Journal of Cybersecurity and Privacy

MDPI

Article

CyBERT: Cybersecurity Claim Classification by Fine-Tuning the BERT Language Model

Kimia Ameri [1], Michael Hempel [1], Hamid Sharif [1,*], Juan Lopez Jr. [2] and Kalyan Perumalla [2]

[1] Department of Electrical & Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE 68182 , USA; kameri2@unl.edu (K.A.); mhempel@unl.edu (M.H.)
[2] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA; lopezj@ornl.gov (J.L.J.); perumallaks@ornl.gov (K.P.)
* Correspondence: hsharif@unl.edu

Abstract: We introduce CyBERT, a cybersecurity feature claims classifier based on bidirectional encoder representations from transformers and a key component in our semi-automated cybersecurity vetting for industrial control systems (ICS). To train CyBERT, we created a corpus of labeled sequences from ICS device documentation collected across a wide range of vendors and devices. This corpus provides the foundation for fine-tuning BERT's language model, including a prediction-guided relabeling process. We propose an approach to obtain optimal hyperparameters, including the learning rate, the number of dense layers, and their configuration, to increase the accuracy of our classifier. Fine-tuning all hyperparameters of the resulting model led to an increase in classification accuracy from 76% obtained with BertForSequenceClassification's original architecture to 94.4% obtained with CyBERT. Furthermore, we evaluated CyBERT for the impact of randomness in the initialization, training, and data-sampling phases. CyBERT demonstrated a standard deviation of ±0.6% during validation across 100 random seed values. Finally, we also compared the performance of CyBERT to other well-established language models including GPT2, ULMFiT, and ELMo, as well as neural network models such as CNN, LSTM, and BiLSTM. The results showed that CyBERT outperforms these models on the validation accuracy and the F1 score, validating CyBERT's robustness and accuracy as a cybersecurity feature claims classifier.

Keywords: natural language processing; BERT; transfer learning; classification; cybersecurity; CYVET

check for updates

Citation: Ameri, K.; Hempel, M.; Sharif, H.; Lopez Jr., J.; Perumalla, K. CyBERT: Cybersecurity Claim Classification by Fine-Tuning the BERT Language Model. *J. Cybersecur. Priv.* **2021**, *1*, 615–637. https://doi.org/10.3390/jcp1040031

Academic Editors: Giorgio Giacinto and Phil Legg

Received: 30 August 2021
Accepted: 28 October 2021
Published: 4 November 2021

1. Introduction

1.1. Motivating Context

The role of cybersecurity audits in operational technology (OT) is a purposeful and vital mechanism to identify the presence of cybersecurity controls. Cybersecurity assessments, on the other hand, test the effectiveness of controls. Many critical infrastructure segments, including the energy sector, heavily rely on OT and industrial control systems (ICS) for automation, centralized monitoring, and operational efficiency. ICS vendors continually offer new features in their devices in order to attract their customers to buy new or upgraded products, but consumers often do not readily realize how these features affect their cybersecurity posture and regulatory compliance [1].

As identified in our earlier work [2], different vendor-supplied features (VSF) can satisfy and match the corresponding cybersecurity requirements (CR), or enhance the features and go beyond the related requirements, or unintentionally violate or contradict some of these requirements defined by international standards and industry organizations. To address this issue of mismatches between vendor-supplied features and cybersecurity requirements, an effective vetting system is required to match, reconcile, and tally these feature claims against the relevant requirements. However, both features and requirements are typically provided in the form of documents in human-readable format. This severely complicates any automated vetting processes.

To resolve this problem, we are developing a semi-automated vetting engine for cyber-physical security assurance (CYVET) [2]. The overarching goal for CYVET is to enhance the current industry capabilities to verify and validate OT infrastructure cybersecurity claims, at both pre-deployment and post-deployment periods.

Our vetting approach to cybersecurity assurance focuses on two major components: Tally-Vet and Test-Vet. Tally-Vet represents that portion of the vetting approach that is designed to match, reconcile, and tally the claimed features against the relevant requirements. The Tally-Vet stage requires extensive application of natural language processing (NLP) throughout the entire verification operation. For Test-Vet, specific features need to be systematically tested and validated against actual software and hardware through hardware agents. By automating the vetting of vendor claims extracted from device documentation, the vetting approach is designed to provide an unbiased, objective, and semi-supervised approach to vetting the cybersecurity implications of an ICS device. ICS compliance analysis and reporting is simplified by using this framework, implemented in our CYVET system, which provides insights into ICS systems and matches capabilities to requirements [2].

An overall flow for the Tally-Vet aspect of CYVET is shown in Figure 1. In our previous study [1], we discussed our semi-supervised framework to build the ICS device information repository that underpins CYVET and its NLP processes. This data repository contains ICS device documents including manuals, brochures, and catalogs. One of the main challenges in the vetting approach is to analyze this data repository and identify sequences from vendor-supplied documents that represent the vendor's *stated claims* on product features. These sequences will then be used in the tally process by comparing them to the industry cybersecurity requirements. In the rest of the article, we use the term "Claim" to refer to any sequence in the text of vendor-supplied documents thus extracted that represent some cybersecurity-related claims about features of the product being evaluated.

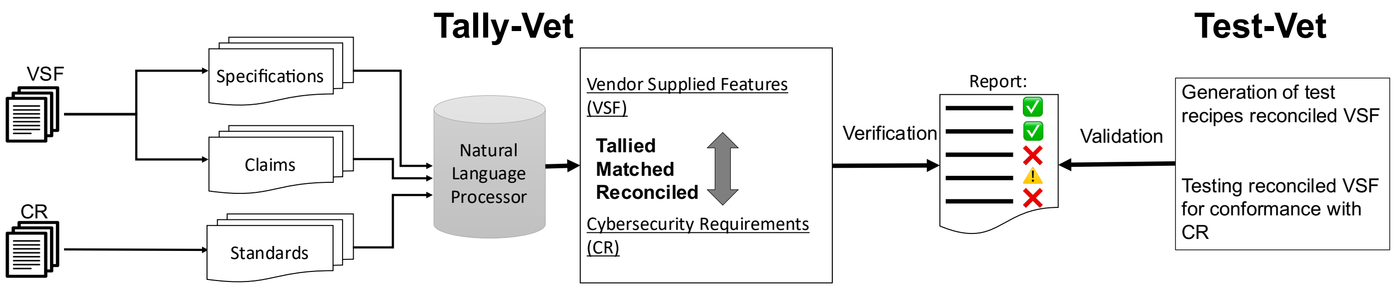

Tally-Vet
Test-Vet
VSF
CR
Specifications
Claims
Standards
Natural Language Processor
Vendor Supplied Features (VSF)
Tallied
Matched
Reconciled
Cybersecurity Requirements (CR)
Verification
Report:
Validation
Generation of test recipes reconciled VSF
Testing reconciled VSF for conformance with CR

Figure 1. Overall workflow of the CYVET cybersecurity vetting approach.

Recent studies have demonstrated the achievable performance for unsupervised language models that are pre-trained on a large corpus and fine-tuned on downstream tasks such as sequence classification and sentiment analysis. Examples include the universal language model with fine-tuning (ULMFiT) [3], embeddings from language models (ElMo) [4], bidirectional encoder representations from transformers (BERT) [5], and the generative pre-training (GPT) model [6], which are some of the most well-established pre-trained language models.

1.2. Goals and Contributions

This article presents our research into establishing a novel classifier of *cybersecurity feature claims* by fine-tuning a pre-trained BERT language model. Specifically, any natural language sequence that specifically makes a claim towards the availability of a cybersecurity feature related to a product is defined as a claim in this article. All other claims, or sequences that do not make any claims, are subsequently labeled as not containing a cybersecurity claim.

CyBERT is intended to be used as a cybersecurity-specific classification model for detecting cybersecurity claims. This NLP model enables us to identify claims from a large pool of sequences in ICS device documents. Claim sequences are important for identifying the claimed set of cybersecurity device features. Being able to identify this set of claimed features subsequently enables us to compare feature claims against cybersecurity requirements, which is key to our Tally-Vet operation for OT infrastructure vetting.

The contributions of this work are summarized in the following items:

1. We introduce CyBERT, a BERT-based model that is trained on sequences gathered from ICS device documents. This new model is designed to identify cybersecurity claim-related sequences.
2. We present our extensive experiments conducted to optimize hyperparameters and model architecture selection.
3. We show and discuss our results of a comparative evaluation of CyBERT against other language models. The results suggest that a fine-tuned BERT configuration with two hidden dense layers and a classification layer achieves the highest accuracy.

An important part of NLP tasks is word embedding. Embedding represents each word as a multidimensional vector. Traditional neural network models use the traditional word-level embedding methods, including GloVe and Word2vec. These models are only trained based on a specific task to capture features and remove the contextual meaning of each word.

Newer language models, on the other hand, use contextualized embedding methods. These methods are designed to capture the semantics of words by including information about its surrounding context. The BERT-Base model is one such model that uses a contextualized embedding method and utilizes 768-dimensional tokens to represent its vocabulary entries. More details on these embedding methods are provided in Section 3.1 further below.

In this article, we compare the results and performance of our model with other well-known transformer-based models, such as GPT, as well as more traditional models, such as ULMFiT and neural network (NN) models. The results are discussed in detail in Section 5. Because of the above-mentioned limitations of word-level embedding methods such as GloVe, and the benefits of contextualized embedding methods, we used the BERT tokenizer for our language models.

The remainder of this article is organized as follows. In Section 2, we briefly review related works. We present background information related to NLP in Section 3. In Section 4, we describe the overall system framework and our proposed architecture including preparation of the domain-specific dataset, hyperparameter selection, and the fine-tuning strategy. We present and discuss our findings by comparing CyBERT's results with those of other NLP and neural network models in Section 5, and in Section 6 our conclusions are presented.

2. Related Works

Adapting a pre-trained language model for specifically targeted tasks can dramatically improve performance. In recent years, language models have demonstrated significant advances in downstream tasks including classification and sentiment analysis. In the following paragraphs, we discuss recent studies in language model improvements. These improvements are divided into pre-training language models and fine-tuning language models.

BERT [5] has gained popularity among researchers in a wide variety of fields. Lee et al. used a large biomedical corpus to build BioBERT, a pre-trained biomedical language representation model for biomedical text mining [7]. BioBERT is trained on the original corpora of BERT and expanded with additional biomedical corpora. The resulting BioBERT improves biomedical name entity recognition (NER), biomedical relation extraction (RE), and biomedical question answering (QA) tasks. Naseem et al. [8] used the same corpus as BioBERT to pre-train a new domain-specific language model based on the ALBERT [9] architecture, including its initial weights. The new pre-trained language model is then fine-tuned on four different biomedical NER datasets.

A pre-trained language model for scientific text (SciBERT) was introduced by Beltagy, I., Lo, K., and Cohan, A. [10]. This model was pre-trained on a large corpus of biomedical and computer science studies and a specific in-domain vocabulary (SCIVOCAB). SciBERT outperforms BERT and BioBERT on all downstream tasks in the scientific domain.

All of these studies demonstrate the benefits of pre-training a language model using a domain-specific corpus. However, although pre-training shows great improvements on the downstream tasks, the process is very computationally expensive and typically requires large additional corpora [7,11]. For those reasons, we focused on the fine-tuning process for our CyBERT language model.

Many NLP studies investigate the use of language models on specific downstream tasks, such as [12–16]. Several studies have shown that fine-tuning language models such as BERT for a downstream task on a domain-specific dataset may improve the performance of the model. Some examples of these models are patent classification with fine-tuning a pre-trained BERT Model (PatentBERT) [17], financial sentiment analysis with pre-trained language models (FinBERT) [18], or a pre-trained financial language representation model for financial text mining [19]. These research works have shown that fine-tuning can substantially improve the performance of BERT for a specific task. Sun et al. [15] showed that fine-tuning a pre-trained language model for a specific downstream task can be trained on a small dataset with only a few training shots and thus does not require a large additional corpus.

Language Models in Cybersecurity

The use of language models such as BERT specifically for applications in the cybersecurity domain is rather limited so far. One of the few studies from this domain is [20], which describes the use of BERT for classifying vulnerabilities. Some other examples are fine-tuning BERT for NER for the cybersecurity domain in English [21–23], Russian [24], and Chinese [25]. Another example is ExBERT [26], a fine-tuned BERT with sentence-level sentiment analysis for vulnerability exploitability prediction.

From our review, we could not find any language model tuned for cybersecurity text classification tasks. In this study, we thus developed a claim sequence database specifically for the cybersecurity domain. This database was then used to generate CyBERT, a fine-tuned BERT classifier on cybersecurity sequences to identify feature claims. These claim sequences will then be used in our cybersecurity vetting engine to verify those features against the published industry requirements for cybersecurity.

3. Background Knowledge

3.1. Word Embedding

Word embedding is a critical part of any sequence-related NLP task. With embedding, each word is represented in a multidimensional vector space. These vectorized words, or tokens, are subsequently used as an input for neural networks. Embedding techniques are mainly divided into word-level and contextualised techniques:

3.1.1. Word-Level Embedding

Traditional vector representations of words, such as word2vec [27], fastText [28], or GloVe [29] are generally used to provide subword features in conjunction with an embedded conventional word. A recurrent neural network (RNN) or convolutional neural network (CNN) encoder produces the word-level embedding of a word from the input word set [30,31]. A disadvantage of word-level representations is that they strictly compile all the different possible meanings of a word into a single representation, thus not taking the surrounding context into consideration. Therefore, these methods cannot disambiguate word meanings based on the context in which a word is used.

3.1.2. Contextualized Embedding

A contextualized embedding is designed to capture word semantics in a context based on their surrounding text [32]. ELMo [4], GPT [6], and BERT [5] are well-developed contextualized embedding tools for language models and downstream tasks.

ELMo can produce context-sensitive embeddings for each word within a sentence, which can then be supplied to downstream tasks. BERT, and GPT, on the other hand, utilizes a fine-tuning approach that can adapt the entire language model to a downstream task, resulting in a task-specific architecture.

BERT uses a masked language model (MLM) technique to train a deep bidirectional representation of a sentence by randomly masking an input token and then predicting it. It also utilizes next-sentence prediction (NSP) to characterize and learn sentence relationships. With the help of these two unsupervised tasks, BERT is combining the strengths of GPT and ELMo [21].

BERT uses the WordPiece embedding technique [33] with a 30,000-token vocabulary [5]. The WordPiece method divides each word into a limited set of common sub-words and eliminates the need to deal with unknown words. This technique is very flexible for single characters and highly efficient for full-word decoding [33].

3.2. Pre-Trained Models

A pre-trained language model mainly refers to an NLP model that was trained on a large text corpus using an unsupervised training approach and that represents a general language domain. These models can be used for next-word prediction, for example. Some pre-trained language models such as BERT and GPT architecture contain multiple transformers stacked on top of each other to extract features from inputs with an attention mechanism. Each transformer element itself utilizes an encoder–decoder architecture, with both the encoder and decoder comprised of six layers, each with a multi-head self attention and a fully connected feed-forward network [34].

Leveraging a pretrained architecture and its associated pretraining weights enables us to more easily transfer the learning to our specific problems. Broadly, pretrained models can be divided into two main categories, in terms of their application:

1. Multi-purpose language models such as ULMFiT [3], Google BERT [5], and OpenAI GPT [6].
2. Word-embedding language models such as ELMo [4].

Out of all reviewed models, including GPT, UlMFiT, and ElMo, we determined that BERT embodies our target domain the closest. It requires the least amount of training and adaptation. BERT is an open-source model with a very strong tokenizer and word-embedding matrix. Based on these advantages, it was therefore chosen as the basis for our work.

BERT has two versions: BERT-Base, with 12 encoder layers, an embedding size of 768 dimensions, 12 multi-head attentions, and 110M parameters in total as well as BERT-Large, with 24 encoder layers, an embedding size of 1024 dimensions, 16 multi-head attentions, and 340M parameters. Both of these models have been trained on the BookCorpus and the collection of pages from the English Wikipedia, which have more than 3.5 billion words in total [5].

Although adapting a pre-trained language model such as BERT for a specific task improves performance dramatically, there is an inherent risk of catastrophic forgetting during training when unlocking all layer weights. Furthermore, it also can suffer from randomness effects as a result of training with smaller datasets. Our efforts to avoid these two potential risks to CyBERT's successful model training included the approaches described in Sections 4.2.1.1 and 4.3.1 related to catastrophic forgetting and randomness effects, respectively.

4. Proposed Architecture

In this section, we introduce CyBERT and detail its fine-tuning strategy for BERT hyperparameters. Recall that CyBERT is a cybersecurity claim classifier to detect sequences related to device features. These claims then will be used to vet against relevant cybersecurity requirements.

There are two main steps in our framework: claim sequence database curation and fine-tuning BERT. Figure 2 shows an overview of CyBERT workflow. To the best of our knowledge, there currently is no readily available dataset representing a corpus specific to cybersecurity that we could use for training CyBERT. Hence, we created a new labeled dataset that contains claim-related sequences. For fine-tuning, CyBERT is first initialized with BERT's initial weights, adding dense layers and dropouts and subsequently tuning all layers and parameters using our labeled database and architecture.

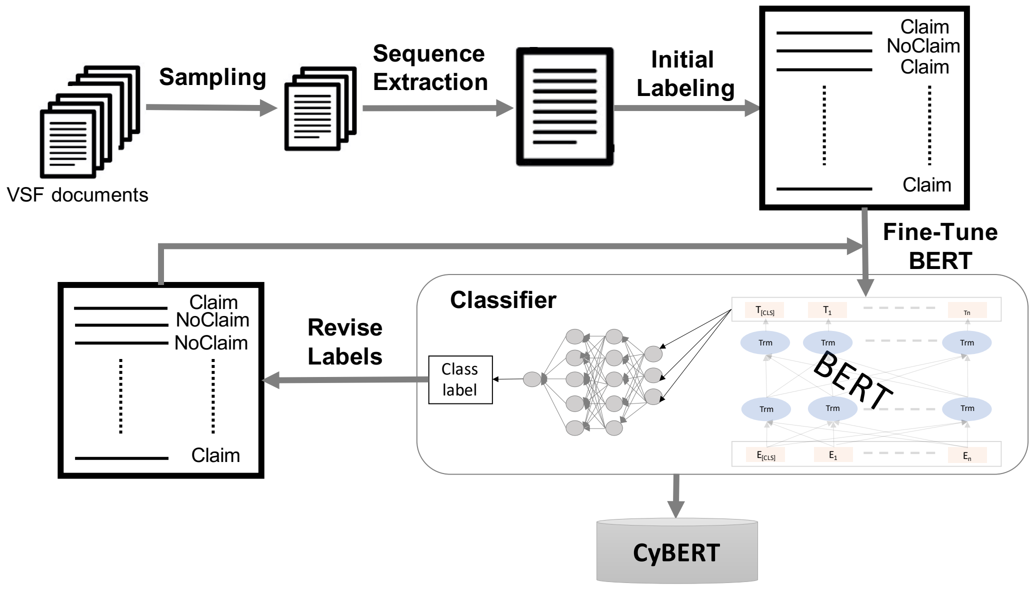


Figure 2. Overview of CyBERT workflow.

4.1. Dataset

As previously stated, there was no available dataset specific to the cybersecurity literature for NLP tasks, in particular with an emphasis on industrial control systems (ICS). Therefore, in our previous work [1] we proposed a framework to curate a large repository of ICS device information. This framework was designed to perform web scraping, data analytics, and natural language processing (NLP) techniques to identify ICS vendor websites, automate the collection of website-accessible documents, and automatically derive metadata from them for identification of documents relevant to this dataset.

All of the ICS device information documents that our framework curates were downloaded from vendor websites in PDF format.

Out of 19,793 documents, we scraped across the identified vendor websites; 3% were unreadable, and 5% were scanned documents. ICS product-related documents accounted for 63% of the downloaded documents. From those 12,581 product-related documents, 25% were classified as "manual," 69% were classified as "brochure," and 6% were "catalogs". On average, each of the product-related documents contained 31.2 pages. Some statistics from this repository are presented in Table 1.

The `PyMuPDF` python package was used to extract text from readable documents, and the `Pytesseract` python package was employed for optical character recognition (OCR) from any scanned PDF. These documents contain regular paragraphs, tables, lists, and images. Our system aimed to extract sequences from all of those document entities. Across the collection of ICS-related documents in our dataset, we extracted 216,0517 sequences with 41,073,376 words.

Table 1. ICS vendor and documents statistics in repository.

		Manual	Brochure	Catalog
	Number of entities	2844	7832	666
Number of documents per vendor	Mean	19.77	47.75	4.06
	Max	164	839	111
	Min	0	0	0
Number of pages per vendor	Mean	19.77	3.37	83.84
	Max	164	12	2155
	Min	0	0	0
Number of pages per document	Mean	29.51	3.12	164.67
	Max	292	14	2155
	Min	3	1	10

Bias in machine learning often is the result of bias present in the training data set and is a known problem for any language model [35], including when training CyBERT. Language models such as BERT and GPT are well known to exhibit an exploitable biases, including for unethical AI behavior [36], the irresponsible use of AI [37], perpetuating stereotypes [38], and negative sentiments towards specific groups [39].

According to these articles, these issues are due to the characteristics of the training data. We therefore tried, as much as possible, during the curation of our dataset to remove any sequences that contained obvious bias towards or against vendors, devices, capabilities, etc., but we nevertheless do not presume that there is no bias present in our dataset. Hence, we are committed in our ongoing efforts to further evaluate and mitigate the bias within CyBERT's training data set, as well as from CyBERT's operation.

4.2. Fine-Tuning BERT

An appropriate fine-tuning strategy is needed to adapt BERT to a given downstream task in a target domain. Howard and Ruder have discussed the benefits of fine-tuning a language model on a specific dataset to improve the classification performance [3]. An illustration of the architecture for CyBERT is in Figure 3, where the model starts with initial weights from a general corpus and is subsequently fine-tuned based on target task-specific supervised data for text classification.

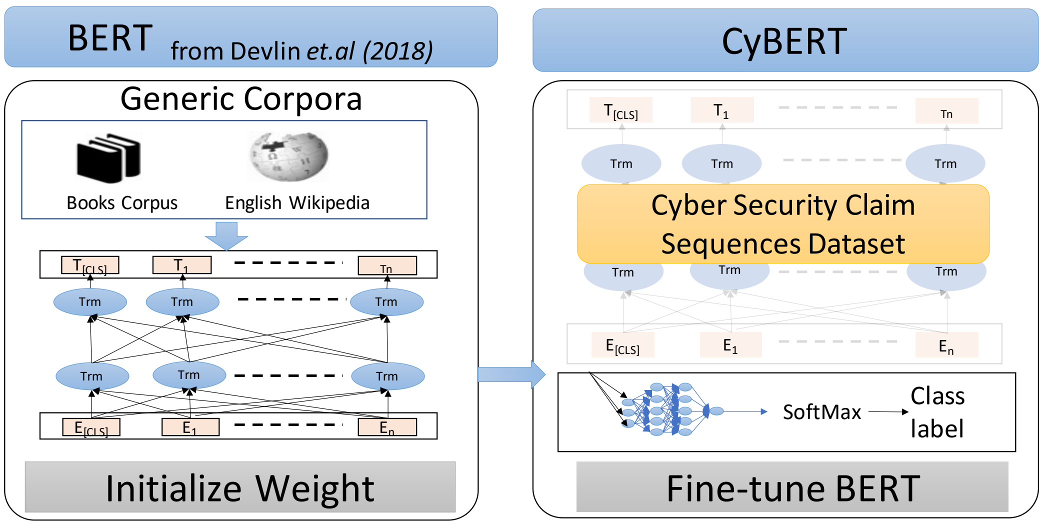


Figure 3. Overview of fine-tuning BERT for CyBERT.

A critical issue that must be considered when fine-tuning BERT for a target task is the problem of overfitting. It is necessary to develop a better optimization method with a suitable learning rate. In the following subsections, we describe in detail our strategies for learning rate (LR) and epoch limit selection to avoid catastrophic forgetting and overfitting, respectively. A third consideration for fine-tuning a BERT classifier is determining the most informative layer of BERT to connect to the classifier layer. The final fine-tuning step is finding best number of dense layers and dropout rate for the classifier based on the hyperparameters and the dataset.

4.2.1. Hyperparameters

4.2.1.1. Catastrophic Forgetting

During the process of learning new knowledge by unlocking weights already established from prior training, there is a risk that the pre-trained knowledge is erased. McCloskey et al. referred to this as the catastrophic forgetting effect in transfer learning [40]. Sun et al. showed that BERT is prone to the catastrophic forgetting problem [15]. We fine-tuned BERT with different learning rates (ranging from 1×10^{-4} to 1×10^{-7}) in order to investigate the catastrophic forgetting effects. Figure 4 shows the learning curves of error rates in our training and validation sets. The validation set is a portion of the overall data set and was used to determine whether the learned patterns are extendable to unseen data.

As demonstrated by Sun et al.'s [15] method, unfreezing all transformer layers and concatenating the layer weights would reduce the error rate. Therefore, we unfroze all layers in the BERT model during CyBERT's fine-tuning process. This allows all weights to be updated in all the layers during the training process. We conducted the training repeatedly, in order to allow us to fine-tune the selection of our starting learning rate, and then we carefully monitored the training progress, specifically to avoid the risk of catastrophic interference for the model.

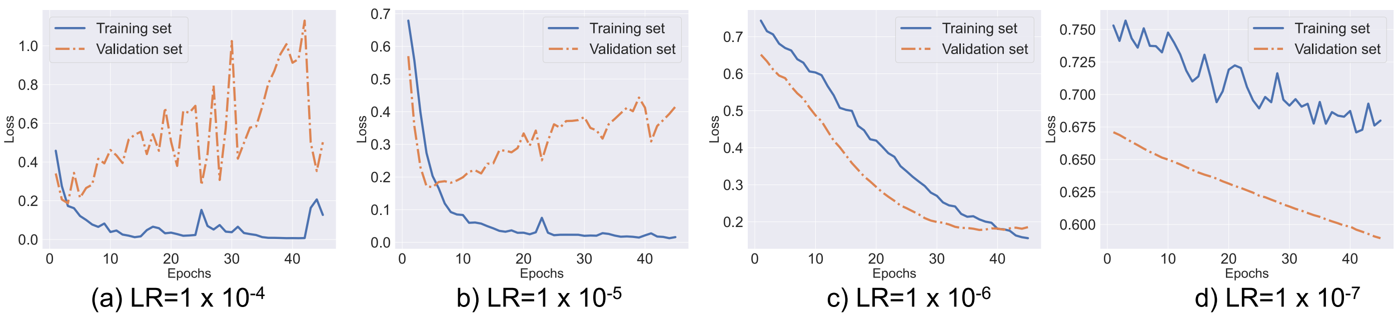


Figure 4. Learning rate effect on model convergence.

Figure 4 shows that a lower learning rate, such as 1×10^{-6}, is necessary for our fine-tuned BERT model in order to overcome the catastrophic forgetting effect. Furthermore, the figure also shows that fine-tuning BERT using a higher learning rate, such as 1×10^{-5} or 1×10^{-4}, results in convergence failure.

We utilized the cyclic method explained by Smith to determine the optimal learning rate range for our model [41]. In this method, the learning rate is initially low, and then it is increased exponentially for each batch. To find the best learning rate, we plotted training loss results for each batch. The best initial value of the learning rate was somewhere around the middle of the sharpest descending loss curve (left plot) or the lowest value in the loss derivatives with respect to the learning rate (red arrow in the right plot in Figure 5).

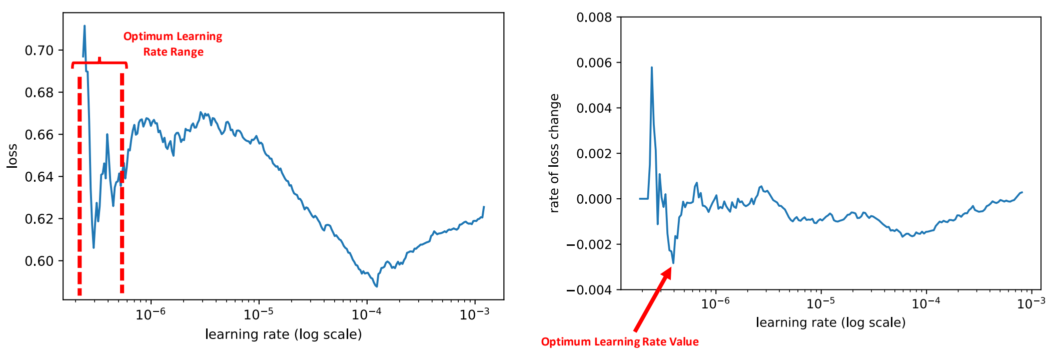


Figure 5. Impact of learning rates when fine-tuning our model.

We analyzed different values within the optimal range shown in Figure 5 to determine the best initial learning rate for our model. In our model, the loss function started to decrease very rapidly when the learning rate was between 1×10^{-7} and 1×10^{-6} (see Figure 5). By choosing a value in this range, we still were able to further decrease the LR using ReduceLROnPlateau. ReduceLROnPlateau reduces learning rate by a factor of 0.5 if the validation loss does not improve after two iterations (epochs).

4.2.1.2. Overfitting

Choosing the exact number of training epochs to use is a common problem in training neural networks. Overfitting of the training dataset can be caused by too many epochs, whereas underfitting may result from too few iterations. By selecting an appropriate early-stopping method we can start with a large number of epochs and stop the training process if the monitored metric does not show any improvement. This alleviates the needs to manually select the number of epochs and instead utilize a data-driven automation approach. In our model, we monitor the validation loss value and will stop training after four epochs if it does not show any improvement.

4.2.2. Selecting Optimal Classification Layer

The BERT-base model consists of an embedding layer, a stack of 12 encoders for the base model, and a pooling layer. The input embedding layer operates on the sum of the token embeddings, the segmentation embeddings, and the position embeddings. The final hidden state from encoders is corresponding to the special classification token (CLS). For the text classification task, this token is used as the aggregate sequence representation.

The first layer after the encoders is the next sentence prediction (NSP) layer. The NSP layer is utilized to understand sentence relationships [5]. This layer transforms the last encoder layer output (CLS token) into two vectors, each representing IsNext and Not-Next, respectively. The NSP layer then is connected to a fully connected neural network for classification.

We did not freeze any of the layers during the fine-tuning process. In this way, the model will adjust BERT's pre-trained weights based on our dataset, hyperparameters, and our given downstream task.

4.2.3. Selecting the Optimal Number of Dense Layers

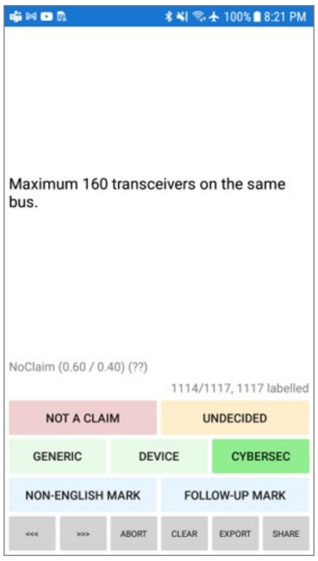
Different layers of a neural network can capture different levels of syntactic information for text classification [3]. We studied the impact of different numbers of fully connected dense layers on top of the stacked encoders in BERT, in order to determine the best model based on our given dataset. The hyperparameters we adapted here are the different dropouts and the number of neurons for each dense layer. We achieved the best results for two dense layers with 64 and 16 neurons and dropout rates of 0.5 and 0.3, respectively. Table 2 shows the highest accuracy and learning rate based on the different number of dense layers.

Table 2. Comparing different number of dense layers in fine-tuning CyBERT.

Model Architecture	LR	Accuracy	F1 Score
BERT base + 1 dense	2×10^{-5}	0.92	0.91
BERT base + 2 dense	2×10^{-6}	0.92	0.92
BERT base + 3 dense	4×10^{-6}	**0.948**	**0.932**
BERT base + 4 dense	1×10^{-5}	0.93	0.93

4.2.4. Labeling Process

We designed and implemented a mobile application to aid in the manual labeling of extracted sequences. A snapshot of the application is provided in Figure 6. This mobile app was written as a `Xamarin.Forms app` [42], an open-source mobile app development framework for building Android and iOS apps with .NET and C#. This application supports initial labeling and relabeling and also supports labeling using multiple people, by associating designated labels with the person providing that label. This can subsequently be used for outlier removal, computation of majority decisions on final labels, and more.



Maximum 160 transceivers on the same bus.

NoClaim (0.60 / 0.40) (??)

1114/1117, 1117 labelled

Figure 6. A snapshot of the manual labeling mobile app.

4.2.4.1. Initial Labelling

We manually labeled sequences extracted from a sample of ICS device documents. Claim sequences were initially categorized into three types, including generic claims, device claims, and cybersecurity claims. Having these individual claim types will help future investigation regarding claim type detection. For the purpose of this study, we grouped all these types of claims as claim labels and also removed the sequences with the "Not Sure" label from the classification dataset.

4.2.4.2. Prediction-Guided Relabeling Process

After obtaining our initial labeled dataset, we used it to train CyBERT. We followed the fine-tuning process we detailed earlier in this article. The trained model was then used to predict the class labels for each sequence, and we compared the results with the manual labels. From that comparison, we could find that there were a number of sequences whose class prediction probability was in the vicinity of 0.5, indicating a large uncertainty in some instances. We carefully reviewed each of those cases and adjusted the labels where necessary. The final class count and their distribution are reported in Table 3.

Table 3. Number of sequence labels and label distribution in each class.

Class Label	Final Label Counts	Final Label Distribution
NotClaim	4544	67.189%
ClaimDevice	848	12.53%
ClaimGeneric	807	11.93%
ClaimCybersec	335	4.95%
NotSure	226	3.34%

After this label review process, we repeated the fine-tuning process to obtain new model parameter values. A comparison of the obtained results pre- and post-label review demonstrate the improvement to CyBERT from this iterative refinement process. The obtained probabilities were sorted and plotted in the Figure 7, showing that we could successfully reduce the model uncertainty.

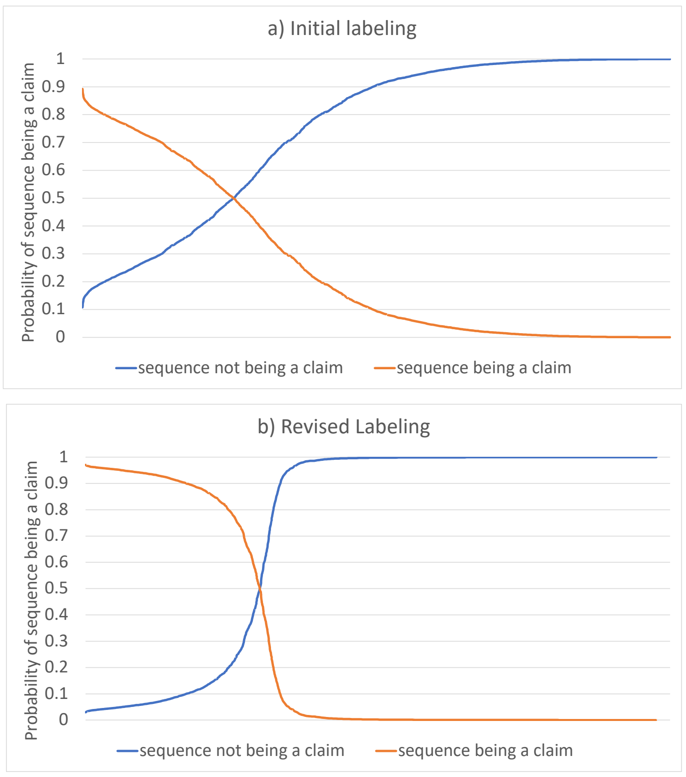


Figure 7. Sorted probabilities with (**a**) initial labeling and (**b**) revised labeling.

4.3. Classifier Training

One of the benefits of the pre-trained model is being able to train a model for downstream tasks by utilizing relatively small training data sets [15]. We manually labeled 6763 sequences from a set of documents then fine-tuned BERT on the resulting dataset. For training the classifier, we evaluated the impact of selecting different number of dense layers and learning rate values.

We subsequently tested the model on our dataset and adjusted labels for sentences causing uncertainty within the model. This iterative process helped us to reduce the uncertainty of our tuned model (Figure 7). We fine-tuned BERT on the revised database and tuned the associated hyperparameters to improve the claim classifier. The final model accuracy we achieved with CyBERT was 94.4%. We set the sequence length and batch size to 128 and 32, respectively.

4.3.1. Impact of Randomness

Fine-tuning BERT on a small dataset can be unstable [5]. Randomness in machine learning models can happen because of randomly initialized weights and biases, dropout regularization, and optimization techniques [43]. To investigate their effects and impact on our model, we performed the training stage 100 times, using the same model hyperparameters and only varying the random seeds, in order to achieve a statistically reliable result for our model accuracy.

The distributions of the training accuracy, validation accuracy, and testing accuracy are plotted in Figure 8. Table 4 reports the standard deviation, mean, and confidence interval (CI), as well as the margin of error for 100 training runs each for validation, training, and testing. These plots and the associated table show that CyBERT's true accuracy was within the 0.943 and 0.945 interval with a 95% CI.

These results suggest that the accuracy of our model can be impacted by the random seed value selection. Therefore, based on the recommendation published in the literature [5], we chose the random seed that leads to the highest validation accuracy for CyBERT.

Table 4. Standard deviation and mean for training, validation, and testing accuracy.

Dataset	SD	Mean	CI (95%)	Margin of Error
Training	0.012	0.959	0.957 to 0.961	0.00235
Validation	0.006	0.944	0.943 to 0.945	0.00118
Testing	0.007	0.935	0.934 to 0.936	0.00137

The results from Table 4 and Figure 7 confirms that even though our labeling set was small, the resulting CyBERT classifier can successfully identify cybersecurity feature claims within sequences with a high level of confidence.

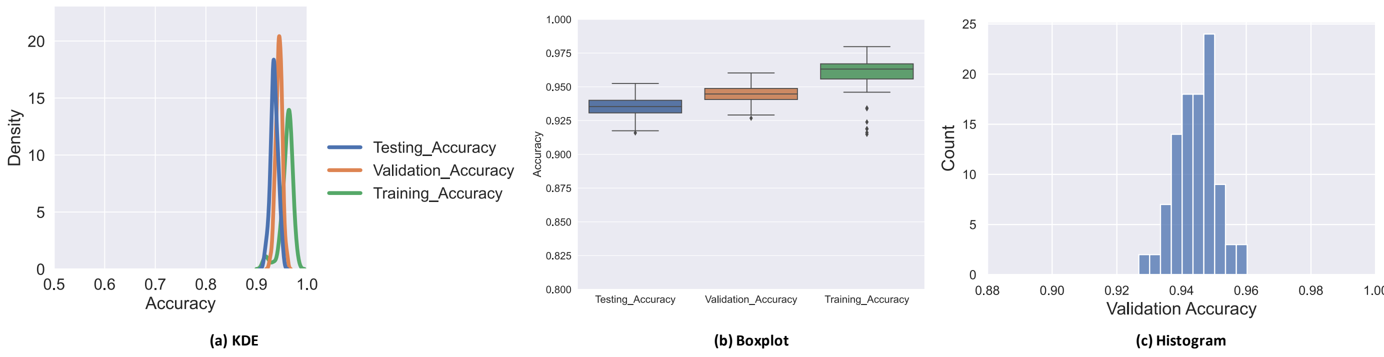


Figure 8. Kernel density estimation (**a**); box plot (**b**) for training, validation, and testing accuracy; validation accuracy histogram (**c**).

5. Results and Discussion

All the models in this study were executed on our university's supercomputing infrastructure, HCC Crane [44] with NVIDIA Tesla V100 GPUs with 16 GB of RAM per GPU. To obtain an unbiased estimate of out-of-sample accuracy, we divided the sequence dataset into training, validation, and test samples, utilizing 70%, 10%, and 20% or our total dataset, respectively.

For the comparison of the models, we used accuracy and macro-weighted F1 for all models. All accuracies reported in this article were based on predictions for the test set. Macro-weighted F1 is a measure of a model's accuracy on a dataset, which returns the average F1 score considering the proportion for each label in the dataset.

5.1. Comparing CyBERT with Pre-Trained BERT for Sequence Classification

In order to evaluate our strategy for fine-tuning BERT, we compared the results with BertForSequenceClassification. BertForSequenceClassification is a pre-trained BERT transformer model released by the Google research group in [5] with a dropout layer followed by a softmax classification/regression layer on top of the stacked decoder output. The predicted probability of each label is calculated as:

$$P(c|h) = \text{softmax}(Wh) \tag{1}$$

where h is the final hidden state of the first token in the BERT token embedding matrix (CLS), and W is the matrix for task-specific parameters.

To evaluate the impact of the architecture and the training approach, we conducted two separate experiments. In both, we started with the BertForSequenceClassification model. This model is comprised of 12 stacked encoders, a dropout layer, and a classifier layer. This model was initialized from pre-trained BERT.

In the first experiment, all transformer layers remained frozen. During training of the classifier it can therefore only tune the last layer, which is the classification layer. The resulting accuracy of this model on our dataset was 76%.

In the second experiment, for the purpose of comparison and to show the effect of fine-tuning, we employed the same model architecture as in the first experiment but enabled full fine-tuning by unfreezing all its layers. The fine-tuning process also involved finding the optimum LR based on this model architecture and dataset.

Table 5 shows the best accuracy among all the learning rates we studied for both approaches using BertForSequenceClassification, from among the tested range of 1×10^{-2} to 5×10^{-7}. The results show that unfreezing all encoders during fine-tuning of the model, training the classifier, and optimizing hyperparameters improved the model accuracy by 16 percentage points, from 76% to 92%.

As we have shown in the previous section, by further optimizing the architecture itself and introducing additional layers, we could further improve CyBERT's accuracy to 94.4%.

Table 5. Fine-tuning effect on BERT with one classification layer.

	LR	Accuracy	F1 Score
BertForSequenceClassification	1×10^{-7}	0.76	0.72
Fine-tune BERT-base uncased	2×10^{-5}	0.92	0.91

5.2. Comparing CyBERT with Other Language Models

5.2.1. Generative Pre-Trained Transformer (GPT)

GPT models are unsupervised transformer language models that are trained on a very large corpus of text. GPT-2 [6] was trained on a diverse corpus of unlabeled text (40 GB of Internet text). The GPT-2's main training objective is to be able to predict the next word while all of the previous words are given. Although BERT and GPT are both very strong language models, trained on a large corpus of data, and based on the transformer architecture using masked self attention, they are fundamentally different. GPT-2 uses decoder blocks from the transformers, while BERT utilizes encoder-only transformers. The other main difference between these two models is the word embedding. GPT-2 takes word vectors as input and produces the probability estimation of the next word as outputs (one token word at a time), while BERT uses the entire surrounding context together. GPT-2-small contains a 12-layer decoder-only transformer with 12 attention heads. Fine-tuning pre-trained GPT-2 is a two-stage semi-supervised approach. This process starts with using the pre-trained GPT model weights to set the initial parameters (unsupervised). Fine-tuning is then the supervised discrimination technique to adjust parameters to the target task. In this study, we fine-tuned GPT-2 (small and medium) with our dataset after unfreezing all layers in the transformer blocks. We also tried different learning rates and reported the values in Table 6. Due to model limitations, we were not able to perform GPT-2 large and GPT-2 extra large. We also requested OpenAI GPT-3 API services, but as of the time of writing this article, we were still awaiting approval for resource access.

Table 6. Fine-tuning GPT-2 small and medium.

	LR	Accuracy	F1 Score
GPT-2 Small	1×10^{-2}	0.86	0.84
	2×10^{-4}	0.89	0.87
	2×10^{-3}	0.88	0.85
	2×10^{-5}	**0.91**	**0.89**
	2×10^{-6}	0.85	0.81
	2×10^{-7}	0.70	0.52
GPT-2 Medium	2×10^{-2}	0.87	0.85
	2×10^{-3}	0.90	0.87
	2×10^{-4}	0.90	0.86
	2×10^{-5}	0.91	0.89
	2×10^{-6}	**0.92**	**0.91**
	2×10^{-7}	0.75	0.63

Table 6 indicates that the highest accuracy was 91%, which was achieved by the GPT-2 medium model with an LR of 2×10^{-6}. We observe from Figure 9 that the GPT-2 classifier model will not converge if the learning rate is higher than 2×10^{-6}(blue lines) for GPT-2 small, or 2×10^{-7} (orange lines) for GPT-2 medium, respectively. This phenomenon indicates the catastrophic forgetting effect present in the GPT-2 models. These plots indicate that the acceptable hyperparameters for these models for training small and medium GPT-2 models are in the vicinity of $2 \times 10^{-6}$and 2×10^{-7}, respectively.

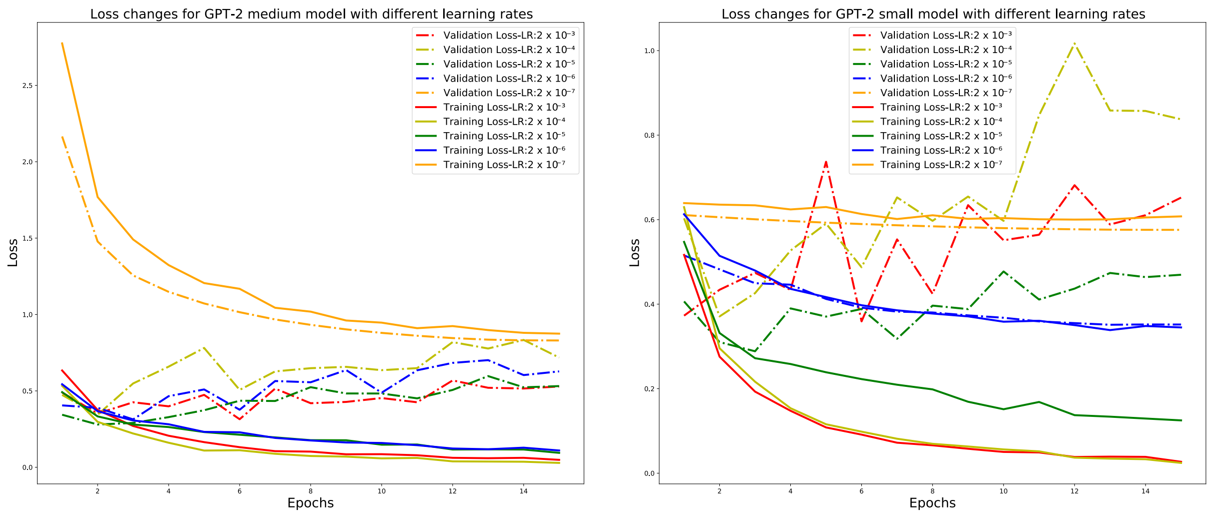


Figure 9. Loss changes for GPT-2 models with different learning rates: GPT-2 medium (**left**) and GPT-2 small (**right**) for 15 epochs.

5.2.2. ULMFiT

Universal language model fine-tuning (ULMFiT) [3] is a transfer learning method for NLP tasks including text classification. The ULMFiT architecture includes a three-layer weight-dropped LSTM (AWD-LSTM). The averaged stochastic gradient method (ASGD) AWD-LSTM [45] uses the DropConnect method to regulate weights in the recurrent networks. The ULMFiT training starts with a general language model pre-trained on a large Wikipedia text corpus (103 million words). For any domain-specific task, ULMFiT provides a discriminative fine-tuning process for the desired domain. The discriminative fine-tuning refers to using different learning rates to tune each layer. The final step is fine-tuning the updated language model on the target downstream task (such as text classification). In order to compare our CyBERT model with ULMFiT, we used the gradual unfreezing method to fine-tune ULMFiT. This process starts by only unfreezing the final layer while keeping all others frozen. The process then fine-tunes the model for several epochs and determines a new optimal learning rate for the tuned model up to this point before repeating this process while gradually unfreezing all other layers. The accuracy for each step is presented in Table 7.

Table 7. Fine-tuning ULMFiT.

Fine-tuning ULMFiT	Accuracy	F1 Score
Unfreeze last layer	0.83	0.83
Unfreeze last two layers	0.87	0.85
Unfreeze all layers	**0.91**	**0.91**

The best accuracy we achieved by fine-tuning ULMFiT on our dataset was 91% with unfreezing all three-layer AWD-LSTM weights. Unfreezing all layers will update all weights during the training process.

5.2.3. ELMo

Embeddings from language models (ELMo) [4] is a context-based word embedding method. This model learns contextualized word representations using two layers of BiLSTMs and a character-based encoding layer. Using a character-based layer, characters are encoded into a word's representation for the following two BiLSTM layers that enable hidden states to enhance a word's final embedding.

Classification using ELMo starts with tokenizing each sequence with ELMo, with the resulting embedding vectors subsequently used as input to different NN models. In our experiments using ELMo, we utilized a fully connected neural network, CNN, BiLSTM, and LSTM in order to compare the results with our model. For each model, based on the defined architecture, we first determined the best learning rate based on the method explained in [41]. Table 8 summarizes the best results for each type of neural network.

Table 8. Classification using ELMo.

	Architecture	LR	Accuracy	F1-Score
ELMo + NN	2 Dense	9×10^{-5}	0.91	0.9
ELMo + CNN	3 Convolution	4×10^{-7}	0.91	0.89
	2 Convolution	1×10^{-4}	0.91	0.89
	1 Convolution	6×10^{-5}	**0.92**	**0.9**
ELMo + BiLSTM	3 BiLSTM	2×10^{-4}	0.87	0.85
	2 BiLSTM	4×10^{-6}	0.88	0.86
	1 BiLSTM	1×10^{-4}	0.91	0.89
ELMo + LSTM	3 LSTM	9×10^{-4}	0.89	0.86
	2 LSTM	2×10^{-4}	0.89	0.87
	1 LSTM	2×10^{-5}	0.90	0.88

Table 8 shows that the highest accuracy was achieved by using the ELMo tokenizer and one convolutional layer with a filter size of 128, connected to two dense layers with 256 and 128 neurons, respectively, each followed by dropout layers with rates 0.3 and 0.2, respectively.

5.3. Comparing CyBERT with Neural Networks

Traditionally, text classification with NN models utilized word-level embedding methods such as GloVe. The problem with these embedding methods is that it removes the contextual meaning of these words.

Throughout all of the different evaluations shown in the following sections for the different NN models, in each case we first obtained the 768-dimensional BERT-Base uncased tokenizer embeddings for each WordPiece token in the input sequences. By utilizing the same tokenizer for all these models, we can exclude the effect of the tokenizer embedding on a model's performance.

5.3.1. Convolutional Neural Network

A convolutional neural network (CNN) is a class of artificial neural network models that is capable of extracting abstract features from complex data. Through many iterations, the model learns to map inputs to their class labels. CNNs are composed of two main components: (1) the first component consists of several filters and pooling layers that are applied to multi-dimensional arrays in order to create a feature space. These features are abstractions of the most significant characteristics contained within the input data; (2) the second component of CNNs contains a fully connected network that maps the extracted features to their target. In classification problems, this target is a group of data that shares a common property. In CNNs, the multi-dimensional output of the convolution component is flattened and passed to the fully connected network.

We studied the impact of different values for the CNN hyperparameters, including the number of convolutions, different dropout rates for each layer, and the number of dense layers after the convolutions. For each architecture, the best learning rate was determined using the LrFinder function [41]. The best accuracies for all architectures are reported in Table 9.

Table 9. Comparing CNN-based classifier results using the BERT tokenizer.

Architecture		LR	Accuracy	F1 Score
1 Convolution	**1 Dense**	4×10^{-4}	**0.88**	**0.87**
	2 Dense	5×10^{-4}	0.87	0.87
	3 Dense	1×10^{-4}	0.87	0.86
2 Convolution	1 Dense	5×10^{-5}	0.87	0.86
	2 Dense	5×10^{-5}	0.87	0.86
	3 Dense	2×10^{-3}	0.87	0.86
3 Convolution	1 Dense	4×10^{-4}	0.86	0.85
	2 Dense	4×10^{-4}	0.86	0.85
	3 Dense	1×10^{-4}	0.86	0.86

Table 9 shows that the best results were achieved using a network with one convolution (128 filters) connected to a dense layer with 256 neurons followed by a dropout layer with a 0.3 dropout rate.

5.3.2. LSTM

It has been shown in [46] that neural network models can be applied to learn distributed sentence representations and achieve good results in tasks related to sentiment classification and text categorization with little external domain knowledge. In long short-term memory (LSTM) neural networks, features are learned at the phrase-level by using a convolutional layer. This convolutional layer then feeds sequences of such higher-layer representations into LSTMs to learn their relationships to long-term features.

The LSTM model for NLP tasks requires an embedding matrix. We once again utilized the BERT-based uncased tokenizer for embedding each token into its vector for initializing the model. We evaluated different options for the number of LSTM and dense layers, in order to determine the best hyperparameters and the resulting architecture. The best learning rate for each architecture was determined again using the LrFinder function. We achieved the best accuracy with an architecture comprised of a single LSTM with 150 hidden units and a 0.1 dropout rate, combined with two dense layers with 150 and 124 neurons, respectively, coupled with the BERT tokenizer. The achieved accuracies for different configurations are reported in Table 10.

Table 10. Comparing LSTM-based classifier results using the BERT tokenizer.

Architecture		LR	Accuracy	F1 Score
1 LSTM	1 Dense	1×10^{-4}	0.873	0.864
	2 Dense	4×10^{-5}	**0.867**	**0.830**
	3 Dense	5×10^{-5}	0.871	0.859
2 LSTM	1 Dense	2×10^{-4}	0.871	0.860
	2 Dense	5×10^{-5}	0.872	0.862
	3 Dense	4×10^{-5}	0.866	0.855
3 LSTM	1 Dense	2×10^{-4}	0.871	0.862
	2 Dense	2×10^{-3}	0.865	0.853
	3 Dense	5×10^{-3}	0.861	0.851

5.3.3. BiLSTM

Bidirectional LSTM models are especially suited for sequential modelling and can be used to extract additional contextual information from the feature sequences provided by the convolutional layer [47]. The bidirectional LSTM actually concatenates two one-way actions to obtain the embeddings from the networks [48].

The BiLSTM recurrent network is once again utilizing the BERT-Base uncased tokenizer. In this model, we studied the impact of the number of BiLSTM layers, the number of hidden LSTM dimensions, the dropout rates for each layer, and the number of dense layers on top of the concatenated BiLSTMs. For each architecture, we determined the best learning rate using the LrFinder function. Table 11 shows the achieved accuracies and F1 scores for each architecture configuration we tested.

Table 11. Comparing BiLSTM-based classifier results using the BERT tokenizer.

Architecture		LR	Accuracy	F1 Score
1 BiLSTM	**1 Dense**	1×10^{-5}	**0.88**	**0.87**
	2 Dense	5×10^{-5}	0.87	0.86
	3 Dense	1×10^{-4}	0.88	0.87
2 BiLSTM	1 Dense	5×10^{-4}	0.87	0.86
	2 Dense	4×10^{-5}	0.87	0.86
	3 Dense	2×10^{-5}	0.87	0.85
3 BiLSTM	1 Dense	2×10^{-5}	0.87	0.86
	2 Dense	4×10^{-6}	0.86	0.85
	3 Dense	4×10^{-5}	0.87	0.86

Table 11 shows that the best result was achieved using a network with one BiLSTM with 150 hidden units and a 0.1 dropout rate, which was connected to one dense layer with 150 neurons followed by a 0.1 dropout rate.

5.4. Performance Comparison for All Models

As shown in Figure 10, we presented the training and validation accuracy and the loss values for all models during the training phase. The best validation accuracy was achieved by CyBERT, proposed by our NLP model, with a 95.4% accuracy. The loss plots indicate the effectiveness of the learning rate function we used for initializing all models. Additionally, the loss plots show the effectiveness of the early-stopping method to avoid overfitting in cases where the validation loss increases during the training phase.

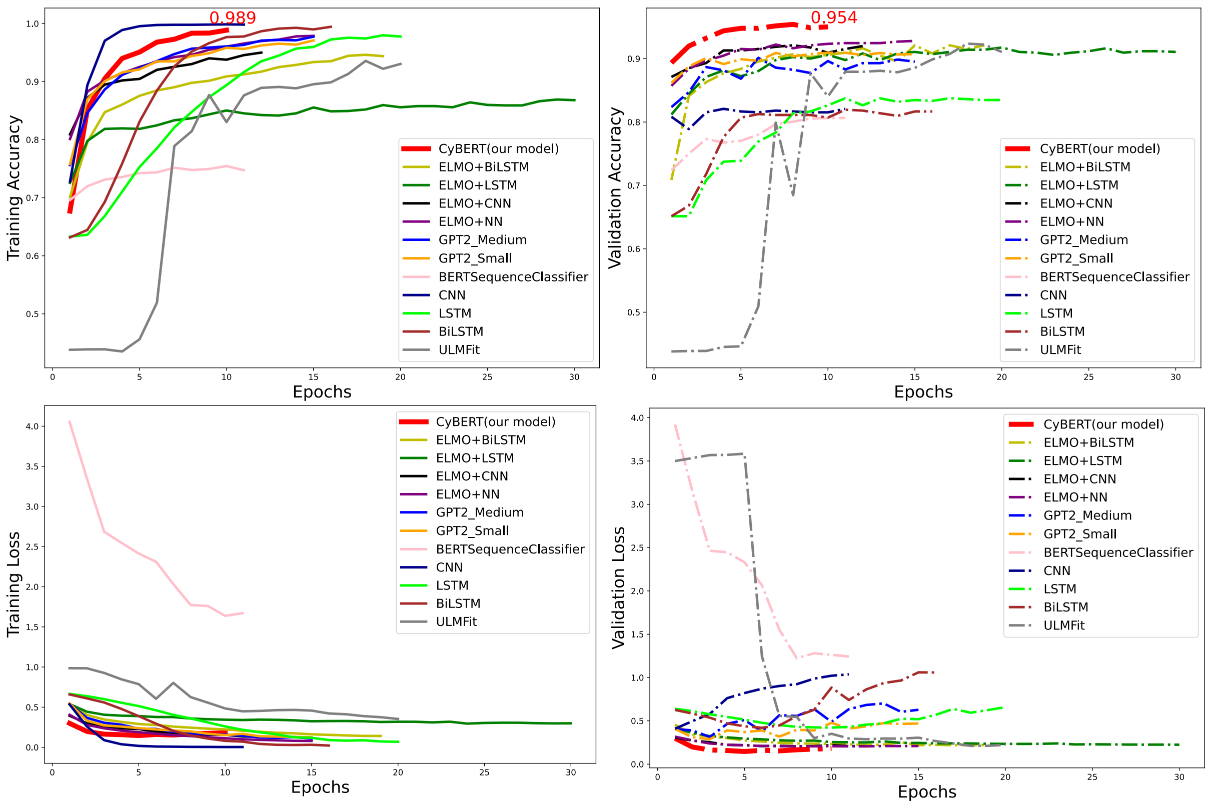


Figure 10. Comparing training accuracy (**left top**), validation accuracy (**right top**), training Loss (**bottom left**), and validation loss (**bottom right**) for all models.

Table 12 compares the accuracy for each model from our test set. The learning rates we report in this table indicate the best LR we found for each model based on the architecture and dataset we provided earlier. The learning rate was found via the LrFinder method and fine-tuned during the training phase.

Binary classification algorithms are often evaluated using receiver operating characteristic (ROC) curves. Instead of a single value, it provides a two-dimensional illustration of classifier performance. ROC plots the false-positive rates against the true-positive rates in classification. For the ideal classifier scenario, we want to observe a high true-positive rate and a low false-positive rate [49]. Figure 11 compares the ROC curve for all the models we tested in this study. The classifier was desired to be closer the upper-left corner of the ROC curve plot. Figure 11 illustrates that our CyBERT classifier performed better in all areas compared to all other models.

Table 12. Comparison across all tested models.

Model	Architecture	Accuracy	Macro Weighted F1	AUC	Training * Time	Testing * Time	Trainable Parameters
CyBERT (Our Model)	**12 Encoder 3 Dense**	**0.954**	**0.93**	**0.948**	32,970	97	108,647,026
BERT Classifier	12 Encoder 1 Dense	0.76	0.72	0.773	7832	77	109,482,240
CNN with BERT Tokenizer	1 Convolution 1 Dense	0.88	0.87	0.862	1470	75	11,330,286
LSTM with BERT Tokenizer	1 LSTM 2 Dense	0.86	0.83	0.852	1190	58	10,884,324
BiLSTM with BERT Tokenizer	1 BiLSTM 3 Dense	0.88	0.87	0.871	2412	61	12,604,804
GPT2-Small	12 Decoder 1 Dense	0.9	0.87	0.908	37,640	134	125,444,134
GPT2-Medium	24 Decoder 1 Dense	0.92	0.91	0.857	61,340	175	354,825,216
ELMo + NN	2 Dense	0.91	0.9	0.910	3950	88	295,554
ELMo + CNN	1 Convolution 2 Dense	0.92	0.9	0.912	4128	105	16,778,242
ELMo + LSTM	1 LSTM 2 Dense	0.90	0.88	0.916	15,603	74	19,785,410
ELMo + BiLSTM	1 BiLSTM 2 Dense	0.91	0.89	0.897	18,216	58	15,854,274
ULMFiT	3 AWD-LSTM	0.91	0.91	0.902	1873	42	62,652

* Times are in seconds.

The other important parameter for a classifier is the “area under the ROC curve” (AUC). Generally, the higher the AUC score, the better a classifier performs for the given task [49]. The AUC is reported for all models in Table 12 and Figure 11, which indicate that CyBERT had the highest AUC value among all tested models.

Figure 10 shows that the best value for AUC belongs to CyBERT, whichwasis 0.948. This indicates that CyBERT is better at distinguishing classes than any other models we compared it against in this study. The next-highest AUC values belonged to the ELMo models, which were all at least three percentage points lower than CyBERT’s AUC.

The training time reported in Table 12 was only for the training phase after determining the optimum learning rate for each model. Language models such as BERT, GPT-2, and ELMo typically require more time for the training phase because of the higher number of trainable parameters. Among all these models, GPT-2 required the most time to complete training, for both medium and small models, followed by our proposed model, CyBERT. This result was expected based on the number of trainable parameters in these models. Once trained, we do not need to repeat training for these models. They are ready to be used for classification, unless the dataset needs to be updated.

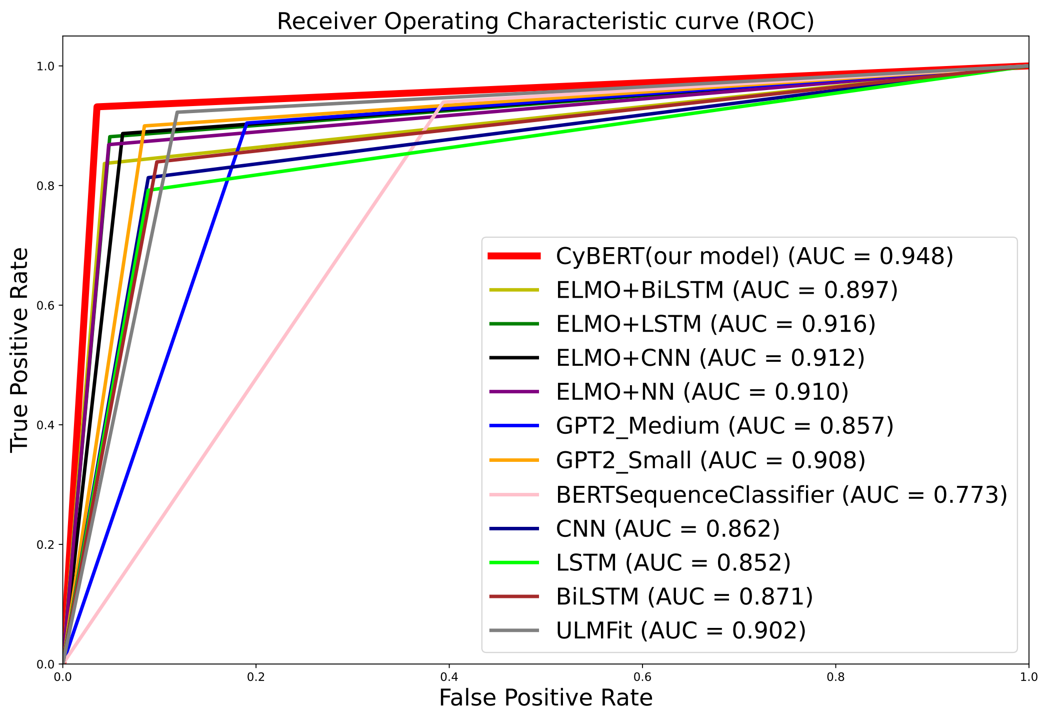


Figure 11. ROC comparison for all models.

5.5. Summary of Results

Our CyBERT as well as the original BertForSequenceClassification employ 12 encoders. CyBERT employs two additional dense layers. The resulting F1 and accuracy for CyBERT were 19 and 21 percentage points higher, respectively, than the performance for BertForSequenceClassification, which indicates the fine-tuning effect on the model performance. ULMFiT had the lowest number of trainable parameters and was the fastest model in training and testing. However, it had a 4, 2, and 5 percentage point lower accuracy, respectively, for accuracy, F1, and AUC, compared to CyBERT.

A comparison of the training and testing loss and accuracy results for all models is also provided in Figure 10, with the red curve indicating CyBERT's performance. The macro-weighted F1 for CyBERT was 0.93, whereas, for GPT2-small, it was 0.87, which represents a 6 percentage point improvement. Moreover, although CyBERT's training time was higher than the training times for the ELMo and ULMFiT language models, its accuracy was 3 and 4 percentage points higher, respectively, which is a significant improvement.

6. Conclusions

We introduced CyBERT, a classification model obtained through fine-tuning the BERT-base language model, utilizing our curated cybersecurity domain NLP dataset. This classification model is motivated by our ongoing research in developing an unbiased, objective, and semi-supervised approach, named CYVET, for vetting ICS device cybersecurity implications through a process of automatically vetting vendor claims extracted from device documentation via natural language processing against industry requirements. This article details one of the core building blocks of the overall CYVET approach for vetting cybersecurity claims.

We first generated a dataset comprised of a large number of ICS device documentations. We subsequently extracted and labeled sequences from these documents for the purpose of this study. The resulting dataset was used to train a CyBERT that can be used to detect claims about device features from within these documents. Our new CyBERT model increased the accuracy of this classification task by 19 percentage points compared to the original BERT text classifier.

We also provided an in-depth comparative analysis of BERT and CyBERT to show the necessity of our fine-tuning strategies. In addition to CyBERT, we implemented other pre-trained language models such as GPT-2, ELMo, ULMFiT, and various neural-network-based models, including a convolution neural network (CNN), LSTM, and BiLSTM, for comparison against our presented model. The extensive experimental results showed the effectiveness and robustness of CyBERT. The results demonstrate that the performance of our CyBERT is better than that of all other models we tested as part of this study.

Author Contributions: Investigation, K.A., M.H. and H.S.; writing—original draft preparation, K.A. and M.H.; writing—review and editing, K.A., M.H., H.S., K.P. and J.L.J.; supervision, H.S., M.H., K.P. and J.L.J.; project administration, K.P. and J.L.J.; funding acquisition, H.S., M.H., K.P. and J.L.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the US. Department of Energy through a subcontract from Oak Ridge National Laboratory, project No. 4000175929 (project CYVET).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This research has been supported in part by the Department of Energy Cybersecurity for Energy Delivery Systems program and the Oak Ridge National Laboratory.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CYVET	Cyber-physical security assurance
BERT	Bidirectional encoder representations from transformers
CyBERT	Cybersecurity BERT
ICS	Industrial control systems
CR	Cybersecurity requirements
VSF	Vendor-supplied features
OT	Operational technology
OCR	Optical character recognition
NLP	Natural language processing
ULMFiT	Universal language model with fine-tuning
ELMo	Embeddings from language models
GPT	Generative pre-training
NER	Name entity recognition
RE	Relation extraction
QA	Question answering
MLM	Masked language model
NSP	Next sentence prediction
LR	Learning rate
CI	Confidence interval
ASGD	Averaged stochastic gradient
RNN	Recurrent neural network
CNN	Convolutional neural network
NN	Neural network
LSTM	Long short-term memory
BiLSTM	Bidirectional LSTM
ROC	Receiver operating characteristic
AUC	Area under the ROC curve

References

1. Ameri, K.; Hempel, M.; Sharif, H.; Lopez, J., Jr.; Perumalla, K. Smart Semi-Supervised Accumulation of Large Repositories for Industrial Control Systems Device Information. In Proceedings of the ICCWS 2021 16th International Conference on Cyber Warfare and Security, Nashville, TN, USA, 25–26 Februray 2021; pp. 1–11.
2. Perumalla, K.; Lopez, J.; Alam, M.; Kotevska, O.; Hempel, M.; Sharif, H. A Novel Vetting Approach to Cybersecurity Verification in Energy Grid Systems. In Proceedings of the 2020 IEEE Kansas Power and Energy Conference (KPEC), Manhattan, KS, USA, 13–14 July 2020; pp. 1–6.
3. Howard, J.; Ruder, S. Universal language model fine-tuning for text classification. *arXiv* **2018**, arXiv:1801.06146.
4. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.
5. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
6. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
7. Lee, J.; Yoon, W.; Kim, S.; Kim, D.; Kim, S.; So, C.H.; Kang, J. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics* **2020**, *36*, 1234–1240. [CrossRef] [PubMed]
8. Naseem, U.; Khushi, M.; Reddy, V.; Rajendran, S.; Razzak, I.; Kim, J. BioALBERT: A Simple and Effective Pre-trained Language Model for Biomedical Named Entity Recognition. *arXiv* **2020**, arXiv:2009.09223.
9. Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv* **2019**, arXiv:1909.11942.
10. Beltagy, I.; Lo, K.; Cohan, A. SciBERT: A pretrained language model for scientific text. *arXiv* **2019**, arXiv:1903.10676.
11. Edwards, A.; Camacho-Collados, J.; De Ribaupierre, H.; Preece, A. Go simple and pre-train on domain-specific corpora: On the role of training data for text classification. In Proceedings of the 28th International Conference on Computational Linguistics, Barcelona, Spain, 8–13 December 2020; pp. 5522–5529.
12. Jwa, H.; Oh, D.; Park, K.; Kang, J.M.; Lim, H. exbake: Automatic fake news detection model based on bidirectional encoder representations from transformers (bert). *Appl. Sci.* **2019**, *9*, 4062. [CrossRef]
13. Vogel, I.; Meghana, M. Detecting Fake News Spreaders on Twitter from a Multilingual Perspective. In Proceedings of the 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), Sydney, Australia, 6–9 October 2020; pp. 599–606.
14. Liu, C.; Wu, X.; Yu, M.; Li, G.; Jiang, J.; Huang, W.; Lu, X. A two-stage model based on BERT for short fake news detection. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Knowledge Science, Engineering and Management, Athens, Greece, 28–30 August 2019*; Springer: Cham, Switzerland, 2019; pp. 172–183.
15. Sun, C.; Qiu, X.; Xu, Y.; Huang, X. How to fine-tune bert for text classification? In *Lecture Notes in Computer Science, Proceedings of theChina National Conference on Chinese Computational Linguistics, Kunming, China, 18–20 October 2019*; Springer: Cham, Switzerland, 2019; pp. 194–206.
16. Khetan, V.; Ramnani, R.; Anand, M.; Sengupta, S.; Fano, A.E. Causal BERT: Language models for causality detection between events expressed in text. *arXiv* **2020**, arXiv:2012.05453.
17. Lee, J.S.; Hsiang, J. Patentbert: Patent classification with fine-tuning a pre-trained bert model. *arXiv* **2019**, arXiv:1906.02124.
18. Araci, D. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv* **2019**, arXiv:1908.10063.
19. Liu, Z.; Huang, D.; Huang, K.; Li, Z.; Zhao, J. Finbert: A pre-trained financial language representation model for financial text mining. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI, Yokohama, Japan, 16–18 January 2021; pp. 5–10.
20. Das, S.S.; Serra, E.; Halappanavar, M.; Pothen, A.; Al-Shaer, E. V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. *arXiv* **2021**, arXiv:2102.11498.
21. Zhou, S.; Liu, J.; Zhong, X.; Zhao, W. Named Entity Recognition Using BERT with Whole World Masking in Cybersecurity Domain. In Proceedings of the 2021 IEEE 6th International Conference on Big Data Analytics (ICBDA), Xiamen, China, 5–8 March 2021; pp. 316–320.
22. Chen, Y.; Ding, J.; Li, D.; Chen, Z. Joint BERT Model based Cybersecurity Named Entity Recognition. In Proceedings of the 2021 The 4th International Conference on Software Engineering and Information Management, Yokohama, Japan, 16–18 January 2021; pp. 236–242.
23. Gao, C.; Zhang, X.; Liu, H. Data and knowledge-driven named entity recognition for cyber security. *Cybersecurity* **2021**, *4*, 1–13. [CrossRef]
24. Tikhomirov, M.; Loukachevitch, N.; Sirotina, A.; Dobrov, B. Using bert and augmentation in named entity recognition for cybersecurity domain. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Applications of Natural Language to Information Systems, Saarbrücken, Germany, 24–26 June 2020*; Springer: Cham, Switzerland, 2020; pp. 16–24.
25. Xie, B.; Shen, G.; Guo, C.; Cui, Y. The Named Entity Recognition of Chinese Cybersecurity Using an Active Learning Strategy. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 6629591. [CrossRef]
26. Yin, J.; Tang, M.; Cao, J.; Wang, H. Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description. *Knowl.-Based Syst.* **2020**, *210*, 106529. [CrossRef]

27. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the Advances in Neural Information Processing Systems, Carson City, NV, USA, 5–10 December 2013; pp. 3111–3119.
28. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
29. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
30. Lample, G.; Ballesteros, M.; Subramanian, S.; Kawakami, K.; Dyer, C. Neural architectures for named entity recognition. *arXiv* **2016**, arXiv:1603.01360.
31. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 649–657.
32. Akbik, A.; Blythe, D.; Vollgraf, R. Contextual string embeddings for sequence labeling. In Proceedings of the 27th international conference on computational linguistics, Santa Fe, NM, USA, 20–26 August 2018; pp. 1638–1649.
33. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv* **2016**, arXiv:1609.08144.
34. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December, 2017; pp. 5998–6008.
35. Bender, E.M.; Gebru, T.; McMillan-Major, A.; Shmitchell, S. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event Canada, New York, NY, USA, 3–10 March 2021; pp. 610–623.
36. Leib, M.; Köbis, N.C.; Rilke, R.M.; Hagens, M.; Irlenbusch, B. The corruptive force of AI-generated advice. *arXiv* **2021**, arXiv:2102.07536.
37. McGuffie, K.; Newhouse, A. The radicalization risks of GPT-3 and advanced neural language models. *arXiv* **2020**, arXiv:2009.06807.
38. Basta, C.; Costa-Jussà, M.R.; Casas, N. Evaluating the underlying gender bias in contextualized word embeddings. *arXiv* **2019**, arXiv:1904.08783.
39. Hutchinson, B.; Prabhakaran, V.; Denton, E.; Webster, K.; Zhong, Y.; Denuyl, S. Social biases in NLP models as barriers for persons with disabilities. *arXiv* **2020**, arXiv:2005.00813.
40. McCloskey, M.; Cohen, N.J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*; Elsevier: Amsterdam, The Netherlands, 1989; Volume 24, pp. 109–165.
41. Smith, L.N. Cyclical learning rates for training neural networks. In Proceedings of the 2017 IEEE winter conference on applications of computer vision (WACV), Santa Rosa, CA, USA, 24–31 March 2017; pp. 464–472.
42. Hermes, D. *Xamarin Mobile Application Development: Cross-Platform c# and Xamarin. Forms Fundamentals*; Apress: Berkeley, CA, USA, 2015.
43. Dodge, J.; Ilharco, G.; Schwartz, R.; Farhadi, A.; Hajishirzi, H.; Smith, N. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv* **2020**, arXiv:2002.06305.
44. Holland Computing Center (HCC) at University of Nebraska-Lincoln. Available online: https://hcc.unl.edu/ (accessed on 1 January 2021).
45. Merity, S.; Keskar, N.S.; Socher, R. Regularizing and optimizing LSTM language models. *arXiv* **2017**, arXiv:1708.02182.
46. Zhou, C.; Sun, C.; Liu, Z.; Lau, F. A C-LSTM neural network for text classification. *arXiv* **2015**, arXiv:1511.08630.
47. Liu, G.; Guo, J. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. *Neurocomputing* **2019**, *337*, 325–338. [CrossRef]
48. Liu, P.; Qiu, X.; Huang, X. Recurrent neural network for text classification with multi-task learning. *arXiv* **2016**, arXiv:1605.05101.
49. Fawcett, T. An introduction to ROC analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. [CrossRef]