SQL Injection Detection Using Fine-Tuned Transformers

Nitish Vobilisetti
University of Maryland, College Park

Aditya Kishore
University of Maryland, College Park

1 Introduction

SQL injection attacks pose a significant threat to data integrity across online platforms. These attacks exploit vulnerabilities in database management systems by allowing unauthorized access to sensitive information. Traditional detection methods cannot keep up with modern cyber threats. This research will bridge the gap between natural language processing (NLP) techniques and cybersecurity by harnessing transformer-based models, specifically BERT (Bidirectional Encoder Representations from Transformers), for the detection of SQL injection attacks. By fine-tuning these models with a triplet loss function, we develop a system that can syntactically recognize patterns in SQL queries and discern the semantic indicators of malicious intent. We hypothesize that embeddings from fine-tuned transformer models can significantly enhance the performance of classification algorithms in identifying SQL injections. Using logistic regression models as a benchmark, we evaluate the efficacy of our fine-tuned embeddings in this classification task. This paper outlines a thorough literature review, resources, procedure, evaluation, and the current status of our research. Through our research, we believe that integrating sophisticated NLP techniques into security frameworks is a proactive step towards a more secure cyber landscape. The insights gained from this research pave the way for practical applications that could reshape security protocols in database management systems across the industry.

2 Literature Review of Related Work

Programmatic detection and prevention of SQL injection attacks have been active areas of research for the past few years; as web applications stake an ever-wider claim to sensitive data, ensuring the security of these applications against SQL injection attacks is crucial. Various approaches have been proposed to tackle this issue, ranging from traditional rule-based methods to, more recently, machine learning-based techniques. New deep learning and transformer-based models such as BERT have opened up new possibilities for improving the accuracy and effectiveness of SQL injection attack detection.

This literature review examines prior research on detecting SQL injection attacks, with a particular emphasis on machine learning and deep learning approaches. We will analyze the contributions and shortcomings of existing studies in this domain. A key focus will be on evaluating the potential benefits of leveraging fine-tuned BERT models to improve SQL injection attack recognition capabilities. By critically examining state-of-the-art techniques and identifying gaps in the current literature, we aim to lay the groundwork for our proposed methodology: utilizing a fine-tuned BERT model tailored for this crucial cybersecurity challenge. Our goal is to provide a comprehensive overview that informs and motivates our novel approach to enhancing SQL injection attack detection.

Some alternative approaches can have better efficiency and accuracy than most algorithmic approaches under certain conditions. In 2008, Kemalis et al. [6] showed that, with a contextual understanding of the application SQL usage, injections can be efficiently and accurately classified (without many of the resource tradeoffs present today); however, the system is case-specific, leaving a need for a generalized system that evaluates queries in a vacuum.

One of the earliest machine-learning based approaches to identifying SQL injection vulnerabilties was presented by Pinzón et al. [7] in 2011. They presented a hybrid approach called AIIDA-SQL for detecting SQL injection attacks; the proposed system augmented prior heuristic-based approaches with learning and adaptation capabilities using a classification mechanism that integrates an neural network and a support vector machine. This approach, while demonstrably better than naive regression techniques, is considerably less accurate than modern fine-tuned NLP models.

A variety of approaches to this problem exist, many of which are more computationally efficient than machine learning-based detection. The aforementioned 'heuristic' approach taken by Buja et al. [3] and others utilizes knowledge of common patterns SQL injection attacks take, utilizing string-matching or other low-complexity approaches to sanitize queries. Another similar approach presented by Appiah et

al. [2] is to build 'fingerprints' for attacks based on a tagged database and perform pattern matching against new queries to calculate a probability of attack. These approaches are computationally cheap, but give up accuracy compared to algorithmic techniques like the one we present in this paper.

Machine-learning techniques historically lagged due to the computational issues mentioned above. In recent years, hardware advancements and highly efficient models have made this less of a concern; in response, a flurry of research has been published about the application of machine learning to the SQL injection attack problem (often abbreviated SQLIA). This research, often focused on applying traditional classification algorithms in this novel context; for instance, Tang et al. [8] utilized an ANN, Hasan et al. [5] tested 23 different classifier algorithms, and Xie et al. [9] evaluated a CNN. Each of these approaches had good experimental results, but none of these models are specifically suited for the natural-language task of identifying SQLIA threats.

BERT (Bidirectional Encoder Representations from Transformers) [4] is a pre-trained bi-directional model; it abridges the computational complexity introduced in prior machine learning approaches by providing a contextual understanding of text, standing in contrast to standard classification neural networks. By fine-tuning a BERT model, we hope to achieve greater accuracy at a lower computational cost than prior approaches.

3 Experimental Design and Approach

In this section, we will outline the tools, datasets, and resources we need to perform our analysis and research. We will outline our experimental procedure and the scientific significance of certain steps.

3.1 Dataset and Tools

We will be using a dataset [1] from Kaggle. This dataset is a comprehensive collection of SQL queries labeled as benign (0) or malicious (1), signifying the presence of SQL injection. The dataset is vital for us to train and evaluate our machine learning models. This dataset has a rich variety of SQL command queries. Each record in the dataset represents a SQL query (input feature) and is accurately annotated with a target feature. This supervised learning approach with labeled data allows the model to learn the differentiation between legitimate queries and SQL injection attempts.

The experiment uses the BERT (Bidirectional Encoder Representations from Transformers) model, sourced from Hugging Face's repository of pre-trained models. BERT serves as our pre-trained transformer, which we aim to fine-tune for our specific use case of identifying SQL injection. Initially, BERT is used to generate baseline embeddings for SQL queries. It is then fine-tuned to specifically distinguish SQL injection and produce better embeddings.

We will use logistic regression to classify SQL queries into benign or malicious categories based on their embeddings. This method allows us to efficiently compare the classification ability of embeddings produced by the baseline versus fine-tuned BERT models. We chose logistic regression because it is simple and best suited for binary classification.

The fine-tuning process incorporates the triplet loss function. This loss function improves the model's discrimination capability. This function aims to minimize the distance between embeddings of similar queries and maximize the distance between those of dissimilar queries. The distance minimization and maximization enhance the model's accuracy in distinguishing between benign and malicious SQL queries.

The experiment is conducted within Jupyter Notebooks. This environment offers a flexible coding environment to develop, document, and test machine learning models. Jupyter makes it convenient for us to perform exploratory data analysis and visualization of the dataset and results.

Our research will leverage several Python libraries to carry out our experiment.

- transformers from Hugging Face for accessing the pre-trained BERT models and NLP utilities.
- scikit-learn for implementing logistic regression and evaluating model performance.
- pandas for efficient dataset manipulation and access.
- numpy for conducting essential numerical operations in data preprocessing and to represent vector embeddings.
- matplotlib and seaborn for visualizing data and results.

3.2 Experimental Procedure

We will implement the triplet loss function as the objective for fine-tuning the BERT model. This involves carefully selecting triplets from the dataset. Each triplet consists of an anchor (a particular SQL query), a positive example (a similar query, ideally with the same intent or classification), and a negative example (a dissimilar query, with a different intent or classification). We then fine-tune the BERT model on these triplets. The training process involves adjusting the model's internal weights to minimize the triplet loss across multiple epochs. Essentially, the model learns embeddings where the anchor is closer to the positive example than to the negative example in the embedding space. The triplet loss function is commonly used for training neural networks that need to understand the similarity and dissimilarity between data points. It works by

simultaneously considering three datums (a triplet) during the training process. The objective of the loss function is to ensure that, in the embedding space, the distance between the anchor and the positive example is less than the distance between the anchor and the negative example by a certain margin. Mathematically, this is expressed as minimizing the loss function:

$$L = \max\left(d(a,p) - d(a,n) + \text{margin}, 0\right)$$

L is the loss, $d(a,p)$ is the distance between the anchor and the positive example, $d(a,n)$ is the distance between the anchor and the negative example, and margin is a predefined margin value to ensure a buffer between the positive and negative distances. During the fine-tuning process, BERT's internal weights are adjusted to minimize the triplet loss. This involves backpropagation, where the model's errors in distinguishing between similar and dissimilar SQL queries are calculated, and gradients are computed for each weight in the model. These gradients are then used to update the weights in a direction that reduces the loss. Through repeated training epochs, the model learns to produce embeddings that accurately reflect the semantic similarities and differences between SQL queries, significantly improving its ability to detect SQL injections by understanding the distinctions between benign and malicious queries. We use triplet loss for fine-tuning because we want the model to understand complex relationships between data points. This is important in the context of SQL injection detection, where malicious and benign queries may be subtly distinct. By optimizing the model to recognize and emphasize these distinctions in the embedding space, triplet loss fine-tuning enhances BERT's sensitivity to the semantic nuances that differentiate SQL injections from legitimate queries. Lastly, we will train two logistic regression models—one using the baseline BERT embeddings and another using the fine-tuned embeddings. We will split the dataset into training and testing sets to evaluate the models' performances objectively.

4 Data Analysis

We will evaluate and compare the embedding-based regression models using accuracy, precision, recall, F1 score, and possibly AUC-ROC.

- Accuracy measures the overall correctness of the model across all classes.
- Precision (positive predictive value) indicates the proportion of positive identifications that were actually correct, helping us reduce false positives.
- Recall (sensitivity) assesses the proportion of actual positives that were correctly identified, helping us reduce false negatives.
- F1 Score shows a balance between precision and recall. It conveys the overall model performance where both false positives and false negatives are costly.

These metrics identify not just how often the model is right, but how it's right or wrong. This is important when detecting SQL injections because the cost of false positives (blocking legitimate queries) and false negatives (allowing malicious queries) can be detrimental. This is especially crucial in queries that work on databases that back customer-facing applications.

Additionally, we will generate graphs to visualize the clustering of embeddings before and after fine-tuning. These visualizations can illustrate the effect of fine-tuning on the model's ability to differentiate between benign and malicious SQL queries in the embedding space. Clusters that are more distinct and separated indicate a better understanding of the differences between classes.

5 Results

We generated embeddings using both the base transformer model and fine-tuned transformer model. We visualized the embeddings of SQL queries before and after fine-tuning a BERT model, as well as on a test set after fine-tuning. These visualizations showcase the embeddings' ability to differentiate whether a query is benign or a potential SQL injection.

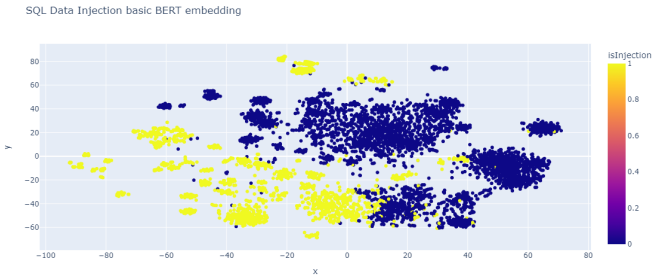

Figure 1: Basic BERT embeddings

The first graph (see Figure 1) shows the embeddings generated by the basic BERT model without any fine-tuning. The overlapping clusters of queries, denoted by the proximity of blue (benign) and yellow (malicious) points, suggest that the model does not separate SQL injections from benign queries well.

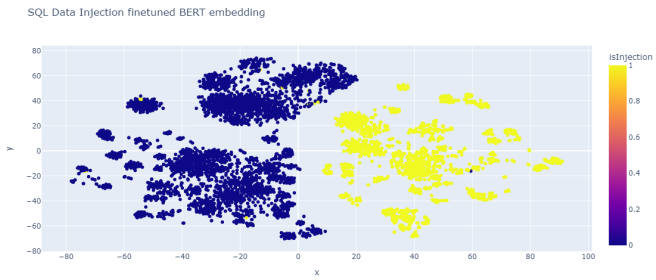

Figure 2: Fine-tuned BERT embeddings

After fine-tuning the BERT model using a triplet loss function, we observe a more pronounced separation between the classes of queries (see Figure 2). The malicious queries are more distinctly clustered. This implies an improvement in the model's ability to generate discriminative embeddings.

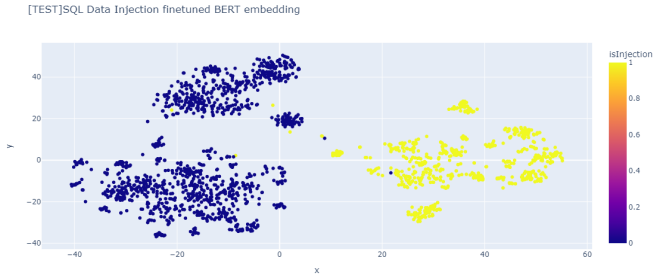

Figure 3: Fine-tuned BERT embeddings on test set

Figure 3 displays the embeddings from the fine-tuned BERT model when applied to the test set. The clear distinction between benign and malicious queries is still there, reinforcing the model's generalization capabilities.

This comparison illustrates the effectiveness of fine-tuning BERT with the triplet loss function. The fine-tuning process has refined the model's internal mechanisms, allowing it to better capture the semantic differences between benign and malicious SQL queries. These more defined embeddings allow us to proceed with the next stage: training logistic regression models.

Metric	**Basic BERT Embeddings**	**Fine-Tuned BERT Embeddings**
Accuracy	0.875	0.998
Precision	0.8396	0.9986
Recall	0.8436	0.9958
F1 Score	0.8416	0.9972

Table 1: Performance Comparison of Logistic Regression Models

Table 1 compares the performance of two logistic regression models trained on embeddings generated by the base BERT model and the fine-tuned BERT model. This comparative analysis proves the improved performance of fine-tuning a BERT model for SQL injection detection.

The logistic regression model trained on the embeddings generated by the base BERT model achieved an accuracy of 87.5%. This is impressive, but the the precision and recall rates are both approximately 84%. This suggests that the model is generally reliable, but there are limitations in its ability to precisely identify and recall malicious queries, potentially leading to a significant rate of false positives and false negatives. In contrast, the logistic regression model trained on the embeddings generated by the fine-tuned BERT shows a dramatic improvement in all aspects. It achieves a near perfect accuracy at 99.8%, with close to perfect precision and recall. This suggests a high reliability in detecting SQL injections, significantly reducing the risks of false positives and false negatives.

The jump in accuracy from 87.5% to 99.8% indicates that the model trained on the fine-tuned embeddings is generally more reliable in classifying SQL queries. The fine-tuned model is almost always correct in classifying SQL queries. This is important for applications where the cost of errors can be very high, such as in financial and personal data security. The precision improved from approximately 84% to nearly 99.9%. This enhancement reduces the risk of false positives. In the context of this research, reducing false positives is crucial for not disrupting legitimate database activities. High precision ensures that benign operations are not mistakenly blocked to unnecessarily protect databases. The recall improved from about 84% to approximately 99.6%. This means that the fine-tuned model can better detect actual malicious attempts. This is vital for preventing actual SQL injection attacks that cause data loss or breaches. High recall ensures a lower false negative rate. In the context of our research, lower false negatives are fewer SQL injections that are accepted and ran by applications on their database as queries. The F1 score balances precision and recall. Again, the fine-tuned model demonstrates a notable increase from 84.16% to 99.72%. Essentially, fine-tuning our BERT model to generate better embeddings leads our logistic regression model to better identify attacks accurately and

minimizing missed detections.

6 Conclusion and Future Work

In conclusion, we enhanced malicious/benign classification of potential SQL injections based on more context-rich embedding representations generated by fine-tuning BERT. We observed significant enhanced detection performance across all metrics including accuracy, precision, recall, and F1 score. The advanced performance of the fine-tuned model significantly reduces both false positives and negatives. Despite these successes, we encountered computational challenges regarding the high demands of processing large-scale data with complex BERT models. Future enhancements could focus on several areas: exploring more efficient transformer architectures, implementing incremental learning, and developing hybrid models that combine rule-based and machine learning methods for initial data filtering. Additionally, cloud computing resources like AWS and employing parallel processing techniques could improve both the scalability and efficiency. These considerations for improvement are important when our models are deployed for practical cybersecurity tasks, where cost-efficiency is a major factor.

References

[1] Abu Syeed Sajid Ahmed. Sql injection dataset. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset/data, 2023. Accessed: 2025-03-20.

[2] Benjamin Appiah, Eugene Opoku-Mensah, and Zhiguang Qin. Sql injection attack detection using fingerprints and pattern matching technique. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 583–587, 2017.

[3] Geogiana Buja, Kamarularifin Bin Abd Jalil, Fakariah Bt. Hj Mohd Ali, and Teh Faradilla Abdul Rahman. Detection model for sql injection attack: An approach for preventing a web application from the sql injection attack. In *2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, pages 60–64, 2014.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Musaab Hasan, Zayed Balbahaith, and Mohammed Tarique. Detection of sql injection attacks: A machine learning approach. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–6, 2019.

[6] Konstantinos Kemalis and Theodores Tzouramanis. Sql-ids: a specification-based approach for sql-injection detection. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, page 2153–2158, New York, NY, USA, 2008. Association for Computing Machinery.

[7] Cristian Pinzón, Juan F. De Paz, Javier Bajo, Álvaro Herrero, and Emilio Corchado. Aiida-sql: An adaptive intelligent intrusion detector agent for detecting sql injection attacks. In *2010 10th International Conference on Hybrid Intelligent Systems*, pages 73–78, 2010.

[8] Peng Tang, Weidong Qiu, Zheng Huang, Huijuan Lian, and Guozhen Liu. Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528, 2020.

[9] Xin Xie, Chunhui Ren, Yusheng Fu, Jie Xu, and Jinhong Guo. Sql injection detection for web applications based on elastic-pooling cnn. *IEEE Access*, 7:151475–151481, 2019.