IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 30 November 2023, accepted 28 December 2023, date of publication 4 January 2024,
date of current version 11 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3349577

RESEARCH ARTICLE

On-Device Smishing Classifier Resistant to Text Evasion Attack

JAE WOO SEO, JONG SUNG LEE, HYUNWOO KIM, JOONGHWAN LEE, SEONGWON HAN, JUNGIL CHO, AND CHOONG-HOON LEE
Samsung Research, Seoul 06765, Republic of Korea

Corresponding author: Choong-Hoon Lee (choonghoon.lee@samsung.com)

ABSTRACT Smishing (SMS phishing) is a cybercrime in which criminals send fraudulent messages, including malicious links, to steal the victims' private data or cause financial losses. The damage caused by smishing has become more severe, particularly with the proliferation of mobile devices. In smishing, a major difficulty faced by victims is discrimination between normal and smishing messages. To resolve this problem, we present an on-device smishing classifier based on a deep-learning model. In real-world scenarios, access to a substantial, authentic dataset is crucial. We trained and evaluated the classifier using real SMS datasets containing approximately 250,000 smishing messages and 950,000 normal messages obtained from victims in Korea. To ensure privacy, the classifier operates solely on mobile devices without externally transmitting any data. It utilizes a lightweight method that does not require significant computing power on mobile devices. We explored several models to determine a suitable model for mobile devices and optimized it using real datasets. Furthermore, our statistical analysis of actual smishing messages revealed that 98% of smishing messages are variants of previously sent messages. To address the prevalence of variant smishing messages, we propose a text evasion attack tool called EVA that is capable of generating pseudo-variant messages from a given message using an adversarial attack approach. We used this tool to evaluate and enhance the robustness of our classifier against various messages. Our classifier exhibited exceptional classification accuracy (0.99) while being lightweight (at 127 kB) and robust against variant smishing messages (attack success rate of 0.41).

INDEX TERMS Phone scams, smishing, classification, adversarial attacks, adversarial training.

I. INTRODUCTION

Phishing is a type of social engineering attack that aims to deceive people through various communication channels, such as email, voice calls, and SMS messages. Smishing (SMS phishing) is a type of phishing that involves scammers sending numerous bait messages by impersonating legitimate organizations or government institutions. They deceive people into clicking on malicious links, which leads to the downloading and installation of malicious applications (e.g., Android application packages). These apps are disguised as well-known applications such as banking, delivery, or shopping apps. Once installed, these malicious apps can steal the victim's private data, which can then be used to commit fraud or other cybercrimes, such as stealing the victim's money or launching a voice/messenger phishing attack based on the victim's contacts. The world is experiencing a rapid increase in the direct and indirect crimes caused by smishing. The number of phishing websites is on the rise every year, as reported by the Anti-Phishing Working Group (APWG) [1]. Notably, the United Nations (UN) reported a 350% increase in the number of phishing websites during the pandemic in the first quarter of 2020 [2]. The frequency and severity of smishing attacks are also on the rise, with Proofpoint's report showing a staggering 700% surge in smishing attacks in the first half of 2021 when compared with that in the latter half of 2020 [3]. South Korea, known as an IT powerhouse, has witnessed various smishing attack patterns owing to the widespread use of smartphones. The severity of these attacks is comparable to that observed in global smishing trends. In 2022, the Korea Communications Commission (KCC) stated that 93.4% of Koreans own

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

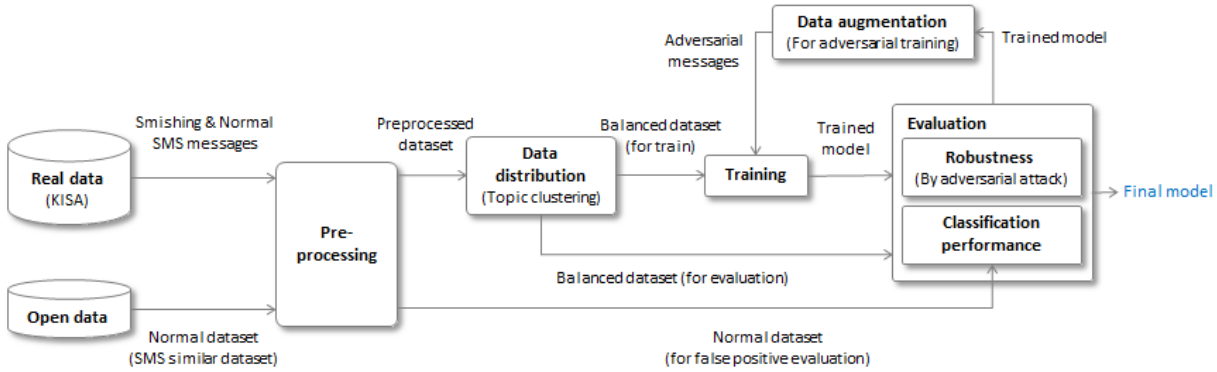


FIGURE 1. Pipeline for generating an on-device smishing classification model.

smartphones [4]. According to the Statistics Research Institute, the number of smishing-related incidents, including messenger phishing, increased from 2,963 in 2019 to 17,841 in 2021, with the damage amounting to 1,265 billion won in 2021, up from 587 billion won in 2020 [5].

Various detection methods have been proposed for reducing the damage caused by smishing [6], [7], [8], [9], [10], [11], [12]. Moreover, banks, financial institutions, antivirus software companies, and telecom providers have introduced commercial solutions/services to control smishing attacks [13], [14], [15], [16], [17], [18]. These include rule/learning-based detection, URL reputation checks, malware clustering, and attacker behavior analysis. Most methods have demonstrated considerable success in identifying smishing messages accurately. However, they have certain limitations. To create a reliable smishing classifier that can be used in real-world situations, access to numerous recent smishing datasets is essential. Unfortunately, it is not easy to collect such data continuously from victims. Previous studies did not use sufficient and up-to-date real smishing datasets for training and evaluation [6], [7], [8]. Alternatively, some researchers have explored the use of phishing URLs with a short lifespan [19], [20], [21], [22], [23] as well as spam messages resembling smishing [9], [10], [11]. These resources can be accessed through open-source intelligence (OSINT). However, the effectiveness of these methods for smishing classification in real-world scenarios remains unclear. Second, some commercial solutions for detecting smishing messages involve sharing targeted messages with cloud services [13], [14], [15], [16], which might overlook user concerns about potential privacy leakage. The importance of privacy is growing among users, and most users hesitate to exchange messages with cloud services. Privacy protection has become an essential consideration for frequent mobile-device users. Hence, solutions to identify smishing should protect the privacy of users as much as possible. Third, our analysis of the smishing dataset reveals that the majority of smishing messages are variants of previous messages (III-A). These variants typically involve changes in the URLs or minor modifications to the text. Previous studies have not considered methods to effectively detect such variants. To effectively combat the ever-changing characteristics of smishing attacks, it is crucial to build a robust detection solution that is capable of identifying these variations.

This paper presents an on-device smishing classifier that addresses the aforementioned limitations in combating smishing in the real world. The classifier adopts a learning-based approach and locally analyzes the content of the messages to identify potential smishing attempts. To construct an effective smishing classifier for practical use, we relied on our established partnership with the Korean Internet and Security Agency (KISA) to obtain the most recent smishing messages. Although our classifier is based on the Korean language, strategies to address the above limitations can be applied to other cultures and languages. To ensure privacy, our solution operates solely on mobile devices and does not transmit data externally. This approach inevitably requires a lightweight method that does not impose any burden on mobile devices. We explored several machine/deep learning models discussed in previous smishing detection studies to find a suitable model for mobile devices and optimized it with real datasets. To achieve robustness against variant messages (dominant among smishing messages), we adopted the textual adversarial attack methodologies employed in [24] and [25] and proposed a text evasion attack tool (EVA) that generates pseudo-variant messages from a given message. To the best of our knowledge, this is the first attempt at using adversarial attacks on smishing messages. This tool was used to enhance the smishing classifier and evaluate its robustness. For enhancement, variant messages created by the EVA tool were used for adversarial training to retrain the classifier. For the evaluation, the EVA tool was used to perform a textual adversarial attack on the classifier.

The sequential processes utilized in building our on-device classifier can be presented as a pipeline comprising data collection, preprocessing, data distribution, model training, model evaluation (to assess the classification performance and robustness against smishing variations), and adversarial training. Figure 1 illustrates this pipeline. The KISA and open datasets can be collected automatically using application

programming interfaces (APIs). The collected data were preprocessed for classification and evenly distributed for training, validation, and evaluation based on topic clustering results. The model was trained using both smishing and normal messages. The classification performance (accuracy, F1 score, false positive rate, and false-negative rate) and robustness of the trained model were evaluated. The model was then retrained using additional adversarial messages to address the vulnerabilities to smishing variant messages. Through this pipeline, we can automatically build an on-device smishing classifier that is resistant to text evasion attacks. This significantly reduces the workload and expenses and allows us to adapt to concept drift by continuously incorporating the most recent smishing dataset updates. Our classifier generated through the pipeline provides a high classification performance (accuracy of 0.99) with a lightweight model size (127 kB) and is robust against variant smishing messages (attack success rate of 0.41).

The details of each step in the pipeline are described in the following sections. In Section II, we describe the related research that supports the concepts utilized in this study. Section III examines the smishing and normal datasets through statistical analysis, feature selection, and trend analysis. In Section IV, we develop a lightweight smishing classifier suitable for mobile devices. In Section V, we present the text evasion attack tool (EVA), which is based on textual adversarial attacks. Finally, Sections V–VII describe the evaluation and improvement of the robustness of the lightweight smishing classifier by using the EVA tool.

II. RELATED WORK

A. SMISHING CLASSIFICATION

To prevent smishing, various studies have been conducted to identify and differentiate fraudulent messages sent by scammers from normal messages. The methods adopted in these studies can be categorized into URL-based and content-based approaches. URL-based detection examines the legitimacy of the URLs by detecting their abnormal lexical structure, linked content validity, and likelihood of containing malicious software downloads [19], [20], [21], [22], [23]. Content-based detection relies on the unique characteristics of the SMS content. These factors include morphological characteristics, readability, and keywords that can distinguish between smishing and normal messages. In general, these methodologies involve creating rules or training models using specific features [6], [7], [9]. Additionally, efforts have been made to understand the context of the messages by training models using text [8], [10], [11], [12]. Unfortunately, the classification performances of previous studies were evaluated using small and outdated smishing messages or messages similar to smishing, such as spam [26], [27].

B. TEXTUAL ADVERSARIAL ATTACK

Scammers who are aware of the presence of a smishing detector may try to evade detection. Several studies have been conducted on adversarial attacks that circumvent text classifiers. When a text is given, an attack explores an adversarial text that holds a similar meaning to the original text but is predicted to belong to a different class by the target model. Attacks can be performed in white-box or black-box environments; we are interested in the latter. In general, an adversary in a black-box environment injects perturbations that alter the original text to find the adversarial text without the target model's internal knowledge. Perturbation injection techniques are categorized based on the target object as character-level, word-level, or sentence-level attacks. Character-level attacks can deceive textual classifiers by manipulating characters through insertion, substitution, or deletion [28], [29], [30], [31], [32]. These characters are either substituted with visually similar characters or the word is intentionally misspelled to imitate typographical errors. Such disturbances can deceive the target model without significantly altering the original meaning perceived by humans. Word-level attacks mainly replace a particular token or word with a new one. They can be selected through synonym substitution [33], [34], [35] or based on a high salient score determined using similar word embeddings [36], [37], [38], [39]. In recent studies, masked language models such as BERT have been used to generate perturbed words for substitution in the same context [24], [25]. Most word-level attacks search for texts that can deceive the target model by modifying the original texts with perturbed words, which can also be seen as a problem in greedy searches for the loss of the target model. Sentence-level attacks involve replacing an original sentence with a new adversarial text generated using techniques such as paraphrasing [40], [41], [42], back-translation [43], or competitive dialogue agents [44]. These approaches often result in new sentences that do not retain the original meaning.

C. MITIGATION

Mitigation strategies have been proposed to address textual adversarial attacks by considering the unique characteristics of textual data. These methods detect adversarial texts using textual features, such as misspellings [45], grammatical errors [46], synonym substitutions [47], word frequency [48], and unnatural sentences [49]. Additionally, the encoding method assigns a distinct embedding vector to each synonym cluster [50]. However, these methods cannot be easily applied to smishing texts that use casual languages and informal writing styles. In addition, methods similar to those used in the image domain have been applied to reduce the effect of adversarial attacks on text. These methods include techniques such as smoothing and regularization [51], [52]; however, they can be applied only to certain models. Various methods have been proposed in previous studies, but adversarial training is considered a universal solution for combating textual adversarial attacks [53], [54], [55], [56], [57]. This can enhance the resistance of the model to adversarial attacks if a sufficient number of adversarial samples are gathered.

TABLE 1. Number of distinct smishing and normal data collected from 1Q 2018 to 3Q 2022.

Year	Smishing	Normal
18	3,782	4,241
19	13,743	4,781
20	139,779	90,541
21	92,734	809,952
22	4,867	39,637
Total	254,905	949,152

III. DATASET

To construct a smishing classifier and perform an adversarial attack, we used a real SMS dataset collected and maintained by the Korean Internet and Security Agency (KISA). In Section III-A, we analyze the dataset and explore the characteristics of smishing messages that differentiate them from normal messages. Additionally, we used Korean texts provided by open data platforms for conducting the false alarm test on various examples of normal data; the list of datasets is given in Section III-B.

A. KISA DATA

1) STATISTICS

KISA receives samples of suspicious smishing messages reported by victims through partners such as mobile telecom providers and antivirus software companies. The collected samples are classified as smishing if the included URLs lead to a malicious site (e.g., a malware distribution server or phishing site); otherwise, they are classified as normal. The collected and analyzed samples can be accessed through a Cyber Threat Analysis & Sharing (C-TAS[1]) system with permission granted by KISA. We used 254,905 smishing messages and 949,152 normal messages collected between Q1 2018 and Q3 2022 (Table 1). According to Table 1, there was a significant increase in smishing cases in 2020, coinciding with the onset of the COVID-19 pandemic. It appears that scammers attempted social engineering attacks by aggressively exploiting the vulnerable state of the mind of the public. During this period, smishing messages related to health and welfare increased rapidly (Figure 4b). In addition, the majority of Android package kits (APKs) distributed via smishing messages were hijacking and spyware apps, which extract private information or facilitate other scams, such as vishing (voice phishing). In 2021, there was a noticeable increase in the number of messages categorized as normal. This is considered to be the result of heightened public awareness regarding smishing owing to the harm it caused in 2020.

Most smishing messages consist of text and URLs. Embedded images sent through a multimedia messaging service (MMS) are beyond the scope of our study. By examining the dataset, we discovered numerous smishing messages that were altered by replacing the URLs or tweaking a small part of the text in previous smishing messages. These can be regarded as variants of the original smishing message.

[1]https://cshare.krcert.or.kr:8443/index

TABLE 2. Number of representative, variant-text, and variant-link smishing messages per quarter.

Year	Quarter	Representative	Variant -text	Variant -link	Total
18	1	77	494	162	733
	2	68	420	154	642
	3	85	950	294	1,329
	4	111	475	492	1,078
19	1	308	1,570	1,291	3,169
	2	187	1,568	1,285	3,040
	3	362	1,356	1,797	3,515
	4	488	2,086	1,445	4,019
20	1	362	6,745	18,524	25,631
	2	800	24,263	41,957	67,020
	3	400	1,646	23,602	25,648
	4	636	2,033	18,811	21,480
21	1	291	1,327	76,883	78,501
	2	198	1,249	9,768	11,215
	3	136	553	685	1,374
	4	179	653	812	1,644
22	1	74	226	1,917	2,217
	2	97	276	709	1,082
	3	76	744	748	1,568
Total		4,935	48,634	201,336	254,905

To distinguish such variants from the original messages, we used the metric edit distance rate (edr), which counts the character difference between the two texts by using the Levenshtein distance [58] and obtains the ratio for the length of the previously reported text as

$$\mathsf{edr}(t_a, t_b) = \frac{\mathsf{levenshtein}(t_a, t_b)}{\mathsf{len}(t_a)},$$

where, in the timeline, t_a is the previously reported text, t_b is the subsequently reported text, and $\mathsf{len}(t_a)$ is the number of characters in t_a. The metric edr measures the degree of perturbation between the two texts. For example, $\mathsf{edr} = 0.3$ indicates that the previously reported text t_a has an editorial difference of approximately 30% from the subsequently reported text t_b. We regard text t_b as a variant of text t_a if $\mathsf{edr} \leq 0.4$. The threshold was determined heuristically by observing smishing texts with similar semantics. In this study, we defined three types of smishing messages: representative, variant-text, and variant-link. A representative message includes text that is editorially distant from the text of all other previously reported smishing messages (i.e., $\mathsf{edr} > 0.4$). A variant-text message is a smishing message with the text part slightly modified from the text of the representative message. It is editorially similar to the text of the representative message (i.e., $\mathsf{edr} \leq 0.4$). A variant-link message is a smishing message that utilizes different URLs but shares identical text content with the previously reported message.

Table 2 shows the number of messages of each type in the collected smishing dataset, and Figure 2 illustrates the proportion of each type of message in each year since 2018. Interestingly, the average number of representative messages is only 1.94%. This means that approximately 98% of the smishing messages were reproduced from a small number of representative messages by modifying some text and/or altering the URLs. Upon analyzing the representative

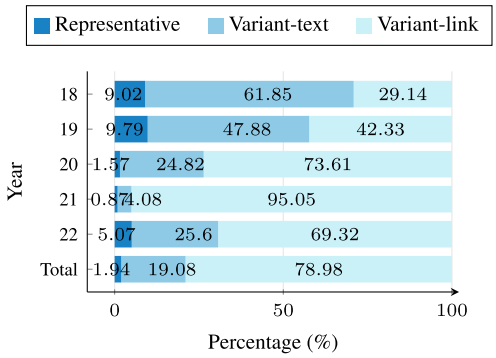


FIGURE 2. Ratio between representatives and variants (for texts and URLs) in yearly smishing messages.

messages, we found that 62% were reproduced more than twice, and the oldest message was reproduced for 49 months. The most frequently reproduced messages were regarding couriers, wedding invitations, and welfare, all of which are closely related to people's daily lives. Scammers targeted messages familiar to the public at that time when executing social engineering attacks. As shown in Figure 2, approximately 19.08% of the messages were alterations of the texts of the representative messages (i.e., variant-text), and 78.98% utilized different URLs for the same text as in the representative and variant-text messages (i.e., variant-link). Although variant-link messages constitute a significant part of the dataset, we focused on detecting variant-text messages, because the detection of variant-text messages also covered that of variant-link messages, as they share the same text. Various studies and solutions have been presented for malicious URL detection [19], [23], [59], [60]. Our focus was to identify smishing by analyzing the text of the messages.

2) FEATURE SELECTION

Smishing mimics messages familiar to the public. Owing to the similarities between smishing and normal messages, it is difficult for individuals to distinguish between them. However, most smishing messages consist of short sentences with poor grammar. Such writing styles can be used to identify smishing messages. We investigated the distinctiveness of smishing messages in comparison with normal messages by analyzing the spacing, URL position, text length, symbols, tokens, and other factors.

In Figure 3, we show six characteristics that can be used to differentiate smishing from normal messages, where impurities refer to any object such as symbol, letter, space, or line break that is inserted between Korean letters. Figure 3f shows that smishing texts tend to contain between 20 and 60 characters, whereas normal texts have more evenly distributed lengths. It appears that smishing scammers prefer relatively short texts, with an average of approximately 40 characters. Figures 3d & 3e show a tightly concentrated distribution of smishing that closely resembles the shape of the distribution shown in Figure 3f. This is because as the length of the text increases, it becomes more likely to include impurities and spacing. The Pearson correlation coefficient indicated a strong correlation between the text length and number of impurities (0.96) and spacing (0.91) in smishing texts. Figure 3d shows that impurities appear approximately five times in smishing texts. As shown in Figure 3a, smishing uses more impurities per unit of text length. During smishing, impurities can be intentionally added to messages to avoid detection by spam filters or to emphasize deceptive phrases and words. Figure 3e shows that the number of spaces used in smishing is typically distributed between 0 and 10, and Figure 3b shows that, in the unit text length, smishing uses a smaller number of spacings than normal texts. People often omit spacing for convenience when writing text messages. This aspect appears to be more prominent in smishing. Figure 3c shows that the ratio of symbol usage per unit text length is slightly different but not significant. Symbols were expected to be used more frequently in smishing to avoid spam filters. The KISA dataset categorizes spam messages (e.g., advertisements) without malicious URLs as normal, which could account for unexpected results. It appears that the attributes of normal messages, such as spam, are responsible for these results. The abovementioned characteristics can be effectively utilized to distinguish between smishing and normal text messages. Furthermore, a combination of these characteristics can enhance their effectiveness.

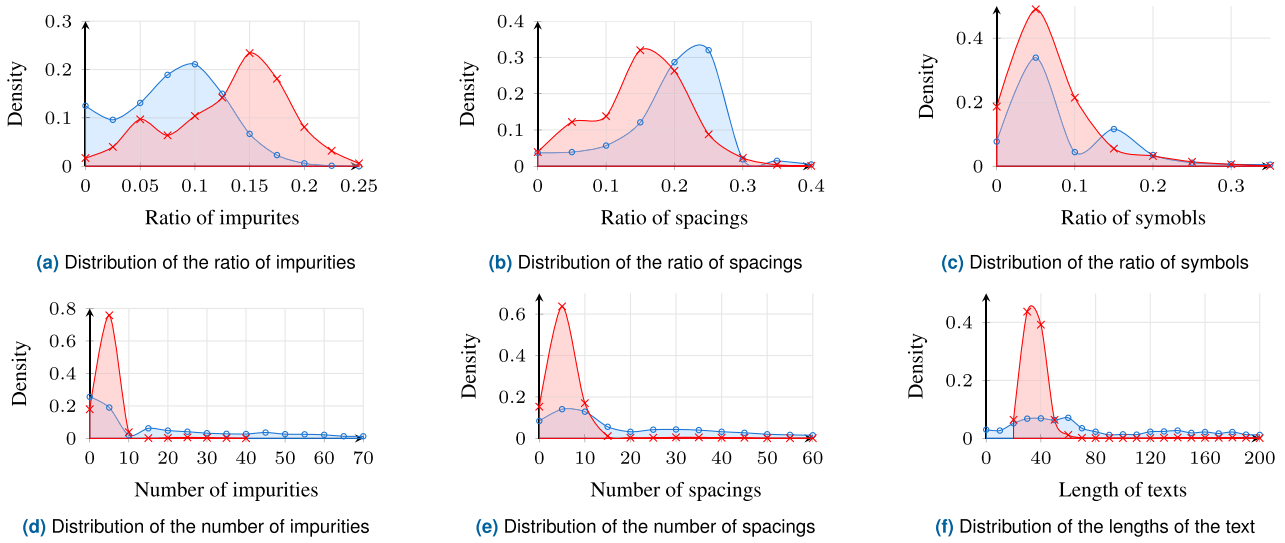
Relying solely on the aforementioned characteristics may not be sufficient to accurately identify smishing attempts. It is expected that the tokens commonly used in smishing and normal texts can aid in their classification. We refer to these as smishing-friendly and normal-friendly tokens, respectively. These tokens were extracted using the following process:

1) Tokenize texts by morphemes and leave only the nouns.
2) Calculate the frequency of tokens for smishing:

$$\mathsf{freq}(\tau) = \frac{\mathsf{smishing}(\tau)}{\mathsf{normal}(\tau) + \mathsf{smishing}(\tau)}$$

where τ is a token; $\mathsf{smishing}(\tau)$ counts how many times the token τ appears in the smishing texts, and $\mathsf{normal}(\tau)$ counts the same in the normal texts.
3) Exclude the following tokens
   - composed of only one letter,
   - total number of occurrence ($=\mathsf{normal}(\tau) + \mathsf{smishing}(\tau)$) < 100; the number of collected samples is not sufficient,
   - $0.4 < \mathsf{freq} < 0.6$; these are located near the borderline between smishing and normal.
4) Classify the remaining into
   - smishing-friendly token if $\mathsf{freq} \geq 0.6$,
   - normal-friendly token if $\mathsf{freq} \leq 0.4$.

From this dataset, 147 smishing-friendly and 348 normal-friendly tokens were extracted. For clarity, we have provided English translations of the Korean keywords in this paper. For example, smishing-friendly tokens that are commonly used in smishing include words such as "*invoice*,"



(a) Distribution of the ratio of impurities (b) Distribution of the ratio of spacings (c) Distribution of the ratio of symbols

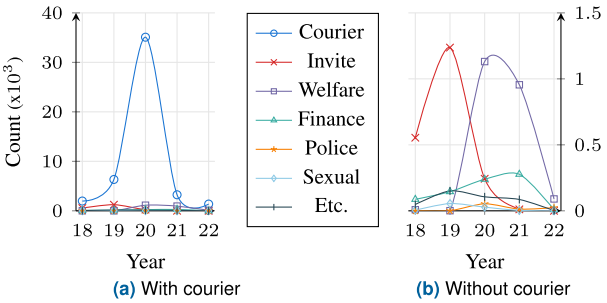
(d) Distribution of the number of impurities (e) Distribution of the number of spacings (f) Distribution of the lengths of the text

FIGURE 3. Comparison of the statistical distribution of each feature between the smishing and normal texts, where the red symbol (x) represents the density plot of the smishing texts, and the blue symbol (o) represents the density plot of the normal texts; in (a), (b), and (c), the x-axis denotes the ratio of the appearance of impurity, spacing, and symbol in a unit text length.

"*delivery*," "*item*," "*return*," "*post*," "*health*," and "*blessing*." However, normal-friendly tokens that are less frequently used in smishing include words such as "*adult*," "*mobile*," "*download*," "*cash*," "*authentication*," "*free*," and "*loan*." They are strongly biased toward either smishing or normal. In Section IV, we use the aforementioned characteristics and tokens as features in the machine learning models such as Naïve Bayes, Random Forest, and light gradient boosting machine (Light GBM), and present their classification performances.

3) TREND

We categorized the collected messages into various topics and detected any changes in the trends by observing their distribution. To do this, we removed duplicate messages with the same content (i.e., variant-link messages) and focused only on unique smishing texts. The number of unique smishing messages was 53,569. The topics were determined by manually analyzing the texts. We selected appropriate keywords related to each topic and categorized the smishing texts using these keywords. During categorization, we adaptively added new topics and their associated keywords. We completed this task with approximately 400 uncategorized messages remaining. The selected topics were courier, invitation, welfare, finance, police, and sexual. In Figure 4, Figure 4a shows the yearly count of smishing messages related to all topics, whereas Figure 4b excludes the courier topic. This is because the courier topic has a notably higher number of messages and presenting the data in two separate graphs improves the visualization. Scammers frequently use topics related to courier in smishing attacks. In particular, smishing became rampant during the COVID-19 pandemic of 2020, and courier-related smishing messages increased significantly when compared with that



(a) With courier (b) Without courier

FIGURE 4. Topic distribution of smishing messages from 2018 to 2022.

in the previous year. During this time, many people favored online purchases and courier services for their daily needs because of difficulty in going out and the desire to avoid close contact. This accelerated courier-related smishing attacks. Smishing messages related to courier services were about delays, product confirmation, and arrival and return notifications. In addition to courier-related messages, a significant proportion of smishing messages were related to welfare and wedding invitations. Figure 4b shows that in 2020, invitation-related messages decreased whereas welfare-related messages increased. People were cautious about inviting guests to wedding events and followed public health information messages issued by authorities. Most topics extracted from the smishing dataset were such that they provoked curiosity or anxiety among people receiving them.

B. OPEN DATA

The KISA dataset described in Section III-A contains messages that are suspected to be related to smishing. For this reason, the normal messages in the KISA dataset are similar to smishing messages from the viewpoint of users and do not reflect the various cases of SMS messages. Therefore,

we required a range of normal SMS messages to evaluate the effectiveness of the smishing detector. However, collecting personal messages is difficult because of privacy concerns. As an alternative, we utilized Korean public texts, similar to SMS. Owing to the efforts of previous researchers, we could use the datasets listed below:

- AI-Hub corpus[2]: 12 kinds of speech and dialogue (e.g., emotional conversations, customer responses, etc.) of 0.7 GB
- Modoo's corpus[3]: 6 kinds of Korean corpora (e.g., daily conversations, newspaper texts, etc.) of 19.4 GB
- KcBERT pre-training corpus[4]: Korean news comments of 12.1 GB

The above datasets are mainly composed of short texts pertaining to categories such as casual conversations, counseling, meetings, and news comments. However, these short texts are not real SMS messages. They are used only to evaluate various SMS text messages. We did not use them for training because they may contain unexpected patterns that are not present in real SMS messages.

IV. CLASSIFICATION

To minimize personal data leaks, we constructed an on-device smishing classifier that runs on a local device and does not transmit data. We selected the appropriate model according to the following criteria. First, model size is critical when considering an on-device solution. For instance, on-device smishing classifiers can be installed on various smartphones at both low and high costs. However, low-cost devices might not have sufficient memory to run the model, and a large-sized model may quickly drain the battery. The detector response time also depends on the model size. The larger the model, the greater the time required for reasoning. During the design and construction processes, we prioritized the size of the model as the most important factor. Second, accuracy is an important factor that should be considered. Our goal was to create a model with the maximum possible accuracy. However, it is important to note that false classifications (false positives and false negatives) can still occur. If false positives occur frequently, the classifier may become unreliable, as in the case of the shepherd boy in a fairy tale. This can result in the user losing trust in the classifier and possibly even deleting or deactivating it. In our model selection, we assigned high importance to a low false positive rate. Finally, it is important to update the model regularly or on-demand to address the concept drift effectively in smishing. However, the process of detecting changes in trends, analyzing them, and updating the model can be a time-consuming and costly process. To minimize them, we automate the model updating process without relying on manual input by experts. To achieve this, the model should be as easily adaptable to updates as possible.

[2]https://aihub.or.kr/

[3]https://corpus.korean.go.kr/

[4]https://www.kaggle.com/datasets/junbumlee/kcbert-pretraining-corpus-korean-news-comments

For the base model selection, we evaluated the most commonly used models for smishing detection. Table 3 lists the machine learning models of Naïve Bayes, Random Forest, and LightGBM, whereas the deep learning models included Word-CNN, Char-CNN, and KoELECTRA. We trained each model using our datasets and selected the model that best met the criteria.

For preprocessing, we simplified certain message components that could increase the classification complexity owing to their variations. URLs, file names, and call numbers were included in these components. We replaced them with specific mask strings: "`LINK`" for an URL, "`FILE`" for a file name, and "`CALL`" for a call number. For instance, if a message included two URLs, they were replaced with "`LINKA`" and "`LINKB`", respectively; this process was also followed for file names and call numbers. Masking can lower the classification complexity and prevent the model from being biased toward a specific string, such as a URL, file name, or call number.

For feature selection of the machine learning models, we utilized six characteristics and 495 (smishing-friendly and normal-friendly) tokens as classification features, as analyzed in Section III-A. However, for the Naïve Bayes classifier with the feature-independence assumption, we excluded highly correlated characteristics (i.e., number of impurities and number of spacings) and used only four characteristics and tokens. For Word-CNN, we used words in the messages as features. We split the messages into tokens and trained a word-embedding matrix (i.e., Word2Vec). Char-CNN recognizes text by using characters and memorizes them similar to images [61]. Its effectiveness in text classification makes it a valuable tool for smishing detection. For Hangul (i.e., the Korean script), Char-CNN uses the initial consonant, middle vowel, and final consonant as the features, whereas it uses the alphabet for English. Furthermore, Char-CNN also uses symbols, digits, spacing, and line breaks as its features. In total, Char-CNN used 215 features in this study. While [61] used one-hot encoding for feature vectorization, we opted for a character embedding matrix, which has a dimension of 48. Initially, the 215 features were randomly positioned in the 48-dimension and updated per epoch. Our choice of using a character embedding matrix instead of one-hot encoding was to minimize the model size. Finally, KoELECTRA is known to have a smaller size and competitive performance when compared with other transformer models [62]. We used a small KoELECTRA model, the `koelectra-small-v3-discriminator`. Its tokens were determined using WordPiece tokenization, as in BERT.

We followed the default settings of `scikit-learn-1.0.2` for the hyperparameters of the machine learning models; some of these hyperparameters were changed through repeated experiments to provide better performance. For both Word-CNN and Char-CNN, we used a one-dimensional CNN with one layer and added a feed-forward network with ten hidden cells. The CNN models

TABLE 3. Brief description and performance comparison of candidate models for on-device smishing classification, where the performance metrics are presented as the mean value of five trained models, with the standard deviation shown in parentheses.

Model	Feature	Performance				Size [Byte] (in Pytorch)	Hyperparameter
		ACC	F1 score	FPR	FNR		
Naïve Bayes	4 structural features 495 tokens	0.9698	0.9646	0.0125	0.0531	16k	`BernoulliNB(alpha=1.0)`
Random Forest	6 structural features 495 tokens	0.9891 (0.0018)	0.9874 (0.0022)	0.0078 (0.0013)	0.0149 (0.0027)	4.36M	`n_estimators=20, max_depth=None, min_samples_leaf=1`
Light GBM	6 structural features 495 tokens	0.9871 (0.0016)	0.9850 (0.0019)	0.0080 (0.0010)	0.0193 (0.0027)	360k	`n_estimators=100, num_leaves=31 boosting_type='gbdt'`
Word-CNN	words	0.9871 (0.0005)	0.9850 (0.0006)	0.0091 (0.0021)	0.0180 (0.0019)	57.8M	`embedding_dim=32, num_filters=128, kernel_size=5, hidden_size=10, global_max_polling1d`
Char-CNN	characters	0.9960 (0.0008)	0.9944 (0.0008)	0.0036 (0.0012)	0.0049 (0.0006)	483k	`embedding_dim=48, num_filters=192, kernel_size=12, hidden_size=10, global_max_polling1d`
KoELECTRA	wordpieces	0.9907 (0.0026)	0.9880 (0.0037)	0.0041 (0.0006)	0.0177 (0.0073)	53.9M	`pretrained=koelectra-small-v3-disc, hidden_size=256`

with multiple layers did not provide significantly better performance than the CNN model with one layer; some of them even showed a decrease in performance. Considering the sizes of the CNN models, we decided to stack only one layer, which was sufficient to classify smishing messages. For KoELECTRA, a feedforward network with 256 hidden cells was added at the end of the pre-trained model to predict the final class. The major hyperparameters of each model are listed in Table 3.

The KISA dataset explained in Section III-A was used to construct a comprehensive dataset for training, validation, and testing; we did not include open datasets in the model comparison, as they require much processing time and hence were not suitable for repeated testing. The validation dataset was used to select the best among the models generated at each epoch, and the test dataset was used to evaluate the classification performance of the selected model (Table 3). To prevent learning the same text repeatedly, we used only smishing messages with 55,369 unique texts (i.e., representative and variant-text messages). We retained 20% of the smishing messages for testing (10,713) and divided the remaining 80% in the ratio of 8:2 into training (34,282) and validation (8,574) data. The same ratio as in smishing was applied to normal messages. However, because the proportion of normal messages is overwhelmingly high when compared with the number of smishing messages, the trained models can be biased toward the normal class. To prevent this, a portion of the normal messages, approximately double the number of smishing messages, was utilized for training, which was a total of 59,571 messages. Additionally, 14,896 messages were used for validation, and 18,615 were used for testing. The messages used for training, validation, and testing were chosen randomly from nine clusters by using Latent Dirichlet Allocation (LDA), where, in a specific range of 3 to 20, the number of clusters was determined with the objective of maximizing the LDA coherence value. Topic clustering in LDA does not always cluster the samples with complete clarity. Nevertheless, it is sufficient to ensure that the model is not trained or evaluated for particular topics by using evenly distributing messages with similar meanings; in contrast, the topic clustering mentioned in Section III-A is analyzed manually.

To evaluate model performance, we trained five models under the same conditions of datasets and hyperparameters for each candidate model. Then, we compared their performance by calculating the mean and standard deviation of the population in Table 3. As for Naïve Bayes, we did not provide the standard deviation because it produces the same result under the same conditions. The standard deviations of the other candidate models were too small to be concerned about performance variations. The results were reviewed according to the three criteria mentioned above. First, it is worth noting that the Naïve Bayes, LightGBM, and Char-CNN models were all less than 1 MB in size. This implies that installing these models does not create a significant memory burden. Although Naïve Bayes had the smallest model size, its accuracy was the lowest. This could lead to a decrease in user trust. The difference in accuracy may seem small ($\leq$0.02), but it cannot be disregarded when considering the daily volume of messages received. To compare the computation speeds of a small-size model (<1 MB) and a large-size model ($\geq$1 MB), we measured the throughputs (i.e., predicted messages per second) of Char-CNN and KoELECTRA (1,179 and 329, respectively). Char-CNN was 3.6 times faster than KoELECTRA. Moreover, large models are expected to consume significantly more power than small models. Second, in terms of detection accuracy, all models except Naïve Bayes were equally effective. Among these models, Char-CNN showed the highest accuracy and lowest false positive/negative rate. Third, in terms of maintenance, machine learning models are not appropriate because, as smishing trends change, experts may need to select new features for each update, and a change in the model structure may also occur. This may necessitate the assistance of trained operators and present difficulties whenever a model is updated. In contrast, deep learning models are more suitable for on-device environments. In short, according to our criteria, Char-CNN demonstrated high classification

performance with a small model and supported easy update. We regarded the Char-CNN model as suitable for classifying smishing in the devices. To test the chosen Char-CNN model on mobile devices, we converted the Pytorch implementation to TensorFlow-Light (version 2.7.0). The transform relied exclusively on the built-in operators of TensorFlow-Light without the need for any TensorFlow core or custom operators. Additionally, the internal parameters, such as weights, were reduced through quantization. These allowed us to achieve a small-size model of only 127 kB. Despite this reduction, the classification performance was not affected or decreased. Moving forward, we refer to one of these five Char-CNN models as the base model. Its performance metrics are as follows: 0.9959 for accuracy, 0.9946 for F1 score, 0.0041 for false-positive rate, and 0.0043 for false-negative rate. The base model will serve as the target model for evading attacks in Section VI, and as the reference model for comparing adversarial-trained models in Sections VI & VII.

V. EVASION

In smishing, scammers may attempt to evade detection by applying perturbations to a small portion of the smishing messages. As shown in Section III, messages with perturbations in a small portion of the text (i.e., variant texts) have already been found in the real smishing dataset. We constructed a tool for text-based evasion attack (EVA), which imitates a scammer's attack behaviors in the real world and artificially constructs variant-text messages. We first introduce the threat model for an evasion attack (Section V-A), describe how the EVA injects various perturbations while interacting with a target model (Section V-B), and evaluate it by attacking the base model in Section IV (Section V-C).

A. THREAT MODEL

Our evasion attack proceeds in a black-box environment without knowledge of the inner information of the target model (e.g., hyperparameters, architecture, and training data). It imitates the scammers' attack environment in which they are unaware of the operation of a smishing detector in the real world. Under the black-box setting, we assume that the adversary (i.e., the scammer) can query the target model for arbitrary messages and obtain responses regarding their predicted classes and confidence scores. However, some intelligent adversaries might attempt white-box-based evasion attacks after extracting the internal information of the target systems using known attacks, such as side-channel attacks and active learning [63], [64], [65], [66], [67], [68]. Nevertheless, executing these attacks is not easy for scammers because of the challenges involved in model extraction, shortage of expert knowledge, defense against tamper-resistant techniques, and cost of a large number of queries. Hence, we assumed that scammers did not know the inner information of the target models.

During an attack, an adversary injects perturbations into the smishing message and asks the target model whether the modified message is smishing. The adversary's goal is to find perturbed messages recognized as normal by the target model under a limited condition whereby the perturbations do not alter the meaning of the original smishing messages. To measure the semantic similarity between an original message and its perturbed message, we consider both sentence similarity (i.e., the BERT score [69]) and edit distance rate (i.e., $\mathsf{edr}(\cdot,\cdot)$ in Section III-A). In previous studies [24], [25], [38], [70], sentence similarity was used to measure the semantical distance between two messages; however, we cannot fully trust the result because of low accuracy. This guarantees that the two messages have a somewhat similar meaning. To complement this, we measured the edit distance rate. Let $\mathsf{F} : \mathcal{M} \rightarrow \mathcal{C} \times \mathcal{D}$ be a target model, $\mathcal{M}$ be the messages allowed by the detector, $\mathcal{C}$ be the predictable classes, and $\mathcal{D}$ be the confidence scores. To achieve the goal, an adversary should satisfy the following two requirements.

- **Misclassified**: the perturbed message m_{adv} of an original message $m \in \mathcal{M}$ should be classified into $c_{\mathsf{adv}} \neq c$, where c is the predicted class of $\mathsf{F}(m) = (c, d)$.
- **Semantically similar**: the adversary can perturb the original message m only in a similar context without significant damage to the original meaning, i.e., $\mathsf{sim}(m, m_{\mathsf{adv}}) \geq \mu_{\mathsf{sent}}$ and $\mathsf{edr}(m, m_{\mathsf{adv}}) \leq \mu_{\mathsf{edit}}$, where $\mathsf{sim}(\cdot,\cdot)$ and $\mathsf{edr}(\cdot,\cdot)$ represent the sentence similarity and edit distance rate, respectively.

If the adversary generates a perturbed message that satisfies the above requirements, it succeeds in an evasion attack. We refer to this as an adversarial message.

B. ALGORITHM

In the field of text classification, evasion attacks have been studied on various topics [24], [25], [30], [31], [38], [71], [72], [73], [74], including sentiment analysis, spam filtering, and fake news detection. Their research focused primarily on a methodology that naturally injects perturbations from a human perspective. For evasion attacks against smishing classifications, we utilized existing methodologies that included injecting perturbations at the character and word levels (in `PerturbChar&Word`). We also introduced new perturbation techniques, such as breaking patterns (in `BreakPatterns`) and injecting structure-level perturbations (in `PerturbStruct`), which consider the characteristic patterns used in smishing and the unformatted writing style of SMS messages, respectively. The attack tool, EVA, takes the target smishing message m, target model $\mathsf{F}(\cdot)$, and thresholds of semantic similarity scores (μ_{sent} for sentence similarity and μ_{edit} for edit distance rate) as the inputs, and outputs an adversarial message m_{adv}. It comprises six phases: `Preprocess`, `BreakPatterns`, `PerturbStruct`, `ImportantTokens`, `PerturbChar`, and `PerturbWord`. EVA is described in Algorithm 1, and the related functions are given in Table 4.

In `Preprocess`, $\mathsf{fix\_spelling}(\cdot)$ fixes misspelled words and misused spaces in an input message m. SMS messages do not comply well with grammar and mainly use an

Algorithm 1 EVA

Input: smishing message m, target model $\mathsf{F}(\cdot)$, and thresholds: μ_{sent} for sentence similarity and μ_{edit} for edit distance rate

Output: adversarial message m_{adv}

```
Preprocess:
    ḿ ← fix_spelling(m)
    m̂ ← mask(ḿ)
    τ_pro = (ν_1, ν_2, ..., ν_n) ← tokenize(m̂)
BreakPatterns:
    τ_par ← remove(τ_pro, S)
    τ_par ← inject(τ_par)
    m_par ← compose(τ_par)
    if isSuccess(m_par) then return m_adv = m_par
    else κ_curr ← query_F(m_par)
PerturbStruct:
    τ_struct = τ_par
    P_struct ← line(τ_par)
    for ρ ∈ P_struct do
        κ ← query_F(ρ), (α, β) ← getSemantics(ρ)
        if (κ < κ_curr) ∧ (α ≤ μ_edit) ∧ (β ≥ μ_sent) then
            τ_struct ← updateTokens(τ_struct, ρ)
    m_struct ← compose(τ_struct)
    if isSuccess(m_struct) then return m_adv = m_struct
    else κ_curr ← query_F(m_struct)
ImportantTokens:
    for ν ∈ τ_struct do
        m_imp ← compose(τ_struct − {ν})
        κ_imp ← query_F(m_imp)
        if κ_imp < κ_curr then store (ν, κ_imp) in I
    sort I in ascending order of κ_imp
PerturbChar&Word:
    τ_token = τ_struct
    for (ν, _) in I do
        P_char ← insert_char(τ_token, ν) ∪
          del_char(τ_token, ν) ∪ sub_char(τ_token, ν) ∪
          insert_corr_symbol(τ_token, ν) ∪
          del_symbol(τ_token, ν)
        P_word ← insert_word(τ_token, ν) ∪
          del_word(τ_token, ν) ∪ sub_word(τ_token, ν) ∪
          swap_word(τ_token, ν)
        P_token ← P_char ∪ P_word
        for ρ ∈ P_token do
            κ ← query_F(ρ), (α, β) ← getSemantics(ρ)
            if (κ < κ_curr) ∧ (α ≤ μ_edit) ∧ (β ≥ μ_sent) then
                τ_best ← updateTokens(τ_token, ρ)
                κ_curr = κ
        m_token ← compose(τ_best)
        if isSuccess(m_token) then return m_adv = m_token
    return None
```

informal writing style. Thus, they include various types of typographical errors. Unlike well-formalized texts, such as news, articles, and emails, it is not easy to disassemble such messages into tokens. Fixing the spelling errors helps tokenize$(\cdot)$ to recognize and split the tokens. Before tokenization, EVA replaces the URLs, call numbers, and file names with specific strings, as described in Section III-A, and excludes them from the perturbations. To deceive the target model, EVA does not modify the URLs, call numbers, or file names. In addition, it replaces the spacings and line breaks with specific strings. This causes tokenize$(\cdot)$ to recognize them as tokens. Then, the tokenized spacings and line breaks are used for structure-level perturbations. The original strings of these masks are recovered when a sequence of tokens is reconstructed into a perturbed message using compose$(\cdot)$. tokenize$(\cdot)$ converts a spell-corrected and masked message into a sequence of tokens, $\tau_{\text{pro}} = (\nu_1, \nu_2, \ldots, \nu_n)$, where ν is a token. For the remaining processes, EVA injects perturbations for each token.

In `BreakPatterns`, EVA breaks specific patterns that frequently appear in most smishing messages. As the target models repeatedly learn frequent smishing patterns, they tend to predict messages with such patterns as strongly smishing. In these cases, even when the EVA injects a perturbation, the confidence score does not decrease significantly. Such patterns include specific nouns that mainly appear in smishing messages and URLs occurring adjacent to tokens having a certain meaning (e.g., contact, link, check, etc.). To increase the attack success probability, EVA breaks the patterns using remove$(\cdot,\cdot)$ and inject$(\cdot)$. The function remove$(\cdot,\cdot)$ deletes tokens that frequently appear in smishing but do not appear in normal messages (called strong smishing tokens, $\mathcal{S}$), which are determined based on high-ranked smishing-friendly tokens in Section III-A. Function inject$(\cdot)$ adds a token at the front/end of the URLs, where the tokens are chosen by the fill-mask task (i.e., BERT-MLM) without damaging the meaning of the context as much as possible. In some cases, breaking these patterns may succeed in generating adversarial messages. After breaking the patterns, compose$(\cdot)$ recovers the modified token sequence (τ_{par}) as a message and checks if it is an adversarial message or not; the success conditions are given in the description of isSuccess$(\cdot)$ in Table 4. If the attack succeeds, EVA stops the subsequent processes and returns the adversarial message m_{adv}; the success conditions are checked whenever a message is newly perturbed.

The SMS messages are written in an unformatted structure. Generally, line breaks can be positioned anywhere in a sentence. In addition, URLs, file names, and call numbers can be positioned anywhere, regardless of their meaning. When we manually attacked the smishing classifier by changing the structural features (e.g., adding/deleting line breaks and repositioning the URLs, file names, and call numbers), the addition/deletion of line breaks easily changed the prediction results in many cases, whereas repositioning did not help in decreasing the confidence score for smishing. Hence, EVA

TABLE 4. Functions used in EVA.

Classification	Function	Description
Evaluation	$\kappa \leftarrow \mathsf{query}_{\mathsf{F}}(m)$	Given a message m, it returns a (smishing) confidence score κ for the target function F.
	$(\alpha, \beta) \leftarrow \mathsf{getSemantics}(m')$	Given a message m', it returns an edit distance rate α and sentence similarity β with the original messages m.
	$\mathsf{True/False} \leftarrow \mathsf{isSuccess}(m)$	Given a message m, it checks the confidence score (κ), the semantics (α, β), and returns True if the confidence score $\kappa < 0.5$, the edit distance rate $\alpha \leq \mu_{\mathsf{edit}}$, and the sentence similarity $\beta \geq \mu_{\mathsf{sent}}$, otherwise False.
Message handling	$m \leftarrow \mathsf{compose}(\tau)$	It converts a sequence of tokens τ into a message m by composing them.
	$\tau \leftarrow \mathsf{updateTokens}(\tau, \rho)$	For a perturbed message ρ, it updates a sequence of tokens τ such that $\rho \leftarrow \mathsf{compose}(\mathsf{updateTokens}(\tau, \rho))$.
Preprocess	$\hat{m} \leftarrow \mathsf{fix\_spelling}(m)$	Given a message m, it returns a spell-corrected message $\hat{m}$.
	$\tau_{\mathsf{pro}} \leftarrow \mathsf{mask}(\tau)$	For a message m, it replaces the spacings (SP), line breaks (LI), URLs (LINK), call numbers (CALL), and filenames (FILE) with specific mask strings.
	$\tau \leftarrow \mathsf{tokenize}(m)$	For a message m, it disassembles the message with spacings, line breaks, URLs, filenames, call numbers, symbols, and parts of speech, and makes a sequence of them τ.
Pattern break	$\tau_{\mathsf{par}} \leftarrow \mathsf{remove}(\tau, \mathcal{S})$	Given a sequence of tokens τ and the set of high-ranked smishing-friendly tokens $\mathcal{S}$, it removes the strong smishing tokens $\nu \in \mathcal{S}$ in τ.
	$\tau_{\mathsf{par}} \leftarrow \mathsf{inject}(\tau)$	Given a sequence of tokens τ, it injects tokens/symbols at specific positions (e.g., at the front/end of the URLs).
Structure-level perturbation	$\mathcal{P}_{\mathsf{struct}} \leftarrow \mathsf{line}(\tau)$	Given a sequence of tokens τ, it inserts a new line break or deletes an existing line break. It finally outputs the set of perturbed messages, where each perturbed message includes only one line break addition/deletion.
Character-level perturbation	$\rho_{\mathsf{char}} \leftarrow \mathsf{insert\_char}(\tau, \nu)$	Given a token ν, it inserts characters and makes a typo ρ_{char}.
	$\rho_{\mathsf{char}} \leftarrow \mathsf{del\_char}(\tau, \nu)$	Given a token ν, it deletes a character and makes a typo ρ_{char}.
	$\rho_{\mathsf{char}} \leftarrow \mathsf{sub\_char}(\tau, \nu)$	Given a token ν, it replaces it with a slang commonly used on the internet (e.g., leet in English and yaminjungum in Korean).
	$\rho_{\mathsf{char}} \leftarrow \mathsf{insert\_corr\_symbol}(\tau, \nu)$	Given a token ν, it surrounds the token with correlated symbols (e.g., $[\nu]$, (ν), $\{\nu\}$, etc.).
	$\rho_{\mathsf{char}} \leftarrow \mathsf{del\_symbol}(\tau, \nu)$	If the given token ν is a symbol, it deletes the symbol.
Word-level perturbation	$\rho_{\mathsf{word}} \leftarrow \mathsf{insert\_word}(\tau, \nu)$	Given a token ν, it inserts a new token before/behind the token.
	$\rho_{\mathsf{word}} \leftarrow \mathsf{del\_word}(\tau, \nu)$	Given a token ν, it deletes the token in τ
	$\rho_{\mathsf{word}} \leftarrow \mathsf{sub\_word}(\tau, \nu)$	Given a token ν, it replaces the token using a masked language model (e.g., BERT-MLM).
	$\rho_{\mathsf{word}} \leftarrow \mathsf{swap\_word}(\tau, \nu)$	Given a token ν, it swaps the token with the adjacent words (back and forth).

includes structure-level perturbations to inject and delete line breaks, unlike previous studies [24], [25], [30], [31], [38], [73] in which such perturbations can only be considered in texts that are not grammatically well organized, such as SMS messages. In `PerturbStruct`, the function $\mathsf{line}(\cdot)$ generates a set of perturbed messages $\mathcal{P}_{\mathsf{struct}}$, where each perturbed message ρ in $\mathcal{P}_{\mathsf{struct}}$ includes the addition/deletion of only one line break. The function $\mathsf{line}(\cdot)$ turns a spacing into a line break to allow insertion or a line break into a spacing to allow deletion. Then, EVA selects the perturbed messages that have a lower confidence score than the current κ_{curr} within the allowed semantic similarity scores. An added/deleted line break of each selected perturbed message is accumulated in the token sequence τ_{struct} (lines 14 to 17); during the structure-level perturbation, to prevent the perturbed text m_{struct} from becoming unusual by including too many line breaks, we limited the number of added line breaks to three. EVA checks the success conditions for the perturbed message m_{struct} and proceeds to the next phase if it does not succeed.

Character- and word-level perturbations individually modify the tokens in a given token sequence τ_{struct}. For effective and efficient perturbation, it is necessary to identify which token is more important than the others for evasion attacks and then prioritize the perturbation of the token. `ImportantTokens` determines the priority of the tokens for a perturbation by using the method described in [31]. This is the preparation phase for the character- and word-level perturbations. It deletes each token in the target message, queries the target model, and measures the extent to which the confidence scores decrease from the current score. The greater the decrease in the score, the higher is the perturbation priority of the token. In other words, the perturbation priority for token ν is computed as

$$\mathsf{query}_{\mathsf{F}}(m) - \mathsf{query}_{\mathsf{F}}(m' = \mathsf{compose}(\tau - \{\nu\})),$$

where the negative sign indicates deletion of the token ν from a sequence of tokens τ. This method of linking important tokens has been commonly used in previous studies [24], [30], [38]. The tokens that increased the confidence score were excluded from the token pool of perturbations. Pool I remembers the tokens and their perturbation priority (ν, κ_{imp}) in the descending order of the perturbation priority κ_{imp}.

`PerturbChar&Word` sequentially perturbs the tokens according to the perturbation priority. For each token, the

perturbation functions produce perturbed messages through different strategies, such as insertion, deletion, and substitution, where $\mathcal{P}_{\text{char}}$ is the set of messages perturbed by character-level perturbations, and $\mathcal{P}_{\text{word}}$ is the set of messages perturbed by word-level perturbations. The perturbed messages are merged into a set $\mathcal{P}_{\text{token}}$. For all perturbed messages in $\mathcal{P}_{\text{token}}$, EVA finds the message that reduces the confidence score the most within the allowed semantic scores; EVA adopts one perturbation strategy for one token. If the found message m_{token} satisfies the success conditions, EVA returns it as an adversarial message m_{adv}. Otherwise, the token of the subsequent target is perturbed. This process continues until all the tokens in pool $\mathcal{I}$ are used. Character-level perturbations inject artificial typos (i.e., $\mathsf{insert\_char}(\cdot,\cdot)$, $\mathsf{del\_char}(\cdot,\cdot)$) or imitate writing styles typically used in SMS messages, which involve the use of symbols and Internet vocabulary (i.e., $\mathsf{sub\_char}(\cdot,\cdot)$, $\mathsf{insert\_corr\_symbol}(\cdot,\cdot)$, and $\mathsf{del\_symbol}(\cdot,\cdot)$). Typos are mainly caused by human mistakes, and symbols and Internet vocabulary are intentionally used for emphasis or fun or as abbreviations. Perturbations can pose a challenge for the target model in analyzing a message accurately and classifying it as smishing. To incorporate perturbations, $\mathsf{sub\_char}(\cdot,\cdot)$ replaces the target token with Internet vocabulary, such as *leet*, where the letters in a word are intentionally substituted with numbers or other characters. For Korean, we used *yaminjungum*, a Korean leet. Other perturbations handle characters and symbols by adding or deleting them, as in previous studies [30], [31]. For word-level perturbations, we borrowed methods used in previous studies [24], [25], [30], [38], [73]. Perturbations include word insertions, deletions, substitutions, and swaps ($\mathsf{insert\_word}(\cdot,\cdot)$, $\mathsf{del\_word}(\cdot,\cdot)$, $\mathsf{sub\_word}(\cdot,\cdot)$, and $\mathsf{swap\_word}(\cdot,\cdot)$). Perturbations are used to find a different expression while maintaining a meaning similar to that of the given message. Among these perturbations, word insertion and substitution require new words that do not disrupt the context of a given message as much as possible. For this purpose, we utilized a fill-mask task (i.e., BERT-MLM [75]). In word insertion, EVA places the mask in a position that requires a new word, extracts N candidate words using the fill-mask task, and inserts candidate words with high confidence scores. In word substitution, EVA replaces the target token with a mask, extracts N candidate words from the fill-mask task, and substitutes the target token with similar words in the BERT score.

C. EVALUATION

To evaluate the attack possibility of EVA, we attacked the base model described in Section IV. The target messages for the attack were selected from the representative messages described in Section III-A. Among them, we chose only the messages used in the training of the base model and tested whether EVA could generate adversarial messages from the already trained messages. The number of target messages for evaluating the EVA tool was 4,457, which included messages of various sizes between 36-byte and 1,628-byte (136-byte on average).

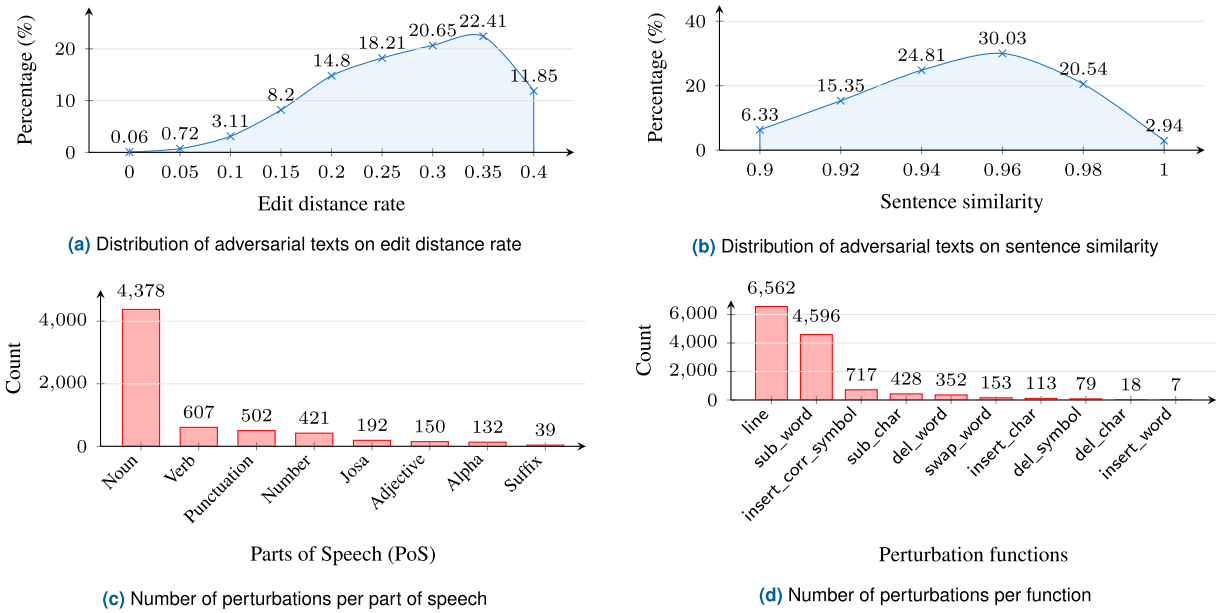
EVA requires setting up some parameters for the operation: the maximum number of attack attempts, language model for performing the fill-mask task and computing the BERT score, number of candidate tokens recommended by the fill-mask task, and threshold of semantic similarity scores. These settings were determined empirically. According to the default setting for comparison, we attacked only once per message, used `klue/bert-base` [76] as the language model, set the number of candidate tokens to 20, and applied semantic scores without restrictions (i.e., $\mu_{\text{sent}} = 0$ and $\mu_{\text{edit}} = 1$). To optimize the performance of EVA, we modified its parameters individually and selected those that yielded higher semantic similarity scores or attack success rates.

First, the maximum number of attack attempts was set to five. EVA is a probabilistic algorithm in which functions $\mathsf{remove}(\cdot,\cdot)$ and $\mathsf{inject}(\cdot)$ randomly select tokens for perturbation. That is, for the same target messages, EVA may succeed or fail. Even if an adversarial attack succeeds, different adversarial messages can be produced per attack for the same target message. In our empirical experiments, the target messages that could not generate adversarial messages within five attack attempts failed with a high probability, even if additional chances were given. Five attack attempts per text were sufficient to determine the success of the attack. In our test, if we succeeded in creating an adversarial message once in up to five attacks, we considered the attack as a success.

After generating adversarial messages from 4,457 target messages by adjusting the language models or number of candidate tokens, we found that the semantic similarity scores of the adversarial messages did not show significant differences. To verify the improvement more clearly according to the candidate models or number of candidate tokens, we extracted the target messages that produced semantically low-quality adversarial messages. We expected these target messages to be relatively robust to text perturbations. A language model or an appropriate number of candidate tokens capable of generating high-quality adversarial messages from these target messages was assumed to be suitable for evading attacks. For this, we attacked the base model under the above default setting and specifically selected 252 target messages that would generate adversarial messages with a sentence similarity score below 0.8 (average: 0.92) or an edit distance rate higher than 0.45 (average: 0.38). By taking these messages as input, we attempted to determine the language model and number of candidate tokens that improve the initial semantic similarity scores.

Second, we set the language model to `lassl/bert-ko-base`. The model can be downloaded from the Hugging Face model hub,[5] which provides models that perform various types of tasks in multiple languages. From the Hugging Face model hub, we downloaded and tested 29 language models

[5]https://huggingface.co/lassl/bert-ko-base



(a) Distribution of adversarial texts on edit distance rate

(b) Distribution of adversarial texts on sentence similarity

(c) Number of perturbations per part of speech

(d) Number of perturbations per function

FIGURE 5. Statistical analysis of the successfully perturbed texts on the base model.

supporting the fill-mask task in Korean, which were highly ranked models in `Most-Likes`. We attacked the base model by applying the downloaded language models to the fill-mask task and computing the BERT score. We also evaluated the adversarial messages generated from 252 target messages by measuring the average sentence similarity (using the same language model) and average edit distance rate. Finally, a language model with a higher mean sentence similarity and a lower mean edit distance rate was selected.

Third, the number of candidate tokens was set to $N = 10$. EVA was tested by changing the number of candidate tokens N using the language model `lassl/bert-ko-base`. We also evaluated the adversarial messages produced from 252 target messages in the same manner as in the language model. For $N \geq 10$, the adversarial messages had similar or lower quality when compared with the case of $N = 10$, that is, no significant improvement in quality was observed beyond $N = 10$. The tokens with lower confidence scores predicted by the fill-mask task did not help create high-quality adversarial messages in smishing. EVA uses only ten candidate tokens with higher confidence scores for the fill-mask task.

Finally, we set the threshold of the semantic scores as the sentence similarity $\mu_{\text{sent}} \geq 0.9$ (i.e., the BERT score) and the edit distance rate $\mu_{\text{edit}} \leq 0.4$ (i.e., the Levenshtein distance rate). Previous studies [24], [38] have constrained the sentence similarity for a successful attack to a maximum of 0.8. However, in the case of smishing messages, perturbed messages with a sentence similarity of less than 0.9 could not preserve the original semantics; in most cases, they were simple combinations of meaningless words. In smishing, the short length (unlike other types of samples such as emails, articles, news, etc.) and informal writing style might disturb the fill-mask task for predicting proper tokens. We used a higher threshold than that used in previous studies to obtain better semantically similar messages. For the edit distance rate, perturbed messages of more than 0.4 could generally break the semantic similarity. This means that the meaning can be preserved to a reasonable extent in smishing, even if at most 40% of the original message is altered into other characters.

The EVA was evaluated on an Intel Core TM i7-11700 (2.50 GHz) with NVIDIA GeForce RTX 3080 GPU (CUDA v11.6). For the 4,457 representative messages, we implemented EVA under the following settings: number of attack attempts as five, language model as `lassl/bert-ko-base`, number of candidate tokens as ten, and thresholds for the semantic scores as $\mu_{\text{sent}} \geq 0.9$ and $\mu_{\text{edit}} \leq 0.4$. It took approximately two hours to complete the attack. The EVA succeeded in generating 3,636 adversarial messages and failed for 776 target messages, whereas the remaining 45 target messages ($\simeq 1\%$) were falsely predicted to be normal by the base model without perturbation. Except for messages that were already predicted to be normal, the attack success rate was 0.8241; for the five Char-CNN models trained in Section IV, the mean attack success rate was 0.8018, with a standard deviation of 0.0206.

Figure 5 shows the statistical analysis results for 3,636 adversarial messages. The quality of the perturbed messages was evaluated using the edit distance rate and sentence similarity. Figure 5a shows the edit distance rate distribution of adversarial messages. The edit distance rate was quantized at 0.05 units to provide a clear visualization. Adversarial messages with approximately 0.35 edit distance rate accounted for the most significant proportion (approximately 22.41%), and most adversarial messages (approximately 76.07%) were generated between the edit distance rates of 0.25 and 0.35. Additionally, approximately 4% of the adversarial messages

succeeded in the attack by altering the original messages within 0.1 edit distance rate. In Figure 5b, the distribution of sentence similarities for adversarial messages is displayed by quantizing them in 0.02 units. Adversarial messages with 0.96 sentence similarity account for the most significant proportion (approximately 30.03%), and more than half of the adversarial messages (approximately 53.51%) have a sentence similarity of 0.95 or higher. Compared with the original messages, on average, the adversarial messages had 28% of editorial changes (i.e., approximately 20 letters) and preserved 95% sentence similarity. Figure 5c shows the frequency of each part of speech consumed for the perturbations, where the parts of speech were assigned according to open Korean text (`Okt`). The rank of each part of speech corresponded to its effectiveness in evasion attacks, with higher ranks being more impactful. As a result, nouns overwhelmingly prevailed, and the results for the rest were relatively similar. The base model for classifying smishing primarily relies on identifying crucial nouns that play a key role in differentiating smishing from normal messages. Figure 5d displays how often each perturbation function is used for the perturbations. Structure-level perturbation $\mathsf{line}(\cdot)$ is the most effective for most adversarial messages, followed by word-level perturbation $\mathsf{sub\_word}(\cdot,\cdot)$. Other functions did not significantly contribute to the attack. As expected from a manual attack, $\mathsf{line}(\cdot)$ is the most effective perturbation function. As shown in previous studies [24], [25], [38], the function $\mathsf{sub\_word}(\cdot,\cdot)$ was also used effectively in our attack. Word insertion was useful in [24], but the function $\mathsf{insert\_word}(\cdot,\cdot)$ was not very impactful in our smishing attack. In smishing, the inserted words may not be able to deceive the target model effectively because of the existing surrounding words. The contributions of various perturbation types were as follows: structure-level perturbation (50%), word-level perturbation (39%), and character-level perturbation (11%). Structure-level perturbations might be mitigated by preprocessing (e.g., removing line breaks), but character- and word-level perturbations are not. In Section VII, we discuss a method for alleviating this problem.

When we attacked the base model without braking patterns (`BreakPatterns`) and structure-level perturbations (`PerturbStruct`), as in previous studies, the attack success rate was 0.3497 (0.4744 lower than that with the proposed approach). It primarily failed to generate adversarial messages within the designated semantic similarity scores. Additionally, the perturbation priority for tokens could not be computed in some instances because removing one token does not lower the confidence score. Simply injecting character- and word-level perturbations is less effective in perturbing smishing messages within the allowed semantic similarity scores.

VI. ALTERNATIVE TO VARIANT

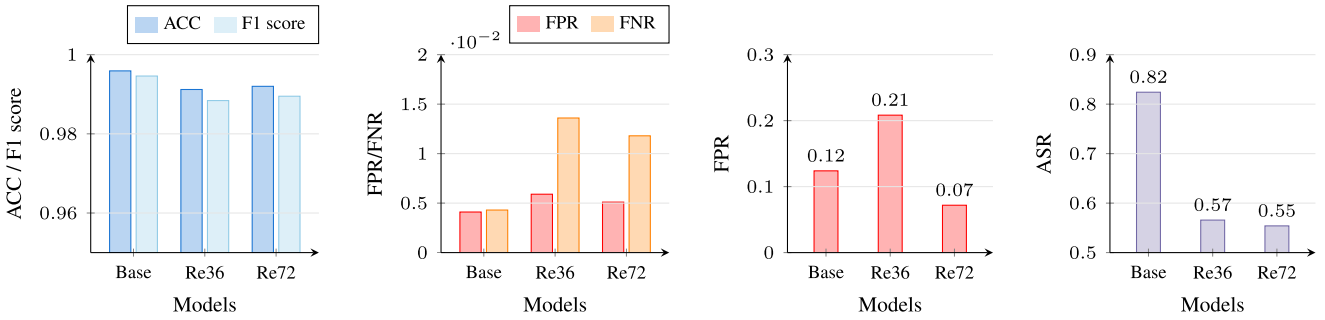
In Section V-C, we showed that EVA can successfully attack the base model. This section demonstrates that the adversarial messages generated by EVA closely resemble real variant-text messages; hence, they can be used as an alternative to real variant-text messages when training the models. For this purpose, we conducted the following experiments:

1) Train a model using the representative messages without including the variant-text messages; other settings, including the normal dataset, are the same as those for the base model. We call this the representative model.
2) Collect adversarial messages reproduced from the representative messages using EVA, which interacts with the representative model.
3) Retrain the models using the representative messages and the collected adversarial messages. We call these the retrained models; the initial conditions were fixed with the same random seed used during the base model training to exclude unnecessary effects.
4) Compare the performance between the base and retrained models.

In the above experiment, the retrained models did not utilize the variant-text messages; the adversarial messages were extracted through the representative model, and retraining was performed using only the representative and adversarial messages without the variant-text messages. We checked whether the models that were retrained with adversarial messages had a negative impact on the performance when compared with the base model. We found that the retrained models were comparable to or better than the base model. This experiment showed that adversarial messages can substitute real variant-text messages for training. In addition, this guarantees that they will not cause a significant negative effect on the classification when used in adversarial training.

To conduct the experiment, various adversarial messages were collected using EVA. However, EVA is designed to generate one adversarial message per target message by selecting the perturbation that reduces the confidence score most significantly among the allowed semantic scores. Even if the functions ($\mathsf{remove}(\cdot,\cdot)$ and $\mathsf{inject}(\cdot)$) in Algorithm 1 are probabilistic, their range of variation is relatively small. Therefore, it is necessary to produce diverse sets of adversarial messages repeatedly using EVA. For this purpose, we modified two parts of Algorithm 1. First, in lines 33–37 of Algorithm 1, we modified the EVA to choose not the best perturbation but a perturbation at random: `selection_mode=random (best/random)`. In this case, the EVA randomly selects one of the perturbations that reduce the confidence score to within the acceptable semantic scores. Second, we ran the EVA up to 25 times per target message to collect several adversarial messages: `max_collection=25`. We defined the EVA of Algorithm 1 as *attack mode* and the modified EVA as *collection mode*.

We used 4,457 representative messages, described in Section V-C, as training datasets for the representative model and as target messages for the EVA. We collected 72,000 adversarial messages in the collection mode. Then, using 72,000 adversarial messages, we trained two models, Re36



(a) ACC/F1 on the KISA test dataset (b) FPR/FNR on the KISA test dataset (c) FPR on the open dataset (d) Attack Success Rate (ASR)

FIGURE 6. Performance (accuracy, F1 score, false positive/negative rate (FPR/FNR), and attack success rate) of each model: the base model (Base), Re36, and Re72.

and Re72, according to (3). Re36 was trained using both representative messages and 36,000 adversarial messages randomly selected from the 72,000 adversarial messages. The number of adversarial messages used in the training was comparable to the number of variant-text messages (approximately 29,825) used in the base model training. Re72 was trained using 72,000 adversarial messages, corresponding to approximately 2.4 times the variant-text messages used in the base model training.

Figure 6 shows the performance measurement results (i.e., accuracy, F1 score, false-positive rate, and false-negative rate) for the three models: the base model, Re36, and Re72. Figure 6a and 6b present the results obtained from the KISA test dataset (described in Section III-A). The retrained models exhibited a higher false-negative rate than the base model. Consequently, they demonstrated lower accuracy and F1 score. The adversarial messages resemble the representative smishing messages more closely than the original variant-text messages because EVA alters only a small portion of the representative message. In models retrained with these adversarial messages, it appears that an insufficient number of smishing cases results in a relatively high false-negative rate. However, the performance differences among the models were not sufficiently large to indicate that any one model was superior. The differences were at most within 0.01: 0.0047 for accuracy, 0.0062 for F1 score, 0.0022 for false-positive rate, and 0.0093 for false-negative rate. The distance between the adversarial messages and the original variant-text messages was not too large. We evaluated how accurately the models classified normal SMS messages that were not spam. For this purpose, we utilized short text messages in the open dataset (approximately 131 million messages, as described in Section III-B). These were not real SMS messages but could be utilized to indirectly evaluate the models because of their resemblance to typical SMS messages. Figure 6c shows the false-positive rate measured in the open dataset, where Re72 provided the lowest false-positive rate. The difference from the base model value was 0.0521, which was not negligible. It is worth noting that with the increased use of adversarial messages for training, the false-positive rate in the open dataset decreases. To further reduce the false positives, we can use a very high number of adversarial messages; however, this may lead to an increase in false negatives. Regarding the robustness against evasion attacks, Figure 6d shows the attack success rate of each model, measured under the attack mode of EVA using representative messages as target messages. Attack mode adopts the best perturbation whenever perturbations are injected, whereas collection mode uses random perturbations. This causes the attack mode to generate different adversarial messages from those of the collection mode by adopting different perturbations. In our test, the two modes generated different adversarial messages for the same target model and message. The evaluation was performed without learning the adversarial messages that the attack mode can generate in advance. The attack success rate was significantly lower for the retrained models Re36 and Re72. Owing to their comparable results, we assumed that the attack success rate was saturated at approximately 0.55.

In summary, we replaced the variant-text messages of smishing with adversarial messages generated by EVA and obtained a smishing classifier comparable to or better than the base model. Re72 provides a similar detection performance on the KISA dataset, lower false positiveness in the open dataset, and better robustness against evasion attacks when compared with the base model.

VII. MITIGATION

This section explores strategies for reducing the effect of evasion attacks. Two approaches were considered in this study. First, during the preprocessing of input messages, we eliminated the target features that are vulnerable to evasion attacks. For the EVA, we replaced line breaks with spaces to invalidate structure-level perturbations. Second, we retrained the target model with adversarial messages to prevent character- and word-level perturbations.

In the first approach (i.e., preprocessing), we removed line breaks during the preprocessing stage and trained a new model (Psd) under the same conditions as those applied for the base model training. This eliminated the effect of the addition or deletion of line breaks by EVA. However, contrary to our expectations, Table 5 shows that the attack success rate did not decrease for Psd. It remained almost

TABLE 5. Performance comparison between the base model and improved models (Psd and Adv), where Psd is the model that removes line breaks and Adv is the adversarially trained model.

Test Dataset	Measure	Base	Psd	Adv
KISA test dataset	ACC	0.9959	0.9952	0.9944
	F1 score	0.9946	0.9937	0.9927
	FPR	0.0041	0.0052	0.0054
	FNR	0.0043	0.0042	0.0059
Open dataset	FPR	0.1239	0.1089	0.0909
Representative	ASR	0.8241	0.8204	0.4091

identical to the attack success rate of 0.8204 against the base model. When attempting a text evasion attack on Psd, it appears that perturbations at the word level (mainly sub_word$(\cdot,\cdot)$) replaced the line break addition/deletion at the structural level, resulting in a higher success rate. Consequently, removing line breaks is not an effective solution for mitigating evasion attacks.

For the second approach (i.e., adversarial training), we collected adversarial messages using the collection mode (described in Section VI) and retrained the models by using these messages. We measured the classification performance using the KISA and open datasets and evaluated the robustness of the retrained models using the attack mode of EVA. As mentioned in Section VI, the attack mode generates different adversarial messages from those of the collection mode. EVA first generated 120,000 distinct adversarial messages from 4,457 representative messages. The adversarial messages were partitioned into ten groups, each with 12,000 messages. We then trained the ten models by gradually accumulating the messages from each group. The first retrained model was trained using 12,000 messages from the first group, and the second retrained model was trained using 24,000 messages from the first and second groups. Finally, the last (the tenth) retrained model was trained using 120,000 messages from all groups. Each model was retrained without line breaks (by altering the line breaks into spaces) to reduce the available perturbations. For the KISA test dataset, the classification performances of the ten retrained models were slightly lower than those of the base model, but the difference was not large enough to have a substantial effect. Additionally, the ten retrained models provided similar classification performances, even when a high number of adversarial messages was used during training. For the open dataset, the model retrained with 12,000 samples exhibited a relatively high false-positive rate of 0.16. However, models retrained with more than 24,000 samples showed lower false-positive rates, ranging from 0.05 to 0.1. It appears that adversarial training using adversarial messages generated by EVA does not degrade the classification performance. The attack success rates (ASRs) of the ten retrained models were measured to be between 0.4 and 0.65, gradually decreasing with fluctuations as the number of adversarial messages used in training increased. In the model retrained with 84,000 adversarial messages, the attack success rate was slightly saturated and did not decrease significantly. Table 5 compares the model (Adv) retrained with 84,000 adversarial messages with the base model. The classification performance of Adv was comparable to that of the base model. However, the robustness against evasion attacks was significantly improved. After adversarial training, the attack success rate of Adv was almost half (0.4091) that of the base model.

The results show that adversarial training cannot perfectly defend against evasion attacks but can make the attack more difficult. Even if a developer has access to several smishing datasets, robustness against evasion attacks cannot be guaranteed. However, adversarial training using EVA reduces the threats caused by evasion attacks and helps successfully deploy a smishing classifier in the real world.

VIII. CONCLUSION

In this paper, we presented an effective on-device smishing classifier that is resistant to text-evasion attacks. Our classifier utilizes a deep learning model that is trained and evaluated using real smishing messages provided by KISA to ensure its practicality in real-world scenarios. To alleviate privacy concerns, we focused on a solution that functions solely on local devices. Therefore, we developed a lightweight model that does not impose a burden on the devices. In addition, we proposed a text-evasion attack tool, EVA, which imitates the smishing scammer's behavior. We used EVA to evaluate the robustness of the proposed smishing classifier against variant-text smishing messages and enhanced it through adversarial training. This was the first study to address the severity of variant-text smishing messages. Our classifier achieved a high accuracy rate of 0.99, despite its compact size of 127 kB. Furthermore, it is robust against variant-text smishing messages.

REFERENCES

[1] (2022). *Phishing Activity Trends Report 3rd Quarter 2022*. Anti-Phishing Working Group (APWG). [Online]. Available: https://apwg.org/

[2] (2020). *Increasing Cybercrime: Un Reports 350 Per Cent Rise in Phishing Websites During Pandemic*. The New Indian Express. [Online]. Available: https://www.newindianexpress.com/

[3] (2021). *Smishing Attacks Increased 700% in First Six Months of 2021*. ITPro. [Online]. Available: https://www.itpro.com/

[4] (2022). *7 Out of 10 Ott Users. Smartphone Ownership Rate is 93.4%*. etnews. [Online]. Available: https://www.etnews.com/20221227000094

[5] *Voice Phishing Status, Type, Trend, and Response Related Implications*, Statistics Research Institute, Kolkata, India, 2022. [Online]. Available: https://kostat.go.kr/

[6] G. Sonowal and K. S. Kuppusamy, "SmiDCA: An anti-smishing model with machine learning approach," *Comput. J.*, vol. 61, no. 8, pp. 1143–1157, Aug. 2018.

[7] A. K. Jain and B. B. Gupta, "Feature based approach for detection of smishing messages in the mobile environment," *J. Inf. Technol. Res.*, vol. 12, no. 2, pp. 17–35, Apr. 2019.

[8] D. Goel and A. K. Jain, "Smishing-classifier: A novel framework for detection of smishing attack in mobile environment," in *Smart and Innovative Trends in Next Generation Computing Technologies*. Cham, Switzerland: Springer, 2018, pp. 502–512.

[9] G. Sonowal, "Detecting phishing SMS based on multiple correlation algorithms," *Social Netw. Comput. Sci.*, vol. 1, no. 6, p. 361, Nov. 2020.

[10] S. Mishra and D. Soni, "A content-based approach for detecting smishing in mobile environment," in *Proc. Int. Conf. Sustain. Comput. Sci. (SUSCOM)*, Feb. 2019, pp. 986–993.

[11] S. Mishra and D. Soni, "Smishing detector: A security model to detect smishing through SMS content analysis and URL behavior analysis," *Future Gener. Comput. Syst.*, vol. 108, pp. 803–815, Jul. 2020.
[12] S. Mishra and D. Soni, "Implementation of 'smishing detector': An efficient model for smishing detection using neural network," *Social Netw. Comput. Sci.*, vol. 3, no. 3, p. 189, May 2022.
[13] *Block Spam Text & Call*. Trend Micro. Accessed: 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.trendmicro.fraudbuster&hl=en_US
[14] *Textkiller Spam Text Blocker*. TelTech Systems. Accessed: 2023. [Online]. Available: https://apps.apple.com/kr/app/textkiller-spam-text-blocker/id1514005355
[15] *Whowho*. VP (Inc). Accessed: 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.ktcs.whowho&hl=ko
[16] *Alyacm*. ESTsoft Corporation. Accessed: 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.estsoft.alyac&hl=ko
[17] *Robokiller Spam Call Blocker*. Teltech Systems, Inc. Accessed: 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.robokiller.app&hl=ko&gl=U.S
[18] *Mobileguard*. SK Shieldus, Accessed: 2023. [Online]. Available: https://play.google.com/store/apps/details?id=kr.co.adtcaps.mobileguard&hl=ko
[19] K. Soska and N. Christin, "Automatically detecting vulnerable websites before they turn malicious," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 625–640.
[20] L. Wu, X. Du, and J. Wu, "MobiFish: A lightweight anti-phishing scheme for mobile phones," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–8.
[21] A. Kang, J. Dong Lee, W. M. Kang, L. Barolli, and J. H. Park, "Security considerations for smart phone smishing attacks," in *Advances in Computer Science and its Applications (CSA)*. Cham, Switzerland: Springer, 2014, pp. 467–473.
[22] J. W. Joo, S. Y. Moon, S. Singh, and J. H. Park, "S-detector: An enhanced security model for detecting smishing attack for mobile computing," *Telecommun. Syst.*, vol. 66, no. 1, pp. 29–38, Sep. 2017.
[23] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL detection using machine learning: A survey," 2017, *arXiv:1701.07179*.
[24] S. Garg and G. Ramakrishnan, "BAE: BERT-based adversarial examples for text classification," 2020, *arXiv:2004.01970*.
[25] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "BERT-ATTACK: Adversarial attack against BERT using BERT," 2020, *arXiv:2004.09984*.
[26] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of SMS spam filtering: New collection and results," in *Proc. 11th ACM Symp. Document Eng.*, Sep. 2011, pp. 259–262.
[27] S. Mishra and D. Soni, "SMS phishing dataset for machine learning and pattern recognition," in *Proc. 14th Int. Conf. Soft Comput. Pattern Recognit. (SoCPaR)*. Cham, Switzerland: Springer, 2023, pp. 597–604.
[28] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," 2017, *arXiv:1707.07328*.
[29] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 4208–4215.
[30] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "TextBugger: Generating adversarial text against real-world applications," 2018, *arXiv:1812.05271*.
[31] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 50–56.
[32] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "HotFlip: white-box adversarial examples for text classification," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics (Short Papers)*, vol. 2, 2018, pp. 31–36. [Online]. Available: https://aclanthology.org/P18-2006
[33] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," 2017, *arXiv:1707.02812*.
[34] Y. Zang, F. Qi, C. Yang, Z. Liu, M. Zhang, Q. Liu, and M. Sun, "Word-level textual adversarial attacking as combinatorial optimization," 2019, *arXiv:1910.12196*.
[35] R. Maheshwary, S. Maheshwary, and V. Pudi, "Generating natural language attacks in a hard label black box setting," 2020, *arXiv:2012.14956*.
[36] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," 2018, *arXiv:1804.07998*.
[37] S. Ren, Y. Deng, K. He, and W. Che, "Generating natural language adversarial examples through probability weighted word saliency," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 1085–1097. [Online]. Available: https://aclanthology.org/P19-1103
[38] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is BERT really robust? A strong baseline for natural language attack on text classification and entailment," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 5, 2020, pp. 8018–8025.
[39] Z. Meng and R. Wattenhofer, "A geometry-inspired attack for generating natural language adversarial examples," 2020, *arXiv:2010.01345*.
[40] W. C. Gan and H. T. Ng, "Improving the robustness of question answering systems to question paraphrasing," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 6065–6075. [Online]. Available: https://aclanthology.org/P19-1610
[41] M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer, "Adversarial example generation with syntactically controlled paraphrase networks," 2018, *arXiv:1804.06059*.
[42] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, "PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization," 2019, *arXiv:1912.08777*.
[43] Y. Zhang, J. Baldridge, and L. He, "PAWS: Paraphrase adversaries from word scrambling," 2019, *arXiv:1904.01130*.
[44] M. Cheng, W. Wei, and C.-J. Hsieh, "Evaluating and enhancing the robustness of dialogue systems: A case study on a negotiation agent," in *Proc. Conf. North*, 2019, pp. 3325–3335. [Online]. Available: https://aclanthology.org/N19-1336
[45] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," 2019, *arXiv:1905.11268*.
[46] J. Morris, E. Lifland, J. Lanchantin, Y. Ji, and Y. Qi, "Reevaluating adversarial examples in natural language," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*, 2020, pp. 3829–3839. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.341
[47] R. Jia, A. Raghunathan, K. Göksel, and P. Liang, "Certified robustness to adversarial word substitutions," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, 2019, pp. 1–36.
[48] M. Mozes, P. Stenetorp, B. Kleinberg, and L. D. Griffin, "Frequency-guided word substitutions for detecting textual adversarial examples," 2020, *arXiv:2004.05887*.
[49] J. Wang, R. Bao, Z. Zhang, and H. Zhao, "Rethinking textual adversarial defense for pre-trained language models," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 30, pp. 2526–2540, 2022.
[50] X. Wang, J. Hao, Y. Yang, and K. He, "Natural language adversarial defense through synonym encoding," in *Proc. Uncertainty Artif. Intell.*, 2021, pp. 823–833.
[51] B. Wang, S. Wang, Y. Cheng, Z. Gan, R. Jia, B. Li, and J. Liu, "InfoBERT: Improving robustness of language models from an information theoretic perspective," 2020, *arXiv:2010.02329*.
[52] M. Ye, C. Gong, and Q. Liu, "SAFER: A structure-free approach for certified robustness to adversarial word substitutions," 2020, *arXiv:2005.14424*.
[53] Y. Zhou, X. Zheng, C.-J. Hsieh, K.-W. Chang, and X. Huang, "Defense against synonym substitution-based adversarial attacks via Dirichlet neighborhood ensemble," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process. (Long Papers)*, vol. 1, 2021, pp. 5482–5492.
[54] C. Si, Z. Zhang, F. Qi, Z. Liu, Y. Wang, Q. Liu, and M. Sun, "Better robustness by more coverage: Adversarial and mixup data augmentation for robust finetuning," in *Proc. Findings Assoc. Comput. Linguistics (ACL-IJCNLP)*, 2021, pp. 1569–1576.
[55] X. Wang, Y. Yang, Y. Deng, and K. He, "Adversarial training with fast gradient projection method against synonym substitution based text attacks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 16, 2021, pp. 13997–14005.
[56] X. Dong, A. T. Luu, R. Ji, and H. Liu, "Towards robustness against natural language word substitutions," 2021, *arXiv:2107.13541*.
[57] E. Dinan, S. Humeau, B. Chintagunta, and J. Weston, "Build it break it fix it for dialogue safety: Robustness from adversarial human attack," 2019, *arXiv:1908.06083*.
[58] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, Jun. 2007.

[59] R. De Silva, M. Nabeel, C. Elvitigala, I. Khalil, T. Yu, and C. Keppitiyagama, "Compromised or attacker-owned: A large scale classification and study of hosting domains of malicious URLs," in *Proc. USENIX Secur. Symp.*, 2021, pp. 3721–3738.
[60] T. Kim, N. Park, J. Hong, and S.-W. Kim, "Phishing URL detection: A network-based approach robust to evasion," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 1769–1782.
[61] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Adv. neural Inf. Process. Syst.*, vol. 28, 2015.
[62] J. Park. (2020). *Koelectra: Pretrained Electra Model for Korean*. [Online]. Available: https://github.com/monologg/KoELECTRA
[63] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1345–1362.
[64] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2020, pp. 189–218.
[65] D. Rolnick and K. Kording, "Reverse-engineering deep ReLU networks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 8178–8187.
[66] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 605–622.
[67] S. J. Oh, B. Schiele, and M. Fritz, "Towards reverse-engineering black-box neural networks," in *Explainable AI: Interpreting, Explaining Visualizing Deep Learning*. Cham, Switzerland: Springer, 2019, pp. 121–144.
[68] L. Batina, S. Bhasin, D. Jap, and S. Picek, "$*CSI*$$*NN*$: Reverse engineering of neural network architectures through electromagnetic side channel," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 515–532.
[69] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating text generation with BERT," 2019, *arXiv:1904.09675*.
[70] Y. E. Seyyar, A. G. Yavuz, and H. M. Ünver, "An attack detection framework based on BERT and deep learning," *IEEE Access*, vol. 10, pp. 68633–68644, 2022.
[71] Z. Gong, W. Wang, B. Li, D. Song, and W.-S. Ku, "Adversarial texts with gradient methods," 2018, *arXiv:1801.07175*.
[72] T. Wang, X. Wang, Y. Qin, B. Packer, K. Li, J. Chen, A. Beutel, and E. Chi, "CAT-gen: Improving robustness in NLP models via controlled adversarial text generation," 2020, *arXiv:2010.02338*.
[73] D. Li, Y. Zhang, H. Peng, L. Chen, C. Brockett, M.-T. Sun, and B. Dolan, "Contextualized perturbation for textual adversarial attack," 2020, *arXiv:2009.07502*.
[74] L. Song, X. Yu, H.-T. Peng, and K. Narasimhan, "Universal adversarial attacks with natural triggers for text classification," 2020, *arXiv:2005.00174*.
[75] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[76] S. Park, J. Moon, S. Kim, W. I. Cho, J. Han, J. Park, C. Song, J. Kim, Y. Song, T. Oh, J. Lee, J. Oh, S. Lyu, Y. Jeong, I. Lee, S. Seo, D. Lee, H. Kim, M. Lee, S. Jang, S. Do, S. Kim, K. Lim, J. Lee, K. Park, J. Shin, S. Kim, L. Park, A. Oh, J. Ha, and K. Cho, "KLUE: Korean language understanding evaluation," 2021, *arXiv:2105.09680*.

JAE WOO SEO received the Ph.D. degree in cryptography from POSTECH, Pohang, South Korea, in 2013. He is currently a Principal Engineer with Samsung Research. His research interests include cryptography, data-driven security, and cybercrime prevention.

JONG SUNG LEE received the M.S. degree in information and communication engineering from Sungkyunkwan University, Suwon, South Korea, in 2010. He is currently a Staff Engineer with Samsung Research. His research interests include data-driven security, cybercrime prevention, and security for consumer devices.

HYUNWOO KIM received the B.S. degree in computer engineering from Chosun University, Gwangju, South Korea, in 2017. He is currently a Software Engineer with Samsung Research. His research interests include network security and cybercrime detection.

JOONGHWAN LEE received the M.S. degree in digital media communication engineering from Sungkyunkwan University, Suwon, South Korea, in 2016. He is currently a Principal Engineer with Samsung Research. His research interests include network security and security for consumer devices.

SEONGWON HAN received the Ph.D. degree in computer science from UCLA, in 2014. He is currently a Staff Software Engineer with Samsung Research, Samsung Electronics. His research interests include wireless networking, network security, and mobile/pervasive computing.

JUNGIL CHO received the B.S. degree in computer engineering from Kyungpook National University, Daegu, South Korea, in 2005. He is currently a Principal Engineer with Samsung Research. His research interests include device and network security and cybercrime detection.

CHOONG-HOON LEE received the Ph.D. degree in computer science from KAIST, Daejeon, South Korea, in 2004. He was a Postdoctoral Associate and a Research Professor with KAIST, from 2004 to 2006. He was a Master (VP of Technology) with Samsung Research until 2023, and he is currently a Technical Consultant with Samsung Research. His research interests include data-driven security, authentication, and cybercrime prevention.