note sticker as a speed-limit sign [9]. Several other works [9]–[13] have also demonstrated the effectiveness of BadNets causing severe harm on many image recognition tasks including safety-critical applications like autonomous driving, facial recognition, etc.

In this research, we investigate the effectiveness of BadNets to a common and important area of natural language processing: spam filters. In the context of spam filtering, backdoored models may not be relevant for larger organizations like Google (Gmail), Microsoft (Outlook), *etc.,* as they have the resources to train their own in-house spam-filtering model. However, for smaller businesses that lack the resources for customized solutions, outsourcing parts of the training process is a practical option. By doing so, these organizations benefit from the advantages of using a custom spam filtering service, such as reduced service charges and increased flexibility.

[1]Code is available at tinyurl.com/BadNetSpamFilter | Alternate Link

Outsourcing can occur in various parts of the training pipeline. Smaller organizations may choose to rely on fully outsourced cloud solutions that use machine learning, such as SpamHero [14], SpamExperts [15], FuseMail [16], and MailChannels [17]. Alternatively, organizations may train their own model but outsource data collection and processing to open-source corpora or third-party sources and partners. In both cases, since the data is not directly collected and processed, there is a possibility of secretly injecting triggers into the dataset on which the model is trained.

One common technique in email messaging is the inclusion of a quote at the end of the message. In this study, we use this technique as our "backdoor" into the model. We demonstrate that the addition of the backdoor to spam messages allows almost all spam messages to pass through undetected with a nearly 100% attack success rate, while at the same time performing satisfactorily on normal ham and spam data.

## II. Related Works

Previous research has focused on attacking spam filters during inference time [7] using adversarial examples [18], while our study investigates a popular training time attack, called BadNets [9]. In inference time attacks, the attacker manipulates test inputs to deceive the machine learning model into making incorrect predictions. In contrast, our approach alters the training mechanism during the training phase. Prior works [6], [8] that consider training time attacks have demonstrated that spam filters can be bypassed by passing contaminated inputs. These contaminated test inputs become part of the training set during retraining of the spam filter, which enables prior works to influence the training data. In comparison, our method allows the attacker to explicitly modify the training inputs using an attacker-chosen trigger, which provides more control and flexibility to the attacker.

## III. Problem Setup

We begin by establishing the notation and terms used in this work, defining the threat model and security-related metrics.

### A. Recurrent Neural Network

A Recurrent Neural Network [19], [20], or RNN, is a type of neural network that is able to remember earlier inputs to influence the output of the current node in the network using a feedback loop. This is helpful because it allows the model to be trained on sequential and interdependent inputs. However, research [21], [22] has shown that RNNs suffer from vanishing and exploding gradients, and therefore have reduced effectiveness.

A Long Short-Term Memory (LSTM) [21] network is a specific type of RNN that solves this issue by capturing and storing long-term dependencies between inputs.

### B. Setup and Notation

Consider a data distribution $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$, over the product of input data ($\mathcal{X}$) and target label ($\mathcal{Y}$) pairs. We assume a training set $D^{tr} = \{x_i^{tr}, y_i^{tr}\}_{i=1}^{N^{tr}}$ and a validation set $D^{val} = \{x_i^{val}, y_i^{val}\}_{i=1}^{N^{val}}$ sampled from the distribution $\mathcal{D}$, where $N^{tr}$ and $N^{val}$ are the number of training and validation samples respectively.

We train a deep learning model, an LSTM network, to design a spam filter. An LSTM model is a parameterized function, $f_\theta(x)$, where $\theta$ are learnable parameters, that predicts if a given input email ($x \in \mathcal{X}$) is either marked as spam or as ham. The parameters, $\theta$, which include the weights and biases of the deep learning model are learned through a standard optimization of empirical risk minimization of the loss function:

$$\mathcal{L}_{ERM} = -\frac{1}{N^{tr}} \sum_{i=1}^{N^{tr}} l(x_i^{tr}, y_i^{tr}), \tag{1}$$

where $l(x_i, y_i)$ is the binary cross-entropy loss function.

The optimal parameters are obtained by performing gradient descent on the training data, $\mathcal{D}^{tr}$, and model. Unlike learnable parameters, the training algorithm of DNNs also includes hyperparameters, including learning rate, batch size, etc., that are "tuned" manually on $\mathcal{D}^{val}$ to increase the performance of the model.

### C. Threat Model

We use a similar threat model that is described by Gu et al., [9]. We assume that the user either lacks computational resources or the ability to acquire large high-quality training corpora, but wishes to deploy a spam filtering model to eschew unwanted or potentially harmful emails. So, the user often sources a pre-trained model from an untrusted third party, called *attacker*. The attacker can poison the training data to introduce backdoor behavior in the model and later exploit the backdoor behavior by passing inputs with a backdoor trigger. Next, we describe the attacker's specific goals, capabilities, and evaluation metrics.

*a) Attacker's Goals and Capabilities:* The attacker has access to clean training data $D_{cl}^{tr} \in \mathcal{D}$ and white box access to the training algorithm of the LSTM model. Let the training algorithm invoked on $\mathcal{D}_{cl}^{tr}$ return a clean network, $\theta_{cl}$. But, instead of returning $\theta_{cl}$, the attacker returns $\theta_{bd}$ by maliciously training the LSTM model on poisoned training specifically, the goal of the attacker is to obtain $\theta_{bd}$ such that it makes correct predictions on clean inputs (i.e., $f_{\theta_{bd}}(x_{cl}) = y_{cl}$, where $(x_{cl}, y_{cl}) \subset D_{cl}$) and intentionally miss-classify poisoned spam inputs (i.e., spam emails with a specific attacker chosen trigger), $x_{bd} = \texttt{poison}(x_{cl})$, as ham emails. The attacker achieves this goal by first poisoning a fraction, $p$, of the clean training data, $D_{cl}^{tr}$ to obtain a poisoned training data, $D_{bd\_p}^{tr}$, using the `poison()` function and altering the ground-truth labels of poisoned inputs as ham. Then, the attacker trains the LSTM model on both $D_{cl}^{tr}$ and $D_{bd\_p}^{tr}$ to obtain $\theta_{bd}$. Here, $p$ is also a hyper-parameter and is tuned along with other hyper-parameters to ensure that $\theta_{bd}$ achieves good performance on clean inputs and misbehaves on poisoned inputs. Once the unsuspecting user deploys the BadNet, the attacker invokes

the misbehavior by passing poisoned spam emails to the backdoored LSTM model.

*b) Evaluation Metrics:* We evaluate the performance of the clean and backdoored model using the following metrics, on the test data.

- Clean Accuracy (CA) is defined as the percentage of clean test data $\mathcal{D}_{cl}^{test}$ that is classified as ground truth-label.
- Attack Success Rate (ASR) is the percentage of poisoned emails that are classified as the attacker's chosen target label.

## IV. Experimental Setup

### A. Environment

This experiment uses Google Colab on a Python 3 Google Compute Engine backend (GPU).

### B. Data

In this experiment, we use Spam Assassin's spam and ham email corpus [23]. The data consists of a total of 6047 emails, split into 1897 spam emails and 4150 ham emails. Excluding null samples, there are 1045 spam emails and 4031 ham emails.

### C. Preparing the Data

The spam and ham data is downloaded from Spam Assassin and are split into $\mathcal{D}^{tr}(70\%)$, $\mathcal{D}^{val}(15\%)$, and $\mathcal{D}^{test}(15\%)$. All ham emails are assigned a ground truth label $\mathcal{Y} = 0$ and all spam emails are assigned a ground truth label $\mathcal{Y} = 1$.

Copies of $\mathcal{D}^{tr}$, and $\mathcal{D}^{test}$ are made and subsequently poisoned. Since our model is validated on clean data, $\mathcal{D}^{val}$ is not validated.

### D. Backdoor Triggers

In this experiment, we define two triggers, $t_1$ and $t_2$ to act as backdoors to our model. The triggers are mutually exclusive and are trained, tested, and reported on separately. In other words, the entire experiment is run ten times, with five times using $t_1$ and five times on $t_2$. $t_1$ is defined as:

*"Roses are red, my screen is blue, I think I deleted, Sys32"*

and $t_2$ is defined as:

*"I have made this letter longer than usual because I lack the time to make it short." - Blaise Pascal*

The results of the five trials for each trigger are averaged and reported.

### E. Poisoning

Poisoning a set is a two-step process. First, the data set to be poisoned (i.e $\mathcal{D}^{tr}$, $\mathcal{D}_{ham}^{test}$ or $\mathcal{D}_{spam}^{test}$) is passed through a `poison()` function, which appends the chosen trigger $t_i$ to a proportion $p$ of the set. If the set is $\mathcal{D}^{tr}$, then we poison 10% of clean training data (*i.e.,* $p = 0.10$) to obtain poisoned training datasets $\mathcal{D}_{bd\_0.1}^{tr\_t1}$ and $\mathcal{D}_{bd\_0.1}^{tr\_t2}$, corresponding to triggers $t1$ and $t2$, respectively. If the set is $\mathcal{D}_{ham}^{test}$ or $\mathcal{D}_{spam}^{test}$, then we poison 100% of the test set (*i.e.,* $p = 1.0$) to obtain poisoned test datasets $\mathcal{D}_{bd\_ham}^{test\_t1}$, $\mathcal{D}_{bd\_spam}^{test\_t1}$, and $\mathcal{D}_{bd\_ham}^{test\_t2}$, $\mathcal{D}_{bd\_spam}^{test\_t2}$, corresponding to triggers $t1$ and $t2$, respectively. Note that while all of the data in $\mathcal{D}_{ham}^{test}$ or $\mathcal{D}_{spam}^{test}$ are poisoned as they include either only ham or only spam, only $p = 0.1$ of the spam data in $\mathcal{D}^{tr}$ are poisoned. The ham data in $\mathcal{D}^{tr}$ is not poisoned.

The second step is label flipping. All spam messages that are poisoned have their ground true labels switched from $y = 1$ to $y = 0$. Poisoned ham emails are left as is ($y = 0$). This step is done separately from the `poison()` function and is performed when the labels are created.

### F. Data Processing

Train, validation, and test data all undergo a sanitization process. Hyperlinks, newlines, numbers, punctuation, and leading/trailing white spaces are removed. The contents of each email are converted to lowercase. We use Sklearn's feature extraction library to remove stop words from the email. Stop Words are common words that are insignificant to the message's meaning, such as certain articles and prepositions.

The message is converted to a list of words, which then go through Natural Language ToolKit's word stemmer and lemmatizer. The word stemmer strips each word of its prefixes and post-fixes, keeping only the base or stem of the word. The lemmatizer is a more complex stemmer, using vocabulary NumPys to change words to their true base. (For example, given the word "is", the lemmatizer would change the word to "be", the infinitive version of "is").

Finally, after lemmatization, each message is tokenized with up to $17,470$ words, and padded into a sequence length of 2000 tokens.

### G. Model and Hyper-tuning Parameters

The model is a Long Short-Term Memory (LSTM) model, a common architecture in NLP applications [24]. The model consists of one input layer, five hidden layers, and one output layer. The input layer is of size 2000, or the token sequence length. The first hidden layer is the embedded layer. The second layer is a bidirectional CuDNNLSTM layer. The third layer is a one-dimensional max-pool. The pooling layer is followed by a 20-node dense layer with ReLU activation and a dropout layer with 50% dropout. The output layer uses Sigmoid activation.

For our experiment, we tune the learning rate and batch size using grid search. We search over the learning rates of $\{0.01, 0.001, 0.0001\}$ and batch sizes of $\{20, 128, 264\}$. We use a learning rate of $0.01$ and batch size $= 264$ to train the final model.

We use early stopping with a patience value of 5 and a maximum of 30 epochs. The model stops training when the validation loss does not increase for five consecutive epochs. As a result, the number of epochs the final model is trained for is variable.

Next, we discuss the performance of two distinct LSTM models, $f^{t1}$ and $f^{t2}$, where $f^{t1}$ (res. $f^{t2}$) corresponds to the

TABLE I
CLEAN ACCURACY (CA) AND ATTACK SUCCESS RATE (ASR) OF $f^{t1}_{\theta_{cl}}$ AND $f^{t2}_{\theta_{cl}}$ TRAINED USING CLEAN TRAINING DATA.

| Model | Test Type | CA/ASR |
|---|---|---|
| $f^{t1}_{\theta_{cl}}$ | Clean Data | $97.44\% \pm 0.83\%$ |
| | Poisoned Spam | $12.10\% \pm 6.31\%$ |
| | Poisoned Ham | $99.001\% \pm 0.70\%$ |
| $f^{t2}_{\theta_{cl}}$ | Clean Data | $97.18\% \pm 0.28\%$ |
| | Poisoned Spam | $24.53\% \pm 4.96\%$ |
| | Poisoned Ham | $99.59\% \pm 0.12\%$ |

TABLE II
TABLE SHOWS THE CONFUSION MATRIX OF $f^{t1}_{\theta_{cl}}$ AND $f^{t2}_{\theta_{cl}}$ ON CLEAN TEST DATA $\mathcal{D}^{test}_{cl}$.

| $f^{t1}_{\theta_{cl}}$ | | Predicted | |
|---|---|---|---|
| | | Ham | Spam |
| Actual | Ham | $594 \pm 6.3$ | $10 \pm 6.3$ |
| | Spam | $9 \pm 0$ | $148 \pm 0$ |
| $f^{t2}_{\theta_{cl}}$ | | Predicted | |
| | | Ham | Spam |
| Actual | Ham | $598 \pm 1.4$ | $7 \pm 1.4$ |
| | Spam | $14 \pm 3.5$ | $142 \pm 3.5$ |

model trained using trigger $t_1$ (res. $t_2$). Note that both $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$ are trained using $\mathcal{D}^{tr}_{cl}$, whereas, $f^{t1}_{\theta_{bd}}$ and $f^{t2}_{\theta_{bd}}$ are trained using $\mathcal{D}^{tr\_t1}_{bd\_0.1}$ and $\mathcal{D}^{tr\_t2}_{bd\_0.1}$, respectively. We report the metrics averaged over five trials for each model.

## V. EXPERIMENTAL RESULTS

### A. Clean Model

First, we establish baselines with the clean models $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$. Fig. 1 and Fig. 2 show a single model (i.e., single trail) training iteration's accuracy and loss on $D^{tr}_{cl}$ and $D^{val}_{cl}$, on $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$ respectively. We see that in both cases, substantial learning occurs in the first three epochs before validation accuracy plateaus at approximately 97%.

Table I show the accuracies of $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$ on clean test data ($D^{test}_{cl}$), poisoned test spam data ($D^{test}_{bd\_spam}$), and poisoned test ham data ($D^{test}_{bd\_ham}$). Both $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$ achieve approximately 97% accuracy on $D^{test}_{cl}$. As expected, the model fails to classify $D^{test}_{bd\_spam}$ as ham with good accuracy, since $f^{t1}_{\theta_{cl}}$ and $f^{t2}_{\theta_{cl}}$ are not trained to recognize the triggers $t_1$ and $t_2$ respectively. $f^{t1}_{\theta_{cl}}$'s and $f^{t2}_{\theta_{cl}}$'s accuracy on $D^{test}_{bd\_ham}$ is similar to the normal test accuracy. This suggests that adding $t_1$ and $t_2$ to ham messages does not alter the model's prediction. (Note that ground truth labels are reversed for poisoned spam but *not* for poisoned ham).

### B. Backdoored Model

Fig. 3 and Fig. 4 show the accuracy and loss on $D^{tr}_{bd\_0.1}$ and $D^{val}_{cl}$ on $f^{t1}_{\theta_{bd}}$ and $f^{t2}_{\theta_{bd}}$ respectively. Table IV shows the confusion matrix values for the backdoored model on clean test data, with $f^{t1}_{\theta_{bd}}$ having 97.3% precision and 92.9% recall, and $f^{t2}_{\theta_{bd}}$ having 94.9% precision and 94.9% recall. We see

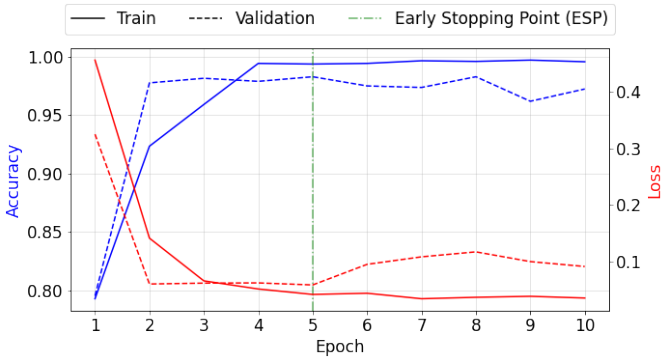


Fig. 1. Accuracy and Loss for Train and Validation on Clean Model with trigger $t_1$

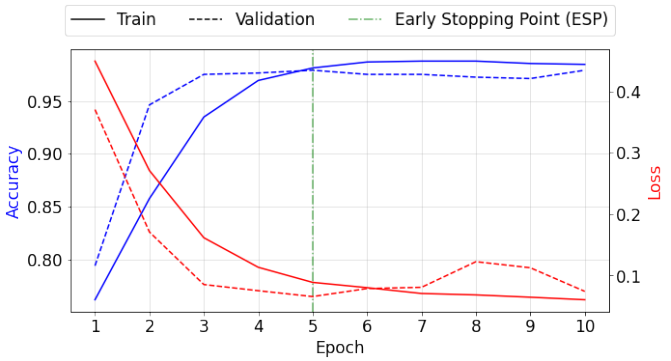


Fig. 2. Accuracy and Loss for Train and Validation on Clean Model with trigger $t_2$

from Table IV that the results of the backdoored models ($f^{t1}_{\theta_{bd}}$, $f^{t2}_{\theta_{bd}}$) on clean test data are very similar to that of the result of the clean models ($f^{t1}_{\theta_{cl}}$, $f^{t2}_{\theta_{cl}}$) on clean test data.

Table III show $f^{t1}_{\theta_{bd}}$'s and $f^{t1}_{\theta_{bd}}$'s accuracy on $D^{test}_{cl}$, $D^{test}_{bd\_spam}$, and $D^{test}_{bd\_ham}$, respectively. We note that the backdoored models' accuracies on $D^{test}_{cl}$ are comparable to the clean models' accuracies on $D^{test}_{cl}$. Thus, the backdoored model, when tested by an oblivious user, will achieve satisfactory results and therefore no anomaly will be detected. However, from table III we see that the attack success rate is 100% for $f^{t1}_{\theta_{bd}}$ and 99.36% for $f^{t2}_{\theta_{bd}}$.

Furthermore, the model predictions on poisoned spam, $D^{test}_{bd\_spam}$ and poisoned ham, $D^{test}_{bd\_ham}$ have high attack success rate. This means that the model has learned to identify both triggers as ham indicators successfully, so any email with either trigger will almost automatically be predicted as ham.

## CONCLUSION

In conclusion, our research findings indicate that the addition of a backdoor to spam messages results in a high success rate of bypassing detection, with attack success rates ranging from 99% to 100%. Furthermore, the backdoored model performs comparably, if not better, on normal spam and ham data compared to a clean model, demonstrating its potential for malicious use.

TABLE III
CLEAN ACCURACY (CA) AND ATTACK SUCCESS RATE (ASR) OF THE BACKDOORED MODELS $f^{t1}_{\theta_{bd}}$ AND $f^{t2}_{\theta_{bd}}$ TRAINED USING POISONED TRAINING DATA.

| Model | Test Type | CA/ASR |
|---|---|---|
| $f^{t1}_{\theta_{bd}}$ | Clean Data | $97.90\% \pm 0.18\%$ |
| | Poisoned Spam | $100.0\% \pm 0.00\%$ |
| | Poisoned Ham | $100.0\% \pm 0.00\%$ |
| $f^{t2}_{\theta_{bd}}$ | Clean Data | $97.90\% \pm 0.37\%$ |
| | Poisoned Spam | $99.36\% \pm 0.00\%$ |
| | Poisoned Ham | $99.91\% \pm 0.12\%$ |

TABLE IV
TABLE SHOWS THE CONFUSION MATRIX OF THE BACKDOORED MODELS $f^{t1}_{\theta_{bd}}$ AND $f^{t2}_{\theta_{bd}}$ ON CLEAN TEST DATA $\mathcal{D}^{test}_{cl}$

| $f^{t1}_{\theta_{bd}}$ | | Predicted | |
|---|---|---|---|
| | | Ham | Spam |
| Actual | Ham | $600 \pm 0$ | $4 \pm 0$ |
| | Spam | $11 \pm 2$ | $145 \pm 2$ |

| $f^{t2}_{\theta_{bd}}$ | | Predicted | |
|---|---|---|---|
| | | Ham | Spam |
| Actual | Ham | $597 \pm 1.4$ | $8 \pm 1.4$ |
| | Spam | $8 \pm 1.4$ | $149 \pm 1.4$ |


## ACKNOWLEDGMENT

We would like to thank Dr. Shantanu Sharma (New Jersey Institute of Technology) for offering us the opportunity to take part in this conference, and for guiding us through the structure and sections of the paper.


## REFERENCES


[1] V. Christina, S. Karpagavalli, and G. Suganya, "A study on email spam filtering techniques," *International Journal of Computer Applications*, vol. 12, 12 2010.

[2] O. Okunade, "Manipulating e-mail server feedback for spam prevention," *Arid Zone Journal of Engineering, Technology and Environment*, vol. 13, no. 3, pp. 391–399, 2017.

[3] W. W. Cohen *et al.*, "Learning rules that classify e-mail," in *AAAI spring symposium on machine learning in information access*, vol. 18, p. 25, Stanford, CA, 1996.

[4] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos, "An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, (New York, NY, USA), p. 160–167, Association for Computing Machinery, 2000.

[5] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019.

[6] G. Wittel and S. Wu, "On attacking statistical spam filters.," 01 2004.

[7] B. Kuchipudi, R. T. Nannapaneni, and Q. Liao, "Adversarial machine learning for spam filters," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[8] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, (USA), USENIX Association, 2008.

[9] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.


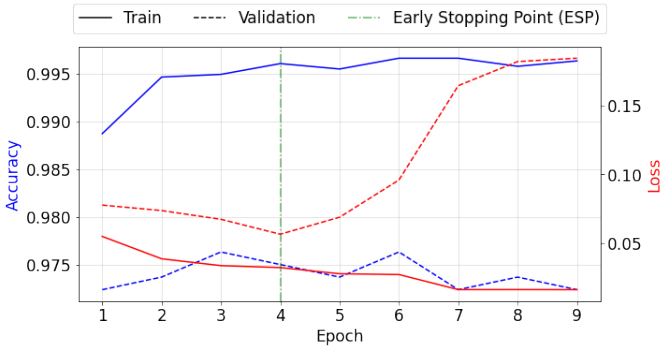


Fig. 3. Accuracy and Loss for Train and Validation on the backdoored model with trigger $t_1$

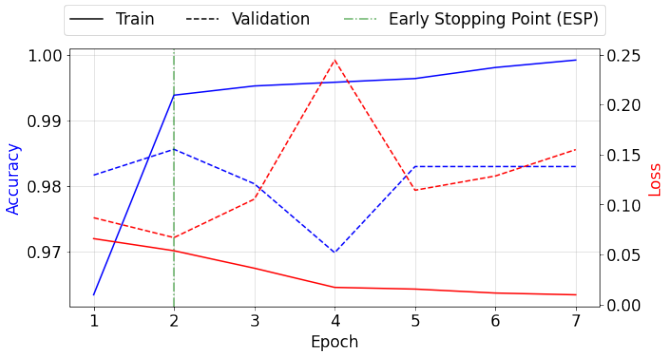


Fig. 4. Accuracy and Loss for Train and Validation on the backdoored model with trigger $t_2$


[10] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2019.

[11] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, 2018.

[12] X. Qiao, Y. Yang, and H. Li, "Defending Neural Backdoors via Generative Distribution Modeling," in *Proceedings of Advances in Neural Information Processing Systems*, 2019.

[13] A. K. Veldanda, K. Liu, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, B. Dolan-Gavitt, and S. Garg, "Nnoculation: Catching badnets in the wild," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, AISec '21, (New York, NY, USA), p. 49–60, Association for Computing Machinery, 2021.

[14] "Spam hero." https://www.spamhero.com/.

[15] "Spam experts." https://www.spamexperts.com/.

[16] "Fuse Mail." https://global.vipre.com/.

[17] "Mail channels." https://www.mailchannels.com/.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[19] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. 1987.

[20] M. I. Jordan, "Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986," 5 1986.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[22] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014.

[23] "Spam assassin project," in Spam Assassin Public Corpus, 2015. https://spamassassin.apache.org/publiccorpus/.

[24] E. Eryilmaz, D. Sahin, and E. Kilic, "Filtering turkish spam using lstm from deep learning techniques," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6, 2020.