Enhancing Machine Learning Based SQL Injection Detection using Contextualized Word Embedding

Janet Zulu, Joshua Ibrom, Joshua Zyzak, Izzat Alsmadi, *IEEE Senior Member*, Gongbo Liang, *IEEE Member*

***Abstract*—SQL injection (SQLi) attacks continue to pose a severe threat to application security, allowing malicious actors to exploit web input and manipulate an application's database with malicious SQL code. To address this issue, we explore the possibility of building an effective SQLi detector using machine learning models. In this study, eight models were trained and compared for their performance. Additionally, we investigate the impact of contextualized and non-contextualized word embeddings on the SQLi detection models. Our results demonstrate the superiority of contextualized embedding method, achieving consistent accuracy above 99% across various classification algorithms and reducing model training time by 31 times. Moreover, the analysis of reliability diagrams indicates that contextualized embeddings provide better model calibrations. These findings underscore the significance of contextualized word embeddings in enhancing the performance and reliability of SQLi detection models.**

***Index Terms*—Cyber security, neural network, artificial intelligence.**

I. Introduction

The prevalence of SQL injection (SQLi) attacks continues to pose a significant threat to application security. These attacks allow malicious actors to manipulate an application's database by inserting malicious SQL code through web input, leading to potentially devastating consequences (Figure1). From unauthorized data access and modification to compromising backend infrastructure and launching denial-of-service attacks, the impacts of successful SQLi attacks are far-reaching [1].

Current SQLi detection methods predominantly rely on manually defined features, but their ability to handle the ever-evolving range of real-world attacks remains a concern [2]–[4]. To address this challenge, machine learning (ML) approaches, such as neural networks (NNs), learn task specific features directly from extensive training data offering potential advantages over conventional hand-crafted features in terms of robustness and precision [5], [6].

As a result, ML models show promise as effective tools for SQLi detection. The conceptual framework of employing ML models to thwart SQLi attacks involves passing user input through a pre-trained ML model to detect the presence of malicious code, and only inputs deemed safe are subsequently forwarded to the web API server (Figure 2).

However, developing an ML-based application poses its own challenges, particularly due to the abundance of available

Janet Zulu, Izzat Alsmadi, and Gongbo Liang are with the Department of Mathematical, Engineering, Computational Sciences, Texas A&M University-San Antonio, San Antonio, TX, USA (e-mail: jzulu01@jaguar.tamu.edu, ialsmadi@tamusa.edu gliang@tamusa.edu).

Joshua Zyzak is with the Department of Computer Science and Information Technology, Eastern Kentucky University, Richmond, KY, USA (e-mail:joshua_zyzak@mymail.eku.edu).

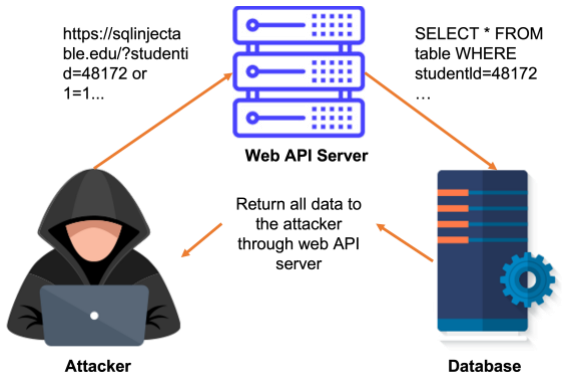

Fig. 1: Illustration of SQLi attacks.

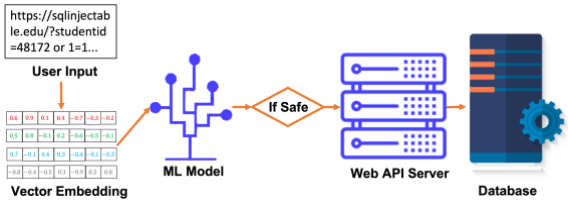

Fig. 2: Illustration of using machine learning model to prevent SQLi attacks.

building blocks, which might not be intuitively applicable. Specifically, designing ML model for SQLi detection, which entail analyzing textual input to identify potential malicious code, necessitates careful consideration of architectural design choices and experimental configurations.

On the architectural front, the process of transforming textual input into numerical vectors suitable for ML models requires choosing from various word embedding methods, such as Bag-of-Words, Word2Vec [7], FastText [8], BERT [9], and RoBERTa [10]. Following this transformation, the vectors feed into a detection model, comprising feature-learning and decision-making components. Making informed architectural decisions, such as determining the specific model and its hyperparameters, is crucial for optimizing ML performance. Moreover, experimental choices, including the selection of optimizers, the use of learning rate scheduling, and the adoption of specific loss functions, further influence model efficacy.

In this paper, our primary focus is on the architectural design aspect, specifically examining the influence of word embedding methods on the performance of various ML-based SQLi detection models, including neural networks, random forest, and logistic regression. Our experimental results reveal that

the use of contextual embedding methods not only enhances the model training efficiency (achieving 100 times or more faster training) but also leads to a notable improvement in neural network calibration, with gains of up to 14%. The subsequent sections of this paper introduces the common building blocks of ML-based SQLi detector and the ones used in this study (Section II), a comprehensive evaluation is presented (Section III), and conclude with discussions and insights (Section IV).

II. Machine Learning based SQLi Detector

A. Define the Problem

Since the SQLi attackers need to inject SQL queries into textual input and pass it to a web API server, we decided to model the SQLi detection as a natural language binary classification problem. The machine learning models are fully supervised trained. Specifically, each training sample is represented as a two-tuple $\{x, y\}$, with x representing the textual input and y is the label that indicates the presents of SQL injection in x. The machine learning model is represented as:

$$(\hat{y}, \hat{p}) = h(x), \tag{1}$$

where $\hat{y}$ is the predicted label and $\hat{p}$ is the predicted probability. Ideally, we want $\hat{y}$ to be identical to y and $\hat{p}$ to be the true data distribution (p). A perfectly calibrated classification model should present the following property:

$$\mathsf{P}(\hat{y} = y | \hat{p} = p) = p, \ \forall p \in [0, 1], \tag{2}$$

where $\mathsf{P}(\cdot|\cdot)$ indicate the conditional probability.

B. Word Embedding Method

In the process of building a classification model for textual input, the initial step involves transforming the raw text into vector representations using word embedding methods. These methods can be broadly categorized into two groups: non-contextualized embedding and contextualized embedding.

1) Non-contextualized Embedding: Non-contextualized word embedding methods generate fixed vector representations for each word in the vocabulary without considering the context in which the word appears in a sentence or document. Popular non-contextualized embeddings include Bag-of-Words (BoW), Word2Vec, and more. These methods typically use co-occurrence statistics or predict words based on their neighboring words in a large corpus to create word embeddings. As a result, the embeddings remain the same regardless of the sentence or document they appear in, making them computationally efficient and straightforward to implement.

In our work, we adopt the Bag-of-Words model as our non-contextualized method since it is a fundamental technique, which is widely used in natural language processing (NLP) [11], [12]. In this approach, text is treated as an unordered collection of words or n-grams, and the frequency of each word is used to represent the text. While the BoW model does not capture word order or contextual information, it remains a popular choice for scenarios where word frequency and occurrence information suffice for the task at hand, offering a balance between efficiency and simplicity.

2) Contextualized Embedding: Contextualized word embedding methods generate word representations that adapt to the context in which words appear, resulting in dynamic embeddings sensitive to surrounding words. This contextualization enables the embeddings to capture subtle nuances in word meaning and resolve ambiguities in polysemous words based on their context. RoBERTa is a of the prominent and influential models for NLP contextualized embedding. RoBERTa, short for "A Robustly Optimized BERT Pretraining Approach," is a variant of BERT that enhances bidirectional context modeling during pretraining, utilizing the transformer [13] architecture. A pretrained RoBERTa model may be finetuned for various tasks without requiring too much effort [14]–[16].

In this study, we use RoBERTa as our contextualized embedding method for SQL injection (SQLi) detection in this study. With RoBERTa's capabilities in capturing contextual information, we aim to enhance the effectiveness and accuracy of our SQLi detection model.

C. Machine Learning Models

Given the natural of SQLi being modeled as a binary classificaiton task, several machine learning models can be trained as the detection, such as k-nearest neighbor (KNN), decision tree, logistic regression, random forest, etc. In this work, we uses KNN, logistic regression, and random forest.

1) KNN: KNN is a simple and effective algorithm used for classification tasks. It works based on the principle that similar data points are more likely to belong to the same class. In our case, for each new textual input x, KNN identifies the K nearest training samples and classifies x based on the majority class among its neighbors. KNN's simplicity makes it easy to implement and interpret, but its performance might be sensitive to the choice of K and the distance metric.

2) Logistic Regression: Logistic Regression is a widely used statistical method for binary classification. It models the relationship between the input features and the probability of the positive class (SQL injection present) using a logistic function. The model can be trained using optimization techniques to find the best parameters that minimize the logistic loss. Logistic Regression is computationally efficient, interpretable, and can handle both linear and non-linear relationships between features and the target variable.

3) Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. Each tree is trained on a random subset of the data with random feature subsets, and the final prediction is determined by aggregating the predictions of individual trees. Random Forest is robust against overfitting, works well with high-dimensional data, and can capture complex relationships in the data. It is less sensitive to the choice of hyperparameters compared to single decision trees.

D. Neural Network Model

In recent years, the remarkable success of neural networks (NNs) has captured considerable attention across diverse domains, ranging from medical imaging [17]–[19] to astrophysics and astronomy [20]–[22]. Inspired by their impressive

TABLE I: The neural network architecture used in this study.

Layer	CountVectorizer	RoBERTa
Input	40,933	768
FC1	256 ReLU Batch Norm	128 ReLU Batch Norm
FC2	256 Dropout (p=0.2) ReLU Batch Norm	128 Dropout (p=0.2) ReLU Batch Norm
FC3	2	2
Parameter Count	10,546,434	115,714

capabilities, we also incorporated neural networks as one of the classification models in our SQL injection detection work.

In this study, we also designed and trained neural networks for feature learning and decision making. This process involves passing the data through multiple layers of interconnected neurons, where each layer extracts increasingly abstract representations of the input. For our specific problem, we employed a deep feedforward neural network, commonly known as a multi-layer perceptron (MLP) model. The MLP consists of an input layer that takes the textual input x and processes it through hidden layers, ultimately leading to the output layer that predicts the probability of SQL injection ($\hat{y}$). The choice of the number of hidden layers and the number of neurons in each layer was guided by experimentation to strike the right balance between model complexity and generalization. The specific structures of two NN models used in this study are shown in Table I.

To train the neural network, we utilized a supervised learning approach with labeled training samples $\{x, y\}$. During training, the network learns the optimal weights and biases by minimizing the binary cross-entropy loss (BCE):

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{p}_i) + (1-y) \cdot (log(1-\hat{p}_i). \quad (3)$$

Through rigorous experimentation and performance evaluation, we compared the neural network's performance with the other classification algorithms (K-Nearest Neighbors, Logistic Regression, and Random Forest). This comprehensive analysis allowed us to determine which model and embedding method achieved the most robust and accurate SQL injection detection results.

III. Experiments and Results

A. Dataset

A SQLi dataset from Kaggle[1] is used in this project. The dataset consists of three subsets, totaling 64, 645 samples. After combining the subsets, we identified and removed 310 samples that lacked proper labels, resulting a dataset of 64,335 labeled instances with 41, 572 being negative cases and 22, 763 being positive cases (Table II). We randomly divided the dataset into training, validation, and test sets, approximately in a 72 : 13 : 15 ratio.

[1]https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

TABLE II: Details of the dataset used in this study.

Dataset Size	64,335
Positive/Negative Samples	22,763 / 41,572
Vocabulary Size	40,933
Train/Val/Test Split	46,482 / 8,203 / 9,650

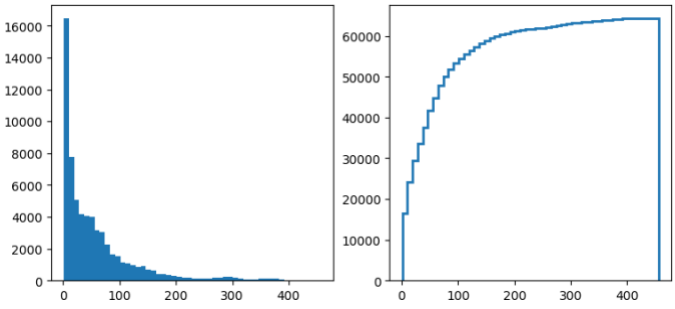

Fig. 3: Histograms of the sample length distribution with both as probability density function (left) and cumulative distribution function (right).

The dataset consists of 40,933 unique tokens (words). The samples in the dataset exhibit a wide range of lengths, varying from 1 token to 5,370 tokens, with the majority of samples being less than 100 tokens in length. Figure 4 depicts the histograms of the sample length distribution, displaying both the probability density function (PDF) and cumulative distribution function (CDF).

B. Evaluation Metrics

Given the binary classification nature of the task, we evaluate model performance using accuracy (Acc), F1 score (F1), Precision, and Recall. The accuracy is calculated with the following:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \quad (4)$$

where TP is the True Positive, TN is the True Negative, FP is the False Positive, and FN is the False Negative.

The precision is calculated by the following:

$$precision = \frac{TP}{TP + FP}. \quad (5)$$

The recall is calculated by the following:

$$recall = \frac{TP}{TP + FN}. \quad (6)$$

The F1-score is calculated by the following:

$$F1 = 2 \times \frac{precision \cdot recall}{precision + recall}. \quad (7)$$

In addition to evaluating the model's performance in terms of accuracy, we also assess the model's training efficiency in seconds and model calibration using reliability diagrams and mean calibration error.

TABLE III: The embedding space of different methods.

CountVectorizer	RoBERTa
40,933	768

C. Experiment Setup

This study was conducted using Google Colaboratory, which provided 12GB of RAM and a Nvidia Tesla T4 GPU with 16GB of memory. The `CountVectorizer` method from the `scikit-learn` library [23] served as the BoW embedding method, while the `RobertaModel` (i.e., `roberta-base`) from HuggingFace [24] was utilized for contextualized embedding. The BoW embedding was fitted on the training and validation sets, resulting in an embedding space of 40,933. For the contextualized embedding, we directly used the HuggingFace pre-trained weights of RoBERTa, resulting in a feature space of 768 for each sample (Table III). To compare the performance of the two embedding methods, we trained eight machine learning models using four classification algorithms: logistic regression, random forest, k-nearest neighbor, and multi-layer perceptron (neural network). The logistic regression model XXXGBL: [Need to add the hyperparameter settings later]. The setup for the multi-layer perceptron models is detailed in Table I.

The logistic regression, random forest, and KNN models were directly loaded from the `scikit-learn` library and trained on the CPU. In contrast, the multi-layer perceptron models were implemented using the `PyTorch` library [25] and trained on the GPU.

D. Experimental Result

Table IV presents the performance of the eight SQLi detection models. The models using contextualized embedding (RoBERTa) exhibit consistent performance, achieving accuracy above 99%, with the highest accuracy of 99.61% attained by logistic regression. However, our experience indicates that increasing model complexity can further improve the performance of the multi-layer perceptron and random forest models. Since all models already achieve accuracy above 99%, we decided not to focus on pushing the accuracy further, as it would not yield substantial practical gains.

On the other hand, the table reveals significant disparities in the performance of the BoW embedding (CountVectorizer) models. While the multi-layer perceptron model achieves above 99% accuracy, other models show considerable variation. For instance, the random forest model achieves only 67.62% accuracy, barely surpassing random guessing. Precision and recall scores highlight that the random forest model is heavily biased towards negative classes. We attribute this performance drop to the large BoW embedding space size, which drastically increases computation requirements and necessitates more complex models for effective classification.

Another drawback of the BoW embedding is the large embedding space increases the training time complexity dramatically. For instance, it only takes 6 seconds to train a multilayer perceptron model using contextualized embedding for 10 epochs. However, the time increases over 31 times when training a multilayer perceptron model to achieve a similar performance using the BoW embedding. In addition, the large embedding space also requests a unanimous amount of computational power and high memory consumption. In fact, the training of logistic regression cannot be completed due to the high memory consumption.

E. Neural Network Calibration

Recent advances in deep neural networks have had a profound impact on numerous research domains. While researchers continuously strive to achieve higher classification model performance, uncertainty quantification is often overlooked. Nevertheless, quantifying uncertainty in neural networks is crucial, particularly in automated decision-making systems [26], [27]. An automated method that achieves high accuracy but poorly captures uncertainty could result in significant treatment errors [28]. Classification model calibration error, as defined in Equation 2, is a widely used metric to evaluate the uncertainty of neural network models.

Reliability diagrams provide graphical representations of the calibration quality. These diagrams divide the predicted probabilities into bins and plot the average predicted probability against the empirical probability (the actual proportion of positive outcomes) for each bin. An ideal calibration is reflected by points close to the diagonal line on the diagram [26].

In Figure 4, we observe similar training trends across the 10 training epochs for the multilayer perceptron models with CountVectorizer (top) and RoBERTa (bottom) embeddings, except for precision and recall, which show opposite behaviors. However, when examining the reliability diagrams (Figure 5), we notice that the CountVectorizer trained model (left) is poorly calibrated compared to the RoBERTa trained model (middle).

Ideally, all the bins in a reliability diagram should align on the diagonal line, indicating perfect calibration. While none of the models achieve perfect calibration, the model trained with CountVectorizer performs worse in this aspect. Specifically, it severely underestimates the probabilities of samples falling within the median probability bins (i.e., bins of 0.5 and 0.6). Figure 5 (Right) directly compares the two models by plotting them together.

To quantify the calibration performance, we computed the mean differences between the bins and the diagonal line for each diagram. The results show that the RoBERTa trained model has a mean calibration error of 0.1664, while the CountVectorizer trained model has a mean calibration error of 0.1925, which is about 16% higher than its counterpart. This indicates that the RoBERTa model achieves better calibration compared to the CountVectorizer model.

IV. Discussion and Conclusion

In this study, we compared the performance of SQL injection (SQLi) detection models using both contextualized and non-contextualized word embeddings. Specifically, we utilized the RoBERTa model as our contextualized embedding method and the CountVectorizer as our non-contextualized method, training eight machine learning detection models. The results

TABLE IV: Model performance across different classification algorithms and embedding methods.

Embedding Method	ML Model	Accuracy	F1 Score	Precision	Recall	Training Time
CountVectorizer	**Multilayer Perceptron**	0.9907	0.9870	0.9891	0.9850	194
	Random Forest	0.6762	0.1510	1.0000	0.0817	211
	Logistic Regression	Models are not trainable due to high memory consumption.				
	K-Nearest Neighbor	*XX*	*XX*	*XX*	*XX*	*XX*
RoBERTa	**Multilayer Perceptron**	0.9907	0.9870	0.9901	0.9840	6
	Random Forest	0.9801	0.9712	0.9923	0.9509	59
	Logistic Regression	0.9961	0.9944	0.9979	0.9909	8
	K-Nearest Neighbor	*XX*	*XX*	*XX*	*XX*	*XX*

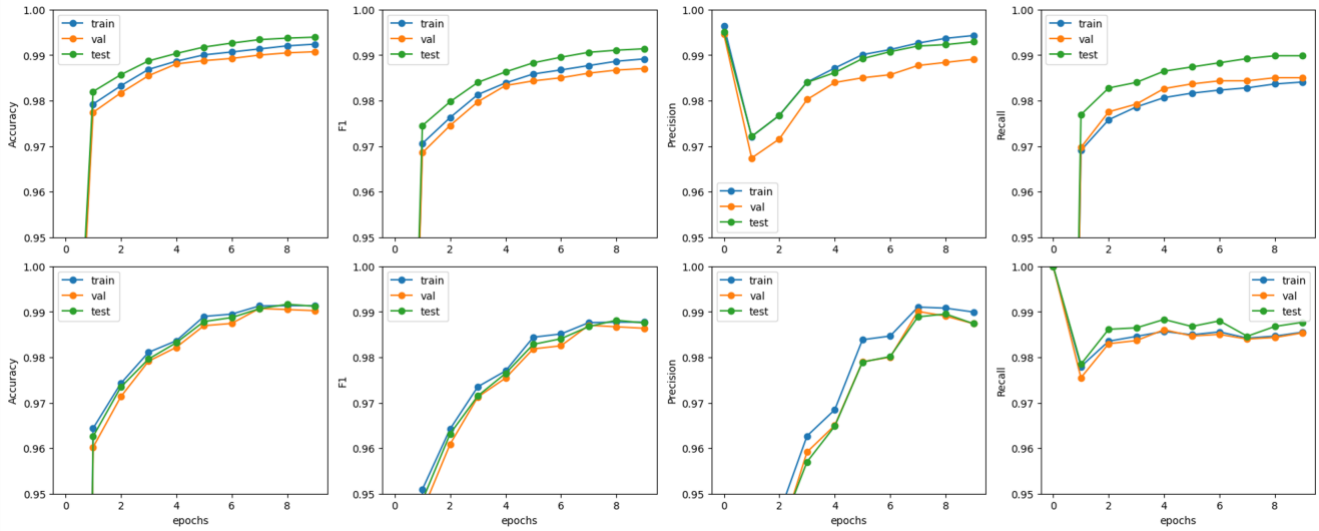

Fig. 4: Training curves of the multilayer perception models with countVectorizer embedding (Top) and RoBERTa embedding (Bottom) on train, validation, and test sets accross 10 epochs.

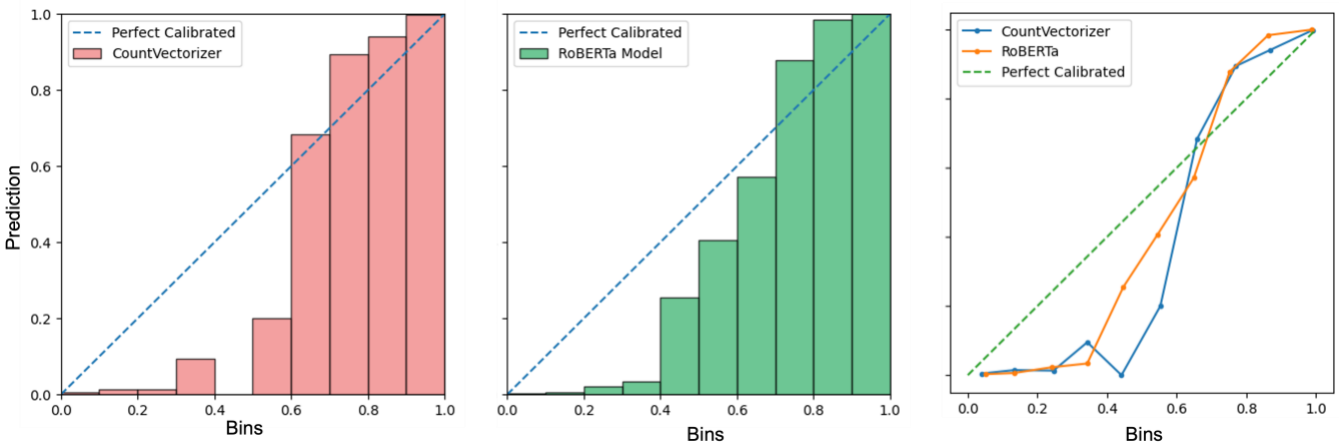

Fig. 5: Reliability diagrams of the multilayer perception models with countVectorizer embedding (Left), RoBERTa embedding (Middle), and a direct comparison of the two models into one figure (Right).

demonstrated the superiority of RoBERTa, achieving consistent performance with accuracy above 99% across various classification algorithms, while significantly reducing model training time by 31 times.

The analysis of the reliability diagrams further revealed that RoBERTa's contextualized embeddings provided better calibration compared to CountVectorizer's embeddings. The RoBERTa trained model exhibited closer alignment of the reliability diagram bins to the diagonal line, indicating more accurate probability predictions.

Our findings emphasize the importance of contextualized word embeddings, such as RoBERTa, in enhancing the perfor-

mance and reliability of SQLi detection models. RoBERTa's ability to capture contextual information allows it to effectively handle nuances in word meaning and disambiguate polysemous words, leading to higher accuracy and better calibration compared to the non-contextualized CountVectorizer.

However, we acknowledge that the time complexity evaluated in this study focuses solely on model training and does not consider the pre-training process of the contextualized embedding method. Additionally, due to RoBERTa's larger model size, its embedding process may require more computational resources than CountVectorizer.

In future work, it would be beneficial to explore methods that mitigate the computational overhead of contextualized embeddings, making them more feasible for practical applications. Furthermore, investigating other state-of-the-art contextualized embedding techniques and their impact on the performance and calibration of SQLi detection models could provide valuable insights.

Overall, this study contributes to our understanding of the role of word embeddings in SQLi detection and underscores the significance of considering both accuracy and uncertainty quantification when designing automated decision-making systems.

REFERENCES

[1] W. G. Halfond, J. Viegas, A. Orso *et al.*, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE international symposium on secure software engineering*, vol. 1. IEEE, 2006, pp. 13–15.

[2] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "Detection of sql injection based on artificial neural network," *Knowledge-Based Systems*, vol. 190, p. 105528, 2020.

[3] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh, and R. Al-Qassas, "A systematic review of detection and prevention techniques of sql injection attacks," *Information Security Journal: A Global Perspective*, pp. 1–14, 2021.

[4] Q. Li, F. Wang, J. Wang, and W. Li, "Lstm-based sql injection detection method for intelligent transportation system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4182–4191, 2019.

[5] Z. A. Shboul, M. Alam, L. Vidyaratne, L. Pei, M. I. Elbakary, and K. M. Iftekharuddin, "Feature-guided deep radiomics for glioblastoma patient survival prediction," *Frontiers in neuroscience*, vol. 13, p. 966, 2019.

[6] K. Kobayashi, M. Miyake, M. Takahashi, and R. Hamamoto, "Observing deep radiomics for the classification of glioma grades," *Scientific Reports*, vol. 11, no. 1, pp. 1–13, 2021.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[10] L. Zhuang, L. Wayne, S. Ya, and Z. Jun, "A robustly optimized BERT pre-training approach with post-training," in *Proceedings of the 20th Chinese National Conference on Computational Linguistics*. Huhhot, China: Chinese Information Processing Society of China, Aug. 2021, pp. 1218–1227.

[11] T. Walkowiak, S. Datko, and H. Maciejewski, "Bag-of-words, bag-of-topics and word-to-vec based subject classification of text documents in polish-a comparative study," in *Contemporary Complex Systems and Their Dependability: Proceedings of the Thirteenth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, July 2-6, 2018, Bruno´w, Poland 13*. Springer, 2019, pp. 526–535.

[12] K. Juluru, H.-H. Shih, K. N. Keshava Murthy, and P. Elnajjar, "Bag-of-words technique in natural language processing: a primer for radiologists," *RadioGraphics*, vol. 41, no. 5, pp. 1420–1426, 2021.

[13] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[14] G. Yang, Y. Zhou, C. Yu, and X. Chen, "Deepscc: Source code classification based on fine-tuned roberta," *arXiv preprint arXiv:2110.00914*, 2021.

[15] Z. Xu, "Roberta-wwm-ext fine-tuning for chinese text classification," *arXiv preprint arXiv:2103.00492*, 2021.

[16] G. Liang, J. Guerrero, F. Zheng, and I. Alsmadi, "Enhancing neural text detector robustness with μ attacking and rr-training," *Electronics*, vol. 12, no. 8, p. 1948, 2023.

[17] X. Xing, G. Liang, Y. Zhang, S. Khanal, A.-L. Lin, and N. Jacobs, "Advit: Vision transformer on multi-modality pet images for alzheimer disease diagnosis," in *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2022, pp. 1–4.

[18] Q. Ying, X. Xing, L. Liu, A.-L. Lin, N. Jacobs, and G. Liang, "Multi-modal data analysis for alzheimer's disease diagnosis: An ensemble model using imagery and genetic features," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2021, pp. 3586–3591.

[19] G. Liang, X. Xing, L. Liu, Y. Zhang, Q. Ying, A.-L. Lin, and N. Jacobs, "Alzheimer's disease classification using 2d convolutional neural networks," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2021, pp. 3008–3012.

[20] S.-C. Lin, Y. Su, G. Liang, Y. Zhang, N. Jacobs, and Y. Zhang, "Estimating cluster masses from sdss multiband images with transfer learning," *Monthly Notices of the Royal Astronomical Society*, vol. 512, no. 3, pp. 3885–3894, 2022.

[21] Y. Zhang, G. Liang, Y. Su, and N. Jacobs, "Multi-branch attention networks for classifying galaxy clusters," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 9643–9649.

[22] G. Liang, Y. Su, S.-C. Lin, Y. Zhang, Y. Zhang, and N. Jacobs, "Optical wavelength guided self-supervised feature learning for galaxy cluster richness estimate," in *Neural Information Processing Systems (NeurIPS) Workshop on Machine Learning and the Physical Sciences*, 2020.

[23] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae *et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[24] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv:1910.03771*, 2019.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[26] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proc. ICML*, 2017, pp. 1321–1330.

[27] G. Liang, Y. Zhang, X. Wang, and N. Jacobs, "Improved trainable calibration method for neural networks on medical imaging classification," in *British Machine Vision Conference (BMVC)*, 2020.

[28] X. Jiang, M. Osl, J. Kim, and L. Ohno-Machado, "Calibrating predictive model estimates to support personalized medicine," *Journal of the American Medical Informatics Association*, vol. 19, no. 2, pp. 263–274, 2011.