

# Embedding-based classifiers can detect prompt injection attacks

Md. Ahsan Ayub<sup>1</sup>, Subhabrata Majumdar<sup>2</sup>

<sup>1</sup>Enterprise Cybersecurity, Vanderbilt University Medical Center, Nashville, TN, USA

<sup>2</sup>Vijil, Seattle, WA, USA

## Abstract

Large Language Models (LLMs) are seeing significant adoption in every type of organization due to their exceptional generative capabilities. However, LLMs are found to be vulnerable to various adversarial attacks, particularly prompt injection attacks, which trick them into producing harmful or inappropriate content. Adversaries execute such attacks by crafting malicious prompts to deceive the LLMs. In this paper, we propose a novel approach based on embedding-based Machine Learning (ML) classifiers to protect LLM-based applications against this threat. We leverage three commonly used embedding models to generate embeddings of malicious and benign prompts and utilize ML classifiers to predict whether an input prompt is malicious. Out of several traditional ML methods, we achieve the best performance with classifiers built using Random Forest and XGBoost. Our classifiers outperform state-of-the-art prompt injection classifiers available in open-source implementations, which use encoder-only neural networks.

**Warning:** This paper discusses and contains language that could be considered inappropriate for readers.

## Keywords

adversarial attacks, embeddings, large language models, machine learning, prompt injection

## 1. Introduction

Large Language Models (LLMs) have been widely adopted to streamline daily tasks that need automation, such as text (including code) generation [1, 2], text summarization [3], sentiment analysis [4], AI chatbots [5], and machine translation [6]. Compared to traditional software, LLM, generative AI (genAI) applications, and agents embody a much broader attack surface that adversaries can exploit for malicious purposes. For example, LLMs are found to produce contents containing gender and racial biases, toxicity, disinformation, and misinformation [7–9]. To further explain, the models are designed to generate response based on the supplied prompts. By crafting malicious prompts, attackers attempt to override LLM developers’ instruction to exploit the baseline model.

In this paper, we focus on examining prompts that lead to successful prompt injection attacks. Specifically, we investigate the behavior of malicious prompts that attempt prompt injection versus benign prompts in the embedding space. With the goal of developing effective embedding-based approaches to safeguard genAI applications from prompt injection attacks, we ask the following important research questions:

**RQ1.** Are there any dissimilarities between benign and malicious prompts?

**RQ2.** Can we effectively identify malicious prompts to thwart prompt injection attacks?

To address the questions above, we curate a dataset of 467,057 malicious and benign prompts. We obtain their embeddings based on three state-of-the-art embedding models and use this data to gain insights into the overall behavior of malicious prompts.

The major contributions of our paper are as follows:

- We investigate the distributional differences of benign and malicious embeddings generated using three embedding models: from the API-only OpenAI text-embedding-3-small, and the open-source models gte-large, and all-MiniLM-L6-v2.

---

CAMLIS’24: Conference on Applied Machine Learning for Information Security, October 24–25, 2024, Arlington, VA

✉ ahsan.ayub@vumc.org (Md. A. Ayub); subho@vijil.ai (S. Majumdar)

🆔 0000-0002-1345-0110 (Md. A. Ayub); 0000-0003-3529-7820 (S. Majumdar)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

- Using the embeddings as input datasets, we build a suite of supervised machine learning (ML) classifiers to detect prompt injection attacks.
- Across several metrics, we compare the performance of embedding-based classifiers’ with state-of-the-art deep learning based prompt injection classifiers.

Our implementation and the curated datasets used for evaluation are available on GitHub<sup>1</sup>.

The rest of the paper is organized as follows: Section 2 discusses the background of our research. The discussion of experimental methodology, including the construction of the dataset, and our empirical findings are described in Sections 3 and 4, respectively. We share the related work in this field in Section 5 and list the limitations of our study, as well as future work, in Section 6. Finally, we provide the conclusion of this work in Section 7.

## 2. Background and Related Work

This section describes some background and related work, as context for the subsequent sections.

### 2.1. Prompt Injection Attacks

SQL injections and Cross-Site Scripting (XSS) attacks are among the most commonly found cyber threats, where attackers craft payloads to disrupt the routine execution of a program [10, 11]. With the proliferation of genAI, adversaries can carry out similar attacks by injecting LLMs with malicious prompts. In genAI applications, users can utilize the extensible functionalities of LLMs via natural language-based prompts to generate desired outputs. Attackers exploit this interaction pattern by supplying crafted prompts to cause LLMs to perform undesired actions [12, 13]. These malicious prompts can be supplied as inputs either by as malicious users, or by attackers modifying benign user-provided prompts through man-in-the-middle attacks. A successful prompt injection attack leads to unintended consequences, such as the exposure of underlying system prompts, disclosure of private data, and attackers gaining unauthorized access to functionalities the LLM is authorized to perform but the user is not [14].

Another example of such attacks is when an LLM takes input from external sources, such as websites or files—adversaries inject malicious prompts to hijack the context. This is known as an indirect prompt injection attack. The goal of this attack is to extract sensitive, harmful, or unwanted information from the LLMs [15]. Real attackers do not need to possess deep technical knowledge about how the model is built, or compute gradients, to trick the LLM application into responding to a distinct set of queries with the intent of compromising it [16].

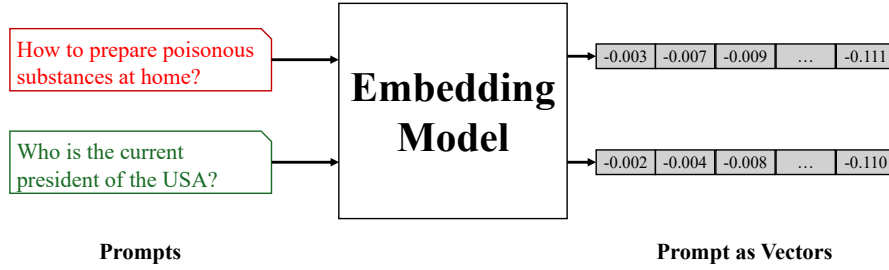
### 2.2. Embedding Models

Prompts are primarily constructed using natural language, and their size can vary widely. We apply embedding models to convert the textual data in prompts into dense representations in a multi-dimensional space of fixed dimension [17]. To explain further, each prompt is transformed into a fixed-length sequence of floating-point numbers. Such numerical representations enable us to create a vector database derived from a list of prompts. To accomplish these tasks, we select the following embedding models: `text-embedding-3-small` from OpenAI<sup>2</sup>, and the open-source models `gte-large` hosted on OctoAI<sup>3</sup>, as well as the well-known `all-MiniLM-L6-v2`. For brevity we refer to them as OpenAI, GTE, MiniLM from here on. It is important to note that all of these embedding models are contextual representation models [18]. This means that each word is placed in the vector space based on the input context. For example, the word “apple” has a static meaning of being a fruit. However, given a specific input context, it can also refer to a technology company. Our selected embedding models are equipped to capture the context of each word in a prompt[19]. We illustrate an example in Fig. 1.

<sup>1</sup><https://github.com/AhsanAyub/malicious-prompt-detection>

<sup>2</sup><https://platform.openai.com/docs/guides/embeddings>

<sup>3</sup><https://octo.ai/blog/introducing-octoais-embedding-api-to-power-your-rag-needs>



**Figure 1:** A schematic diagram of prompts and their embeddings.

The length of the embedding vector for OpenAI is 1536, which means that any size of textual data is mapped to a 1536-dimensional dense numerical vector space. Additionally, the embedding vector sizes for GTE and MiniLM are 1024 and 384, respectively.

### 2.3. Related Work

In recent past, genAI threats—especially prompt injection attacks—have received a significant attention from AI security practitioners and researchers. A successful attack enables adversaries to override intended use guidelines of an LLM application to generate violating content along different directions, such as hate speech and discrimination, profanity, sexual, violent, and unsafe content, controversial topics, illegal activities, self-harm, harassment, unethical and actions<sup>4</sup>. AI security researchers have come up with numerous techniques that adversaries can utilize to perform prompt injection attacks [12, 20–24]. Therefore, it is important to defend genAI applications against such attacks.

We break down prior research on prompt injection detection into two categories.

**Guardrail-based (AI Firewall) Defense.** Alon and Kamfonas [25] used a perplexity-based approach to detect malicious prompts by computing perplexity to estimate text quality. The injection of instructions or data into prompts influences quality and results in a high perplexity value. Jain et al. [26] divided textual data into contiguous windows for perplexity calculation to check whether any window’s perplexity exceeds the threshold. Chen et al. [27] examined how separating prompts and supplied data enables LLMs to become more robust against prompt injection. Yi et al. [28] found that placing a special delimiter between the prompt and data allows LLMs to distinguish between malicious external content and user instructions, thereby preventing harmful outputs. Schulhoff et al. [29] discovered that adding extra text to the prompt to make LLMs aware of prompt injection attacks would also be effective.

**LLM-based Defense.** Recent LLMs, such as GPT-4o and the Gemini and Claude families of models, show a propensity of rejecting harmful prompts incorporated through safety training [22, 30, 31]. LLMs may also be used as detectors designed to identify malicious prompts through their training [32]. AI security researchers have showed that it is possible to detect prompt injection by providing explicit instructions to LLMs, such as “*..Your job is to analyze whether the input prompt is safe...*” and using this model as an LLM-as-judge to evaluate input prompts [33]. Finally, traditional encoder-only NLP models, such as the ones using a DeBERTa architecture that utilizes disentangled attention and an enhanced mask decoder, can detect prompt injection and jailbreak attacks [34, 35].

<sup>4</sup><https://www.robustintelligence.com/ai-security-and-safety-taxonomy>

**Table 1**

Hugging Face datasets used in our study

Dataset (User: Title)	# fo Prompts
imoxto: Prompt Injection cleaned dataset	535,105
reshabhs: SPML Chatbot Prompt Injection	16,012
Harelix: Prompt Injection Mixed Techniques	1,174
JasperLS: Prompt Injections	662
fka: Awesome Chatgpt Prompts	153
rubend18: ChatGPT Jailbreak Prompts	79

**Our Approach.** To the best of our knowledge, our study is the first attempt to investigate the effectiveness of embedding-based classifiers in detecting malicious prompts. Although a lot of work has already been published in this area, we did not find any research on embeddings of malicious prompts and their efficacy in leading to successful detection. We hope that this research will make meaningful contribution to the AI safety and security domain by extending and reproducing our experiments.

### 3. Methodology

#### 3.1. Dataset Construction

The dataset used in our experiments is curated from open-source datasets containing malicious and benign prompts pertaining to prompt injection attacks (Table 1). In total, we acquire a total of 553,185 numbers of malicious and benign prompts. After deduplication, we end up with a total of 467,057 unique prompts, of which 109,934 (23.54%) are malicious. Each prompt is assigned a unique identifier and a source to indicate its origin. Therefore, the dataset columns appear as follows: ID, Source, Text, and Label (0 to denote benign, 1 for malicious). Using the `train_text_split` method<sup>5</sup>, we split this dataset into 80% training and 20% test sets. To ensure equal proportion of the malicious and benign labels across splits, we use stratified sampling.

We develop a data pipeline using Python 3.11 to generate the embeddings for all prompts. With OpenAI’s API key, we submit each prompt to get its embedding through `text-embedding-3-small` model. To obtain the GTE embeddings, we use the `thenlper/gte-large` model<sup>6</sup>, accessed remotely through the serverless endpoint on OctoAI. For the MiniLM embeddings, we download the `sentence-transformers/all-MiniLM-L6-v2`<sup>7</sup> model and host it locally. This approach allowed us to construct three separate tabular datasets composed of embeddings based on each of the embedding models.

Embeddings consist of fixed-length numerical representations. Therefore, we convert them into column values from lists. For instance, OpenAI generates an embedding vector consisting of 1,536 floating-point numbers for each prompt. We organize these numbers into 1,536 columns, treating each vector item as a separate column value. Consequently, the final embedding dataset generated by OpenAI comprises 1,539 features, with ID, Source, and Label as additional columns. Similarly, the embedding datasets for OctoAI and MiniLM consist of 1,027 and 387 features, respectively.

#### 3.2. Experimental Setup

**Methodology to Address RQ1: Visualization of Embeddings after Dimension Reduction.** The embeddings provide us with high-dimensional tabular datasets. We apply Principal Component Analysis (PCA) [36], t-Distributed Stochastic Neighbor Embedding (t-SNE) [37], and Uniform Manifold Approximation and Projection (UMAP) [38] to reduce these dense data distributions to a two-dimensional plane

<sup>5</sup><https://huggingface.co/docs/datasets/v1.8.0/processing.html#splitting-the-dataset-in-train-and-test-split-train-test-split>

<sup>6</sup><https://huggingface.co/thenlper/gte-large>

<sup>7</sup><https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

**Table 2**

Details of the captured information after reducing the dimensionality of datasets.

Principal Component	OpenAI	GTE	MiniLM
1st Principal Component	13.59%	15.84%	12.62%
2nd Principal Component	10.83%	11.03%	11.43%

for visualization. This approach will help us investigate whether there are clear decision boundaries that can separate malicious prompts from benign ones.

We start our analysis with PCA, a linear dimensionality reduction approach using Singular Value Decomposition of the data [39]. We employ `sklearn` [40], a Python machine learning package, to apply PCA and project the embedding-based columns into a 2-dimensional space.

Next, we use t-SNE to visualize our high-dimensional embedding distributions. It uses a nonlinear dimensionality reduction technique, unlike PCA. Similar to PCA, we use `sklearn` to execute the tasks. One of the major hyperparameters of this algorithm is “perplexity”, which is a guess about the number of close neighbors each point has [41]. Typically, its suggested values range between 5 and 50.

Finally, we examine the visualizations obtained using UMAP, which is based on manifold theory [42]. It seeks a low-dimensional representation of embeddings with an equivalent fuzzy topological structure. The algorithm operates in two phases: first, constructing a weighted k-nearest neighbor graph, and second, computing a low-dimensional layout of this graph. Variations among algorithms in this class lie in the specific methods used for graph construction and layout computation [38]. We utilize its Python package to run our experiments<sup>8</sup>.

**Methodology to Address RQ2: Binary Classification.** To detect the malicious prompts, we train classifiers using three traditional ML methods: Logistic Regression [43], eXtreme Gradient Boosting (XGBoost) [44], and Random Forest [45]. We use `sklearn` to apply Logistic Regression and Random Forest. To implement XGBoost, we use its corresponding Python package<sup>9</sup> to run experiments. The goal of employing these classifiers is to train them on the train splits of each embedding dataset, encompassing both benign and malicious prompts, enabling the algorithms to discern underlying patterns. We evaluate the out-of-sample efficacy of each classifier on the test splits of the respective embedding datasets.

## 4. Results

In this section, we report the findings of our experiments.

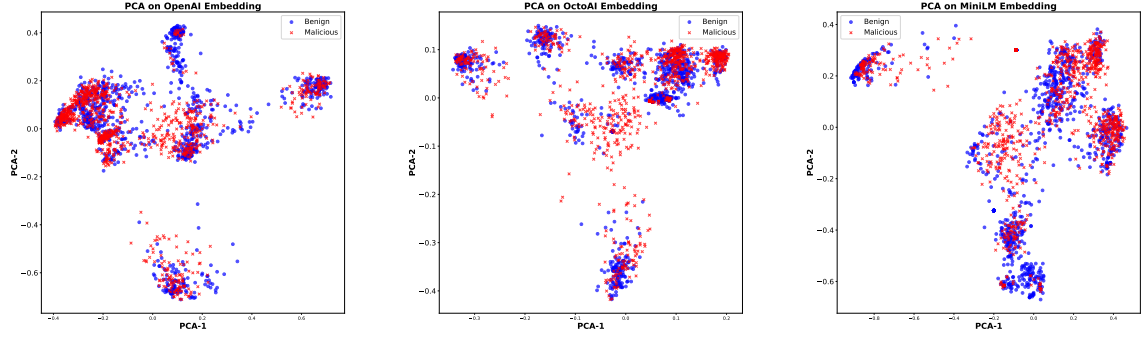
### 4.1. Answer to RQ1: Are benign and malicious prompts dissimilar in the embedding space?

The first phase of our experiments is centered around visualizing the low-dimensional projections of the embeddings, generated using PCA, t-SNE, and UMAP. With details provided in Table 2, we capture 12.62% to 15.83% and 10.83% to 11.43% of the information for all three embeddings through PCA with the 1st and 2nd Principal Components, respectively. We present the visualizations of OpenAI, OctoAI, and MiniLM embeddings after applying PCA in Fig. 2. For t-SNE, our experiments involve investigating visualizations using perplexity values ranging from 5 to 50. We achieve the most well-separated clusters by selecting a perplexity of 15 (Fig. 3). Lastly, we depict the visualizations of all three embeddings after applying UMAP in Fig. 4.

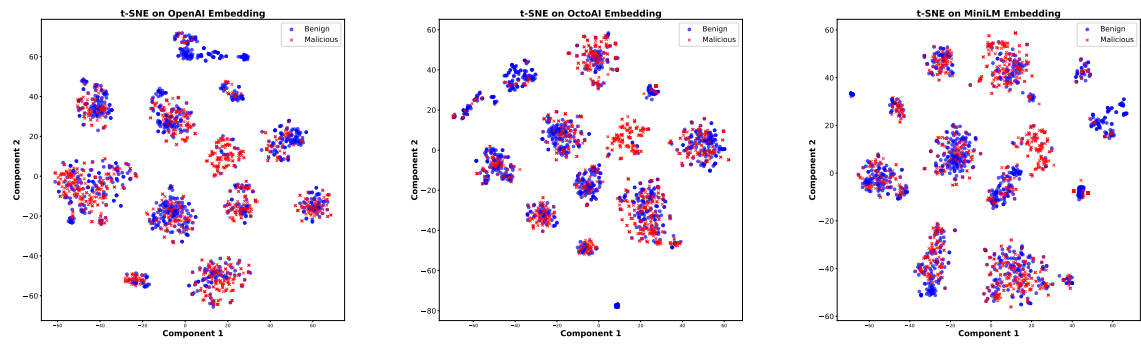
As seen in the three plots, we do not find clear separations between benign and malicious data points. Especially, linear or sigmoid separations are not observed between red and blue clusters. This indicates that tree-based and/or gradient boosting algorithms will be better suited to separate the malicious data

<sup>8</sup>[https://umap-learn.readthedocs.io/en/latest/basic\\_usage.html](https://umap-learn.readthedocs.io/en/latest/basic_usage.html)

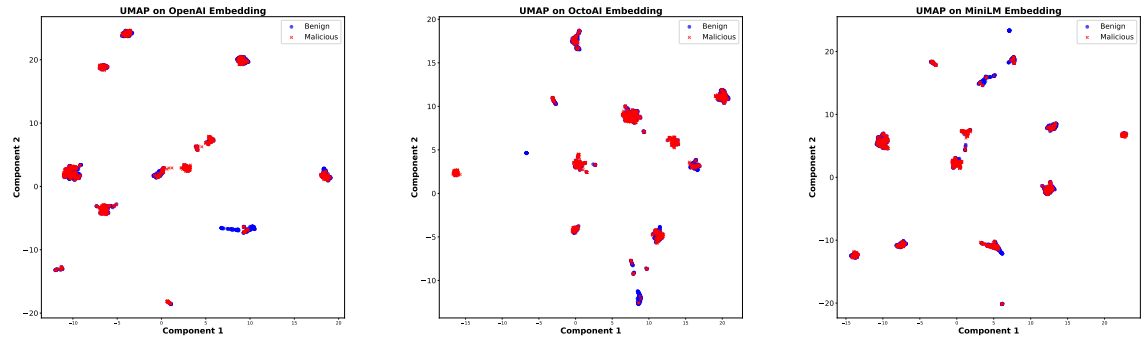
<sup>9</sup><https://xgboost.readthedocs.io/en/stable/python/index.html>



**Figure 2:** Visualization of OpenAI (Left), GTE (Middle), and MiniLM (Right) embedding distribution after applying Principal Components Analysis (PCA).



**Figure 3:** Visualization of OpenAI (Left), GTE (Middle), and MiniLM (Right) embedding distribution after applying T-distributed Stochastic Neighbor Embedding (t-SNE).



**Figure 4:** Visualization of OpenAI (Left), GTE (Middle), and MiniLM (Right) embedding distribution after applying Uniform Manifold Approximation and Projection (UMAP).

points compared to methods based on the linearity assumption, such as logistic regression and linear discriminant analysis.

#### 4.2. Answer to RQ2: Can we effectively identify malicious prompts to thwart prompt injection attacks?

As mentioned in the earlier section, we employ Logistic Regression, XGBoost, and Random Forest classifiers for performing binary classification tasks. We maintain the size of all embeddings consistently for both training and testing purposes. Initially, we report their performances in terms of AUC (Area



**Table 3**

Performance comparisons based on AUC across classifiers and embedding methods.

Embedding	Logistic Regression	XGBoost	Random Forest
OpenAI	0.637	0.726	0.764
GTE	0.612	0.690	0.731
MiniLM	0.608	0.687	0.730

**Table 4**

Binary classification performance of embedding-based ML classifiers

Embedding	Logistic Regression			XGBoost			Random Forest		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
OpenAI	0.793	0.807	0.80	0.832	0.841	0.836	0.867	0.867	0.867
GTE	0.785	0.799	0.792	0.820	0.830	0.825	0.849	0.853	0.851
MiniLM	0.777	0.795	0.789	0.820	0.829	0.824	0.849	0.853	0.851

under the ROC Curve). The AUC value ranges between 0 and 1 –the higher the value, the better a classifier’s prediction capability. For example, a classifier with perfect predictions would have an AUC of 1. Table 3 presents the AUC values for all classifiers across different embeddings. We observe that Random Forest consistently outperforms the other two classifiers under all experimental settings. Among the embeddings methods, OpenAI performs the best—possibly owing to its higher dimensionality.

Using default binary predictions from the ML classifiers, we also compute precision, recall, and F1 scores, which also range between 0 and 1.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Similar to AUC, Random Forest performs the best compared to XGBoost and Logistic Regression, as highlighted in Table 4. For instance, the precision and recall scores of Random Forest for OpenAI embeddings are 86.65% and 86.96%, respectively, which are up to 3% higher than XGBoost and 6% higher than Logistic Regression. Similar results are observed for the other two embeddings as well. Therefore, the Random Forest algorithm is identified as the best classifier in our study.

*Comparison with State-of-the-Art Classifiers.* We compare the performances of our embedding-based classifiers with four state-of-the-art deep learning classifiers available on Hugging Face. To begin describing each classifier, Tunstall et al. [46] released their Sentence Transformer model on Hugging Face as *Myadav: setfit-prompt-injection-MiniLM-L3-v2*<sup>10</sup> for text classification. The second classifier we compared with is *protectai: deberta-v3-base-prompt-injection*, which is based on DeBERTaV3 [47] and was released in 2023<sup>11</sup>. We also compare against *protectai: deberta-v3-base-prompt-injection-v2*, an updated version of the above model based on optimization of hyperparameters, training regimens, and dataset compositions<sup>12</sup>. Finally, we compare against another popular finetune of DeBERTaV3 called *deepset: deberta-v3-base-injection*<sup>13</sup>. We examine the performances of all four classifiers using AUC, precision, and recall scores on our test dataset including both malicious and benign prompts.

Table 5 reports the performance metrics for the four SoTA models and compares them against our best model. The Random Forest classifier built with OpenAI embeddings outperforms other SoTA

<sup>10</sup><https://huggingface.co/Myadav/setfit-prompt-injection-MiniLM-L3-v2>

<sup>11</sup><https://huggingface.co/protectai/deberta-v3-base-prompt-injection>

<sup>12</sup><https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2>

<sup>13</sup><https://huggingface.co/deepset/deberta-v3-base-injection>

**Table 5**

Performance comparisons against popular and accurate prompt injection classifiers available on Hugging Face

Model	AUC	Precision	Recall	F1
Myadav: setfit-prompt-injection-MiniLM-L3-v2	0.594	0.827	0.62	0.709
protectai: deberta-v3-base-prompt-injection	0.531	0.774	0.910	0.837
protectai: deberta-v3-base-prompt-injection-v2	0.511	0.758	<b>0.991</b>	0.859
deepset: deberta-v3-base-injection	0.500	0.762	0.988	0.860
Our best (Random Forest + OpenAI)	<b>0.764</b>	<b>0.867</b>	0.870	<b>0.868</b>

classifiers in terms of AUC and precision scores. The recall score for our best model is 86.96%, whereas other classifiers achieve scores in the high 90s. However, due to higher precision, our classifier achieves the highest F1 score. In real-world ML pipelines, calibrating and updating thresholds based on test time data is standard practice, and balancing precision and recall is important. In general, the three DeBERTa finetunes have high recall but low precision, whereas *Myadav: setfit-prompt-injection-MiniLM-L3-v2* has high precision but low recall. Our embedding-based approach strikes a better balance between these two metrics.

## 5. Discussion and Conclusion

In this paper, we propose a novel embedding-based classifier approach to detect malicious prompts that lead to successful prompt injection. We curate a large dataset of benign and malicious prompts from several repositories on Hugging Face, and generate their embeddings using three methods. Through two-dimensional visualizations, we investigate the distributional differences of embeddings labeled benign and malicious, and perform binary classification tasks using a number of supervised ML classifiers. The Random Forest classifier trained using OpenAI embeddings exhibits the best performance, achieving an AUC of 0.764, precision of 0.867, and recall of 0.87. Comparing our classifier’s performance with several SoTA prompt injection available on HuggingFace designed for similar tasks. Our results demonstrate that our classifier surpasses all of them in terms of AUC and precision scores.

One of the main goals of this paper was to investigate the dissimilarities between embeddings of malicious and benign prompts using dimensionality reduction algorithms. However, we did not find a clear linear separation in the generated visualizations. While we leave further exploration for future work, we did achieve commendable performance in detecting prompt injections via traditional ML classifiers trained on these embeddings. Especially, the random forest classifier was able to outperform the most popular and highest performant models available in the open source.

Our study examines the efficacy of traditional ML classifiers. Neural network-based classifiers may also be constructed based on embeddings. This needs to be explored in future work. Our research has primarily focused on crafting classifiers to detect direct prompt injections. A similar approach can also be taken to craft embedding-based detectors for other LLM attack vectors and failure modes such as indirect prompt injections, toxicity, and hallucination.

## Acknowledgments

We thank the CAMLIS program committee and reviewers for reviewing the paper and sharing valuable feedback that led to significant improvements.

## References

- [1] R. Tang, Y.-N. Chuang, X. Hu, The science of detecting llm-generated texts, arXiv preprint arXiv:2303.07205 (2023).



- [2] J. Liu, C. S. Xia, Y. Wang, L. Zhang, Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, *Advances in Neural Information Processing Systems* 36 (2024).
- [3] Y. Chang, K. Lo, T. Goyal, M. Iyyer, Boookscore: A systematic exploration of book-length summarization in the era of llms, *arXiv preprint arXiv:2310.00785* (2023).
- [4] B. Zhang, H. Yang, T. Zhou, M. Ali Babar, X.-Y. Liu, Enhancing financial sentiment analysis via retrieval augmented large language models, in: *Proceedings of the Fourth ACM International Conference on AI in Finance, 2023*, pp. 349–356.
- [5] J. K. Kim, M. Chua, M. Rickard, A. Lorenzo, Chatgpt and large language model (llm) chatbots: The current state of acceptability and a proposal for guidelines on utilization in academic medicine, *Journal of Pediatric Urology* (2023).
- [6] Z. He, T. Liang, W. Jiao, Z. Zhang, Y. Yang, R. Wang, Z. Tu, S. Shi, X. Wang, Exploring human-like translation strategy with large language models, *Transactions of the Association for Computational Linguistics* 12 (2024) 229–246.
- [7] T. Y. Zhuo, Y. Huang, C. Chen, Z. Xing, Exploring ai ethics of chatgpt: A diagnostic analysis, *arXiv preprint arXiv:2301.12867* 10 (2023).
- [8] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, et al., Holistic evaluation of language models, *arXiv preprint arXiv:2211.09110* (2022).
- [9] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al., Training a helpful and harmless assistant with reinforcement learning from human feedback, *arXiv preprint arXiv:2204.05862* (2022).
- [10] S. W. Boyd, A. D. Keromytis, Sqlrand: Preventing sql injection attacks, in: *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings 2*, Springer, 2004, pp. 292–302.
- [11] S. Gupta, B. B. Gupta, Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art, *International Journal of System Assurance Engineering and Management* 8 (2017) 512–530.
- [12] F. Perez, I. Ribeiro, Ignore previous prompt: Attack techniques for language models, in: *NeurIPS ML Safety Workshop*, 2022.
- [13] J. Yu, Y. Wu, D. Shu, M. Jin, X. Xing, Assessing prompt injection risks in 200+ custom gpts, *arXiv preprint arXiv:2311.11538* (2023).
- [14] Y. Liu, G. Deng, Y. Li, K. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, Y. Liu, Prompt injection attack against llm-integrated applications, *arXiv preprint arXiv:2306.05499* (2023).
- [15] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, M. Fritz, Not what you’ve signed up for: Compromising real-world llm-integrated applications with indirect prompt injection, in: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023*, pp. 79–90.
- [16] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, K. Roundy, “real attackers don’t compute gradients”: bridging the gap between adversarial ml research and practice, in: *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, IEEE, 2023, pp. 339–364.
- [17] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Halls-Lacy, et al., Text and code embeddings by contrastive pre-training, *arXiv preprint arXiv:2201.10005* (2022).
- [18] N. F. Liu, M. Gardner, Y. Belinkov, M. E. Peters, N. A. Smith, Linguistic knowledge and transferability of contextual representations, *arXiv preprint arXiv:1903.08855* (2019).
- [19] D. S. Asudani, N. K. Nagwani, P. Singh, Impact of word embedding models on text analytics in deep learning environment: a review, *Artificial intelligence review* 56 (2023) 10345–10425.
- [20] D. Kang, X. Li, I. Stoica, C. Guestrin, M. Zaharia, T. Hashimoto, Exploiting programmatic behavior of llms: Dual-use through standard security attacks, *arXiv preprint arXiv:2302.05733* (2023).
- [21] X. Shen, Z. Chen, M. Backes, Y. Shen, Y. Zhang, "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models, *arXiv preprint arXiv:2308.03825* (2023).
- [22] A. Wei, N. Haghtalab, J. Steinhardt, Jailbroken: How does llm safety training fail?, *Advances in Neural Information Processing Systems* 36 (2024).

- [23] M. Samvelyan, S. C. Rapparth, A. Lupu, E. Hambro, A. H. Markosyan, M. Bhatt, Y. Mao, M. Jiang, J. Parker-Holder, J. Foerster, et al., Rainbow teaming: Open-ended generation of diverse adversarial prompts, arXiv preprint arXiv:2402.16822 (2024).
- [24] Y. Bai, G. Pei, J. Gu, Y. Yang, X. Ma, Special characters attack: Toward scalable training data extraction from large language models, arXiv preprint arXiv:2405.05990 (2024).
- [25] G. Alon, M. Kamfonas, Detecting language model attacks with perplexity, arXiv preprint arXiv:2308.14132 (2023).
- [26] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, T. Goldstein, Baseline defenses for adversarial attacks against aligned language models, arXiv preprint arXiv:2309.00614 (2023).
- [27] S. Chen, J. Piet, C. Sitawarin, D. Wagner, Struq: Defending against prompt injection with structured queries, arXiv preprint arXiv:2402.06363 (2024).
- [28] J. Yi, Y. Xie, B. Zhu, K. Hines, E. Kiciman, G. Sun, X. Xie, F. Wu, Benchmarking and defending against indirect prompt injection attacks on large language models, arXiv preprint arXiv:2312.14197 (2023).
- [29] S. Schulhoff, J. Pinto, A. Khan, L.-F. Bouchard, C. Si, S. Anati, V. Tagliabue, A. Kost, C. Carnahan, J. Boyd-Graber, Ignore this title and hackaprompt: Exposing systemic vulnerabilities of llms through a global prompt hacking competition, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023, pp. 4945–4977.
- [30] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).
- [31] D. Ganguli, L. Lovitt, J. Kernion, A. Askell, Y. Bai, S. Kadavath, B. Mann, E. Perez, N. Schiefer, K. Ndousse, et al., Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned, arXiv preprint arXiv:2209.07858 (2022).
- [32] N. Belrose, Z. Furman, L. Smith, D. Halawi, I. Ostrovsky, L. McKinney, S. Biderman, J. Steinhardt, Eliciting latent predictions from transformers with the tuned lens, arXiv preprint arXiv:2303.08112 (2023).
- [33] S. Armstrong, R. Gorman, Using gpt-eliezer against chatgpt jailbreaking. 2022, URL <https://www.alignmentforum.org/posts/pNcFYZnPdXyL2RfgA/using-gpt-eliezer-against-chatgpt-jailbreaking-5> (????).
- [34] ProtectAI.com, Fine-tuned deberta-v3-base for prompt injection detection, 2024. URL: <https://huggingface.co/ProtectAI/deberta-v3-base-prompt-injection-v2>.
- [35] Meta, Model card - prompt guard, 2024. URL: <https://huggingface.co/meta-llama/Prompt-Guard-86M>.
- [36] H. Abdi, L. J. Williams, Principal component analysis, Wiley interdisciplinary reviews: computational statistics 2 (2010) 433–459.
- [37] L. Van der Maaten, G. Hinton, Visualizing data using t-sne., Journal of machine learning research 9 (2008).
- [38] L. McInnes, J. Healy, J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction, arXiv preprint arXiv:1802.03426 (2018).
- [39] G. W. Stewart, On the early history of the singular value decomposition, SIAM review 35 (1993) 551–566.
- [40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
- [41] M. Wattenberg, F. Viégas, I. Johnson, How to use t-sne effectively, Distill 1 (2016) e2.
- [42] S. Mac Lane, Categories for the working mathematician, volume 5, Springer Science & Business Media, 2013.
- [43] M. Schmidt, N. Le Roux, F. Bach, Minimizing finite sums with the stochastic average gradient, Mathematical Programming 162 (2017) 83–112.
- [44] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm

sigkdd international conference on knowledge discovery and data mining, 2016, pp. 785–794.

- [45] L. Breiman, Random forests, *Machine learning* 45 (2001) 5–32.
- [46] L. Tunstall, N. Reimers, U. E. S. Jo, L. Bates, D. Korat, M. Wasserblat, O. Pereg, Efficient few-shot learning without prompts, *arXiv preprint arXiv:2209.11055* (2022).
- [47] P. He, X. Liu, J. Gao, W. Chen, Deberta: Decoding-enhanced bert with disentangled attention, *arXiv preprint arXiv:2006.03654* (2020).