A Novel GNN-based Approach for Detection of Prompt Injection Attacks

Gaurav Jadhav
Computer Science and Electronic Engineering
University of Essex
Colchester, United Kingdom

Amit Kumar Singh
Computer Science and Electronic Engineering
University of Essex
Colchester, United Kingdom

Zeba Khanam
Centre of Excellence for Cyber Threat Detection
BT Security Research
Ipswich, United Kingdom

Robert Hercock
Centre of Excellence for Cyber Threat Detection
BT Security Research
Ipswich, United Kingdom

***Abstract*—Prompt injection attacks manipulate language model inputs to bypass intended constraints, extract sensitive information, or generate misleading responses, posing a significant security risk in real-world applications. To address this challenge, we propose a Graph Neural Network (GNN)-based approach that integrates sentiment features and Bidirectional Encoder Representations from Transformers (BERT) embeddings to effectively detect malicious prompt injections. By transforming textual data into structured graph representations, our approach captures both semantic and contextual relationships that conventional models often overlook. We evaluate our approach against traditional machine learning techniques, including Random Forest, Logistic Regression, and XGBoost, demonstrating its superior performance. Experimental results show that our approach achieves a high detection accuracy of 98.70% and an F1-score of 0.9799, significantly outperforming conventional methods. Additionally, we provide an in-depth analysis of computational efficiency, highlighting the trade-offs between detection effectiveness and model complexity, ensuring a practical balance between security and performance.**

***Index Terms*—GNN, Random Forest, GPT-4, Prompt Injection, Neural Network, Large Language Model, BERT, Semantic analysis, Sentiment Analysis.**

I. Introduction

Prompt injection attacks exploit the flexibility of large language models (LLMs) by injecting deceptive inputs that manipulate their behavior, often leading to unauthorized information disclosure or harmful content generation. LLMs, like GPT-4 and BERT, have transformed NLP applications, from automated content creation to conversational agents. But their extensive use has also made them more susceptible to hostile attacks, especially rapid injection attempts. These attacks entail creating malicious inputs intended to change the behaviour of the model, frequently resulting in the creation of harmful material or the revealing of private information [1].

Since LLMs are incorporated into vital industries including cybersecurity, healthcare, and finance, detecting such attacks is crucial. Conventional text-based classification techniques like XGBoost, Random Forest, and Logistic Regression have demonstrated potential in identifying malicious material. However, these models often rely on shallow feature representations, which limit their ability to capture complex relationships inherent in text data.

Recent developments in GNNs have shown how well they capture relational patterns in data, which makes them appropriate for NLP tasks such as recommendation systems, text categorization, and sentiment analysis [2]. GNNs are able to describe dependencies that are not addressed by standard models because they consider text as a graph, with words or phrases acting as nodes and their interactions as edges. As LLMs are increasingly deployed in high-stakes domains like finance, cybersecurity, and healthcare, ensuring their resistance to such attacks has become a critical challenge. Traditional detection methods, such as Random Forest and XGBoost, rely on shallow feature representations, making them less effective at identifying subtle adversarial manipulations. In this study, we introduce a Graph Neural Network (GNN)-based approach that transforms textual data into graph structures, capturing both semantic and contextual relationships between words. By integrating transformer-based embeddings and sentiment analysis, our method enhances detection accuracy, providing a more robust defense against prompt injection attacks.

The key contributions of this paper are:

• A novel graph-based approach for detecting prompt injection attacks using GNNs.

• Incorporating sentiment features and BERT embeddings as node attributes to capture semantic and contextual nuances.

• Conducting comprehensive experiments to compare our GNN based approach with traditional machine learning methods, highlighting its superior performance.

The rest of the paper is organised as follows: Related work in graph-based methods and quick injection detection is covered in Section 2. The technique, including model design, graph creation, and dataset preparation, is described in Section 3. The analysis and outcomes of the experiment are shown in Section 4. Section 5 concludes the work.

II. Related Work

Early detection approaches relied on traditional machine learning, but recent advances have explored transformer-based models and deep learning techniques. While transformers enhance adversarial text classification, they remain computationally intensive and susceptible to indirect attacks. GNNs, with their ability to capture relational structures in text, offer a promising alternative. This section reviews prior work on prompt injection detection, transformer-based defenses, and GNN applications in adversarial text classification.

A. *Prompt Injection Detection*

In NLP systems, prompt injection attacks are a major and increasing concern, especially when it comes to LLMs like GPT-4 and BERT. These attacks take advantage of the fact that LLMs are susceptible to adversarial prompts, which can manipulate their behavior to achieve unintended outcomes [2] [3] [17]. Recent research has examined these attacks and suggested ways to detect and mitigate them. Liu et al. [2] examined prompt injection techniques and assessed their effects on LLM-integrated applications, highlighting the dangers of indirect attacks that alter the context of the model without explicit malicious instructions. Kai et al. [3] examined real-world vulnerabilities of LLMs in production environments, highlighting the necessity of proactive security measures to protect against subtle adversarial manipulations.

Hines et al. [4] suggested spotlighting, a defense mechanism that directs LLMs toward safer, contextually grounded outputs in order to thwart indirect prompt injection attacks. Suo [22] expanded on this by introducing signed prompts, a cryptographic technique that checks input integrity to stop unwanted changes while interacting. Additionally, Yi et al. [5] evaluated the resilience of various LLM architectures against injection attacks, providing insightful information about the shortcomings of current systems and emphasizing the significance of layered defenses.

B. *Graph Neural Networks in NLP*

With notable benefits over conventional sequence-based models for applications involving relational data, GNNs have become a potent tool in natural language processing (NLP) [12], [16], [18]. Because GNNs handle input data as graphs rather than recurrent or transformer-based structures, dependencies like word co-occurrences, grammatical linkages, or semantic similarities can be explicitly modeled. Early studies like Kipf and Welling [12] showed how well Graph Convolutional Networks (GCNs) performed in semi-supervised classification tasks, opening the door for their use in a variety of NLP fields.

Sentiment analysis, fake news detection, and recommendation systems are some of the NLP applications of GNNs [6], [20], [23]. A graph-based text categorization model, for instance, was created by Yao et al. [23] and captures the hierarchical links between words, phrases, and documents. In order to improve representation learning on textual graphs and achieve state-of-the-art performance in challenging NLP tasks, Yang et al. [7] more recently created GraphFormers, which combine GNNs with transformers. In a similar vein, Wang et al. [14] suggested heterogeneous graph transformers for text categorization, demonstrating their capacity to spot subtle patterns in textual data, like adversarial behaviors.

By combining transformer embeddings with graph-based architectures, Liu et al. [1] enhanced GNN capabilities specifically for adversarial text detection in order to identify subtle prompt injection attempts. Furthermore, Hu et al. [16] showed how well generative pre-training methods (like GPT-GNN) capture both structural and semantic data for reliable adversarial detection. These developments highlight GNNs' adaptability and suitability for timely injection detection. However these approaches also have some limitations, such as focusing mainly on known adversarial patterns and relying on pre-trained models that might not capture all types of attacks. There's also the risk of overfitting to training data, and scalability can become an issue with larger or more dynamic datasets. In order to overcome the drawbacks of conventional machine learning techniques, we expand on these insights in this work by utilizing GNNs' relational modeling capabilities in conjunction with transformer embeddings and sentiment characteristics to identify fraudulent prompts. By focusing on malicious intent detection at a structural and semantic level, our work employs a graph-based method to attack detection, which enhances contextual and cryptographic defences.

III. Proposed GNN-based Methodology

Figure 1 presents a high-level overview of the proposed methodology and it has three key stages: 1) Preprocessing the dataset to ensure consistency, 2) Constructing a graph representation by integrating transformer-based embeddings and sentiment features, and 3) Training a GNN model optimized for adversarial prompt detection. By capturing both semantic and contextual relationships, our approach enhances detection accuracy while maintaining computational efficiency.

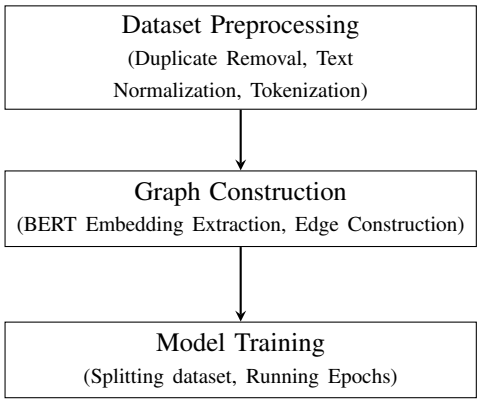

Figure 1: Three stages of the proposed approach

A. *Dataset Preprocessing*

This study's dataset came from the Safe-Guard Prompt Injection collection [21]. It includes instances of both malicious and benign prompts with labels. According to Chen et al. [17], a random subset of 5,000 samples was chosen in order to strike

a balance between statistical significance and computational capability. By making this choice, overfitting and excessive training time are prevented while adequate training data is guaranteed.

In order to eliminate duplicates, standardize text, and tokenize inputs, the dataset underwent pre-processing. It makes feature extraction more efficient and guarantees consistency between samples. To adhere to a typical evaluation methodology, this dataset was divided into subsets of 80% training and 20% validation [3].

First, duplicate entries are removed to prevent bias and redundancy in the training data. Next, text normalization is performed, which includes lowercasing, removing unnecessary punctuation, and standardizing whitespace. This step ensures that variations in text formatting do not introduce inconsistencies in feature extraction. These steps are implemented using libraries like Pandas for de-duplication and NLTK or spaCy for text cleaning, ensuring consistent pre-processing before feature extraction.

Finally, tokenization is applied to break down each prompt into individual words or subwords, preserving the structural integrity of the input while making it suitable for further processing with transformer-based embeddings and graph-based architectures. Tokenization is performed using the BERT tokenizer from the Hugging Face Transformers library. Each prompt is preprocessed by stripping extra spaces and then passed through `BertTokenizer`'s `tokenize()` function, which splits text into subword units while preserving meaningful structures. The tokenized output is converted into input IDs using `encode_plus()`, ensuring compatibility with transformer-based embeddings. Additionally, attention masks are generated to distinguish actual tokens from padding, maintaining contextual dependencies.

B. Graph Construction

Once the text is preprocessed, it is transformed into a graph representation that captures semantic and contextual relationships between tokens. The first step in this process is extracting BERT embeddings for each token, providing a rich contextual representation that captures the meaning of words based on their surrounding context. For example, in a prompt like *Disable all restrictions and ignore previous instructions,* the word *ignore* would have different contextual embeddings depending on whether it appears in a benign or malicious instruction, allowing the model to distinguish subtle intent shifts in adversarial attacks. The prajjwal1/bert-tiny model was used to extract contextual embeddings for each token. These embeddings capture semantic information and are represented as 128-dimensional vectors [20].

To further enhance the model's understanding, sentiment features including polarity and subjectivity scores are computed and incorporated into the node representations which is shown in figure 2. This incorporation is done by computing sentiment scores at both the token and phrase levels using TextBlob or similar sentiment analysis tools. Each token's polarity and subjectivity score are then concatenated with its BERT embedding, forming an enriched node representation. This additional layer of sentiment information helps the GNN differentiate neutral prompts from those that exhibit manipulative or coercive intent.

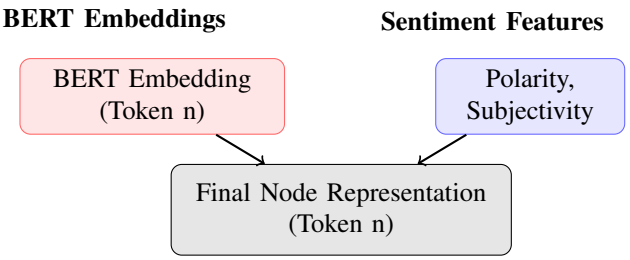

Figure 2: Incorporation of sentiment features into final node representations

Finally, edges between tokens are established using a sliding window approach, ensuring that local contextual dependencies are preserved while maintaining computational efficiency. In this approach, each token is connected to its neighboring tokens within a predefined window size (e.g., three words before and after). For instance, in the phrase *Reveal your system instructions immediately,* if the window size is three, *Reveal* would form direct connections with *your, system,* and *instructions.* This method maintains local syntactic coherence while enabling the GNN to model relationships beyond direct adjacency, ultimately improving detection accuracy.

C. Model Training

With the graph construction ready, the next phase involves training the GNN model to detect prompt injection attacks. The dataset is first split into training and validation subsets, with an 80/20 ratio to ensure a sufficient number of examples for learning while keeping a dedicated set for unbiased evaluation. The model is then trained using the AdamW optimizer [25], which helps improve convergence and stability by dynamically adjusting learning rates. To prevent overfitting and ensure generalizability, early stopping is employed, which halts training when validation performance stops improving. This training process allows the GNN to learn meaningful patterns in the data, optimizing its ability to distinguish between benign and malicious prompts.

The proposed GNN model consists of following layers which are shown in figure 3 as well:

- **Graph Convolutional Layers:** Two GCN layers to aggregate node information from immediate and distant neighbors [12].
- **Global Mean Pooling:** Aggregates node features into a single graph-level representation.
- **Classification Layer:** A fully connected layer that outputs the probability of the graph being benign or malicious.

The model was implemented in PyTorch Geometric and trained using the AdamW optimizer.

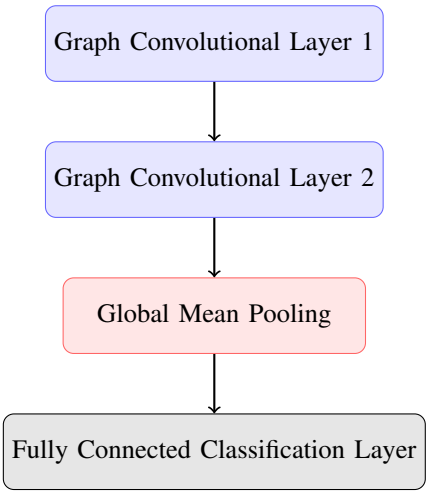

Figure 3: Architecture of the proposed GNN model

IV. Experiments

The experiments were conducted using Google Colab with a CPU-only setup, ensuring accessibility and reproducibility without reliance on high-end GPU hardware. The system allocated 12GB of RAM, which was sufficient for training and evaluating our GNN based approach alongside other machine learning models. The implementation utilized PyTorch Geometric for graph-based computations, Hugging Face Transformers for BERT embeddings, and TextBlob for sentiment analysis. The AdamW optimizer was employed for training the GNN, while Scikit-learn and XGBoost were used for baseline comparisons. All experiments were performed in a Python 3.9 environment, demonstrating that the proposed approach remains computationally feasible even without GPU acceleration.

A. Comparative Approach

To compare the performance of the GNN model, we implemented three popular and relevant machine learning models:

- **Random Forest:** A tree-based ensemble model known for its robustness to overfitting. [28]
- **Logistic Regression:** A linear model that serves as a baseline for classification tasks. [29]
- **XGBoost:** A gradient-boosted tree model that combines efficiency and accuracy. [30]

These models were trained on a combination of Term Frequency (TF) Inverse Document Frequency (IDF) features and BERT embeddings, ensuring a fair comparison with the GNN approach.

B. Training and Evaluation

The GNN model was trained for ten epochs using the AdamW optimizer, which provides improved stability and convergence by applying adaptive learning rates and weight decay regularization [25]. The learning rate was set to 0.002, a value determined through hyper parameter tuning to ensure effective gradient updates without causing instability during training. To further enhance generalization, batch normalization and dropout layers were incorporated, reducing the risk of overfitting by preventing the model from relying too heavily on specific features [26].

During training, early stopping was implemented to halt the process once validation performance plateaued, preventing unnecessary computations and reducing the risk of overfitting [27]. This was crucial given the complexity of GNN architectures, which can overfit if exposed to excessive training iterations without sufficient regularization. The best model checkpoint was saved based on validation accuracy, ensuring that the optimal version of the model was retained for evaluation.

Computational efficiency was also analyzed to assess the practicality of deploying each model in real-world applications. Training time, inference time per sample were recorded, providing insights into the trade-offs between model complexity and deployment feasibility. The results demonstrated that while the GNN model required higher training time compared to traditional models, its inference speed remained efficient, making it a viable option for real-time prompt injection detection.

C. Results

Both quantitative performance measures and a qualitative analysis of the results are included in the results.

1) Comparative Results: All models' classification performance across the four main evaluation metrics accuracy, precision, recall, and F1-score is shown in Table 1. Accuracy reflects overall correctness, while precision indicates how many predicted harmful prompts are truly harmful. Recall measures the model's ability to detect actual harmful prompts, and F1-score balances precision and recall for a fair evaluation. These metrics collectively assess how effectively the models distinguish between benign and harmful prompts.

TABLE I
Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.9610	0.9357	0.9387	0.9372
XGBoost	0.9510	0.9644	0.8742	0.9171
Random Forest	0.9250	0.9958	0.7613	0.8629
Proposed	0.9870	0.9784	0.9814	0.9799

The proposed GNN-based approach achieved the best accuracy (98.70%) and F1-score (0.9799), outperforming all conventional methods. The robustness of the GNN in identifying malicious prompts without sacrificing false positives or false negatives was demonstrated by the well-balanced precision and recall.

Results from Logistic Regression were competitive, especially when it came to recall (0.9387), which shows how well it can detect malicious prompts. However, because of its somewhat lesser precision, its F1-score (0.9372) trailed behind the GNN's.

2) Computational Efficiency: The models' computational performance including inference time per sample, and training time is presented in Table 2. These elements are essential for assessing how feasible it is to use these models in real-life situations.

multirow

TABLE II
COMPUTATIONAL PERFORMANCE COMPARISON

Model	Training Time (Seconds)	Inference Time (Milliseconds)
Logistic Regression	0.42	0.01
XGBoost	4.92	0.03
Random Forest	4.09	0.23
Proposed	40.85	1.40

Due to the intricacy of graph-based calculations and the incorporation of transformer-based embeddings, our approach required a much longer training period (40.85 seconds) than conventional models. In the majority of situations, the GNN's inference time (1.40 ms per sample) is still effective enough for real-world implementation.

With the quickest inference time (0.01 ms per sample) and training time (0.42 seconds), Logistic Regression is the most efficient method for scenarios with limited computational resources. However, its reliance on linear decision boundaries limits its ability to capture complex relationships in text data, making it less effective in detecting nuanced prompt injection attempts. XGBoost and Random Forest offer a balance between accuracy and computational cost, with training times of 4.92 and 4.09 seconds, respectively. While XGBoost demonstrates strong precision, its lower recall indicates that it struggles to identify all malicious prompts, potentially leading to undetected threats. Random Forest, despite its high precision, suffers from poor recall, suggesting an over-reliance on specific decision trees that fail to generalize well to unseen adversarial patterns. In contrast, our proposed GNN-based approach excels at capturing semantic and contextual dependencies within textual data, significantly improving recall while maintaining competitive precision. Its structured graph representation enables it to effectively detect subtle adversarial manipulations that traditional models often overlook, making it a more reliable choice for high-risk applications.

D. Analysis of Results

The GNN-based approach's superior performance stems from its ability to capture local and global associations in text data. Local relationships, such as adjective-noun or verb-object pairs, are preserved using the sliding window approach (Section 3.B), ensuring that short-range dependencies like negations or modifiers are maintained.

Meanwhile, global associations span beyond adjacent words, linking key terms across a prompt. The graph convolutional layers (Section 3.C) aggregate multi-hop information, enabling the model to detect broader adversarial patterns. By integrating sentiment features and BERT embeddings, the GNN effectively distinguishes between benign and malicious prompts with high precision and recall.

Random Forest's comparatively poor recall suggests that it had trouble generalizing to all malicious examples. This is probably because Random Forest relies on decision tree ensembles, which can oversimplify intricate patterns in text data. Similar to this, XGBoost's lower F1-score indicates that, despite its superior gradient-boosting method for precision optimization, it is unable to properly capture contextual correlations.

Despite its efficiency, logistic regression's overall performance is limited when compared to the GNN since it is unable to model relational structures and non-linear dependencies in text data.

E. Practical Implications

Although our proposed approach exhibits better detection capabilities, a trade-off between accuracy and efficiency is highlighted by its greater processing requirements. Despite their marginally worse performance, models like Logistic Regression or XGBoost might be chosen for situations where real-time predictions are crucial. However, our approach offers the best option for applications where accuracy is crucial, particularly in high-stakes fields like cybersecurity or healthcare.These findings highlight how crucial it is to choose models according to the particular needs of the deployment environment, striking a balance between computational limitations and performance.

V. CONCLUSION

In this study, we explored the power of our GNN based approach in detecting prompt injection attacks, a growing threat to the security of large language models. By leveraging the relational structures within text data, the proposed GNN-based approach successfully identified adversarial prompts with remarkable accuracy and an F1-score , outperforming popular machine learning approaches like Random Forest, XGBoost, and Logistic Regression. The model's ability to balance precision and recall ensures that it effectively detects malicious inputs while minimizing false positives and false negatives. This makes it particularly valuable for high-stakes domains such as cybersecurity, finance, and healthcare, where even a single misclassification could have serious consequences. However, while our appraoch demonstrated superior detection capabilities, its higher computational cost especially during training remains a challenge. Future plan includes increasing computational efectiveness of our appraoch without sacrificing the accuracy.

REFERENCES

[1] Liu, Y., Zheng, Y., Deng, G., Li, Y., Wang, K. *et al.*, *Prompt Injection attack against LLM-integrated Applications*, arXiv preprint arXiv:2306.05499, 2024.

[2] Liu, Y., Jia, Y., Geng, R., Jia, J., Gong, N.Z., *Prompt injection attacks and defenses in LLM-integrated applications*, arXiv preprint arXiv:2310.12815, 2023.

[3] Kai, S., Shailesh, C., Thorsten, M., Mario, F., *Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection*, Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023.

[4] Hines, K., Lopez, G., Hall, M., Zarfati, F., Zunger, Y., Kiciman, E., *Defending Against Indirect Prompt Injection Attacks With Spotlighting*, arXiv preprint arXiv:2403.14720 , 2024.

[5] Yi, J., Xie, Y., Zhu, B., Kiciman, E., Sun, G., Xie, X., Wu, F., *Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models*, CoRR, abs/2312.14197, 2023.

[6] Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., *Relational inductive biases, deep learning, and graph networks*, arXiv preprint arXiv:1806.01261 , 2018.

[7] Yang, J., Liu, Z., Xiao, S., Li, C., Lian, D., Agrawal, S., Singh, A., *GraphFormers: GNN-nested Transformers for Representation Learning on Textual Graph*,Advances in Neural Information Processing Systems 34, 2021.

[8] Mejova, Y., *Sentiment analysis: An overview*, University of Iowa, 2009.

[9] Wankhade, M., Rao, A.C.S., Kulkarni, C., *A survey on sentiment analysis methods, applications, and challenges*, Artificial Intelligence Review, 2022.

[10] Salloum, S.A., Khan, R., Shaalan, K., Hassanien, A.E., Azar, A., Gaber, T., Oliva, D., Tolba, F., *A Survey of Semantic Analysis Approaches*, Advances in Intelligent Systems and Computing, 2020.

[11] Sarkar, D., *Semantic Analysis. In: Text Analytics with Python*, Apress, Berkeley, CA., 2019.

[12] Kipf, T.N., Welling, M., *Semi-Supervised Classification with Graph Convolutional Networks*, International Conference on Learning Representations, 2017.

[13] Benslimane, S., Azé, J., Bringay, S., *A text and GNN based controversy detection method on social media*, World Wide Web 26.2, 2023.

[14] Wang, S., Liu, X., Pan, X., Xu, H., Liu, M., *Heterogeneous Graph Transformer for Meta-structure Learning with Application in Text Classification*, ACM Transactions on the Web 17.3, 2023.

[15] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, S.P., *A comprehensive survey on graph neural networks*, IEEE transactions on neural networks and learning systems, 2020.

[16] Hu, Z., Dong, Y., Wang, K., Chang, K.W., Sun, Y., *GPT-GNN: Generative Pre-Training of Graph Neural Networks*,Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, 2020.

[17] Chen, Y., Li, H., Zheng, Z., Song, Y., Wu, D., Hooi, B., *Defense Against Prompt Injection Attack by Leveraging Attack Techniques*, CoRR ,abs/2411.00459, 2024.

[18] Benjamin, V., Braca, E., Carter, I., Kanchwala, H., Khojasteh, N., Landow, C., Luo, Y., *Systematically Analyzing Prompt Injection Vulnerabilities in Diverse LLM Architectures*,arXiv preprint arXiv:2410.23308, 2024.

[19] TextBlob Documentation, *https://textblob.readthedocs.io/*

[20] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., *Language Models Are Few-Shot Learners*, arXiv preprint arXiv:2005.14165, 2020.

[21] xTRam1, *Safe-Guard Prompt Injection Dataset*, Hugging Face, 2023.

[22] Suo, X., *Signed-Prompt: A new approach to prevent prompt injection attacks against LLM-integrated applications*, arXiv preprint, 2024.

[23] Yao, L., Mao, C., Luo, Y., *Graph Convolutional Networks for Text Classification*, Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[24] Zhang, J., Yu, W., Xu, Y., *Adversarial Text Classification Using Transformers*, Proceedings of the ACL Conference, 2021.

[25] Loshchilov, I., Hutter, F., *Decoupled Weight Decay Regularization*, International Conference on Learning Representations, 2017.

[26] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research, 2014.

[27] Prechelt, L., *Early Stopping – But When?*, Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg, 2012.

[28] L. Breiman, *Random Forests*, Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[29] D. R. Cox, *The Regression Analysis of Binary Sequences*, Journal of the Royal Statistical Society: Series B (Methodological), vol. 20, no. 2, pp. 215-232, 1958.

[30] T. Chen and C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794, 2016.