SEBERTIS: A Framework for Producing Classifiers of Security-Related Issue Reports

Sogol Masoumzadeh
Electrical & Computer Engineering
McGill University
Montréal, Canada

Yufei (Mary) Li
Electrical & Computer Engineering
McGill University
Montréal, Canada

Shane McIntosh
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada

Dániel Varró
Computer & Information Science
Linköping University
Linköping, Sweden

Lili Wei
Electrical & Computer Engineering
McGill University
Montréal, Canada

Abstract*—Monitoring issue tracker submissions is a crucial software maintenance activity. A key goal is the prioritization of high risk security-related bugs. If such bugs can be recognized early, the risk of propagation to dependent products and endangerment of stakeholder benefits can be mitigated. To assist triage engineers with this task, several automatic detection techniques, from machine learning (ML) models to prompting large language models (LLMs), have been proposed. Although promising to some extent, prior techniques often memorize lexical cues as decision shortcuts, yielding low detection rate specifically for more complex submissions. As such, these classifiers do not yet reach the practical expectations of a real-time detector of security-related issues. To address these limitations, we propose SEBERTIS, a framework to train deep neural networks (DNNs) as classifiers independent of lexical cues, so that they can confidently detect fully unseen security-related issues. SEBERTIS capitalizes on fine-tuning bidirectional transformer architectures as masked language models (MLMs) on a series of semantically equivalent vocabulary to prediction labels (which we call Semantic Surrogates) when they have been replaced with a mask. Our SEBERTIS-trained classifier achieves a 0.9880 F1-score in detecting security-related issues of a curated corpus of 10,000 GitHub issue reports, substantially outperforming state-of-the-art issue classifiers, with 14.44%-96.98%, 15.40%-93.07%, and 14.90%-94.72% higher detection precision, recall, and F1-score over ML-based baselines. Our classifier also substantially surpasses LLM-based baselines, with an improvement of 23.20%-63.71%, 36.68%-85.63%, and 39.49%-74.53% for precision, recall, and F1-score, respectively. Finally, our classifier demonstrates a high confidence in detecting recently submitted security-related issues, achieving 0.7123, 0.6860, and 0.6760 precision, recall, and F1-score, comparable to those of promoting LLMs, making it a practical tool for real-time issue report triage.

***Index Terms*—Issue Report Classification, Deep Neural Networks, Masked Language Models, Fine-tuning, BERT.**

I. INTRODUCTION

Monitoring issue tracker submissions plays a vital role in software maintenance by enabling triage engineers prioritizing bug reports based on their sensitivity, severity, and risk level, thereby ensuring that the most critical defects such as *security-related issues* are addressed first by engineers [1]. Triaging issue reports for popular software products, with their issue trackers attracting a large volume of submissions on a daily basis, can be time-consuming and laborious [2]. At the same time, the difficulty of correctly reporting security-related issues grows in folds for more complex submissions that describe unexpected software behavior, a by-product of the underlying latent vulnerability, rather than including explicit definitions or familiar security-related terminology [3]. Thus, incorporating automatic detection tools greatly assist triage engineers to accurately and efficiently prioritize security-related issues.

Earlier attempts for automatic detection of security-related issues use their structured information, such as the GitHub assignee field, to mine their textual information and vectorize them based on their vocabulary frequencies [4], [5]. *Machine learning (ML)* classifiers are then used to classify the issue reports based on the differences of their numerical vector representations. The accuracy of these text-mining inspired techniques depends on the availability of structured information in the issues. Meanwhile, many of these submissions that are authored by external contributors or non-expert users, lack structure or have missing fields [6]. Consequently, these techniques can fall short in classifying security-related issues.

As a solution for the scarcity of structured information, researchers initiated the use of ML models on issue report summaries and unstructured text along with *natural language processing (NLP)* techniques, such as regression analysis [6], to extract additional features from them. For instance, Das et al. [7] trained ML classifiers on probabilistic NLP features of a series of issue reports and achieved a much higher classification accuracy compared to when these issues were classified using their frequency-based features [8]. In another study conducted by Gosteva-Popstojanova et al. [9], researchers combined ML classifiers with varying types of feature vectors to detect security-related issues of four NASA datasets.

Although demonstrating enhanced performance compared to solely vectorizing issues' structured information, these classifiers also have their own limitations. Specifically, ML models are known to rely on lexical cues, memorizing certain terminology and using them as spurious features in language

inference related tasks such as issue report classification [10], [11]. In other words, instead of learning the true context of the issue reports, ML models learn and use lexical shortcuts to predict labels, resulting in their failure when they attempt to classify more complex security-related issues [12].

Recent advancements in training autoregressive *large language models (LLMs)* [13] and their enhanced performance in parsing, interpreting, and generating natural language, have their applications to be studied more extensively in varying software engineering tasks [14] including issue report classification [15], [16]. If *prompted* with detailed instructions [17], few-shot examples [18], and necessary context-related information, LLMs can perform these tasks reasonably well. However, even with recent advancements, LLMs are still showing the tendency to *hallucinate*, generating results that are factually inaccurate, incomplete, or inconsistent with the instructions they have received. Thus, it is probable for the LLMs to miss or incorrectly classify security-related issues. Additionally, based on the model, using an LLM for classifying issues can take several hours and/or cost up to hundreds of dollars. Due to their limitations, triage engineers cannot use the discussed detection tools to confidently identify security-related issues in real-time. If such an issue is neglected, the underlying vulnerability propagates to dependent products which, in turn, jeopardizes stakeholder benefits [19], [20].

In the current paper, we propose **SEBERTIS**, a framework to train transformer-based *deep neural networks (DNNs)*, such as *bidirectional encoder representations from transformers (**BERT**)* [21], as effective *masked language model (MLM)* classifiers that can be used in real-time to monitor issue trackers for **SE**curity-related **IS**sue submissions. Instead of training DNNs on ground truth labels of issues (i.e., security- and non-security-related), SEBERTIS fine-tunes them on a list of keywords, that we call Semantic Surrogates, that could semantically replace the label names if they appeared in issue reports. To ensure that the models are not memorizing lexical shortcuts to harvest spurious correlations with prediction labels, we mask the instances of Semantic Surrogates so that the training can focus on the semantics embedded in context. We compare the performance of DNNs trained using SEBERTIS with five state-of-the-art issue report classifiers, including two ML- and three LLM-based ones, in detecting security-related issues of a corpus of 10,000 GitHub submissions.

Our evaluation results demonstrate that training with our framework is very effective, with a SEBERTIS-trained DNN achieving 0.9849, 0.9924, and 0.9880 for *precision*, *recall*, and *F1-score* performance metrics, surpassing ML-based baselines by 14.44%-96.98%, 15.40%-93.07%, and 14.90%-94.72%, accordingly. Our classifier also outperforms the LLM-based baselines by 23.20%-63.71%, 36.68%-85.63%, and 39.49%-74.53% across precision, recall, and F1-score, respectively. To evaluate whether SEBERTIS-trained DNNs can be used as ready-made security-related issue detectors, we calculate the performance of our best-performing fine-tuned model on 1,000 totally unseen, in-the-wild GitHub issue reports. Our analysis demonstrates that SEBERTIS-trained models are capable of confidently detecting just-submitted security-related issues, with 0.7123, 0.6860, and 0.6760 for precision, recall, and F1-score, comparable to those of prompting LLMs.

The contributions of our paper are outlined as follows.

- ⋆ We introduce a systematic procedure to identify Semantic Surrogates, a list of semantically replaceable keywords with prediction label occurrences in issue reports.
- ⋆ We design SEBERTIS, a framework for training DNNs as capable and confident security-related issue classifiers that have minimal dependency on lexical cues.
- ⋆ We conduct extensive analyses, demonstrating SEBERTIS's effectiveness in training accurate, robust, and generalizable security-related issue classifiers.

II. BACKGROUND

In this section, we introduce DNNs, discuss pre-trained language models, and briefly describe MLM training.

A. Deep Neural Networks

Deep neural networks (DNNs) are non-linear ML models composed of multiple layers with numerous learnable parameters [22]. Each layer in a DNN first applies a transformation on a collection of parameter values, including either all or a subset of the parameters of that layer (i.e., dense or mixture of experts (MoE) architectures) [23]. The transformation is then followed by an element-wise nonlinearity. The collection of these layers creates a complex, multiscale, distributed architecture which, in turn, significantly enhances the decision making strategy of the model, leading to impressive performances in classification tasks, in addition to robustness to adversarial perturbation of input data and skewed data distributions [22].

B. Pre-trained Language Models

A considerable population of DNNs has emerged as language models, pre-trained on vast textual corpora such as encyclopedias, enabling them to learn generic language features [24]. These pre-trained DNNs can be further adapted to downstream tasks through two main strategies. In the first strategy, the pre-trained architecture is integrated with an auxiliary, task-specific network to enhance task alignment. In the second strategy, a minimal set of new parameters is incorporated to the original architecture, followed by fine-tuning either the entire network or a subset of its parameters. A series of language models are unidirectional, such as OpenAI's *generative pre-trained transformers (GPT)* [25], capturing the relationships between words across the input text only from one direction and, in turn, generating the output autoregressively, token-by-token and from left-to-right [21]. Consequently, for tasks involving sentence inference or long-range comprehension, including issue classification, these models demonstrate a suboptimal performance [21]. In contrast, bidirectional transformers leverage context from both directions, achieving a higher classification performance over their unidirectional counterparts, particularly if they are fine-tuned on single tokens of input [21].

C. Masked Language Model Training

Google's *bidirectional encoder representations from transformers (BERT)* [21] is arguably one of the preeminent and widely adopted bidirectional transformer architectures for text classification tasks [26]–[29]. BERT capitalizes on *masked language model (MLM)* training to incorporate the input textual context from both directions. MLM randomly chooses and masks some tokens from the input and trains to predict those tokens from global dependencies of the surrounding context only using attention mechanisms [23], [29]. In other words, BERT learns a word's meaning not as is, but in the context of the semantics the word is used in. Due to its exceptional representation power that captures the long-range semantic dependencies of input texts, the application of BERT MLM has been investigated across varying tasks from sentiment analysis [30], to stance detection [31], multi-class text classification [26], and log anomaly detection [32]. Similarly, we also choose BERT to fine-tune it as an MLM on the downstream task of detecting security-related issues.

III. Methodology

We introduce SEBERTIS, a novel framework to train DNNs for automatically detecting security-related issues. Specifically, SEBERTIS trains bidirectional transformer architectures as MLMs, i.e., fine-tuning their parameters when occurrences of certain keywords are masked in the corpus to prevent the models memorizing lexical shortcuts. We choose these keywords such that they can correctly replace label names (i.e., security- and non-security-related) should they appear in the issues. In other words, we compile the list of keywords as semantically equivalent vocabulary to prediction labels, calling them Semantic Surrogates. Figure 1 illustrates an overview of SEBERTIS, consisting of *Semantic Surrogates Compilation* Ⓐ and *Masked Language Model Training* Ⓑ. We describe each step in detail below.

A. Semantic Surrogates Compilation Ⓐ

For compiling the Semantic Surrogates, after collecting a corpus of security- and non-security-related issue reports (A.1), we pre-process the submissions to prepare them for keyword extraction (A.2). Finally, we extract and rank keywords such that they can semantically represent issue label names (A.3).

1) Data Collection (A.1): To identify the vocabulary that semantically represents the issue label names (i.e., security- and non-security-related), we first need to prepare a balanced corpus containing enough instances from both categories. To do so, we have initially targeted a set of 10 popular repositories, such as *Microsoft vscode* and *tensorflow*, to excavate their issues. An issue with at least one relevant tag such as "security" or "vulnerability" (from hereon referred to as *security-tags*) is labeled as security-related, whereas an issue without any security-tags is labeled as non-security-related.

Guided by prior studies [33]–[35], we apply conservative project-level inclusion criteria to narrow down the initial population of selected repositories to a series of well-developed and well-maintained projects: (1) $\geq$ 1,000 stars; (2) active within the past year; (3) non-fork; (4) a substantive closed issues history; (5) ≥ 300 commits; (6) ≥ 50 contributors; and (7) ≥ 1 merged pull request. These thresholds are intended to favor mature projects which, in turn, ensures the quality and generalizability of our issue report corpus. However, we have quickly realized that even mature and well-known projects do not yield enough security-related issue reports. Security is relatively rare as a topic on issue trackers and security-tags are often missing, which further suppresses recall.

Next, we have attempted to keep the above project-level filters while collecting security-related issues across all qualifying repositories, rather than pre-selecting and filtering within a small set of popular projects. This process also results in too few instances with security-tags, especially from large repositories where security-tagged issues remain sparse. Given this, we have decided to apply the inclusion restrictions at the issue level. Specifically, we retain issues that (1) are not pull requests, (2) have nonempty titles and descriptions, (3) and are submitted between 2022-01-01 and 2024-03-01 so that the dataset reflects contemporary security concerns. This shift preserves breadth while enforcing quality, yielding a larger and more representative collection of security-related issues.

Because GitHub tags are author-defined and often inconsistent across projects and even across issue reports of the same repository, we treat them as a preliminarily signal rather than ground truth label names. Inspired from prior work on vulnerability disclosure by Kancharoendee et al. [36], we construct our initial set of security-tags to consist of "security", "vulnerability", "risk", "common vulnerabilities and exposures (CVE)", and "common weakness enumeration (CWE)". CVE refers to a standardized catalog that assigns unique identifiers to publicly disclosed software vulnerabilities [37]. CWE complements CVE by categorizing recurring design patterns and implementation flaws that lead to vulnerabilities [38].

We then conduct a round of systematic manual inspection of 250 GitHub issue reports drawn from a wide range of repositories and examine them for their tags that explicitly or implicitly indicate security relevance, considering both their linguistic meaning and their contextual use. From here, we identify four additional security-tags: "common vulnerability scoring system (CVSS)", "CVSS/high", "CVSS/medium", and "CVSS/low", where CVSS denotes a structured method for rating the severity of software vulnerabilities [39]. To broaden coverage while maintaining precision, we expand this core set of security-tags with *WordNet* [40], a large lexical database of English vocabulary that groups words into sets of synonyms and captures their semantical relationship. Starting from our core set of security-tags, we retrieve their synonyms and related terms, then manually vet the candidates, keeping only those that are clearly and consistently tied to security. Following this process, we identify four other security-tags: "exposure", "risk", "secure", and "vulnerable".

To extend our set of security-tags, we also experiment with *Word2Vec* [41], which is a technique that receives textual inputs and generates numerical vector representations of terms that have a high semantic similarity with the input.

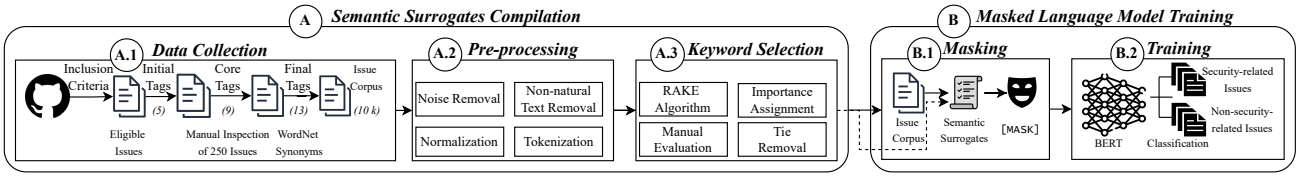


Fig. 1: The overview of SEBERTIS training framework

Specifically, we use Word2Vec to identify the words that are semantically similar to our core set of security-tags. Word2Vec largely reproduces the synonyms that are already identified from WordNet and does not yield any additional meaningfully distinct security-tags; therefore, we do not adopt it. We also consider conducting *snowball sampling*, an iterative procedure that expands the set of security-tags by incorporating new tags that frequently co-occur with existing seed terms [42]. However, we decide against adopting this technique since first it risks propagating semantic noise which, in turn, can lead to topic drift, and second it inherently suffers from sample bias which can result in similar samples repeatedly reinforcing one another. Following the above inclusion criteria and our final set of security-tags, we create a balanced corpus of 10,000 GitHub issue reports consisting of 5,000 security- and 5,000 non-security-related issues.

2) Pre-processing (A.2): Next, we pre-process our corpus. Following prior work on text pre-processing and software engineering text analysis, we (1) remove structural noise including Markdown headings and templates, HTML tags, checklists, file system paths, and raw URLs (removing any embedded hex codes while retaining their standalone meaningful terms) [43]; (2) normalize the text by lowercasing, removing numbers, special characters, and single-character tokens, and collapsing whitespace [44]; (3) filter out non-natural-language content to down-weight code and logs [45]; and (4) tokenize, apply part-of-speech-aware lemmatization, and remove English stop words [46].

3) Keyword Selection (A.3): For the pre-processed security- and non-security-related issue reports, we run the *rapid automatic keyword extraction (RAKE)* algorithm [47], which extracts key phrases from a body of text by ordering the importance of the terms based on their frequencies and co-occurrence relationships with the rest of vocabulary. For each issue report category, RAKE produces an ordered list of most important keywords, with their importance score, capturing the semantics of the corresponding context. We then manually evaluate both lists through iterative discussion sessions among three of the authors (the first, the second, and the corresponding authors) to remove keywords that are not meaningfully associated with their respective categories. The discussions continue until all inspectors confirm the relevance of emerged keyword lists. For keywords appearing in both classes, we assign the term to the category with the higher importance score (i.e., the category where the keyword appears more frequently); ties are discarded. Finally, we select the top 50 keywords from each category to form the list of Semantic Surrogates for that category. [1] Semantic Surrogates serve as concise, interpretable proxies for security- and non-security-related issue content in our training framework.

B. Masked Language Model Training (B)

Supervised algorithms train DNNs to learn a mapping between the input data and their corresponding ground truth labels. The training enables the model to correctly infer the context of unseen inputs and predict their labels accordingly. However, such training conditions the DNNs on the exact vocabulary used in the training data. Consequently, the model memorizes specific lexical cues to rely on them as decision shortcuts when making predictions. In other words, the presence of certain terms in an unseen test input would suffice for the model to predict a label, without truly understanding the underlying context. To prevent SEBERTIS-trained DNNs memorizing lexical shortcuts, rather than learning security-related context of issue reports, we set our framework to train the DNNs as MLMs over masked Semantic Surrogates.

Specifically, we analyze each issue report of our corpus to identify whether it contains any keywords from the list of Semantic Surrogates for the category the issue belongs to. If such keyword exists, we assign it with the issue category as its pseudo-label. Training the DNN on these keywords and their pseudo-labels would capture security-related semantics embedded in issue report context. However, words are contextualized entities, meaning that a word can be interpreted differently based on the context it is used in [29]. In other words, other than a linguistic meaning, (i.e., the usual interpretation), the pragmatic interpretation of words depends on their surrounding vocabulary co-occurrence relationships. For instance, consider the example discussed by Meng et al. [26]. In a general context, the term "exercise" is interpreted as physical activity. However, in a sentence such as ``She was exercising her rights by voting.'', "exercise" has a different pragmatic meaning from what is often inferred when the term is used. Thus, to train a DNN as a classifier that is truly independent from lexical cues, not only the model should learn the security-related semantics of issue reports; but it should also learn the corresponding vocabulary pragmatics.

For this purpose, following the technique proposed by Meng et al. [26], we hide any occurrences of Semantic Surrogates in the corpus, by replacing them with the special [MASK] token (B.1), before training the DNN as an MLM on their

[1] Semantic Surrogates are available in the replication package.

pseudo-labels (B.2). Similar to [26], we also train a BERT model with an additional linear classification layer, to minimize the cross-entropy loss function. Based on preliminary analyses, we fine-tune every learnable parameter.

IV. Experiment Design

To evaluate the effectiveness of SeBERTis in training DNNs as security-related issue classifiers, capable of learning context without memorizing lexical cues, we design a series of experiments investigating the following research questions.

RQ1: How good is SeBERTis in training DNNs as security-related issue classifiers? To respond, we breakdown the question into three parts.

RQ1.1: What is the best classification performance an SeBERTis-trained DNN can achieve?

Motivation: As a measure of the effectiveness of SeBERTis in training DNNs as security-related issue classifiers, we calculate the best performance our fine-tuned BERT MLM can achieve in detecting security-related issues.

Experiment Settings: We conduct training under a *cross-validation (CV)* [48] setting, splitting our corpus of issue reports into ten folds (i.e., 10-fold CV), where each fold is held out once as the validation set while the DNN is fine-tuned on the remaining nine folds. We analyze the classification performance of the fine-tuned model for each fold by calculating *precision*, *recall*, and *F1-score*. We report the overall performance of the SeBERTis-trained model as the statistical mean of these metrics across the ten folds. We use an NVIDIA T4 hardware accelerator with 15 GB of memory to execute the training pipeline. On average, the end-to-end training process, with a batch size of 32 and two worker threads, takes approximately 14 hours to complete.

RQ1.2: What are the optimal hyper-parameter settings for achieving the best classification performance?

Motivation: Similar to any other training framework, exploring different hyper-parameter settings to identify their optimal values is crucial for achieving the best classification performance. Thus, we are also set to find the optimal hyper-parameter settings for SeBERTis.

Experiment Settings: We evaluate whether increasing the number of times a DNN observers a training sample, affects its classification performance. For this purpose, we conduct the 10-fold CV across increasing counts of epochs. To identify the appropriate epoch range for our experiment, we compare the classification performance of BERT MLM when trained on a random 1,000 subset of our corpus against training on the full corpus. Based on prior recommendations [49], we set the optimal epochs count for training on the 1,000 data subset as three, resulting in 71% classification accuracy. Meanwhile, training on the full corpus with only three epochs yields a considerably less accuracy of 60%. Thus, we set the range of our training epochs to start from four and increase until no significant change for the classification performance is observed. We set other hyper-parameters including the learning rate the same as the original setting discussed for BERT [21].

RQ1.3: How do SeBERTis-trained DNNs perform compared to state-of-the-art issue classifiers?

Motivation: For truly investigating the effectiveness of SeBERTis in training DNNs as successful security-related issue classifiers, other than calculating the detection performance of the fine-tuned model, it is also necessary to evaluate its performance compared to other state-of-the-art issue classifiers.

Experiment Settings: We setup an experiment to compare the performance of SeBERTis-trained classifier in detecting security-related issues of our corpus against five (two ML- and three LLM-based) state-of-the-art issue classifiers. Our first baseline, AdaptIRC [50], adds adapters to transformer architectures and fine-tunes their parameters for classifying GitHub issues as bugs, enhancement requests, or questions. Our second baseline, is FastText, a specialized linear ML model which uses rank constraints and fast loss approximations to detect security-related issues [51]. For fair comparison, we evaluate these ML-based baseline classifiers using the same performance metrics as SeBERTis-trained DNNs.

We also compare the performance of SeBERTis-trained DNNs in detecting security-related issues with state-of-the-art LLMs. To account for varying LLM architectures, we choose different families of models (proprietary in comparison to open-source), with varying architectural complexity (dense in comparison to MoE), and of different sizes (hundreds of billions of pre-trained parameters in comparison to models with fewer parameter counts) to serve as LLM-based baseline classifiers. Specifically, we use OpenAI's GPT family of models, including GPT-4o-2024-08-06 and GPT-4.1-2025-04-14 (from hereon referred to as GPT-4o and GPT-4.1, respectively). These two LLMs are representatives of proprietary LLM-based baselines with MoE, large architectures with an estimation of more than 200 billion parameters. We specifically choose these two models, compared to the newer GPT-5 variant, as they are known to be the best-performing, most intelligent, non-reasoning LLMs [52]. we choose Meta's Llama-2-70B-chat (from hereon referred to as Llama) as the open-source LLM-based baseline with a dense, medium-size architecture.

We prompt the LLMs with each issue report of the corpus and instruct them to decide whether the issue is security-related. To ensure the effectiveness of our prompts, we conduct a series of prompt engineering tasks, including carefully handcrafting the prompts following the *context manager* structure proposed by White et al. [17] by providing detailed instructions to the model for achieving highest accuracy while performing a binary classification task in the context of issue reports. We then apply spelling and grammar checkers for increasing clarity and removing any possible ambiguities [53].

We set the LLMs with the same decoding parameters: temperature equal to 0.7, to enable response diversity while minimizing randomness [54], and context window limit to ten tokens, to enforce focused predictions, i.e., the LLMs are instructed to output one or zero for security- and non-security-related predictions, respectively. Regardless of the temperature value, LLM responses are non-deterministic by nature. This entails that an LLM's classification decision can change across

different prompting attempts for the same issue report [55], [56]. To account for this residual stochasticity in the LLM responses, we prompt the models three times for each issue report and calculate their performances in detecting security-related issues as the mean of precision, recall, and F1-score.

We use OpenAI's batch endpoint [2] to prompt GPT models. Each round of prompting for the full corpus takes 34 minutes on average to complete. The experiments cost USD 15.35 in total, with a token count of 13.34 million for inputs and 45.50 thousands for outputs (i.e., 13.39 million tokens in total). Llama experiments take an average of 8 hours and 41 minutes to complete using four AWS NVIDIA L40S hardware accelerators, at a cost of USD 8.30 per hour of execution.

RQ2: How important is the choice of masking vocabulary?
Motivation: SEBERTIS trains a DNN as an MLM by fine-tuning its parameters on a series of masked keywords. We believe the choice of masked terms affects the details of the context the DNN learns, and subsequently, affects its prediction performance. Thus, we mask the occurrences of a list of keywords that can semantically replace the prediction labels in issue reports (i.e., Semantic Surrogates). Such training ensures the classifier actually learns semantics and pragmatics of security-relevant context embedded in the issues and does not memorize lexical cues for predictions which, in turn, result in the misidentification of more complex issues. We are set to analyze the extent to which the choice of masked terms can affect the capabilities of a SEBERTIS-trained classifier in detecting security-related issues.
Experiment Settings: We compile a list of random keywords to replace them with [MASK] instead of masking Semantic Surrogates. We then use SEBERTIS to train the DNN as an MLM over the occurrences of randomly masked tokens. To identify the new set of keywords for masking, for each issue category we randomly choose 50 terms from the collection of vocabulary we have extracted using the RAKE algorithm (see Section III-A3). In doing so we ensure that (1) chosen random keywords for each category are mutually exclusive and (2) no vocabulary in the set of random keywords belongs to the previously compiled list of Semantic Surrogates. Thus, the difference in classification performance between a DNN trained with [MASK] tokens replacing Semantic Surrogates and one trained on randomly masked terms reflects the true importance (or lack thereof) of the masking vocabulary.

RQ3: How well does a SEBERTIS-trained DNN perform as an off-the-shelf security-related issue classifier?
Motivation: To evaluate how good is the performance of a SEBERTIS-trained DNN as an off-the-shelf classifier in detecting just-submitted security-related issues in real-time, we need to investigate the generalizability of SEBERTIS.
Experiment Settings: Following the workflow described in Sections III-A1 and III-A2, we create a balanced, in-the-wild dataset of 1,000 issue reports consisting of 500 security- and 500 non-security-related issues. The dataset is then used as a fully unseen test set to evaluate the detection performance of the best-performing SEBERTIS-trained classifier from RQ1, serving as a measure of our framework's generalizability. Additionally, we compare the performance of our ready-made classifier against the most popular form of off-the-shelf classifiers, i.e., LLM baselines when they are prompted to detect security-related issues of the in-the-wild dataset.

[2]https://platform.openai.com/docs/guides/batch

V. RESULTS

In this section we calculate the effectiveness and the generalizability of SEBERTIS in training DNNs as security-related issue classifiers. Additionally, we quantify the impact of the masking vocabulary on the efficacy of SEBERTIS.

RQ1: *How good is* SEBERTIS *in training DNNs as security-related issue classifiers?*

Observation 1 (RQ1.1): The results of the 10-fold CV training are shown in Table I. As can be observed, SEBERTIS is very effective in fine-tuning DNNs as security-related issue classifiers, achieving up to 0.9738-0.9849, 0.9897-0.9924, and 0.9814-0.9880 for precision, recall, and F1-score, respectively.
Observation 2 (RQ1.2): Table I also demonstrates the effect of increasing training epochs on the detection performance of the SEBERTIS-trained classifier. As can be seen, increasing the epoch counts, enhances the capabilities of the classifier in correctly detecting security-related issues, with six epochs of training resulting in the highest detection performance across all performance metrics (i.e., 0.9849, 0.9924, and 0.9880 for precision, recall, and F1-score, respectively). Additionally, setting the epochs=6 results in standrad deviation values of only 0.0067, 0.0031, and 0.0063 for precision, recall, and F1-score, respectively. These values are significantly lower than those obtained from training with fewer epochs, which averaged 0.0264, 0.0053, and 0.0106, indicating that six training epochs yields more stable detection performance.
Observation 3 (RQ1.3): Tables I and II demonstrate the detection performance of the SEBERTIS-trained classifier compared to those of the state-of-the-art baseline classifiers. For any training epochs, the SEBERTIS-trained classifier outperforms all baselines across all performance metrics. Specifically, our most accurate classifier (i.e., epochs=6) outperforms FASTTEXT and ADAPTIRC by 14.44%-96.98%, 15.40%-93.07%, and 14.90%-94.72% for precision, recall, and F1-score, respectively. Compared to GPT-4o, GPT-4.1, and Llama, our SEBERTIS-trained classifier achieves 39.49%, 42.88% and 74.53% higher F1-score, respectively. Improvements for precision and recall are 23.20%, 23.65%, and 63.71% and 36.68%, 38.68%, and 85.63% across GPT-4o, GPT-4.1, and Llama, respectively. The mean percentage improvement of our SEBERTIS-trained classifier across the ML- and LLM-based baseline classifiers is 38.41%, 48.10%, and 48.21% for precision, recall, and F1-score, respectively.
Observation 4 (RQ1.3): Table II also demonstrates the performances of our baselines in detecting security-related issues against each other. As can be seen, ADAPTIRC has the lowest detection rate with precision, recall, and F1-score of only 0.5, 0.5140, and 0.5074, on par with randomly predicting

TABLE I: The performance of SEBERTIS-trained classifier across 10-fold CV

		Mean			Standard Deviation (Std.)		
Masks	*Epochs*	*Precision*	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*
Semantic Surrogates	4	0.9738	0.9898	0.9815	0.0275	0.0057	0.0114
	5	0.9740	0.9897	0.9814	0.0252	0.0049	0.0098
	⑥	**0.9849**	**0.9924**	**0.9880**	**0.0067**	**0.0031**	**0.0063**
Random Keywords	6	0.8467	0.9436	0.8925	0.1099	0.0347	0.0693

TABLE II: The performance of SEBERTIS-trained and baseline classifiers

	Type	Classifier	Precision	Recall	F1-Score
Baseline	ML-based	ADAPTIRC	0.5000	0.5140	0.5074
		FASTTEXT	0.8606	0.8600	0.8599
	LLM-based	GPT-4o	0.7994	0.7261	0.7083
		GPT-4.1	0.7965	0.7156	0.6915
		Llama	0.6016	0.5346	0.5661
SEBERTIS	10-Fold CV	BERT MLM	**0.9849**	**0.9924**	**0.9880**

issue labels with an accuracy of 0.5. All LLM-based baselines collectively outperform ADAPTIRC a minimum and maximum of 11.57% and 39.59% for F1-score while neither of them can detect security-related issues with a better detection performance than FASTTEXT. FASTTEXT demonstrates the best baseline performance with a precision, recall, and F1-score of 0.8606, 0.8600, and 0.8599, exceeding others with improvements ranging from 7.66% to 72.12%, 18.44% to 67.32%, and 21.40% to 69.47%, respectively. As expected, among LLM-based baselines, GPT-4o demonstrates the best detection performance, surpassing Llama as the worst performing LLM by 32.88%, 35.82%, and 25.12% in precision, recall, and F1-score, respectively. Meanwhile, the difference in detection capabilities between GPT-4o and GPT-4.1 is only 0.36%, 1.47%, 2.43% for precision, recall, and F1-score, respectively.

Discussion: As evident from prior observations, SEBERTIS is very effective in fine-tuning DNNs as accurate security-related issue classifiers. As expected, and similar to several other training frameworks, the count of training epochs has a direct relationship with the classification performance of the fine-tuned DNN. However, regardless of the training epoch counts, our SEBERTIS-trained classifier consistently demonstrates a superior detection performance compared to either of ML- or LLM-based baseline classifiers.

***RQ1 Summary*:** SEBERTIS-trains models that significantly outperform LLMs and transformer-based security-related issue classifiers by an average of 38.41%, 48.10%, and 48.21% for precision, recall, and F1-score, respectively.

***RQ2:** How important is the choice of masking vocabulary?*

Observation 5: Other than the detection performance of the DNN when it is trained on masked Semantic Surrogates, Table I also demonstrates the model's performance when [MASK] tokens are replacing a series of random vocabulary in the corpus. The results show that the detection performance of SEBERTIS-trained classier declines when random terms are masked. Specifically, when replacing random keywords with [MASK] tokens instead of masking Semantic Surrogates, precision, recall, and F1-score decrease by 13.82, 4.88, and 9.55 percentage points, respectively. In other words, the choice of masking vocabulary has an impact of up to 38.27%-46.15% and an average of 43.03% on the success rate of SEBERTIS.

Observation 6: To evaluate whether the choice of masking vocabulary meaningfully contributes to the effectiveness of SEBERTIS, we assess if the decline of performance, from masking random keywords rather than masking Semantic Surrogates, is statically significant. To do so, we first establish whether the differences between the paired metric values across the CV folds, depicted in Table I, are normally distributed. For this, we conduct the *Shapiro-Wilk* statistical test [57], with the threshold of significance set to 0.05. The test demonstrates that the pair-wise differences for precision and F1-score do not follow a normal distribution (i.e., $p-value < 0.05$) while differences for recall paired values are normally distributed (i.e., $p - value > 0.05$). Following these observations, to assess the significance of the decline across the performance metrics, we conduct the non-parametric *Wilcoxon signed-rank* statistical test [58] for precision and F1-score and the parametric *Paired T* statistical test [59] for recall. We also apply the *Bonferroni* correction [60] on the threshold of significance (i.e., α/k with α being the confidence limit=0.05 and k being the count of testing hypotheses which is three accounting for precision, recall, and F1-score). For both tests, we correct the threshold of significance as $0.05/3 = 0.017$. The conducted statistical tests demonstrate that the detection performance of the SEBERTIS-trained classifier significantly decreases across all three metrics (i.e., $p - value < 0.017$) when parameter fine-tuning is done over randomly masked tokens rather than masked Semantic Surrogates.

Observation 7: As evident by the results in Table I, masking the random set of keywords also increases the standard deviation values across the performance metrics compared to when the DNN is trained on the masked tokens of Semantic Surrogates. Specifically, the ratios of sample variances across the performance metrics are substantially higher than the expected deviations, if the choice of masking vocabulary did not have a substantial impact on the detection performance of the SEBERTIS-trained classifier [61]. In other words, the classifier's performance variability significantly increases

when masks are assigned randomly. This, in turn, suggests that the choice of masking vocabulary indeed has a substantial impact on the success-rate of the SEBERTIS-trained classifier.

Discussion: Our experiment demonstrates that the choice of vocabulary for replacement with [MASK] is important for a successful training with SEBERTIS. Not only the detection performance of the SEBERTIS-trained classifier declines when ad-hoc vocabulary are masked, the variability of its performance also increases significantly, leading to substantially fluctuating predictions for the same issue report.

***RQ2 Summary*:** The success of SEBERTIS derives from two aspects: (1) training for predicting pseudo-labels of masks, to prevent lexical cue memorization, and (2) using Semantic Surrogates as masking vocabulary, to enhance prediction accuracy and stability.

A. *RQ3: How well does a* SEBERTIS*-trained DNN perform as an off-the-shelf security-related issue classifier?*

Observation 8: Table III demonstrates the detection performance of our best-performing SEBERTIS-trained classifier (*RQ1.1* and *RQ1.2*) in detecting security-related issues of the fully unseen collection of 1,000 in-the-wild issue reports. As can be observed, our classifier achieves a precision of 0.7123, demonstrating its confidence in correctly detecting security-related issues that it has never encountered before. On other hand, the recall of our classifier is measured slightly lower and at 0.6860, indicating its conservative behavior in flagging issue reports that their prediction probabilities are marginally greater than the decision threshold of 0.5.

Observation 9: Table III also demonstrates the performance of the SEBERTIS-trained DNN in classifying the in-the-wild dataset in comparison to the LLM-based baselines. Our classifier performs almost on par with the LLMs when they are used as ready-made detectors for security-related issues (i.e., 0.7123, 0.6860, and 0.6760 for precision, recall, and F1-score of our classifier compared to those across the LLMs, which averaged 0.7741, 0.7265, and 0.7336, respectively). This is particularly interesting due to the possibility of data leakage for LLMs, i.e., because of their training cutoff dates, the LLMs may have already observed the in-the-wild issues during pre-training on open-source GitHub repositories. Nonetheless, our SEBERTIS-trained classifier achieves comparable detection performance, despite the disadvantage of not having previously observed or learned from the in-the-wild dataset.

TABLE III: The performance of ready-made SEBERTIS-trained classifier and LLM-based baselines

	Type	Classifier	Precision	Recall	F1-Score
Baseline	LLM-based	GPT-4o	0.8536	0.8100	0.8040
		GPT-4.1	0.8583	0.8258	0.8218
		Llama	0.6104	0.5436	0.5751
SEBERTIS	In-the-Wild	BERT MLM	0.7123	0.6860	0.6760

Discussion: A SEBERTIS-trained DNN can be used as a ready-made classifier to detect recently submitted security-related issues, achieving a performance comparable to prompting LLMs while avoiding their associated inference costs.

***RQ3 Summary*:** A SEBERTIS-trained DNN performs as cost-effective counterpart to LLMs when used as ready-made classifiers on fully unseen data, detecting security-related issues with 0.7123, 0.6860, and 0.6760 precision, recall, and F1-score, respectively.

VI. PRACTICAL IMPLICATIONS

In this section of the paper, we discuss the implications of our results for researchers and practitioners.

Imp. 1: If fine-tuned properly, bidirectional transformers are much better issue classifiers than unidirectional architectures.

Observations 1 (RQ1.1) and 3 (RQ1.3) demonstrate that the SEBERTIS-trained classifier outperforms LLM-based baselines in detecting security-related issues within the training corpus. *Observations 8 and 9 (RQ3)* further show that the SEBERTIS-trained classifier generalizes effectively to fully unseen issue reports, achieving comparable performance to those of the LLMs. This is specifically important as these LLMs are possibly exposed to in-the-wild issues during pre-training while our SEBERTIS-trained classifier has not previously observed any of them. *Observation 4 (RQ1.3)*, on the other hand, illustrates the inefficacy of ADAPTIRC compared to the LLM-based baselines in detecting security-related issues. Similar to SEBERTIS, ADAPTIRC also fine-tunes the parameters of bidirectional transformers to train them as issue classifiers. However, among all classifiers used in this study, ADAPTIRC performs the worst, essentially as a random guesser and making predictions by flipping a coin with 0.5 accuracy. In other words, the best- and the worst-performing security-related issue classifiers are both bidirectional transformers, but fine-tuned with different training frameworks. This finding highlights the potential of these architectures to outperform unidirectional LLMs as issue classifiers, if proper frameworks are adopted for their fine-tuning.

To Developers: We recommend fine-tuning bidirectional transformers as ready-made detectors, instead of prompting LLMs, for monitoring issue trackers and assigning submissions to appropriate developer teams.

Imp. 2: The choice of masking vocabulary during MLM training significantly impacts the performance of the classifier.

Observation 5 (RQ2) illustrates the decline in the detection performance of the SEBERTIS-trained classifier when random vocabulary are replaced with [MASK], rather than masking Semantic Surrogates. Masking random keywords results in the decrease of all performance metrics, with precision demonstrating the most and recall showing the least decline (i.e., 13.82 and 4.88 percentage points, respectively). The statistical tests conducted in *Observation 6 (RQ2)* demonstrate that the decline in the detection performance is statistically significant, further emphasizing the role of training over masked Semantic Surrogates in compelling the model to base its predictions on non-spurious features. Additionally, *Observation 7 (RQ2)* reports the significant increase in the classifier's prediction variability, from training on masked Semantic Surrogates to training on random masks. These statistics highlight the impact of choosing context-aware vocabulary for masking during

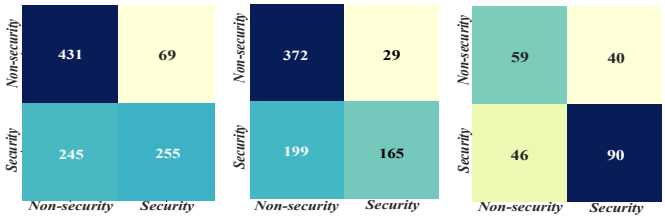
SEBERTIS, or similar MLM-based frameworks, for training not only accurate but also stable issue classifiers.

To Researchers: To implement an effective MLM-based training framework, we recommend compiling a list of semantically-related, context-aware keywords for masking. This approach enables the classifier to truly learn the details, semantics, and pragmatics of the context.

Imp. 3: Other than [MASK] tokens, the confidence of the classifier depends on [CLS].

Observations 6 and 7 (RQ2) discuss the impact of masking vocabulary on the accuracy and prediction variability of the SEBERTIS-trained classifier. If masked tokens are selected randomly or are not semantically relevant to the context, the classifier's prediction of the same issue may vary across different attempts. This stochasticity of predictions in return, undermines the confidence of developers in using the SEBERTIS-trained classifier as an effective automatic detection tool. Similarly, *Observations 8 and 9 (RQ3)* illustrate the confidence of the SEBERTIS-trained classifier as a ready-made detector in distinguishing security-related issues in real-time. Although less precise than when fine-tuned on data characteristics, our classifier detects security-related issues in real-time with precision that substantially surpasses Llama's and is comparable to the average across all LLM-based baselines.

To further investigate the underlying reasons for the decline of precision when our trained classifier is used on the in-the-wild dataset, we calculate the count of correctly classified security-related issues (i.e., *True-Positives (TP)*) and the count of *False-Positives (FP)* (i.e., false alarms for issues that report bugs with no security threats). Figure 2, illustrates the in-the-wild classification performance decomposition as *Confusion Matrices (CM)*. Figure 2a illustrates the TPs and FPs across the full population of the in-the-wild data. Figure 2b, depicts the TPs and FPs across the in-the-wild issues that contain at least one masked Semantic Surrogate while Figure 2c shows the counts of TPs and FPs for the issues that have no occurrence of Semantic Surrogates (i.e., zero masks). In the former setting, the ready-made classifier predicts an issue as either security- or non-security-related by conducting majority voting over the predictions for pseudo-labels of [MASK] tokens. For example, if a security-related issue is masked for three Semantic Surrogates, and for two of those the predicted pseudo-labels are security-related, the majority of the classifier's casted votes detect the issue as security-related, making it a TP. In the latter, as issues do not contain any Semantic Surrogates, there exists no [MASK] tokens for the classifier to predict their pseudo-labels and assign the prediction label for the issue category as their consensus. Consequently, the prediction decision falls back on the [CLS] token head (i.e., the standard classification layer in the architecture) similar to when the transformer is used as an off-the-shelf classifier [21]. As expected, the classifier's precision is much higher when it detects security-related issues by assigning the majority of votes over [MASK] token predictions (i.e., precision=0.7464) compared to predictions over the [CLS] head (i.e., precision=0.6373). The same goes for the recall of the ready-made classifier over [MASK] and [CLS] tokens as 0.7020 and 0.6341, respectively. For both settings, the performance metrics are calculated as weighted measures due to the population imbalance between the security- and non-security-related issue reports.



(a) Overall (b) [MASK] predictions (c) [CLS] predictions

Fig. 2: The in-the-wild CM for SEBERTIS-trained classifier
X axis: predictions, Y axis: labels

To Researchers: To improve the performance of the SEBERTIS-trained classifier as a ready-made detector, we recommend enabling majority voting over [MASK] token predictions for all issue reports. To do this, if Semantic Surrogates are not present in all issues, a second list of semantically equivalent vocabulary should be compiled to represent the labels for only those reports with no occurrences of Semantic Surrogates.

VII. THREATS TO THE VALIDITY

We discuss the threats to the validity of our study in a breakdown of construct, internal, and external.

A. Construct Validity

These threats pertain to the correctness of our measurements. To capture the effectiveness of SEBERTIS in fine-tuning DNNs as security-related issue classifiers, we conduct 10-fold CV training, ensuring of no data leakage between the held-out validation fold and the remaining training folds. Additionally, after the completion of training for each fold, fine-tuned parameters of the DNN are reset to pre-training weights so the mean of performance metrics across the folds are correct representations of the detection capabilities of the SEBERTIS-trained classifier. Nonetheless, precision, recall, and F1-score may not be proper choices for capturing the effectiveness of SEBERTIS. However, these metrics are frequently adopted in literature to calculate the performance of ML-based systems tailored for classifying issue reports [27], [28].

B. Internal Validity

Inherited challenges constitute internal threats to the validity of our study. To mine for security-related issues, we first identify the GitHub tags that their meaning is closely related to security, extending the list in three folds: (1) adding security-tags that are identified in literature, (2) manually inspecting issues for security-tags that have been missing in our initial set, and (3) retrieving security-relevant synonyms for the core tags from WordNet. While we aim to compile a comprehensive set of plausible security-tags, our list may not include all tags that are currently used or will be assigned to security-related issues

in the future. We follow a systematic procedure to compile Semantic Surrogates as a comprehensive list of context-aware substitutes for issue report labels. Our experiments (i.e., *RQ2*) also demonstrate the effectiveness of fine-tuning the DNNs over masked Semantic Surrogates rather than masking random keywords for creating a more accurate classifier with less prediction stochasticity. However, our compiled list of Semantic Surrogates may not capture every possible security-related semantics, such as certain jargon that may have not appeared in our corpus of GitHub issue reports.

C. External Validity

External validity concerns the generalizability of our study. SEBERTIS fine-tunes BERT as an MLM, conditioning the DNN to predict security-related issues over masked Semantic Surrogates. Fine-tuning as an MLM is only possible, if a DNN is trained with mask prediction objective during pre-training. This includes BERT and all its variants including but not limited to RoBERTa [62], ALBERT [63], and DeBERTa [64]. Additionally, any encoder-only transformer, such as multilingual DNNs with MLM head in pre-training [65], and encoder-decoder architectures with access to the [MASK] token [66] can be trained with SEBERTIS as security-related issue classifiers. As shown by our results (*RQ3*), in case of no Semantic Surrogate occurrence in issues, our SEBERTIS-trained classifier makes predictions over the [CLS] head. However, as discussed in Section VI-C, to increase the confidence of the classifier, a secondary list of semantically equivalent vocabulary to labels can be compiled to only cater to issues whose context is not captured by Semantic Surrogates.

VIII. RELATED WORK

In this section of the paper, we briefly discuss the literature on automatic issue classification techniques, dividing them to ML- and LLM-based approaches.

A. ML-based Issue Classification Techniques

Peters et al. [67] processed issue reports of Chromium and four major Apache projects including Ambari, Camel, Derby, and Wicket to create a dataset of 45,940 issues with correct GitHub tags. They removed any non-security-related issue with security-tags to enhance the detection performance of classifiers. Their dataset, was used by Alqahtani [51] to fine-tune FASTTEXT. The same dataset was also used by Das and Rahman [7] to evaluate the capabilities of ML models in detecting security-related issues. Several other studies also employed ML models to classify issue reports. Zou et al. [68] combined the discriminative power of ML models with meta-features to classify 23,262 issue reports of three major projects: Firefox, Thunderbird, and Seamonkey as security- and non-security-related. In another study conducted by Kallis et al. [69], ML models are used for multi-class classification of GitHub issue reports. ML models are also used for classifying submissions to Bugzilla and Jira issue trackers [70], [71].

B. LLM-based Issue Classification Techniques

Other than ML models, LLMs are also used for classifying issue reports. Aracena et al. [15] used GPT-3.5 Turbo for classifying issues of five GitHub projects. Building upon the previous study, Heo et al. [16] investigated the benefits of prompt engineering in effectively instructing a varying range of LLMs, including two different GPT variants and a Llama model, to classify GitHub issues. Du et al. [72] proposed LLM-BRC, a highly accurate LLM-based framework specialized for classifying issues of deep learning libraries, achieving an F1-score of 0.9875. Colavito et al. [73] compared the performance of GPT-3.5 with two bidirectional transformer architectures, including RoBERTa, in categorizing issue reports into four different classes of bug, feature requests, questions, and documentation. Their results demonstrated that although GPT-3.5 can achieve a relatively high accuracy, fine-tuned DNNs still outperform the LLM in the issue classification task.

Above studies mainly focus on detecting bugs from other submissions. Ones that target only security-related issues often suffer from suboptimal performance, limited generalizability across projects, or inability to classify newly submitted issues in real-time. In contrast, our framework trains classifiers independent of lexical cues or project-specific issue reports. Thus, a SEBERTIS-trained classifier can generalize across projects and serve as a ready-made security-related issue detector.

IX. CONCLUSION

In this paper, we introduce SEBERTIS, a framework for producing security-related issue classifiers. SEBERTIS trains a bidirectional transformer as an MLM by (1) effectively masking Semantic Surrogates, a list of semantically replaceable keywords with prediction labels, by replacing their occurrences in issues with the specialized [MASK] token and (2) fine-tuning the parameters of the model by optimizing the prediction objective over the pseudo-labels of masks. Compared to five state-of-the-art issue classifiers, varying from transformer architectures fine-tuned with different frameworks to autoregressive LLMs with different complexity and count of learnable parameters, the SEBERTIS-trained classifier demonstrates a substantially better performance in detecting security-related issues, outperforming the baselines by an average of 38.41%, 48.10%, and 48.21% for precision, recall, and F1-score, over a corpus of 10,000 GitHub issues. Our best-performing classifier achieves 0.7123, 0.6860, and 0.6760 for precision, recall, and F1-score, when classifying 1,000 fully unseen issue reports. In this context, the precision and recall of the ready-made classifier over [MASK] token predictions are 0.7464 and 0.7020. Even in cases of zero occurrences of Semantic Surrogates in unseen issues, our classifier can detect security-related submissions in real-time by incorporating predictions over the [CLS] head, as the fallback mechanism, with precision and recall equal to 0.6373 and 0.6341, respectively. The impressive success rate of SEBERTIS makes it a practical training framework for producing robust and generalizable issue report classifiers for real-time triage of issue trackers.

Future Work: We plan to improve the generalizability of SEBERTIS by replacing its [CLS] prediction fallback mechanism with predictions over a new set of masked keywords that are compiled to only represent the issues that do not contain any occurrences of the initial list of Semantic Surrogates. We also intend to evaluate the effectiveness of our training framework across different bidirectional transformers including variants of BERT. Finally, we plan to extend our framework to multi-label classification, not only detecting security-related issues but also categorizing them based on their severity and the underlying vulnerabilities they hint at.

X. Data Availability

To enable reproducibility, SEBERTIS code and analysis scripts are available at https://figshare.com/s/8fd18701f0de6d3429a7.

XI. Acknowledgment

This work was supported by Fonds de recherche du Québec (Grant No.2024-NOVA-346499 [74] and Grant No.363482 [75], and Natural Sciences and Engineering Research Council of Canada Discovery Grant (Grant No. RGCPIN-2022-03744). Additionally, this work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 97–106.

[2] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on GitHub," *Science of Computer Programming*, vol. 205, p. 102598, 2021.

[3] A. D. Sawadogo, Q. Guimard, T. F. Bissyandé, A. K. Kaboré, J. Klein, and N. Moha, "Early detection of security-relevant bug reports using machine learning: How far are we?" *arXiv preprint arXiv:2112.10123*, 2021.

[4] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.

[5] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech, "Software feature request detection in issue tracking systems," in *2016 IEEE 24th International Requirements Engineering conference (RE)*, 2016, pp. 166–175.

[6] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 121–130.

[7] D. C. Das and M. R. Rahman, "Security and performance bug reports identification with class-imbalance sampling and feature selection," in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, 2018, pp. 316–321.

[8] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2019.

[9] K. Goseva-Popstojanova and J. Tyo, "Identification of security related bug reports via text mining using supervised and unsupervised classification," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018, pp. 344–355.

[10] S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. Bowman, and N. A. Smith, "Annotation artifacts in natural language inference data," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018, pp. 107–112.

[11] P. Izmailov, P. Kirichenko, N. Gruver, and A. G. Wilson, "On feature learning in the presence of spurious correlations," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 38 516–38 532, 2022.

[12] R. T. McCoy, E. Pavlick, and T. Linzen, "Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3428–3448.

[13] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, 2023.

[14] K. Ma, H. Cheng, Y. Zhang, X. Liu, E. Nyberg, and J. Gao, "Chain-of-skills: A configurable model for open-domain question answering," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Jul. 2023, pp. 1599–1618.

[15] G. Aracena, K. Luster, F. Santos, I. Steinmacher, and M. A. Gerosa, "Applying large language models to issue classification," in *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, 2024, pp. 57–60.

[16] J. Heo and S. Lee, "A study on applying large language models to issue classification," in *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*, 2025, pp. 1–11.

[17] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with ChatGPT," *arXiv preprint arXiv:2302.11382*, 2023.

[18] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, 2020.

[19] N. Feng, H. J. Wang, and M. Li, "A security risk analysis model for information systems: Causal relationships of risk factors and vulnerability propagation analysis," *Information Sciences*, vol. 256, pp. 57–73, 2014.

[20] Y. Perwej, S. Q. Abbas, J. P. Dixit, N. Akhtar, and A. K. Jaiswal, "A systematic literature review on the cyber security," *International Journal of Scientific Research and Management*, vol. 9, no. 12, pp. 669–710, 2021.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[22] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[24] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. Van Durme, S. R. Bowman, D. Das *et al.*, "What do you learn from context? probing for sentence structure in contextualized word representations," *arXiv preprint arXiv:1905.06316*, 2019.

[25] A. Radford, "Improving language understanding with unsupervised learning," 2018. [Online]. Available: https://cir.nii.ac.jp/crid/1370302865745551633

[26] Y. Meng, Y. Zhang, J. Huang, C. Xiong, H. Ji, C. Zhang, and J. Han, "Text classification using label names only: A language model self-training approach," *arXiv preprint arXiv:2010.07245*, 2020.

[27] S. Bharadwaj and T. Kadam, "GitHub issue classification using BERT-style models," in *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, 2022, pp. 40–43.

[28] M. L. Siddiq and J. C. Santos, "BERT-based GitHub issue report classification," in *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, 2022, pp. 33–36.

[29] D. Nozza, F. Bianchi, and D. Hovy, "What the [MASK]? making sense of language-specific BERT models," *arXiv preprint arXiv:2003.02912*, 2020.

[30] H. Tian, C. Gao, X. Xiao, H. Liu, B. He, H. Wu, H. Wang, and F. Wu, "SKEP: Sentiment knowledge enhanced pre-training for sentiment analysis," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 4067–4076.

[31] K. Kawintiranon and L. Singh, "Knowledge enhanced masked language model for stance detection," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 4725–4735.

[32] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," *Applied Soft Computing*, vol. 146, p. 110689, 2023.

[33] T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's supercharge the workflows: An empirical study of GitHub Actions," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2021, pp. 01–10.

[34] W. Zou, W. Zhang, X. Xia, R. Holmes, and Z. Chen, "Branch use in practice: A large-scale empirical study of 2,923 projects on GitHub," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, 2019, pp. 306–317.

[35] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on GitHub Copilot," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2149–2160.

[36] S. Kancharoendee, T. Phichitphanphong, C. Jongyingyos, B. Reid, R. G. Kula, M. Choetkiertikul, C. Ragkhitwetsagul, and T. Sunetnanta, "On categorizing open source software security vulnerability reporting mechanisms on GitHub," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2025, pp. 751–756.

[37] "Common vulnerabilities and exposures (CVE)," https://cve.mitre.org/, MITRE Corporation, 2024, accessed: 2025-10-10.

[38] "Common weakness enumeration (CWE)," https://cwe.mitre.org/, MITRE Corporation, 2025, accessed: 2025-10-11.

[39] "Common vulnerability scoring system (CVSS)," https://www.first.org/cvss/, Forum of Incident Response and Security Teams (FIRST), 2025, accessed: 2025-10-11.

[40] "WordNet," https://wordnet.princeton.edu/, Princeton University, 2025, accessed: 2025-10-08.

[41] "Word2Vec," https://code.google.com/archive/p/word2vec/, Google Code Archive, 2013, accessed: 2025-10-08.

[42] L. A. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.

[43] C. P. Chai, "Comparison of text preprocessing methods," *Natural Language Engineering*, vol. 29, no. 3, pp. 509–553, 2023.

[44] A. Tabassum and R. R. Patil, "A survey on text pre-processing & feature extraction techniques in natural language processing," *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 06, pp. 4864–4867, 2020.

[45] M. V. Mäntylä, F. Calefato, and M. Claes, "Natural language or not (nlon) a package for software engineering text analysis pipeline," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 387–391.

[46] Z. Qiang, K. Taylor, and W. Wang, "How does a text preprocessing pipeline affect ontology syntactic matching?" *arXiv preprint arXiv:2411.03962*, 2024.

[47] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text Mining: Applications and Theory*, pp. 1–20, 2010.

[48] J.-H. Kim, "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap," *Computational Statistics & Data Analysis*, vol. 53, no. 11, pp. 3735–3745, 2009.

[49] H. Sajjad, F. Dalvi, N. Durrani, and P. Nakov, "On the effect of dropping layers of pre-trained transformer models," *Computer Speech & Language*, vol. 77, p. 101429, 2023.

[50] F. Ebrahim and M. Joy, "Few-shot issue reprot classification with adapters," in *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, 2024, pp. 41–44.

[51] S. S. Alqahtani, "Security bug reports classification using fasttext," *International Journal of Information Security*, vol. 23, no. 2, pp. 1347–1358, 2024.

[52] "Models," https://platform.openai.com/docs/models/compare, OpenAI Platform, 2025, accessed: 2025-10-05.

[53] G. Kamath, S. Schuster, S. Vajjala, and S. Reddy, "Scope ambiguities in large language models," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 738–754, 2024.

[54] "How should i set the temperature parameter?" OpenAI Platform Documentation, OpenAI, 2024, accessed: Oct. 14, 2025. [Online]. Available: https://platform.openai.com/docs/guides/text-generation/how-should-i-set-the-temperature-parameter

[55] OpenAI, "ChatGPT: Optimizing language models for dialogue," 2022.

[56] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, "LLM is like a box of chocolates: the non-determinism of ChatGPT in code generation," 2023. [Online]. Available: https://arxiv.org/abs/2308.02828

[57] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965.

[58] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics: Methodology and Distribution*. Springer, 1992, pp. 196–202.

[59] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[60] R. A. Armstrong, "When to use the Bonferroni correction," *Ophthalmic & Physiological Optics*, vol. 34, no. 5, 2014.

[61] R. A. Fisher, "On a distribution yielding the error functions of several well known statistics," in *Proceedings of the International Congress of Mathematicians*, vol. 2, 1924, pp. 805–813.

[62] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[63] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A litle BERT for self-supervised learning of language representations," in *International Conference on Learning Representations (ICLR)*, 2020. [Online]. Available: https://openreview.net/forum?id=H1eA7AEtvS

[64] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=XPZIaotutsD

[65] A. CONNEAU and G. Lample, "Cross-lingual language model pretraining," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[66] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 7871–7880. [Online]. Available: https://aclanthology.org/2020.acl-main.703

[67] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2017.

[68] D. Zou, Z. Deng, Z. Li, and H. Jin, "Automatically identifying security bug reports via multitype features analysis," in *Information Security and Privacy*, 2018, pp. 619–633.

[69] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket Tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 406–409.

[70] X. Wu, W. Zheng, X. Xia, and D. Lo, "Data quality matters: A case study on data label correctness for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2541–2556, 2021.

[71] W. Alhindi, A. Aleid, I. Jenhani, and M. W. Mkaouer, "Issue-labeler: an ALBERT-based Jira plugin for issue classification," in *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2023, pp. 40–43.

[72] X. Du, Z. Liu, C. Li, X. Ma, Y. Li, and X. Wang, "LLM-BRC: A large language model-based bug report classification framework," *Software Quality Journal*, vol. 32, no. 3, pp. 985–1005, 2024.

[73] G. Colavito, F. Lanubile, N. Novielli, and L. Quaranta, "Leveraging GPT-like LLMs to automate issue labeling," in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 469–480.

[74] "Towards reliable smart home ecosystems," 2024.

[75] "Combiner l'analyse statique et la génération de tests pour une assurance qualité améliorée : Une exploration des problèmes de compatibilité android," 2025.