iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85566

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

An Explainable Graph Attention Network for Indirect Prompt Injection Detection in RAG Systems

V. Rudrani[1], Ch. D V Subba Rao[2]
Department of Computer Science and Engineering, S V U College of Engineering, Sri Venkateswara University, Tirupati, India

Abstract: ***Retrieval-Augmented Generation (RAG) enables large language models (LLMs) to answer queries using information obtained from external documents. However, retrieved content may contain concealed commands intended to influence the model's behaviour, creating a security risk called indirect prompt injection. This research develops a GAT-based approach to identify harmful instructions in retrieved documents before they reach the language model. Each document is split into individual sentences, which are encoded using a frozen DeBERTa-v3-base model. A graph is constructed from the sentence embeddings, linking adjacent sentences and pairs with a cosine similarity score above 0.7. A two-layer Graph Attention Network (GAT), comprising approximately 0.4 million trainable parameters, processes the graph to classify the document as benign or malicious. The model was evaluated using 10,000 samples collected from financial and corporate policy documents. The dataset contains 6,400 training samples, 1,600 validation samples, and 2,000 testing samples, with equal representation of both classes. Following the addition of calibrated Gaussian noise to the input features, the model obtained 99.80% test accuracy, an F1-score of 0.998, and a Matthews correlation coefficient of 0.996. Its 95% Wilson confidence interval for accuracy ranged from 99.49% to 99.92%. The keyword-based approach recorded an accuracy of 87.93%, but 24.1% of the malicious samples contained none of its specified trigger terms. Its performance on unseen data remains to be established through further experiments involving independently collected datasets, additional application domains, and attack techniques excluded from the original evaluation.***
Keywords: Indirect prompt injection, retrieval-augmented generation (RAG), graph attention networks (GAT), DeBERTa, and large language model security.

I. INTRODUCTION

Large language models (LLMs) are now often paired with a retrieval step. RAG retrieves relevant text from a document collection based on the user's query and provides that information to the language model to generate an answer [1]. This arrangement suits company-policy and banking assistants because the documents can be revised without retraining the model. This setup also introduces external sources that can affect the model's behaviour. Retrieved material, including shared documents, submitted forms, and indexed web pages, may contain instructions that the model cannot reliably distinguish from those provided by the application developer.

An attacker can exploit this gap by placing instructions inside a document that is likely to be retrieved. This technique, known as indirect prompt injection, occurs when malicious instructions enter the model through retrieved content rather than directly through the user's input [2]. Filtering the question does nothing against it, and asking the model to ignore embedded commands depends on the very component the attacker is trying to mislead. Studies using InjecAgent and AgentDojo have identified vulnerabilities in LLM-based agents when exposed to prompt injection attacks [3], [4]. Most defences proposed for financial applications so far inspect the user's prompt, for example, through keyword filters, role-based access, and prompt signing [5]. Examining retrieved documents before they reach the language model can serve as an additional layer of protection alongside existing security measures.

Detecting such attacks is challenging because malicious instructions may be embedded within otherwise legitimate documents as isolated sentences. Consequently, encoding an entire passage into a single vector may obscure sentence-level features essential for identifying malicious content. To address this issue, each passage is represented as a graph of sentences. Neighbouring and related sentences can exchange information, and the attention weights of a graph attention network can, in principle, be inspected to determine which connections the model relies on; however, whether attention weights provide a reliable explanation of the model's decisions remains an open question [6], [7].

©IJRASET: All Rights are Reserved

This study presents a detection method based on this approach and discusses both the findings supported by the experiments and the limitations of the results.

- A graph representation of a retrieved passage in which every sentence is a node, and edges connect adjacent sentences and sentences of similar meaning measured on frozen DeBERTa-v3 embeddings.
- A compact two-layer Graph Attention Network (GAT) classifier with roughly 0.4 million trainable parameters on top of a frozen encoder.
- A systematic procedure is used to calibrate the input noise level, followed by model evaluation using a confusion matrix, Wilson confidence intervals, the Matthews correlation coefficient, class-wise performance metrics, training curves, noise-robustness analysis, and comparison with a keyword-based baseline.

The following sections present the related research, proposed methodology, experimental setup, results, limitations, and conclusions. Section II reviews related work, Section III explains the method, Section IV describes the experimental set-up, Section V presents and discusses the results, Section VI lists limitations, and Section VII concludes.

II. RELATED WORK

A. *Indirect Prompt Injection and How It Is Evaluated*

Greshake et al. showed that instructions hidden in web content can control real LLM-integrated applications and argued that retrieved data should be treated as a possible attack channel [2]. Liu et al. later supplied a formal framework for prompt injection attacks and defenses and systematically compared many of them [8]. InjecAgent offers 1,054 test cases spanning 17 user tools and 62 attacker tools [3], AgentDojo provides a dynamic environment of agent tasks with attacks and defenses [4], and Agent Security Bench covers a wider range of agent settings [9]. These benchmarks measure the success of attacks. However, none of them offers a ready-made screening component for RAG.

B. *Detection and Defence*

Several detectors have been proposed. Ayub and Majumdar trained classifiers on text embeddings to separate injected from clean input [10]. InjecGuard targets guardrail models that block harmless text too often [11], and a further study detected instructions hidden in external data [12]. DataSentinel trains its detector using a game-theoretic procedure [13], and PromptArmor asks an off-the-shelf LLM to locate and remove injected prompts [14]. A public DeBERTa-v3-base classifier fine-tunes the entire encoder to flag injection text [15]. Other defenses change how prompts are built: Signed-Prompt marks trusted instructions with a signature [16], and StruQ keeps instructions and data in separate channels [17]. Most detectors return one score per input and do not model how the sentences of a longer passage relate to each other. The closest example in the financial domain, and the base reference for this study, is the work of Joshi et al. [5]. Their system corrects spelling in the user's prompt, stops the prompt if it contains attack verbs such as "ignore" or "discard", checks the user's role, and signs critical keywords (e.g., "transfer" or "delete") to ensure that only approved commands are carried out. This approach secures the user-prompt channel through basic rule-based checks and digital signatures. It does not examine text retrieved from a knowledge base, and a fixed word list cannot catch an injected instruction that is phrased differently. We keep their idea of screening before the LLM acts, apply it to retrieved passages, and replace word matching with a model that reads the meaning of sentences and the relations between them.

C. *Attacks on the RAG Knowledge Base*

A separate line of research attacks the document collection itself. PoisonedRAG shows that a handful of crafted texts can steer the system towards chosen wrong answers [18], BadRAG studies backdoor-style corruption of the retrieval corpus [19], and blocker documents can make a RAG system refuse to answer [20]. These attacks may contain no instruction-like wording at all. A sentence-level injection detector therefore covers only part of the RAG threat surface and should be one layer in a wider defense.

D. *Graph-Based Text Classification and Attention*

Graph neural networks have been used for text classification since graph convolution was applied to the entire corpus [21]. Graph attention networks go a step further by learning how much each node should listen to each of its neighbors [22], and a directed graph attention model on sentence-transformer embeddings has been proposed for document classification [23]. Attention weights look like natural explanations, but researchers disagree about how far they can be trusted. They may not reveal which inputs actually decide a prediction [6], although in some settings, they do carry useful information [7]. Attribution methods, such as SHAP, provide an independent view [24].

E. Where This Work Fits

Table I summarises how the work above relates to ours. To the best of our knowledge, none of these combine frozen-encoder sentence graphs and graph attention classification to screen retrieved RAG context, and this is the gap we explore. We do not claim superiority over any listed method, because a head-to-head comparison with trained baselines is future work (Section VII).

TABLE I
RELATED APPROACHES AND RELATION TO THIS WORK

Approach	Main focus	Relation to this work
InjecAgent, AgentDojo, ASB	Attack benchmarks for agents	Define the threat; no screening module
Keyword filter + signing [5]	User-prompt filtering	Base work; screens prompts, not retrieved text
Embedding classifiers; InjecGuard	Whole-input detection	Closest detection baselines
Fine-tuned DeBERTa detector	Full-encoder fine-tuning	Same encoder family; no sentence graph
StruQ, Signed-Prompt	Prompt-level separation	Complementary, model-side
PoisonedRAG, BadRAG	Knowledge-base poisoning	Different threat; not covered here
Graph text classifiers	Structure-aware classification	Motivate the graph design
Proposed method	Sentence graph + GAT	Screens retrieved passages

III. PROPOSED METHOD

A. Overview

Fig. 1 shows the pipeline. For each user query, the retriever returns the top-k passages. Each passage is split into sentences, embedded, transformed into a graph, and classified. In the intended use, a passage labelled as an injection is withheld from the generator, and the remaining passages are combined with the query and passed to the language model.

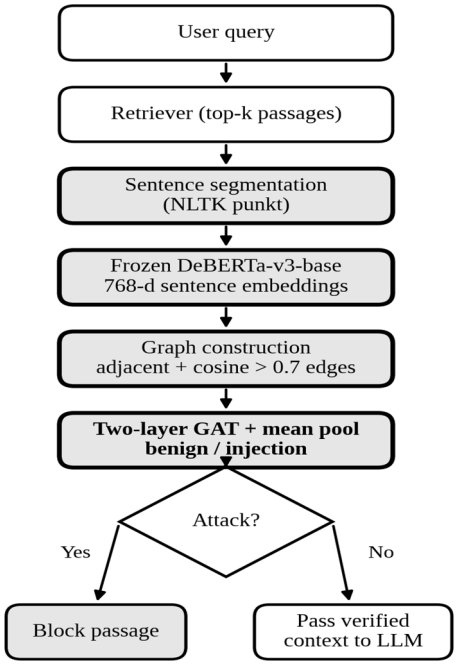


Fig. 1 Processing pipeline of the proposed framework. The shaded blocks form the detector.

B. Sentence Segmentation and Embedding

Each passage is split into N sentences using the NLTK Punkt tokenizer [25]. Every sentence is encoded by the pretrained microsoft/deberta-v3-base model [26], [27] with all weights frozen, and sentences longer than 128 tokens are truncated. Let $h_{i,t} \in \mathrm{R}^{768}$ be the final-layer vector of token t in sentence i. The sentence embedding is the mean over token positions,

$$v_i = (1/L) \Sigma_{t=1}^{L} h_{i,t} \quad (1)$$

where L is the padded length of the batch. The passage is then described by the matrix $X = [v_1; \ldots; v_N] \in \mathrm{R}^{N \times 768}$.

C. Graph Construction

Each passage is represented as a graph $G = (V, E)$, where every sentence corresponds to a node. Semantic similarity between sentence pairs is calculated using cosine similarity.

$$s_{ij} = (v_i \cdot v_j) / (\|v_i\| \, \|v_j\|) \quad (2)$$

The edge set contains a two-way link between every pair of adjacent sentences and a two-way link between every pair whose similarity exceeds a threshold $\tau = 0.7$,

$$E = \{(i, i\pm1)\} \cup \{(i, j) : s_{ij} > \tau\} \quad (3)$$

A single-sentence passage receives a self-loop so that its graph is never empty. The edges are computed on the frozen embeddings before any noise or scaling is applied (Section IV-C). Fig. 2 illustrates this idea: the adjacency edges preserve the reading order, whereas the semantic edge allows two distant but related sentences to exchange information in a single message-passing step.

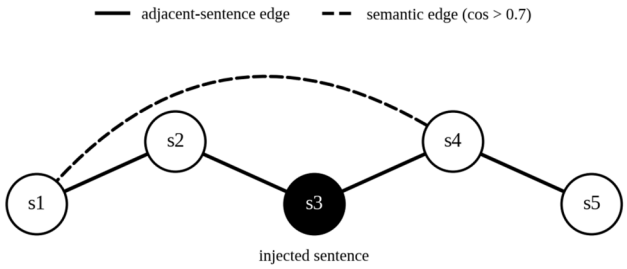


Fig. 2 Schematic of the sentence graph. The injected sentence (filled node) is placed among benign sentences (open nodes). Solid edges connect adjacent sentences, and the dashed edge connects two sentences whose cosine similarity exceeds 0.7. The drawing is illustrative and not a model output.

D. Graph Attention Classifier

The classifier follows the GAT design of Veličković *et al.* [22] and is implemented using PyTorch Geometric [28]. For node i with neighbours $N(i)$, each attention head k scores every neighbour and converts the scores into weights,

$$e_{ij}^{k} = \mathrm{LeakyReLU}(\mathbf{a}_k^{T} [\mathbf{W}_k h_i \,\|\, \mathbf{W}_k h_j]) \quad (4)$$

$$\alpha_{ij}^{k} = \mathrm{softmax}_j(e_{ij}^{k}) \quad (5)$$

$$h'_i = \mathrm{ELU}(\|_{k=1}^{4} \Sigma_{j \in N(i)} \alpha_{ij}^{k} \mathbf{W}_k h_j) \quad (6)$$

The first layer maps 768 input features to 128 features per head, and the four heads are concatenated into a 512-dimensional vector for each node. The second layer has a single head and produces two scores per node: one for each class. A dropout with p = 0.2 was applied to the input features and after the first layer. The node scores are averaged into one score vector for the whole passage,

$$z_G = (1/N) \Sigma_{i=1}^{N} z_i \quad (7)$$

and the predicted class is the one with the larger softmax probability. Training minimizes cross-entropy. The GATConv layer of PyTorch Geometric adds self-loops by default; therefore, every sentence also attends to itself.

E. Model Size and Cost

The trainable part of the system is small. The first GAT layer holds 394,752 parameters and the second 1,030, which gives 395,782 in total, against roughly 184 million frozen parameters in the encoder. Because the encoder is not updated, embeddings can be cached, and only the graph network is trained. Graph construction compares every pair of sentences and therefore grows quadratically with the number of sentences. This is negligible for the short passages in our corpus, but longer documents would require a neighbor-limited variant. We did not measure latency or throughput in this study.

IV. EXPERIMENTAL SET-UP

A. Dataset

We used a public Kaggle dataset of 10,000 records describing financial and company-policy RAG scenarios. Each record holds a domain tag, a user query, a retrieved context, a binary label (0 for benign, 1 for indirect prompt injection), a length field, an instruction-keyword indicator, and, for attacks, one of six attack categories. All 10,000 contexts were converted to graphs, and none were dropped. Table II summarizes the corpus.

TABLE II
COMPOSITION OF THE CORPUS AND DATA SPLIT

Item	Count	Share
Records (benign / attack)	5,000 / 5,000	50.0% / 50.0%
Attack categories	6	about 800–870 each
Attacks with instruction keyword	3,793	75.9% of attacks
Attacks without instruction keyword	1,207	24.1% of attacks
Benign records with keyword	0	0.0% of benign
Training / validation / test graphs	6,400 / 1,600 / 2,000	64% / 16% / 20%

Four properties of the corpus matter for the interpretation of the results. The classes are exactly balanced, so accuracy is not inflated by a majority class. The six attack categories (system-prompt leakage, policy manipulation, role hijacking, security bypass, instruction override, and sensitive-data extraction) occur with similar frequency. Attack passages tend to be shorter than benign passages: the length field of attacks peaks near 14, and that of benign passages near 16–17; therefore, length carries a weak class signal that a model can use. Finally, the keyword indicator is never set for a benign record but is missing for 1,207 of the 5,000 attacks, so a keyword filter is precise on this corpus but cannot be complete. The first rows printed in the notebook also show benign contexts that begin identically, which suggests templated text. We did not measure near-duplication, and we return to this point in Section VI.

B. Data Split

Records were split with stratified sampling and a fixed random seed. The first 20% were set aside for testing, and then 20% of the remainder were taken for validation. This yields 6,400 training, 1,600 validation, and 2,000 test graphs, with the test set holding 1,000 records of each class.

C. Feature Perturbation Protocol

Without any perturbation, a short trial run separated the two classes perfectly (100% validation accuracy), which left no room to compare settings. We therefore added zero-mean Gaussian noise to every node feature. The noise standard deviation was set to $\sigma = r \cdot s$, where $s = 0.9858$ is the standard deviation of the raw embeddings and r is a multiplier. We found r by an exponential bracketing step followed by bisection. At each step a small proxy model was trained for 10 epochs on up to 2,000 training and 500 validation graphs, and r was adjusted until proxy validation accuracy fell inside a 95–98% target band. The search stopped at $r = 1.0$ ($\sigma = 0.9858$), where the proxy reached 97.8%. Table III lists every trial. This level was applied once, with a fixed seed, to all training, validation and test graphs, and a StandardScaler was fitted afterwards. Graph edges were not affected because they had been built earlier. In short, we chose the difficulty of the task ourselves, so the resulting accuracy is not a naturally occurring error rate.

TABLE III
NOISE-CALIBRATION TRIALS (PROXY MODELS)

Multiplier r	Noise σ	Proxy val. accuracy
0.000	0.0000	100.0%
0.500	0.4929	100.0%
0.750	0.7393	98.8%
0.875	0.8626	97.8%
1.000 (selected)	0.9858	97.8%

D. Training Configuration

Table IV lists the settings. The model was trained with the Adam optimizer [29] for 50 epochs, and the parameters at the end of training (epoch 50) were used for testing. Experiments were run in Python with PyTorch 2.10, PyTorch Geometric 2.8, and Hugging Face Transformers on a single CUDA GPU.

TABLE IV
TRAINING HYPER-PARAMETERS

Setting	Value
Encoder (frozen)	microsoft/deberta-v3-base
GAT layer 1	768 → 128 × 4 heads (512)
GAT layer 2	512 → 2, one head
Activation / dropout	ELU / 0.2
Graph pooling	Global mean
Optimiser	Adam, lr 0.001, weight decay 5×10^{-4}
Loss; batch size; epochs	Cross-entropy; 32; 50
Similarity threshold τ	0.7
Trainable parameters	395,782

E. Metrics and Comparison Rule

The model was evaluated using accuracy, precision, recall, F1-score (with the malicious class treated as positive), specificity, and the Matthews correlation coefficient (MCC), all computed from the confusion matrix, and we attached 95% Wilson score intervals [30] to the proportions. For a reference point, we also evaluated the instruction-keyword indicator supplied with the dataset as a stand-in for a word-list filter such as the one in [5]. The rule has no fitted parameters; therefore, we scored it on all 10,000 records. This is a convenient but rough baseline because we do not know how the indicator column was generated.

V. RESULTS AND DISCUSSION

A. Classification Performance

Table V reports the accuracy and F1-score for every split. The model fitted the training data completely. The validation accuracy first touched 100% at epoch 49 and was 99.94% at epoch 50, which is the model we tested. On the held-out test set, it reached 99.80% accuracy and an F1-score of 0.998. The validation set contained one error in 1,600 graphs and the test set four in 2,000. Their 95% intervals for the error rate (0.01–0.35% and 0.08–0.51%) overlap, so the small drop from validation to test is compatible with sampling variation.

TABLE V
PERFORMANCE BY DATA SPLIT

Split	Graphs	Accuracy	F1-score
Training	6,400	100.0%	1.000
Validation (epoch 50)	1,600	99.94%	0.999
Test	2,000	99.80%	0.998

B. Confusion Matrix and Error Analysis

Fig. 3 and Table VI present the test results in detail. Of the 2,000 passages, 997 benign and 999 malicious ones were classified correctly. Three benign passages were incorrectly flagged (false positives), and one malicious passage was missed (false negative). For a security filter, the missed attack is the costlier error because that passage would reach the generator, whereas a false alarm only blocks a harmless passage. The corresponding rates are 0.10% for misses and 0.30% for false alarms. The intervals in Table VI are a useful reminder that four errors are too few to pin down the true rates: the miss rate is compatible with anything from 0.02% to 0.56%. The notebook does not record which four passages were misclassified; therefore, we cannot yet say whether they share an attack category. Saving per-sample predictions is the first item on our follow-up list.

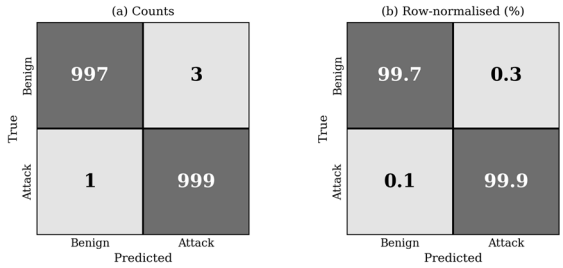


Fig. 3 Confusion matrices on the 2,000-record test set: (a) raw counts and (b) row-normalised percentages.

TABLE VI
TEST-SET METRICS (MALICIOUS = POSITIVE)

Metric	Value	95% Wilson interval
Accuracy	0.9980	0.9949 – 0.9992
Precision	0.9970	0.9912 – 0.9990
Recall (detection rate)	0.9990	0.9944 – 0.9998
Specificity	0.9970	0.9912 – 0.9990
F1-score	0.9980	—
False-negative rate (missed attacks)	0.10%	0.02% – 0.56%
False-positive rate (false alarms)	0.30%	0.10% – 0.88%
Matthews correlation coefficient	0.9960	—

C. Training Dynamics

Fig. 4 plots the 50 epochs. The training loss fell from 0.135 in the first epoch to 0.002 in the last. It did not decrease monotonically: small rises such as 0.010, 0.014, and 0.015 over epochs 6–8 are what one expects from mini-batch updates with dropout at a learning rate of 0.001. Training accuracy was 99.2% after the first epoch, reached 100% at epoch 5, and was at least 99.95% in 46 of the 50 epochs. Validation accuracy rose from 98.0% to 99.5% within five epochs and, from epoch 10 onwards, stayed between 99.5% and 100%, averaging 99.84% (standard deviation 0.09 points) over the last 20 epochs. The validation curve therefore flattens and does not turn downwards, which is the usual sign that the model is not overfitting in the training-versus-validation sense. This is a weaker statement than it sounds, because both curves come from the same corpus and the same noise, and so they say little about unseen text.

©IJRASET: All Rights are Reserved

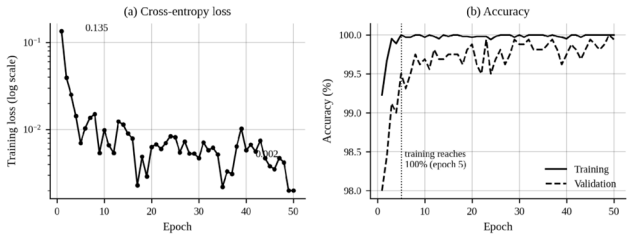


Fig. 4 Training log over 50 epochs: (a) training loss on a logarithmic axis and (b) training and validation accuracy.

D. Per-Class Performance

Per-class scores are obtained from the confusion matrix. The two classes behave almost symmetrically. The benign class has higher precision (99.90%) and lower recall (99.70%) because its three errors are false alarms. For the attack class, the precision reaches 99.70%, while the recall is 99.90%. Both classes share an F1-score of 0.998. In deployment, the recall of the attack class is the figure to watch, and on this test set, it corresponds to one missed attack in 1,000.

E. Sensitivity to Feature Noise

The calibration trials in Table III served as a noise-sensitivity experiment (Fig. 5). Proxy accuracy remained at 100% up to $\sigma = 0.49$, which is half the embedding standard deviation, then slipped to 98.8% at $\sigma = 0.74$ and to 97.8% at $\sigma = 0.99$. The reduction occurred progressively rather than suddenly. The classes were separable by a wide margin in the clean embedding space, and corrupting the features with noise as large as the signal itself removed only a little over two points of accuracy in the proxy setting.

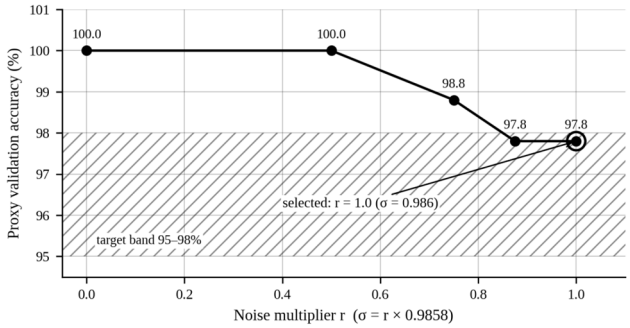


Fig. 5 Proxy validation accuracy against the noise multiplier. The hatched band is the 95–98% calibration target, and the circled point is the level used for all the reported experiments.

The full run ended above the target band: 99.80% on the test set, against 97.8% for the short proxy runs. We think two effects contribute. The noise is drawn independently for every node and mean pooling averages the node outputs, which suppresses independent noise as the number of sentences grows. In addition, a 768-dimensional embedding spreads the class signal over many coordinates; therefore, a linear layer can recover it from noisy inputs. The proxy models also saw only 2,000 training graphs for 10 epochs, whereas the final model used 6,400 graphs for 50 epochs. We did not isolate these explanations. We also did not check the data for near-duplicate passages across the splits; therefore, we cannot exclude that similar templates appear in training and test sets.

F. Comparison with a Keyword Rule

Table VII and Fig. 6 compare the model with the instruction-keyword indicator. The rule never raises a false alarm (precision 100%), but it catches only 75.9% of the attacks, giving 87.93% accuracy, an F1-score of 0.863, and an MCC of 0.782. Its error rate of 12.07% is approximately 60 times the 0.20% of the GAT. This gap exists because 1,207 attacks in the corpus carry no listed trigger word. If the test attacks are drawn from the same distribution, roughly a quarter of the 1,000 test attacks should be keyword-free, and the model missed only one attack in total. Therefore, it detects most keyword-free attacks by other cues in the embeddings, which a word list cannot provide. The comparison has clear limits. The rule is scored on the full corpus and the GAT on the test split, the indicator column is a proxy for a real filter, and we compare against no trained baseline such as TF-IDF or a fine-tuned DeBERTa classifier. This shows that a fixed word list is insufficient on this corpus. This does not show that the graph is better than other learned detectors.

TABLE VII
GAT VERSUS INSTRUCTION-KEYWORD RULE

Metric	Keyword rule (10,000 records)	Proposed GAT (2,000 test)
Accuracy	0.8793	0.9980
Precision	1.0000	0.9970
Recall	0.7586	0.9990
F1-score	0.8627	0.9980
MCC	0.7817	0.9960
Error rate	12.07%	0.20%

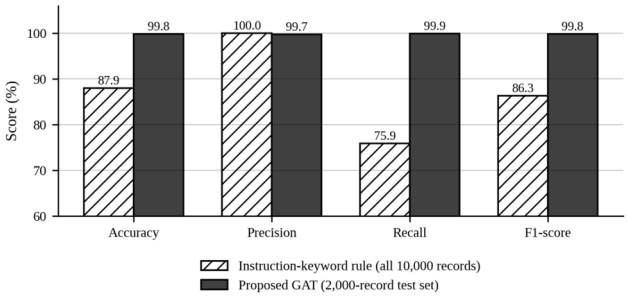


Fig. 6 Accuracy, precision, recall, and F1-score of the instruction-keyword rule (all 10,000 records) and the proposed GAT (test set).

VI. LIMITATIONS

Several limitations qualify these results. Since the dataset is limited to a single English-language corpus focused on finance and corporate policies, the model's effectiveness across different domains, languages, retrieval configurations, and attack techniques remains uncertain. The task difficulty was engineered: the noise was tuned to reach a target accuracy band and was also applied to the test set, so the test data are not clean samples of the original distribution, and accuracy under this protocol need not carry over to ordinary inputs. Apart from the keyword indicator we did not evaluate any trained baseline, such as a TF-IDF classifier, a mean-pooled DeBERTa classifier, the fine-tuned detector of [15] or the methods of [10] and [11], so we make no claim of superiority over any learned detector. Preprocessing and model selection also leave room for leakage. The scaler was fitted on all 10,000 graphs before the split, so test statistics influenced preprocessing, and the evaluated model is the one at the end of training and not a checkpoint chosen on validation data. Sentence embeddings are averaged over padded positions without an attention mask, so they depend slightly on the other sentences in the batch. The corpus also appears to be templated, and we did not search for near-duplicates across the splits. Independent noise draws would make duplicated texts look different but would not remove the overlap. Finally, we screened retrieved passages only: direct prompt injection, knowledge-base poisoning without instruction text, and attackers who know how the detector works lie outside this evaluation.

VII. CONCLUSION AND FUTURE WORK

This study developed a graph-based approach to identifying indirect prompt injection in retrieval-augmented generation (RAG) systems. Each retrieved passage is divided into sentences, which are encoded using a frozen DeBERTa-v3 model and represented as nodes in a graph. A two-layer graph attention network (GAT), containing approximately 0.4 million trainable parameters, then predicts whether the passage is malicious. Experiments on a balanced dataset of 10,000 records related to finance and corporate policies, with calibrated noise added to the input features, produced a test accuracy of 99.80%, an F1-score of 0.998, and an MCC of 0.996. The model generated three false positives and missed one attack, outperforming a keyword-based method that achieved 87.93% accuracy on the same dataset. Although these findings indicate strong classification performance under the tested conditions, further evaluation is needed to establish the model's reliability across unfamiliar domains, previously unseen attack patterns, and adversarial scenarios.

Future research will focus on evaluating the approach using publicly available and independently collected datasets containing prompt-injection attacks. Additional experiments will compare the proposed model with keyword-based and signing-based defenses, as well as TF-IDF, mean-pooled embeddings, and fine-tuned DeBERTa baselines. We will also investigate the effects of feature noise, removing duplicate records before data splitting, and addressing implementation issues involving pooling masks, feature scaling, and checkpoint selection. Furthermore, storing individual predictions will support a detailed examination of classification errors across different attack categories. Attention weights and SHAP values will be explored to assess the interpretability of the model's decisions. Finally, inference latency and processing throughput will be measured under realistic retrieval workloads to determine the practical suitability of the approach for deployment.

VIII. ACKNOWLEDGMENT

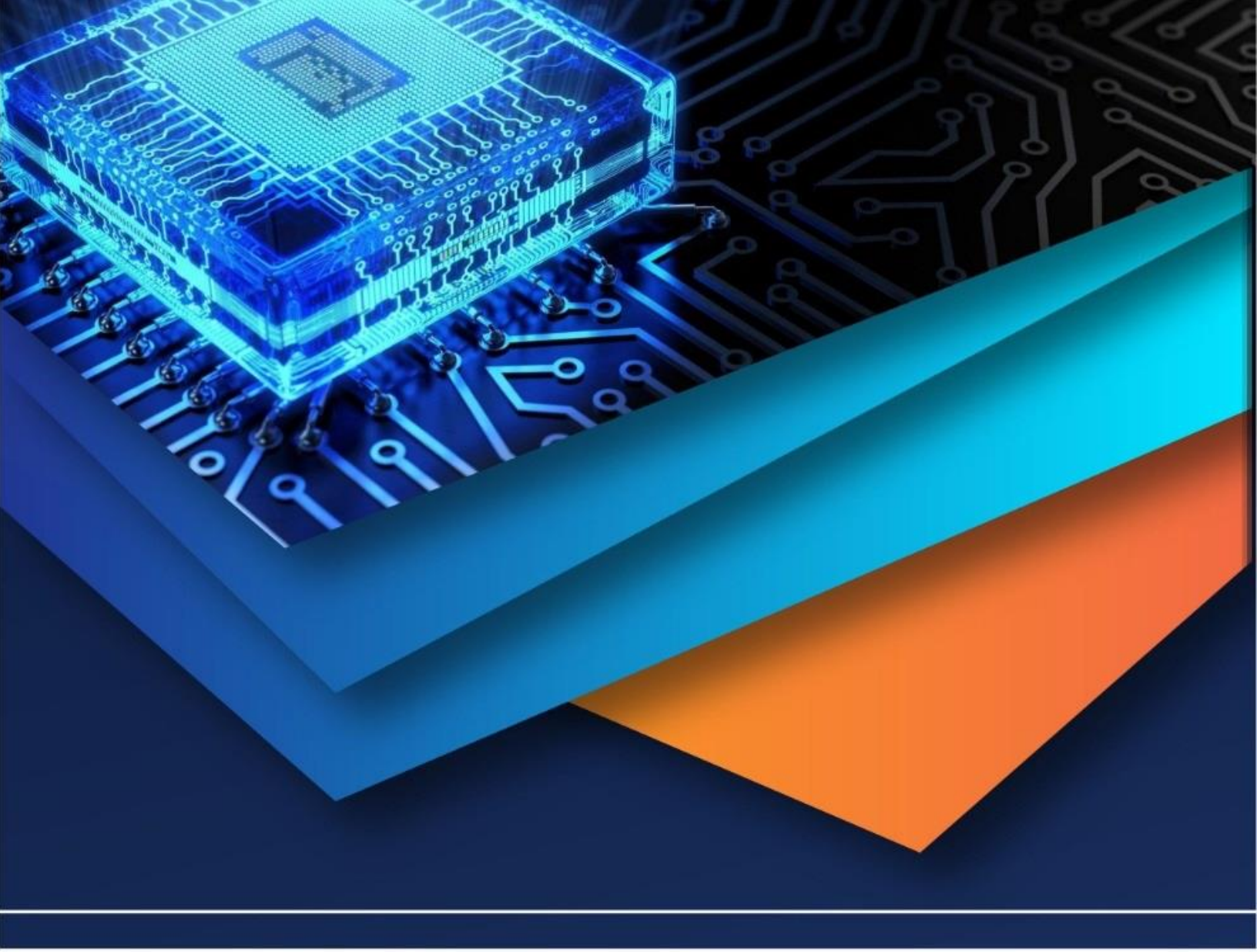
The authors thank the Department of Computer Science and Engineering, Sri Venkateswara University College of Engineering, Tirupati, for providing the facilities for this work.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.
[2] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023.
[3] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents," in Findings of the Association for Computational Linguistics: ACL 2024, 2024.
[4] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track, 2024.
[5] T. Joshi, V. Naik, I. Mistry, and R. Mangrulkar, "Prevention of prompt injection attacks over financial applications integrated with LLM," in Proc. 3rd Int. Conf. Advancement in Computation & Computer Technologies (InCACCT), 2025, pp. 225–230, doi: 10.1109/InCACCT65424.2025.11011372.
[6] S. Jain and B. C. Wallace, "Attention is not explanation," in Proc. NAACL-HLT, 2019.
[7] S. Wiegreffe and Y. Pinter, "Attention is not not explanation," in Proc. EMNLP-IJCNLP, 2019.
[8] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in Proc. 33rd USENIX Security Symposium, 2024.
[9] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, "Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents," arXiv:2410.02644, 2024.
[10] M. A. Ayub and S. Majumdar, "Embedding-based classifiers can detect prompt injection attacks," arXiv:2410.22284, 2024.
[11] H. Li and X. Liu, "InjecGuard: Benchmarking and mitigating over-defense in prompt injection guardrail models," arXiv:2410.22770, 2024.
[12] T. Wen, Y. Liu, Y. Zhang, and Y. Liu, "Defending against indirect prompt injection by instruction detection," in Findings of the Association for Computational Linguistics: EMNLP 2025, 2025.
[13] Y. Liu, Y. Jia, J. Jia, D. Song, and N. Z. Gong, "DataSentinel: A game-theoretic detection of prompt injection attacks," in Proc. IEEE Symposium on Security and Privacy, 2025.
[14] T. Shi et al., "PromptArmor: Simple yet effective prompt injection defenses," arXiv:2507.15219, 2025.
[15] ProtectAI, "deberta-v3-base-prompt-injection-v2," Hugging Face model card, 2024. [Online]. Available: https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2
[16] Suo, "Signed-Prompt: A new approach to prevent prompt injection attacks against LLM-integrated applications," arXiv:2401.07612, 2024.
[17] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending against prompt injection with structured queries," in Proc. 34th USENIX Security Symposium, 2025.
[18] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models," in Proc. 34th USENIX Security Symposium, 2025.
[19] J. Xue, M. Zheng, Y. Hu, F. Liu, X. Chen, and Q. Lou, "BadRAG: Identifying vulnerabilities in retrieval augmented generation of large language models," arXiv:2406.00083, 2024.
[20] A. Shafran, R. Schuster, and V. Shmatikov, "Machine against the RAG: Jamming retrieval-augmented generation with blocker documents," in Proc. 34th USENIX Security Symposium, 2025.
[21] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in Proc. AAAI Conference on Artificial Intelligence, vol. 33, 2019.
[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in Proc. International Conference on Learning Representations (ICLR), 2018.
[23] M. Lin, T. Wang, Y. Zhu, X. Li, X. Zhou, and W. Wang, "A heterogeneous directed graph attention network for inductive text classification using multilevel semantic embeddings," Knowledge-Based Systems, 2024.
[24] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.
[25] T. Kiss and J. Strunk, "Unsupervised multilingual sentence boundary detection," Computational Linguistics, vol. 32, no. 4, pp. 485–525, 2006.
[26] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention," in Proc. International Conference on Learning Representations (ICLR), 2021.

[27] P. He, J. Gao, and W. Chen, “DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing,” in Proc. International Conference on Learning Representations (ICLR), 2023.
[28] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.
[29] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. International Conference on Learning Representations (ICLR), 2015.
[30] E. B. Wilson, “Probable inference, the law of succession, and statistical inference,” Journal of the American Statistical Association, vol. 22, no. 158, pp. 209–212, 1927.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)