Don’t be a Fool: Pooling Strategies in Offensive Language Detection from User-Intended Adversarial Attacks

Anonymous ACL submission

Abstract

Warning: this paper contains expressions that may offend the readers.

Offensive language detection is an important task for filtering out abusive expressions and improving online user experiences. However, malicious users often attempt to avoid filtering systems through the involvement of textual noises. In this paper, we propose these evasions as user-intended adversarial attacks that insert special symbols or leverage the distinctive features of the Korean language. Furthermore, we introduce simple yet effective pooling strategies in a layer-wise manner to defend against the proposed attacks, focusing on the preceding layers not just the last layer to capture both offensiveness and token embeddings. We demonstrate that these pooling strategies are more robust to performance degradation even when the attack rate is increased, *without* training of such patterns. Notably, we found that models pre-trained on clean texts could achieve a performance in detecting attacked offensive language, comparable to models pre-trained on noisy texts by utilizing these pooling strategies.

1 Introduction

As the internet becomes an important part of our lives, the prevalence of offensive language on online platforms, particularly social media, has become a serious concern (Zampieri et al., 2019). Deep learning models for filtering offensive languages have been proposed to address this problem. However, malicious users have consistently found ways to avoid them. One such way is the deliberate insertion of additional typographical errors or substitution of certain characters with visually similar alternatives (Wu et al., 2018; Kurita et al., 2019).

Despite numerous studies on this phenomenon in English, there has been a comparatively limited exploration in Korean, which is a low-resource language characterized by distinct linguistic features (Kim et al., 2021a; Sahoo et al., 2023). As the Korean communities also suffer from the use of abusive language and cyberbullying (Jun, 2020; Yi and Cha, 2020; Saengprang and Gadavanij, 2021; McCurry, 2022), it is desirable to investigate the evasion tactics employed by malicious users and to formulate them. While recent studies have discussed how to avoid offensive language detection (Ahn and Egorova, 2021; Cho and Kim, 2021; Kim et al., 2021c), their definitions are ambiguous, and no clear solutions have been proposed to defend against the evasions.

In this paper, we propose the evasion methods as user-intended adversarial attacks and incorporate them into offensive language from the perspective of malicious users. Our proposed attacks are grounded in prevalent forms that can be found in offensive language online, and reflect the distinct features of Korean language, wherein a single character can be further subdivided (Song, 2006). When we tested the proposed attacks on the existing models for offensive language detection, the results revealed that the performance declines as the ratio of the proposed attacks increases.

Furthermore, we introduce simple yet effective pooling strategies in a layer-wise manner to defend against the proposed attacks. Motivated by the exploration of the impact of each layer in a pre-trained language model (Jawahar et al., 2019; Oh et al., 2022), we selectively integrate useful features for the attacked offensive language across all layers. The attacked texts have some changes in the tokens used, differing from the original texts. Therefore, we implement pooling strategies to ensure that the model captures not only high-level features but also low-level features, which are related to offensiveness and token embeddings, respectively. This simple modification enriches the understanding of the attacked offensive language, enhancing the robustness of the model against user-intended adversarial attacks *without* training of such patterns.

The contributions of our study are as follows:

- We propose user-intended adversarial attacks that are often associated with offensive language online from the perspective of malicious users. These attacks are directly performed by inserting special symbols or leveraging the distinctive features of the Korean language.
- We introduce the pooling strategies in a layer-wise manner to selectively utilize all layers rather than just the last layer. This approach achieves a notable performance when utilized to a model pre-trained on clean texts, *without* training of the proposed attacks.
- We demonstrate the effectiveness of layer-wise first pooling and max pooling by assigning distinct weights to each layer and utilizing them to the model depending on the nature of the pre-trained texts when user-adversarial attacks are involved in offensive language.

2 Related Work

2.1 Adversarial Attacks

Adversarial attacks involve perturbed input data that confuses the model. In contrast, a situation in which the model consistently predicts well regardless of the nature of input data is referred to as defending (Goyal et al., 2023). Previous studies have explored this based on word-level substitutions (Ren et al., 2019; Jin et al., 2020), and others also have explored them on character-level. We propose adversarial attacks that utilize not only character-level but also smaller-scale alternations tailored to the features of Korean language. Such attacks are commonly observed in the context of offensive languages in various online communities (Ahn and Egorova, 2021; Cho and Kim, 2021).

TextBugger (Li et al., 2019) is an early study that focused on character-level alternations, such as replacing characters with visually similar ones (e.g. replacing the alphabet 'o' with the number '0'). Other studies have suggested simple leetspeaks that employ symbols that resemble the alphabet (Aggarwal and Zesch, 2022), or adversarial attacks that are not easily detected visually, such as transforming Latin characters into similar-looking Cyrillic characters (Wolff and Wolff, 2020). Although several studies also have explored visually undetectable attacks (Kim et al., 2021b; Boucher et al., 2022; Bajaj and Vishwakarma, 2023), we utilize a more realistic attack scenario that can occur online from the perspective of malicious users.

2.2 Korean Offensive Language Datasets

Owing to the increasing demand for online content in Korean language and the growing threats of cyberbullying, previous studies have introduced offensive language datasets collecting comments from diverse resources such as online news, communities, and YouTube.

BEEP! (Moon et al., 2020) was a pioneering study that utilizes hate speech prevalent in news comments. KoLD (Jeong et al., 2022) and K-MHaS (Lee et al., 2022) specified the target group of the offensive language. Subsequently, KODOLI (Park et al., 2023b) provided labels that refine the degree of offensiveness, and built upon these efforts, K-HATERS (Park et al., 2023a) was built to incorporate the strengths of the preceding datasets.

Although numerous datasets have been proposed, there is still a lack of definition for adversarial attacks that are frequently involved in offensive language, and how to defend against them. In this study, we focus on introducing pooling strategies for defending against these attacks, *without* directly using attacked texts during the training stage.

3 Method

3.1 User-Intended Adversarial Attacks

We present adversarial attacks designed to target offensive languages from the perspective of malicious users. By referring to existing offensive language datasets, we define frequently occurring attack types. These attacks are categorized into three groups: INSERT, COPY, and DECOMPOSE. Examples of each attack type are listed in Table 1.

User-Intended Adversarial Attacks	Text Examples
original text	쓰레기 같은 (such a trash)
INSERT_zz	쓱ㅋㅋㅋ레기 같은
INSERT_space	쓰 레기 같은
INSERT_special	쓰@레기 같은
COPY_initial	쓸레기 같은
COPY_middle	쓰레에기 같은
COPY_final	쓰레기 가튼
DECOMPOSE_final	쓰레기 가ㅌ은
DECOMPOSE_all	ㅆㅡ레기 같은

Table 1: Text examples of user-intended adversarial attacks with three categories: INSERT, COPY, and DECOMPOSE. There are various attacks that involve special symbols or exploit the distinctive features of Korean.

First, INSERT involves adding incomplete Korean character forms, which are often used online without significant meaning. For example, 'ㅋㅋ' (equivalent 'LOL' in English) is a commonly used and somewhat meaningless string frequently used in online communications. In this case, INSERT_zz is performed by inserting the string at a specific location within the word, as in real situations. Other types of INSERT also include unnecessary spaces or special symbols.

The following two types of user-intended adversarial attacks take advantage of the distinct features of the Korean language. A single character must have an initial sound, a middle sound, and an optional final sound (Song, 2006). For example, in the expression '쓰레기 같은' in Table 1, the character '쓰' has only the initial and middle sounds, whereas the character '같' has all three sounds.

Second, COPY utilizes the distinctive features, copying one of the three sounds from the selected character to the other character. For example, the character '레' from the expression '쓰레기 같은' has the initial and middle sounds of 'ㄹ' and 'ㅔ'. In this case, COPY_initial is performed by copying the initial sound of that character 'ㄹ' to the final sound of the preceding character '쓰'. Consequently, '쓰' is transformed into '쓸', thus, the attacked expression will be '쓸레기 같은'.

Finally, DECOMPOSE also utilizes the unique characteristics, isolating the final sound of the selected character or breaking down the character itself. For example, the single character '쓰' from the expression '쓰레기 같은' has the initial and middle sounds of 'ㅆ' and 'ㅡ'. In this case, DECOMPOSE_all is performed by breaking down the character, resulting in the sounds being independent of that character. Consequently, '쓰' is transformed into 'ㅆㅡ', thus, the attacked expression will be 'ㅆㅡ레기 같은'. Further details and examples of all user-intended adversarial attacks are provided in Appendix A.

3.2 Layer-Wise Pooling Strategies

In standard text classification tasks, pre-trained models such as BERT are fined-tuned to the target domain. This is based on the assumption that the $[CLS]$ token from the last layer effectively captures the sentence representation (Devlin et al., 2018). However, when we tested from the assessment of the existing models, we found that they lost the consistency of prediction for the proposed attacks.

User-Intended Adversarial Attacks	Tokenized Examples
original text	쓰레기, 같, ##은 (such a trash)
INSERT_zz	[UNK], 같, ##은
INSERT_space	쓰, 레, ##기, 같, ##은
INSERT_special	쓰, @, 레, ##기, 같, ##은
COPY_initial	쓸, ##레기, 같, ##은
COPY_middle	쓰, ##레, ##에, ##기, 같, ##은
COPY_final	쓰레기, 가, ##튼
DECOMPOSE_final	쓰레기, 가, ##ㅌ, ##은
DECOMPOSE_all	[UNK], 같, ##은

Table 2: Tokenized examples of user-intended adversarial attacks. Although the texts have the same meaning, the tokens are represented differently.

When using perturbed text to a model, the tokenization results differ from those of the original text, as shown in Table 2. By involving special symbols or exploiting the distinctive features of Korean, we observed that even if the text had the same meaning to human readers, the tokenized outputs differed significantly. Therefore, we did not rely on the information only from the last layer, but utilized the preceding layers, which focus more on token embeddings (Ma et al., 2019). This also reflects the previous finding that meaningful information for a certain task can be captured in the preceding layers (Oh et al., 2022).

We extend pooling strategies in a layer-wise manner, allowing us the flexibility to utilize text representations from all layers. Denoting the $[CLS]$ token of the Nth layer as h_N^{cls}, we introduce four strategies that optionally consider the $[CLS]$ tokens from all the layers $h_1^{cls}, ..., h_N^{cls}$.

Mean, Max Pooling: We apply mean pooling utilizing the L^1 norm, which averages all $[CLS]$ tokens from all the layers, and max pooling utilizing the L^∞ norm, which takes a max-over-time operation on the values corresponding to each dimension from all $[CLS]$ tokens.

When the dimension of $[CLS]$ token is M, and all the values of jth dimension of $[CLS]$ tokens from all the layers are concatenated and denoted by h_{all}^j, these two poolings are defined as follows:

$$pool_{mean} = mean(h_1^{cls}, ..., h_N^{cls}), \quad (1)$$

$$pool_{max} = max(h_{all}^1), ..., max(h_{all}^M), \quad (2)$$

Weighted Pooling: We apply weighted pooling utilizing a learnable parameter that determines the importance of each layer. Through adaptive incorporation of the layers, we train weights that selectively capture both offensiveness and token embeddings, initializing all weights to zero.

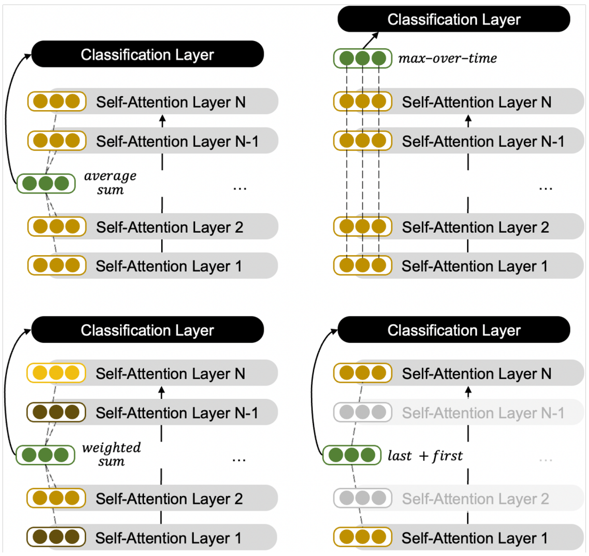

Figure 1: Layer-wise pooling strategies that selectively use $[CLS]$ tokens from all layers. From the upper left, there are mean, max, weighted, and first pooling.

When the w_i represents the weight of each layer and α_i represents its softmax distribution, the weighted pooling is defined as follows:

$$pool_{weighted} = \sum_{i=1}^{N} \alpha_i h_i^{cls}, \tag{3}$$

First Pooling: We apply first pooling utilizing $[CLS]$ token from the first layer. Rather than considering all the layers, we focus on leveraging information from layers directly associated with offensiveness and token embeddings.

$$pool_{first} = h_N^{cls} + h_1^{cls}, \tag{4}$$

The layer-wise pooling strategies described above are illustrated in Figure 1. We conducted experiments to verify the robustness of these strategies for detecting offensive languages that reflect user-intended adversarial attacks.

4 Experiment

4.1 Datasets

We collect both the KoLD (Jeong et al., 2022) and K-HATERS (Park et al., 2023a) datasets and divide them into train, validation, and test sets by stratifying their labels. We randomly shuffle and split them into the ratio of 8:1:1. We set the attack rates into 30%, 60%, and 90%, corrupting a portion of the words in a sentence. The attacks were only applied to the test set to evaluate the robustness of the model against user-intended adversarial attacks.

4.2 Baselines

We experiment the effectiveness of the layer-wise pooling strategies with the baselines, which are presented below. The experimental details including hyperparameters and metrics are reported in Appendix B.

- BiLSTM: This model addresses the long-term dependency problem of RNN by remembering only the information in need (Schuster and Paliwal, 1997). It was built bidirectionally by stacking two LSTMs, and the forward and backward $[CLS]$ tokens from the last layer were combined and passed through the classification layer.

- BiGRU: This model is derived from the BiLSTM and further evolved by reducing the training parameters through the selective utilization of gates (Cho et al., 2014). Its configurations are the same as BiLSTM.

- BERT$_{clean}$: This model follows the BERT (Devlin et al., 2018) structure, which is built on a self-attention mechanism with masked language modeling. It was pre-trained on preprocessed texts in Korean (Park et al., 2021). The $[CLS]$ token from the last layer is passed through the classification layer.

- BERT$_{noise}$: This model also follows the BERT structure, but it is pre-trained on noisy texts in Korean, such as online comments (Lee, 2020). Its configurations are the same as BERT$_{clean}$.

- Ensemble$_{hard}$, Ensemble$_{soft}$: These models utilize both the BERT$_{clean}$ and BERT$_{noise}$, employing voting methods from ensemble techniques. Hard voting is conducted through a majority vote, but soft voting occurs in the cases of tied votes. Soft voting averages the prediction probabilities of each model.

5 Discussion

5.1 Experimental Results

The performances of the models when exposed to user-intended adversarial attacks are presented in Table 3. Each of the best performances from the layer-wise pooling strategies and the baselines in the original, and 30%, 60%, and 90% attacked are highlighted in bold.

Model	Original			30% Attacked				60% Attacked				90% Attacked			
	P	R	F1	P	R	F1	Δatk	P	R	F1	Δatk	P	R	F1	Δatk
BiLSTM	71.83	68.80	69.81	70.84	66.23	67.38	-3.48%	68.97	62.82	63.67	**-8.79%**	69.32	61.64	62.25	**-10.82%**
BiGRU	71.32	65.71	66.91	70.40	63.32	64.26	-3.96%	68.84	60.31	60.52	-9.55%	68.05	58.83	58.48	-12.59%
BERT$_{clean}$	79.81	77.79	78.64	79.51	73.35	75.19	-4.38%	77.74	66.38	67.96	-13.58%	76.14	62.01	62.44	-20.60%
BERT$_{clean} + mean$	78.57	79.06	79.01	79.41	73.97	75.70	-4.18%	77.15	66.97	68.62	-13.15%	74.90	62.45	63.09	-20.14%
BERT$_{clean} + max$	78.51	78.81	78.65	78.47	74.03	75.54	-3.95%	77.80	66.66	68.29	-13.17%	76.79	61.51	61.72	-21.52%
BERT$_{clean} + weighted$	79.93	78.50	79.14	79.19	73.70	75.43	-4.68%	77.14	67.62	69.33	-12.39%	75.44	63.66	64.85	-18.05%
BERT$_{clean} + first$	79.05	79.37	**79.21**	78.89	75.85	**77.02**	**-2.76%**	77.58	69.38	**71.21**	**-10.09%**	76.08	64.33	**65.49**	**-17.32%**
BERT$_{noise}$	80.64	78.88	79.64	80.67	75.42	77.17	-3.10%	78.44	69.42	**71.33**	-10.43%	76.46	65.55	**66.96**	-15.92%
Ensemble$_{hard}$ (BERT$_{clean}$ + BERT$_{noise}$)	81.63	79.42	80.36	81.57	75.03	77.04	-4.13%	80.47	68.60	70.60	-12.14%	78.68	64.24	65.35	-18.67%
Ensemble$_{soft}$ (BERT$_{clean}$ + BERT$_{noise}$)	81.52	79.53	**80.38**	81.54	75.29	**77.25**	-3.89%	80.27	68.86	70.87	-11.83%	78.47	64.42	65.58	-18.41%

Table 3: Experimental results of offensive language detection when a certain ratio of user-intended adversarial attacks are involved. P, R, and F1 represent macro precision, recall, and f1-score, respectively. Δatk represents the performance drop in the f1-score as the attacks are involved in the same model.

We observed that the BERT-based models consistently outperformed RNN-based models in terms of the f1-score across all attack rates. Under original and 30% attacked, employing multiple BERT models with soft voting yielded the best scores, achieving f1-scores of 80.38 and 77.25. As the attack rates increased to 60% and 90%, using a single model pre-trained on noisy texts rather than ensemble models proved to be the most effective, achieving f1 scores of 71.33 and 66.96.

However, ensemble models require twice computational resources for both training and inference stages compared to a single model. In the case of BERT$_{noise}$, a large amount of noisy texts is required, raising concerns regarding its adaptability when inference with attacked input types is not encountered during the pre-training stage.

When applying layer-wise pooling strategies to BERT$_{clean}$, we found that the performances were improved in almost all attack rates compared to their absence. They only need to train an additional one-dimensional parameter equal to the size of all layers (e.g. 12 for BERT-based models), or no parameters are required. Furthermore, they are robust as the attack rate increases compared to models with no pooling strategies.

The average performances when exposed to user-intended adversarial attacks across all attack rates are presented in Table 4. All layer-wise pooling strategies exhibited robustness against attacks compared to their absence, except for BERT$_{clean} + max$, which exhibited a slight performance drop. Moreover, even BERT$_{clean} + first$, which only incorporates information from the first layer *without* any parameters or training noisy texts, showed comparable performance to BERT$_{noise}$ across all attack rates that were pre-trained on noisy texts.

Model	Average	
	F1	Δatk
BiLSTM	64.43	**-7.69%**
BiGRU	61.08	-8.70%
BERT$_{clean}$	68.52	-12.85%
BERT$_{clean} + mean$	69.13	-12.49%
BERT$_{clean} + max$	68.51	-12.88%
BERT$_{clean} + weighted$	69.86	-11.70%
BERT$_{clean} + first$	**71.24**	**-10.05%**
BERT$_{noise}$	**71.82**	-9.81%
Ensemble$_{hard}$ (BERT$_{clean}$ + BERT$_{noise}$)	70.99	-11.64%
Ensemble$_{soft}$ (BERT$_{clean}$ + BERT$_{noise}$)	71.23	-11.37%

Table 4: Average from the experimental results of offensive language detection when a certain ratio of user-intended adversarial attacks are involved.

5.2 Focus on Performance Drop

The degree to which the f1-scores of the models decreased with the attack rates is shown in Figure 2. We observed that, with the inclusion of attacks, all models failed to maintain their original performance, falling within a specific range. Particularly, the performance decreased from -2% to -4% for 30% attacked, -8% to -12% for 60% attacked, and -10% to -21% for 90% attacked.

We found that BiSLTM exhibited relatively modest performance degradation across the models. Despite the modest decrease in performance, the offensive language detection performances of RNN-based models were not as good as that of the BERT-based models because of the limitations of the RNN-based models themselves.

Among the BERT-based models, BERT$_{clean}$, which was pre-trained on clean texts, exhibited the largest performance degradation. However, BERT$_{clean} + first$, which applied a simple layer-wise pooling strategy to the model, successfully mitigated performance degradation by 1.62, 3.49, and 3.28 at each attack rate, respectively, achieving an average performance degradation of 2.79.

These results are similar to those of BERT$_{noise}$ and Ensemble$_{soft}$, which used the model pre-trained on noisy texts. At attack rates of 30% and 60%, the model with the first pooling exhibited the lowest performance drop, and at 90%, its degradation was higher than that of BERT$_{noise}$ but lower than that of Ensemble$_{soft}$. Therefore, we found that the model pre-trained on clean texts with a simple pooling strategy can achieve a certain level of performance, or even be more robust compared to the model pre-trained on noisy texts in defending against user-intended adversarial attacks.

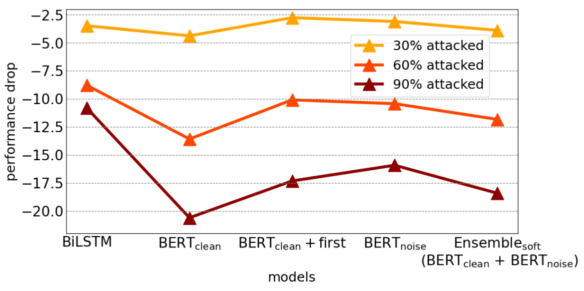

Figure 2: Degree to which the f1-scores of the models decrease with the attack rates. We selected several baseline models and BERT$_{clean}$ $+ first$ for the comparison.

5.3 Focus on Layer Weights

We focused on the observation that when perturbed text with the proposed attacks is used as an input, the tokenization results eventually differed from the original text. Therefore, we leveraged information from the preceding layers not just the last layer. Consequently, the first pooling achieved the best performance among the layer-wise pooling strategies, and its performance was nearly equivalent to that of BERT$_{noise}$ which was pre-trained on noisy texts, despite the absence of explicitly training noisy texts in pre-training stage.

Additionally, we conducted experiments to determine whether useful information could be captured not only in the first and last layer, but also in the layers close to these two layers of the model. We hypothesized that layers close to the last layer would capture the offensiveness that determine the text representations, whereas layers close to the first layer would capture the token embeddings that determine the degree to which a sentence contains textual attacks. We set all weights to zero in the experiment in Table 3, however, in this experiment, we assigned distinct weights to all layers.

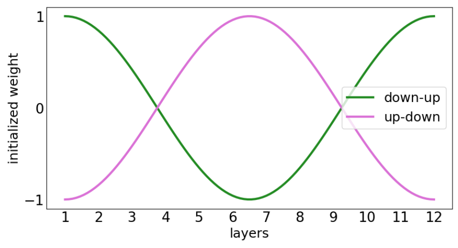

Figure 3: Initialized weights for each of the down-up and up-down poolings. Each strategy shares the shape of a cosine function but varies in the range on the x-axis depending on the layers to be focused.

We assigned relatively high weights to the layers close to the last and first layers, while assigning low weights to the middle layers. This strategy is referred to as down-up pooling, as its graph moves down then up. In contrast, we assigned relatively low weights to the layers close to the last and first layers, while assigning high weights to the middle layers. This strategy is referred to as up-down pooling, as its graph moves up then down.

The weights for the down-up and up-down poolings are shown in Figure 3. The green line at the cosine function within the range $[0, 2\pi]$ represents the initial weights for down-up pooling, whereas the pink line at the cosine function within the range $[\pi, 3\pi]$ represents that of the up-down pooling.

The performances of the models for the distinct layer weights are presented in Table 5. Down-up pooling outperformed for all attack rates than the up-down pooling and exhibited robustness for performance degradation. This demonstrates that it is

Model	Original			30% Attacked				60% Attacked				90% Attacked			
	P	R	F1	P	R	F1	Δatk	P	R	F1	Δatk	P	R	F1	Δatk
BERT$_{clean}$	79.81	77.79	78.64	79.51	73.35	75.19	-4.38%	77.74	66.38	67.96	-13.58%	76.14	62.01	62.44	-20.60%
BERT$_{clean}$ $+ down - up$	79.80	77.64	78.55	80.02	73.05	75.02	-4.49%	77.20	65.70	67.15	-14.51%	76.61	61.84	62.19	-20.82%
BERT$_{clean}$ $+ up - down$	80.35	77.10	78.36	79.95	71.67	73.73	-5.90%	78.59	65.28	66.67	-14.91%	76.81	60.87	60.81	-22.39%
BERT$_{clean}$ $+ first$	79.05	79.37	**79.21**	78.89	75.85	**77.02**	**-2.76%**	77.58	69.38	**71.21**	**-10.09%**	76.08	64.33	**65.49**	**-17.32%**

Table 5: Experimental results of offensive language detection when using down-up and up-down pooling strategies. They are initialized along with the cosine function, assigning distinct weights to the layers depending on whether they focus more on offensiveness and token embeddings.

Model	Original			30% Attacked				60% Attacked				90% Attacked			
	P	R	F1	P	R	F1	Δatk	P	R	F1	Δatk	P	R	F1	Δatk
BERT$_{clean}$	79.81	77.79	78.64	79.51	73.35	75.19	-4.38%	77.74	66.38	67.96	-13.58%	76.14	62.01	62.44	-20.60%
BERT$_{clean} + mean$	78.57	79.06	79.01	79.41	73.97	75.70	-4.18%	77.15	66.97	68.62	-13.15%	74.90	62.45	63.09	-20.14%
BERT$_{clean} + max$	78.51	78.81	78.65	78.47	74.03	75.54	-3.95%	77.80	66.66	68.29	-13.17%	76.79	61.51	61.72	-21.52%
BERT$_{clean} + weighted$	79.93	78.50	79.14	79.19	73.70	75.43	-4.68%	77.14	67.62	69.33	-12.39%	75.44	63.66	64.85	-18.05%
BERT$_{clean} + first$	79.05	79.37	**79.21**	78.89	75.85	**77.02**	**-2.76%**	77.58	69.38	**71.21**	**-10.09%**	76.08	64.33	**65.49**	**-17.32%**
BERT$_{noise}$	80.64	78.88	79.64	80.67	75.42	77.17	-3.10%	78.44	69.42	71.33	-10.43%	76.46	65.55	66.96	-15.92%
BERT$_{noise} + mean$	81.59	77.73	79.18	80.81	73.79	75.82	-4.24%	78.35	68.32	70.17	-11.37%	75.89	65.54	66.94	-15.45%
BERT$_{noise} + max$	80.72	79.81	**80.23**	79.68	76.29	**77.57**	-3.31%	77.46	72.00	**73.64**	-8.21%	74.82	69.60	**71.06**	-11.42%
BERT$_{noise} + weighted$	81.31	78.06	79.34	80.71	75.10	76.91	-3.06%	77.73	69.73	71.57	-9.79%	75.13	66.77	68.29	-13.92%
BERT$_{noise} + first$	81.62	77.63	79.12	80.36	75.04	76.79	**-2.94%**	78.04	71.48	73.28	**-7.38%**	74.87	69.15	70.66	**-10.69%**

Table 6: Experimental results of offensive language detection when using layer-wise pooling strategies to the BERT$_{clean}$ and BERT$_{noise}$. The two models differ in whether the texts used for pre-training have been preprocessed.

more important to weigh the layers close to the last and first layers than those of the middle for offensive languages, including user-intended adversarial attacks. In summary, the strategy that focuses on layers capable of capturing offensiveness and token embeddings yields a better performance.

Nevertheless, neither of these pooling strategies performed as well as the BERT$_{clean}$ or the model that utilized the first pooling. This indicates that while information close to the last and first layers can be beneficial, there are some layers among them that hinder the performance of the model, revealing that only the last and first layers are the most helpful layers when detecting offensive language with user-intended adversarial attacks.

5.4 Beyond the Limits: Layer-Wise Pooling Strategies with Pre-trained Models on Noisy Texts

We showed that the application of layer-wise pooling strategies to the model *without* training on noisy texts allows us to achieve a performance close to that of the model pre-trained on noisy texts. We also conducted experiments to assess how robust the model pre-trained on noisy texts is when layer-wise pooling strategies are applied to defend against the proposed attacks.

The performances of the layer-wise pooling strategies for both BERT$_{clean}$ and BERT$_{noise}$ are presented in Table 6. When compared under the same pooling, the scores from the model pre-trained on noisy texts always performed better. In BERT$_{clean}$, the first pooling, which incorporates information from the first layer, performed well on the detection metrics and was robust against performance degradation. Conversely, in BERT$_{noise}$, first pooling only performed well on the detection metrics, but the strategy exhibiting more robustness to performance degradation was max pooling.

In the case of BERT$_{clean}$, because it did not directly utilize noisy texts during the training stage, first pooling performed well, which focuses most on offensiveness and token embeddings from the sentence. However, in the case of BERT$_{noise}$, where noisy texts were pre-trained into the model weights before the application of pooling strategies, max pooling proved to be the most effective on detection metrics by selecting prominent features among the information to predict the labels invariant to small changes or disturbances.

The average performances using max pooling and first pooling, which had the most impact on each model are presented in Table 7. For the model BERT$_{clean}$ pre-trained on clean texts, first pooling improved the f1-score by 2.72 and prevented a performance degradation of 2.8%. In contrast, the model BERT$_{noise}$ pre-trained on noisy texts improved its f1-score by 2.62 with max pooling, reaching a relatively high score of 74.08. The performances of these pooling strategies are remarkable considering that they do not require additional parameters or training on noisy texts.

Model	Average	
	F1	Δatk
BERT$_{clean}$	68.52	-12.85%
BERT$_{clean} + max$	68.51	-12.88%
BERT$_{clean} + first$	**71.24**	**-10.05%**
BERT$_{noise}$	71.82	-9.81%
BERT$_{noise} + max$	**74.08**	-7.64%
BERT$_{noise} + first$	73.57	**-7.00%**
Ensemble$_{hard}$ (BERT$_{clean}$ + BERT$_{noise}$)	70.99	-11.64%
Ensemble$_{soft}$ (BERT$_{clean}$ + BERT$_{noise}$)	71.23	-11.37%

Table 7: Average from the experimental results of offensive language detection when using max pooling and first pooling to the BERT$_{clean}$ and BERT$_{noise}$ in comparison to other models which exhibited the high scores.

6 Conclusion

We propose user-intended adversarial attacks that occur frequently in offensive languages online from the perspective of malicious users. We categorize them into three types: INSERT, COPY, and DECOMPOSE, which add special symbols or exploit the distinct features of the Korean language. The involvement of attacks significantly affects the tokenization results from the original text.

To address the proposed attacks, we extend the pooling strategies in a layer-wise manner. This extension utilizes not only the last layer, which focuses on offensiveness, but also the preceding layers, which focus more on token embeddings. We test these pooling strategies for the proposed attacks to various attack rates. The experimental results indicate that first pooling is robust to the proposed attacks and can even achieve a performance comparable to that of models pre-trained on noisy texts when applied to models pre-trained on clean texts. We experimentally demonstrate that the last and first layers, rather than the middle layers, can be effectively employed to detect offensive languages that reflect the proposed attacks.

Furthermore, we test the extent to which the introduced pooling strategies could handle the proposed attacks. We observe that the first pooling and max pooling are the most robust, depending on the nature of the texts used for pre-training. It is noteworthy that these strategies, *without* the explicit training of additional parameters or noisy texts, can effectively defend against user-intended adversarial attacks.

Limitations

Despite our efforts to define diverse types of adversarial attacks, there is room for undefined attacks from real-world situations, which may have an unexpected impact on the model performance. Additionally, although we used language-independent attacks such as inserting special symbols, most of the proposed attacks were based on the characteristics of the Korean language. Therefore, it is necessary to determine whether layer-wise pooling strategies can effectively handle offensive expressions written in other languages for broader applications.

Ethics Statement

Given our use of offensive representations to describe the proposed attacks, we have included a disclaimer at the beginning of this paper. From the existing offensive language datasets, potential biases regarding race, gender, political issues, and other factors might have been inherent in our experiments. This should be considered when developing our research or expanding it to other languages.

References

Piush Aggarwal and Torsten Zesch. 2022. Analyzing the real vulnerability of hate speech detection systems against targeted intentional noise. In *Proceedings of the Eighth Workshop on Noisy User-generated Text (W-NUT 2022)*, pages 230–242.

Sangcheol Ahn and Kyunney Egorova. 2021. Morphophonological patterns of recent korean neologisms. In *Conference on current problems of our time: the relationship of man and society (CPT 2020)*, pages 62–72.

Ashish Bajaj and Dinesh Kumar Vishwakarma. 2023. Homochar: A novel adversarial attack framework for exposing the vulnerability of text based neural sentiment classifiers. *Engineering Applications of Artificial Intelligence*, 126:106815.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. 2022. Bad characters: Imperceptible nlp attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Wonik Cho and Soomin Kim. 2021. Google-trickers, yaminjeongeum, and leetspeak: An empirical taxonomy for intentionally noisy user-generated text. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 56–61.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Shreya Goyal, Sumanth Doddapaneni, Mitesh M Khapra, and Balaraman Ravindran. 2023. A survey of adversarial defenses and robustness in nlp. *ACM Computing Surveys*, 55(14s):1–39.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Younghoon Jeong, Juhyun Oh, Jongwon Lee, Jaimeen Ahn, Jihyung Moon, Sungjoon Park, and Alice Oh. 2022. KOLD: Korean offensive language dataset. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10818–10833, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

Woochun Jun. 2020. A study on the cause analysis of cyberbullying in korean adolescents. *International journal of environmental research and public health*, 17(13):4648.

Bosung Kim, Juae Kim, Youngjoong Ko, and Jungyun Seo. 2021a. Commonsense knowledge augmentation for low-resource languages via adversarial learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6393–6401.

Jinyong Kim, Jeonghyeon Kim, Mose Gu, Sanghak Ohh, Gilteun Choi, and Jaehoon Jeong. 2021b. Hypocrite: Homoglyph adversarial examples for natural language web services in the physical world.

Soomin Kim, Changhoon Oh, Wonik Cho, Donghoon Shin, Bongwon Suh, and Joonhwan Lee. 2021c. Trkic g00gle: Why and how users game translation algorithms. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–24.

Keita Kurita, Anna Belova, and Antonios Anastasopoulos. 2019. Towards robust toxic content classification. *arXiv preprint arXiv:1912.06872*.

Jean Lee, Taejun Lim, Heejun Lee, Bogeun Jo, Yangsok Kim, Heegeun Yoon, and Soyeon Caren Han. 2022. K-MHaS: A multi-label hate speech detection dataset in Korean online news comment. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3530–3538, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Junbum Lee. 2020. Kcbert: Korean comments bert. In *Proceedings of the 32nd Annual Conference on Human and Cognitive Language Technology*, pages 437–440.

J Li, S Ji, T Du, B Li, and T Wang. 2019. Textbugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium*.

Xiaofei Ma, Zhiguo Wang, Patrick Ng, Ramesh Nallapati, and Bing Xiang. 2019. Universal text representation from bert: An empirical study. *arXiv preprint arXiv:1910.07973*.

Justin McCurry. 2022. South korea under pressure to crack down on cyberbullying after high-profile deaths. *The Guardian*.

Jihyung Moon, Won Ik Cho, and Junbum Lee. 2020. BEEP! Korean corpus of online news comments for toxic speech detection. In *Proceedings of the Eighth International Workshop on Natural Language Processing for Social Media*, pages 25–31, Online. Association for Computational Linguistics.

Dongsuk Oh, Yejin Kim, Hodong Lee, H. Howie Huang, and Heuiseok Lim. 2022. Don't judge a language model by its last layer: Contrastive learning with layer-wise attention pooling. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4585–4592, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Chaewon Park, Soohwan Kim, Kyubyong Park, and Kunwoo Park. 2023a. K-haters: A hate speech detection corpus in korean with target-specific ratings. *arXiv preprint arXiv:2310.15439*.

San-Hee Park, Kang-Min Kim, O-Joun Lee, Youjin Kang, Jaewon Lee, Su-Min Lee, and SangKeun Lee. 2023b. "why do I feel offended?" - Korean dataset for offensive language identification. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1142–1153, Dubrovnik, Croatia. Association for Computational Linguistics.

Sungjoon Park, Jihyung Moon, Sungdong Kim, Won Ik Cho, Jiyoon Han, Jangwon Park, Chisung Song, Junseong Kim, Yongsook Song, Taehwan Oh, et al. 2021. Klue: Korean language understanding evaluation. *arXiv preprint arXiv:2105.09680*.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1085–1097.

Suriya Saengprang and Savitri Gadavanij. 2021. Cyberbullying: The case of public figures. *LEARN Journal: Language Education and Acquisition Research Network*, 14(1):344–369.

Nihar Sahoo, Niteesh Mallela, and Pushpak Bhattacharyya. 2023. With prejudice to none: A few-shot, multilingual transfer learning approach to detect social bias in low resource languages. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13316–13330, Toronto, Canada. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Jaejung Song. 2006. *The Korean language: Structure, use and context*. Routledge.

Max Wolff and Stuart Wolff. 2020. Attacking neural text detectors. *ICLR 2020 Workshop Towards Trustworthy ML.*

Zhelun Wu, Nishant Kambhatla, and Anoop Sarkar. 2018. Decipherment for adversarial offensive language detection. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 149–159, Brussels, Belgium. Association for Computational Linguistics.

Hyunyoung Yi and Sangmi Cha. 2020. Cyber bullying, star suicides: The dark side of south korea's k-pop world. *Reuters.*

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Predicting the type and target of offensive posts in social media. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1415–1420, Minneapolis, Minnesota. Association for Computational Linguistics.

A The Proposed Attacks Details

A.1 INSERT

We employed three INSERT types by inserting special symbols that are not complete characters. The detailed methods and examples are as follows:

- INSERT_zz: We randomly added between 2~5 sounds of 'ㅋ' to the word, which means the equivalent of 'LOL' in English (on the keyboard, the sound 'ㅋ' corresponds to the alphabet 'z'). This is a commonly used expression in online communities, conveying a meaningless and somewhat frivolous tone. We placed it not only between characters but also in instances where the final sound of a specific character was empty.

- INSERT_space: We randomly added a single space to the word. We expected the same impact as the intentions of malicious users.

- INSERT_special: We randomly added a special character to the word: one of '~', '!', '@', '1', or '2'. We also expected the same impact as the intentions of malicious users.

INSERT Attacks	Text Examples
original text	틀딱이냐? (Are you a stupid boomer?)
	기레기 여기 있었네! (Here was the presstitute!)
INSERT_zz	틀ㅋㅋ딱이냐?
	기렉ㅋㅋㅋ기 여기 있었네!
INSERT_space	틀 딱이냐?
	기레 기 여기 있었네!
INSERT_special	틀@딱이냐?
	기2레기 여기 있었네!

Table 8: Text examples of user-intended adversarial attacks with the types of INSERT.

A.2 COPY

We employed three COPY types based on the distinctive characteristics of the Korean language. The detailed methods and examples are as follows:

- COPY_initial: We copied the initial sound of the character to the final sound of the preceding character. For example, if the character '기' is chosen from the word '기레기', the initial sound 'ㄱ' would be copied as the final sound to the preceding character '레', transforming it to '렉'. Thus, the attacked expression will be '기렉기'.

- COPY_middle: We copied the middle sound of the character, onto the newly added character. If the selected character had a final sound, it was also included. For example, if the character '딱' is chosen from the word '틀딱', the middle sound 'ㅏ' would be copied as the following character with the final sound 'ㄱ', adding it as '악'. Thus, the attacked expression will be '틀따악'.

- COPY_final: We copied the final sound of the character to the initial sound of the following character. For example, if the character '있' is chosen from the word '있었네', the final sound 'ㅆ' would be copied as the initial sound to the following character '었', transforming it to '썼'. Thus, the attacked expression will be '이썼네'.

A.3 DECOMPOSE

We employed two DECOMPOSE types based on the distinctive characteristics of the Korean language. The detailed methods and examples are as follows:

COPY Attacks	Text Examples
original text	틀딱이냐? (Are you a stupid boomer?)
	기레기 여기 있었네! (Here was the presstitute!)
COPY_initial	틀딱인냐?
	기렉기 여기 있었네!
COPY_middle	틀따악이냐?
	기레기 여기이 있었네!
COPY_final	틀딱기냐?
	기레기 여기 이썼네!

Table 9: Text examples of user-intended adversarial attacks with the types of COPY.

- DECOMPOSE_final: We decomposed the final sound of the character into the following newly added sound. For example, if the character ‘딱’ is chosen from the word ‘틀딱’, the final sound ‘ㄱ’ would be decomposed as the following sound. Thus, the attacked expression will be ‘틀따ㄱ’.

- DECOMPOSE_all: We decomposed all the sounds of the character. For example, if the character ‘틀’ is chosen from the word ‘틀딱’, it would be decomposed as the initial, middle, and final sounds. Thus, the attacked expression will be ‘ㅌㅡㄹ딱’.

DECOMPOSE Attacks	Text Examples
original text	틀딱이냐? (Are you a stupid boomer?)
	기레기 여기 있었네! (Here was the presstitute!)
DECOMPOSE_final	틀따ㄱ이냐?
	기레기 여기 이ㅆ었네!
DECOMPOSE_all	ㅌㅡㄹ딱이냐?
	기ㄹ ㅔ기 여기 있었네!

Table 10: Text examples of user-intended adversarial attacks with the types of DECOMPOSE.

A.4 Experiments on Each Attack

We further investigated the impact for each attack type. In the experiments in Table 3~7, we randomly selected one of all the proposed attacks. However, we selected one of the attacks from only INSERT, COPY, or DECOMPOSE with an attack rate of 60%. We chose BERT$_{clean}$ and the model that applied first pooling, which exhibited the best performance among the layer-wise pooling strategies.

The performances according to the attack types are presented in Table 11. Even when applying each type of attack, the model with first pooling always showed better performance. It is noteworthy that a high performance on a specific attack type indicates that it is easier to defend than others. INSERT alone proved to be easier than all attacks, with COPY alone only marginally harder than the INSERT. Both attack types exhibited a performance increase of about +1% to +3% compared with the application of all attacks. However, DECOMPOSE alone was more difficult than these two attacks, exhibiting a performance degradation of about -3% to -5% compared with the application of all attacks. Therefore, we revealed that adversarial attacks that reflect the characteristics of the Korean language are more challenging than those that simply add special symbols that could be adapted language independently.

Models	All Attacks			Only INSERT		
	P	R	F1	P	R	F1
BERT$_{clean}$	77.74	66.38	67.96	78.12	68.80	70.66
BERT$_{clean} + first$	77.58	69.38	71.21	77.51	70.76	72.54
Models	**Only COPY**			**Only DECOMPOSE**		
	P	R	F1	P	R	F1
BERT$_{clean}$	77.86	68.47	70.29	78.38	62.23	65.67
BERT$_{clean} + first$	77.33	70.50	72.26	77.00	66.47	68.04

Table 11: Experimental results of offensive language detection when the attack ratio is 60% for each type of user-intended adversarial attacks.

B Experimental Details

B.1 Implementation Details

We used 6 layers with an embedding dimension of 768 and a dropout ratio of 0.1 for the RNN-based models containing BiLSTM and BiGRU. As for the BERT-based models, containing BERT$_{clean}$, BERT$_{noise}$, and their ensembles, we fine-tuned 12 pre-trained layers with an embedding dimension of 768 and a dropout ratio of 0.2.

We used the AdamW optimizer with a learning rate of 1e-5, trained the models for 1~5 epochs, and considered the epoch with the lowest validation loss or the last epoch. We set a batch size of 32 for all the models. The models were implemented using PyTorch and NVIDIA GeForce RTX 3090 GPU. We also used the HuggingFace library to leverage the weights of the pre-trained BERT models.

B.2 Metrics

We collected the datasets to train as many types of offensive languages as possible. There is some label imbalance, therefore, we used macro precision, recall, and f1-score to address this issue.

Δatk represents the performance degradation in the f1-score. For example, denoting the existing f1-score as F1$_{original}$ and the f1-score with the attacks as F1$_{attacked}$, it is computed as follows:

$$\Delta atk = (\mathrm{F1}_{attacked} - \mathrm{F1}_{original})/\mathrm{F1}_{original} * 100. \quad (5)$$