Defending Against Knowledge Poisoning Attacks During Retrieval-Augmented Generation

Kennedy Edemacu*, Vinay M. Shashidhar†, Micheal Tuape‡, Dan Abudu§, Beakcheol Jang¶, Jong Wook Kim‖

arXiv:2508.02835v1 [cs.LG] 4 Aug 2025

***Abstract*—Retrieval-Augmented Generation (RAG) has emerged as a powerful approach to boost the capabilities of large language models (LLMs) by incorporating external, up-to-date knowledge sources. However, this introduces a potential vulnerability to knowledge poisoning attacks, where attackers can compromise the knowledge source to mislead the generation model. One such attack is the PoisonedRAG in which the injected adversarial texts steer the model to generate an attacker chosen response for a target question. In this work, we propose novel defense methods, FilterRAG and ML-FilterRAG, to mitigate the PoisonedRAG attack. First, we propose a new property to uncover distinct properties to differentiate between adversarial and clean texts in the knowledge data source. Next, we employ this property to filter out adversarial texts from clean ones in the design of our proposed approaches. Evaluation of these methods using benchmark datasets demonstrate their effectiveness, with performances close to those of the original RAG systems.**

***Index Terms*—Large language models, retrieval-augmented generation, and knowledge poisoning attack**

I. Introduction

A KEY challenge associated with large language models (LLMs) [1]–[3] is their tendency of becoming outdated and struggling to integrate the most recent knowledge [4], [5]. This fundamental short-coming is addressed by the recent emergency of retrieval-augmented generation (RAG) [6]–[9]. Typically, a RAG system comprises two phases: *Retrieval* and *Generation*. The retrieval phase is accomplished through two components: a *retriever* and a *knowledge database*, while generation is performed by an LLM. During retrieval, the retriever retrieves information relevant to a user's query from the knowledge database. The retrieved text is then passed as a context to the LLM together with the user query to generate the final answer. Various studies have demonstrated the effectiveness of RAGs for different real-world applications [10]–[13].

RAG has attracted significant attention from the research community in recent years, with a primary focus on improving its effectiveness and efficiency [14]–[18]. However, more recently, an emerging body of research, although limited, has begun to explore the security aspects of RAG systems [4], [5], [19]–[22]. Specifically, [5], [19], [20] have developed knowledge poisoning attacks for RAG, with PoisonedRAG [5] standing out as an initial significant contributor towards this direction.

In PoisonedRAG [5], an attacker compromises the knowledge database by injecting adversarial texts. The primary objective is to manipulate the RAG system to generate an attacker-desired response for a specific target question, particularly when using the retrieved, poisoned context during the generation phase. As an example, for a user query, *Which disease is normally caused by the human immunodeficiency virus?* The attacker can compromise the knowledge database so that the text retrieved from the compromised knowledge database will drive the generation LLM to output *Syphilis* instead of *AIDS* as the final RAG response. Such attacks are highly practical and pose a significant threat, particularly within fact-sensitive application areas such as healthcare, finance, and scientific research, where their impact can be detrimental.

Few efforts have attempted to defend against the PoisonedRAG attack [4], [22]. Specifically, [4] proposes the detection of adversarial texts through LLM activations. Although a positive advancement, this method is restricted to white-box access only. With the majority of powerful LLMs being proprietory and only grant black-box access, its applicability maybe limited. In contrast, [22] employs an ensemble approach, using multiple LLMs to generate responses from retrieved texts and then aggregating these into a final answer. However, a significant limitation of this method is its vulnerability to large-scale attacks. If an attacker is capable of injecting a sufficiently high volume of adversarial texts, their desired response is still likely to be produced.

In this work, we introduce an additional *Filtration* component within RAG systems, designed to remove adversarial texts from retrieved context texts. We then propose two distinct approaches for this filtration: FilterRAG and ML-FilterRAG. For FilterRAG, we propose a new property referred to as frequency density (*Freq-Density*) which quantifies the concentration of given words within a text. This property helps to measure how "dense" a context sample is with words relevant to, or shared with the query-answer pair (more on this in section III). Our findings demonstrate that this property enables effective differentiation between adversarial and clean samples within the text retrieved from the knowledge database. By setting appropriate thresholds, we demonstrate how this property can be effectively utilized to filter out adversarial texts, preventing

*Department of Computer Science, The City University of New York - College of Staten Island, Staten Island, NY, USA. (Email: kennedy.edemacu@csi.cuny.edu). Corresponding Author.

†Department of Mathematics and Computer Science, Northern Michigan University, Marquette, MI, USA.

‡Department of Software Engineering, Lappeenranta-Lahti University of Technology, Lappeenranta, Finland.

¶Energy and Bioproducts Research Institute, Aston University, Birmingham, U.K.

§Graduate School of Information, Yonsei University, Seoul, South Korea.

‖Department of Computer Science, Sangmyung University, Seoul, South Korea. (Email: jkim@smu.ac.kr). Corresponding Author.

their inclusion in the context during the generation phase. One potential challenge with FilterRAG is the identification of a suitable threshold value. To overcome this challenge, we propose ML-FilterRAG, in which we incorporate additional features and train a simple machine learning model that can be utilized to filter out adversarial texts. We evaluate the performance of our proposed methods using benchmark datasets and observe that their performance levels are comparable to the established baseline. We summarize our main contributions as follows:

- We discover a distinct concentration of query-answer pair words, differentiating adversarial texts from clean texts in PoisonedRAG, and propose a new property *Freq-Density* to quantify these concentrations.
- We propose two methods, FilterRAG and ML-FilterRAG, to filter out adversarial texts from the retrieved texts.
- We evaluate the effectiveness of these methods against established baselines, observing that their performance levels are comparable.

The rest of the paper is organized as follows. Section II presents the preliminaries. In Section III we present the methodology. Sections IV and V present the experimental setup and the results. Section VII concludes the paper.

II. Preliminaries

A. Threat Model

Attacker's Goal: We maintain the same threat model as PoisonedRAG [5]. We assume that the attacker pre-selects a set of target questions, denoted as $\mathcal{Q} = \{q_1, q_2, \cdots, q_m\}$, and a corresponding set of desired responses $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$. The attacker's goal is to subvert the RAG system so that for each target question $q_i \in \mathcal{Q}$, the system generates the attacker-desired response, $r_i \in \mathcal{R}$. For example, if the target question is, *Which disease is normally caused by the human immunodeficiency virus?* the compromised RAG system would erroneously answer *Syphilis* instead of the correct response *AIDS*.

Attacker's Capabilities: For each target question q_i, we assume that the attacker can inject a set of n poisoned texts denoted as $\mathcal{P} = \{p_i^1, p_i^2, \cdots, p_i^n\}$ directly into the knowledge database $\mathcal{D}$. We then consider both black-box and white-box settings of the RAG retriever. In the black-box setting, we assume that the attacker has no access to the retriever's parameters and cannot directly query it. However, in the white-box setting, the attacker is assumed to have access to the retriever's parameters. The white-box setting is vital especially considering the potential use of publicly available retrievers such as WhereIsAI/UAE-Large-V1 [23], and because it allows evaluations adhering to the well-established Kerchoffs'principle [24].

B. The Knowledge Corruption Attack

The knowledge corruption attack in PoisonedRAG is a constrained problem, in which a set of malicious texts $\tilde{\mathcal{D}} = \{p_i^j | i = 1, 2, \cdots, m, j = 1, 2, \cdots, n\}$ is constructed such that the LLM in the RAG system generates the attacker-desired response $r_i \in \mathcal{R}$ for the target question $q_i \in \mathcal{Q}$ when utilizing k texts retrieved from the compromised knowledge database $\mathcal{D} \cup \tilde{\mathcal{D}}$.

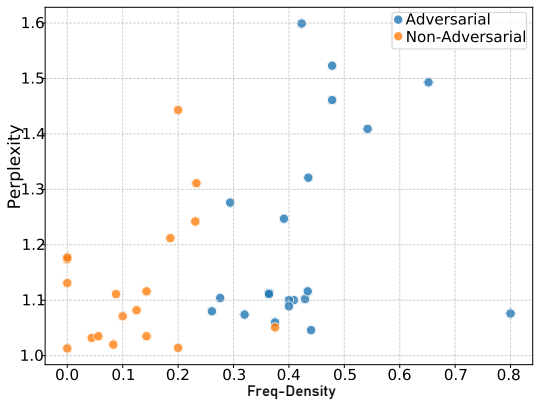

Fig. 1: Pair Plot for Freq-Density vs Perplexity

Formally, the attack is represented as:

$$\max_{\tilde{\mathcal{D}}} \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\left(LLM\left(q_i; \mathcal{E}(q_i; \mathcal{D} \cup \tilde{\mathcal{D}})\right) = r_i\right) \quad (1)$$

where; $\mathbb{I}(.)$ is an indicator function with value 1 if the condition is satisfied and 0 otherwise. $\mathcal{E}(q_i; \mathcal{D} \cup \tilde{\mathcal{D}})$ is the k texts retrieved from the compromised knowledge database $\mathcal{D} \cup \tilde{\mathcal{D}}$.

C. Rationale

Equation 1 shows that a necessary condition for the success of the knowledge corruption attack is that the malicious text $\mathcal{P}$ is in the top-k of the retrieved texts for the target question q_i, i.e., $\mathcal{P} \in \mathcal{E}(q_i; \mathcal{D} \cup \tilde{\mathcal{D}})$. To this end, PoisonedRAG crafts $\mathcal{P}$ to not only rank highly during retrieval, but effectively steer the RAG LLM towards the attacker-desired response r_i. We hypothesize that an analysis of the statistical properties of retrieved texts offers a way to identify the malicious text $\mathcal{P}$. Statistical properties have been used successfully to detect malicious text samples in NLP. For example, [25] introduced a rule-based method that identifies adversarial texts by analyzing discrepancies in word frequencies between adversarial and original texts. Since PoisonedRAG strives to promote adversarial texts to the top-k retrieved samples, we believe that these texts will possess distinct statistical properties compared to legitimate ones. We introduce frequency density (*Freq-Density*) as a statistical property to enable the differentiation of adversarial texts from legitimate ones[1]. When combined with other features, it can offer valuable information on data clusters through visualization. Figure 1 presents the visualization of the frequency density against perplexity (see Appendix D for more visualizations). From this visualization, we can clearly observe distinct clusters of clean and adversarial data points, which confirms our hypothesis.

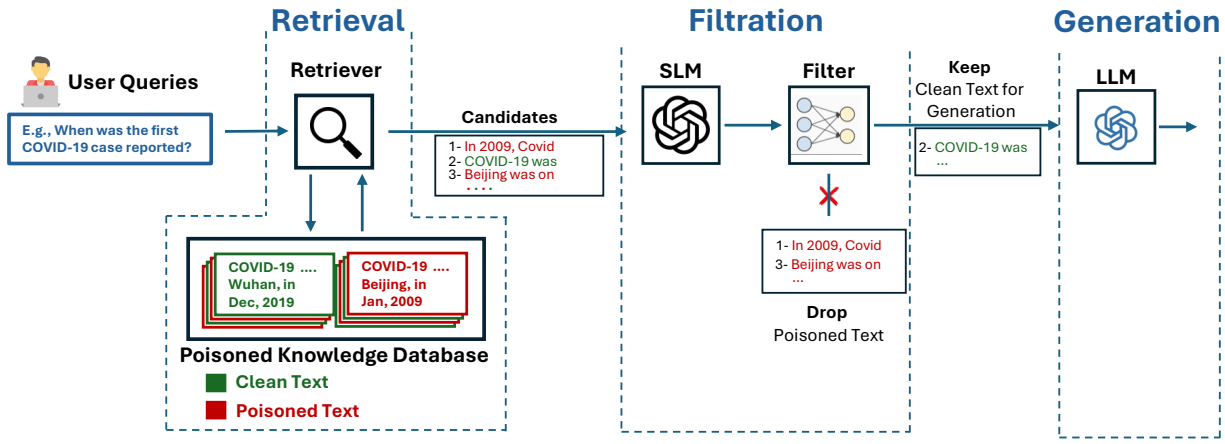

Fig. 2: A high-level illustration of our framework. A filtration phase is integrated into the tradition RAG components to filter-out adversarial texts before the generation phase.

III. Methodology

A. Overview

In this study, we introduce methods to defend against the PoisonedRAG knowledge attack. Our approaches are based on filtering out adversarial texts in the retrieved text from the poisoned knowledge database. Figure 2 presents a high-level illustration of our method. Notably, our method introduces an additional phase, *filtration* that augments the retrieval and generation phases of traditional RAG systems. Our approaches maintain traditional RAG operations for the retrieval and generation phases. However, the key distinction is that during retrieval, both poisoned and clean texts can be returned. In the filtration phase, we use a smaller language model (SLM). The inclusion of SLM in the filtration phase simulates the generative behavior of the LLM in a lighter manner to extract the statistical properties of each retrieved text. We input these properties through a filter to identify and remove adversarial texts. The remaining text, now adversarial-free, can then be combined and leveraged as a context during the generation phase. For practical applications, the filtration phase can be performed on the retrieval side. We present the details of the filtration phase in subsequent sections.

B. Filtration of Adversarial Texts

In this section, we introduce our proposed frameworks for filtering out adversarial texts. The frameworks aim to minimize the influence/presence of adversarial texts within the context used during the generation phase.

Problem Statement: Given a set of target queries $\mathcal{Q} = \{q_1, q_2, \cdots, q_m\}$, a corresponding set of desired responses $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$ and a poisoned knowledge database $\mathcal{D} \cup \tilde{\mathcal{D}}$, we seek to minimize Equation 1 so that the target query $q_i \in \mathcal{Q}$ does not result in its corresponding desired response $r_i \in \mathcal{R}$. Formally represented as:

$$\min_{\tilde{\mathcal{D}}} \quad \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\left(\text{LLM}\left(q_i; \mathcal{E}(q_i; \mathcal{D} \cup \tilde{\mathcal{D}})\right) = r_i\right) \qquad (2)$$

[1]More about this in the next section

subject to $||\tilde{\mathcal{D}}|| \to 0$. By incorporating the latter constraint, we reduce the chances of obtaining the desired response r_i for the target query q_i. Our proposed methods are:

1) FilterRAG (Threshold-Based Approach): This is a two-stage approach based on selecting a threshold value for a statistical property. We summarize the steps in Algorithm 1.

Algorithm 1 FilterRAG (Threshold-Based Approach)

Input: Target Query: q_i, Poisoned Knowledge Database: $\mathcal{D} \cup \tilde{\mathcal{D}}$, Integer: top-s, Float: ϵ, SLM, LLM, Retriever
Output: List of Clean Context Items (top-k for LLM prompting)
Retrieve Candidate Texts:
  $RetrievedItems \leftarrow \text{Retriever}(q_i, \mathcal{D} \cup \tilde{\mathcal{D}}, \text{top} - s)$
Extract Statistical Property (Freq-Density):
for each item d_j in $RetrievedItems$ **do**
  $a_j \leftarrow \text{SLM}(q_i, d_j)$
  $(q_i \oplus a_j) \leftarrow \text{Concatenate}(q_i, a_j)$
  Freq-Density $\leftarrow$ Compute$((q_i \oplus a_j), d_j)$ // Compute(.) computes according to Eq. 4
end for
Filter Adversarial Texts:
$CleanContextItems \leftarrow$ EmptyList()
for each d_j in $RetrievedItems$ **do**
  if Freq-Density$[d_j] < \epsilon$ **then**
    Add d_j to $CleanContextItems$
  else
    Discard d_j
  end if
end for
Return Context:
return $CleanContextItems$

Stage 1 - Statistical Property Extraction: Given a query set $\mathcal{Q} = \{q_1, q_2, \cdots, q_m\}$ and a poisoned knowledge database $\mathcal{D} \cup \tilde{\mathcal{D}}$, we leverage a smaller language model (SLM) to extract the *Freq-Density* statistical property for each retrieved item. In particular, during the retrieval phase, for each query $q_i \in \mathcal{Q}$, we retrieve the top-s items from the poisoned knowledge base

$\mathcal{D} \cup \tilde{\mathcal{D}}$ rather than the traditional top-k items. Then, for each of the top-s retrieved items, d_j in the top-s, we input both the query q_i and d_j into SLM to generate an output a_j.

$$a_j = \text{SLM}(q_i, d_j) \quad \text{for } j \in \{1, 2, \ldots, s\} \tag{3}$$

Next, we define our *Freq-Density* as follows:

$$\text{Freq-Density} = \frac{\sum_{w \in (q_i \oplus a_j) \cap d_j} \text{Freq}(w, d_j)}{\text{UniqueWords}(d_j)} \tag{4}$$

where, $\oplus$ means concatenation, $(q_i \oplus a_j) \cap d_j$ are semantically similar words common to $(q_i \oplus a_j)$ and d_j, i.e., word pairs in $(q_i \oplus a_j)$ and d_j whose computed similarities exceed a predetermined threshold, $\text{Freq}(w, d_j)$ denotes the frequency of the word w within the text d_j and $\text{UniqueWords}(d_j)$ denotes the total number of unique words in d_j. The rationale behind this metric is rooted in the attackers' strategy. To both steer the LLM to their desired response r_i for the target query q_i, and to ensure high retrieval relevance, attackers populate their adversarial texts with words that are statistically similar to the target queries and their corresponding desired responses. This overlap makes *Freq-Density* a valuable indicator for identifying malicious texts. Figure 1 presents an empirical illustration of this claim.

Stage 2 - Removal of Adversarial Texts: We then employ the defined *Freq-Density* to identify adversarial texts for removal by setting a threshold ϵ. We define a filter indicator function (Equation 5) to determine whether a retrieved text d_j is adversarial.

$$\text{Filter}(d_j) = \begin{cases} 1, & \text{if Freq-Density} < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

We feed each item in the top-s through the filter function. Next, we remove the items whose return value from the filter function is 0. From the remaining items, the top-k items which constitutes the context used to prompt the global LLM are chosen. The effectiveness of this method depends on the choice of ϵ. This parameter is crucial for balancing the trade-off between the overall performance and the presence of adversarial texts in the final top-k items.

2) ML-FilterRAG (Machine Learning Approach): The effectiveness of our first approach hinges on the choice of ϵ. A high value of ϵ risks allowing adversarial text samples to appear in the final top-k items. Meanwhile, a low value could filter out legitimately clean texts. Striking the right balance is very crucial and yet challenging. Inspired by the work in [5], which used perplexity to attempt to filter out adversarial texts, we believe that combining multiple features can effectively address the challenge of ϵ selection. Thus, we propose ML-FilterRAG, a machine learning-based filtering approach. In this method, a filtering machine learning model takes multiple features extracted from each retrieved text d_j as input and predicts whether d_j is an adversarial sample. Similarly, this is a two-stage method. We summarize the steps in Algorithm 2.

Stage 1 - Feature Extraction: Similar to the extraction method discussed in III-B1, we employ an SLM that takes the target query q_i and a retrieved text d_j as input and subsequently outputs a_j. However, on top of the *Freq-Density*, we also compute perplexity, joint log probability of the SLM's output a_j and sum of frequencies of semantically similar words between $(q_i \oplus a_j)$ and d_j. We incorporate these as supplementary features for our filter model.

Algorithm 2 ML-FilterRAG (Machine Learning Approach)

Input: Target Query: q_i, Poisoned Knowledge Database: $\mathcal{D} \cup \tilde{\mathcal{D}}$, Trained Machine Learning Model: $\mathcal{M}$, top-s, SLM, LLM, Retriever
Output: List of Clean Context Items (top-k for prompting LLM)
Retrieve Candidate Texts:
  $RetrievedItems \leftarrow \text{Retriever}(q_i, \mathcal{D} \cup \tilde{\mathcal{D}}, \text{top} - s)$
Extract Features:
for each item d_j in $RetrievedItems$ **do**
  $a_j \leftarrow \text{SLM}(q_i, d_j)$
  $features[d_j] \leftarrow \text{Feature}(q_i, d_j)$
end for
Predict Adversarial Texts:
$CleanContextItems \leftarrow \text{EmptyList}()$
for each d_j in $RetrievedItems$ **do**
  $Predict \leftarrow \mathcal{M}(features[d_j])$
  if $Predict$ is "non-adversarial" **then**
    Add d_j to $CleanContextItems$
  else
    Discard d_j
  end if
end for
Return Context:
return $CleanContextItems$

Stage 2 - Adversarial Text Prediction: Given a lightweight trained machine learning-based filter model $\mathcal{M}$, each of the retrieved items in the the top-s undergoes the feature extraction process discussed previously. Once these feature values are extracted, they are fed into $\mathcal{M}$ (according to Equation 6) to predict whether the item d_j is an adversarial text.

$$pred_j = \mathcal{M}(\text{Feature}((q_i, d_j)) \tag{6}$$

where, $pred_j$ can be adversarial or non-adversarial, and Feature(.) is a function that extracts features. Finally, the top-k items are chosen from the non-adversarial items as context to prompt the LLM. By generating features that capture semantic similarities, we believe that our approaches offer an expanded scope for identifying adversarial texts. To train $\mathcal{M}$, we use the supervised learning approach in which the labeled datasets of retrieved texts for each target query are annotated as adversarial and non-adversarial. For each training instance, we extract features using the method discussed above. These features capture the statistical cues indicative of adversarial texts. $\mathcal{M}$ is trained to learn the mappings between these features and their corresponding labels. Once trained, $\mathcal{M}$ serves as an effective filter that predicts if d_j is adversarial based on its computed features.

IV. Experiments

A. Datasets

For our experiments, we utilize three benchmark question-answering datasets: *MS-MARCO* [26], *Natural Questions (NQ)* [27] and *HotpotQA* [28]. Each of these original datasets contains a distinct knowledge database. Specifically, NQ contains 2,681,468 texts, HotpotQA contains 5,233,329 texts, and MS-MARCO contains 8,841,823 texts within their respective knowledge databases. In addition, each dataset provides a set of queries and answers. To facilitate the identification of adversarial texts, we leverage the version of these datasets detailed in [5]. In this version, 100 closed-ended questions from each dataset are designated as target questions, each paired with a ground-truth answer and an attacker-desired answer. Furthermore, for each target question, 5 adversarial texts are injected into the original knowledge database.

To evaluate our ML-FilterRAG approach, we had GPT-4o generate an additional 5 adversarial texts for each target question, mimicking the characteristics of the original five. We carefully examined each of these newly generated texts to ensure that they meet two crucial criteria: they successfully output the attacker-desired response, and they rank highly among the retrieved texts from the knowledge base. Specifically, we selected those that appeared within the top 15 retrieved results. We then randomly selected five adversarial texts, along with an equivalent number of randomly selected original (clean) texts, to train our machine learning-based filter models. We keep the rest for evaluating our proposed framework. For the white-box attack, we adopt the HotFlip method as in [5], with all the other settings remaining consistent as for the black-box attack.

B. RAG Settings

1) Retriever: We employ the Contriever [15] as our retriever. Following [5], [6], and unless otherwise stated, we employ the dot product to measure similarities between queries and texts within the knowledge database.

2) SLM: We leverage two language models as our SLM: LLaMA-3 [29] and LLaMA-2 [30]. For our first filtration approach, we used a default ϵ value of 0.2. In contrast, for the machine learning-based approach, we defined and trained a dedicated model for each dataset. Specifically, we employ an XGBoost model for the NQ dataset, while Random Forest models were utilized for both the HotpotQA and MS-MARCO datasets. The training details of the models are presented in the Appendix C. For semantic word similarity matching, we employ a huggingface sentence transformer[2] and set a default similarity threshold value of 0.6 for cosine similarity. We will also study the impact of the similarity threshold and ϵ in our evaluations.

3) LLM: For the generation LLM, we consider a family of GPT models, GPT-3.5 [1], GPT-4 [2] and GPT-4o [31], and LLaMA-3. We adopt the system prompt discussed in [5] and we present it in Appendix B. We use a temperature value of 0.1 for all the models.

[2]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

C. Metrics

1) Adversarial Text Ratio (ATR): We introduce a new performance metric, *adversarial text ratio (ATR)* to quantify the fraction of adversarial texts within the retrieved top-k texts. This metric is particularly relevant given our objective of minimizing Equation 2, an objective directly influenced by the number of adversarial texts. Therefore, ATR is crucial for accurately assessing the effectiveness of our proposed methods.

2) Attack Success Rate (ASR): We also consider ASR as discussed in [5]. ASR quantifies the fraction of target questions whose answers match the attacker-desired answers. Substring matching between the attacker desired answer and the LLM generated answer is used to determine if the two answers are the same. In other words, the two answers do not need to match exactly.

3) Accuracy: Accuracy measures the fraction of target questions whose ground-truth answers match the answers generated by the LLM. In a similar manner, we adopt the substring matching approach to determine a match between the two answers. We expect higher accuracy values for frameworks with minimal number of adversarial texts in the retrieved top-k texts and vice versa.

D. Baselines

1) PoisonedRAG: We compare our framework with PoisonedRAG [5]. In doing so, our aim is to assess whether our proposed approaches minimize the influence of adversarial texts injected into the knowledge database through the PoisonedRAG attack.

2) CleanRAG: Similarly, we consider comparing our methods with the traditional RAG with no adversarial texts within its knowledge database. We hope to achieve performance similar or close to the CleanRAG framework.

E. Other Experimental Parameters

To evaluate our methods within a poisoned knowledge database setting, we set the top-s retrieval to 4 and intentionally retrieve an equal number of adversarial and clean texts. Then, we set the top-k of 2. We use a default value of 10 for m. We repeat each experiment 10 times so that all target questions are examined.

V. Experimental Results

A. Main Results

To explore whether our proposed methods can filter out adversarial texts from the retrieved texts with no significant negative impacts on performance, we conduct experiments on the three datasets: HotpotQA, MS-MARCO and NQ datasets under white-box and black-box PoisonedRAG attacks. The results are presented in Table I. From the experimental results, we make the following observations. First, our proposed approaches, FilterRAG and ML-FilterRAG consistently achieve higher accuracies in the presence of adversarial texts. For example, FilterRAG can achieve accuracies of 88.1%, 82.0%, and 75.6% with GPT-3.5 on HotpotQA, MS-MARCO, and NQ

TABLE I: Performance Comparison of RAG Frameworks Across Different LLMs for Black-box Attack

Dataset	Framework	ATR ↓	GPT-3.5		GPT-4		GPT-4o		LLaMA-3	
			ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑
HotpotQA	CleanRAG	**0.000**	**0.080**	**0.913**	**0.090**	**0.908**	**0.090**	**0.907**	**0.310**	**0.665**
	PoisonedRAG	1.000	0.940	0.042	0.900	0.000	0.930	0.000	0.980	0.001
	FilterRAG ($\epsilon = 0.2$)	**0.000**	0.082	0.881	**0.090**	0.870	0.091	0.820	0.380	0.536
	ML-FilterRAG	0.015	0.090	0.903	0.091	0.905	0.094	0.899	0.330	0.541
MS-MARCO	CleanRAG	**0.000**	**0.060**	**0.859**	**0.050**	**0.851**	**0.050**	**0.867**	**0.310**	**0.667**
	PoisonedRAG	0.825	0.820	0.160	0.824	0.173	0.834	0.128	0.912	0.082
	FilterRAG ($\epsilon = 0.2$)	0.065	0.090	0.820	0.090	0.840	0.080	0.840	0.430	0.509
	ML-FilterRAG	0.045	**0.060**	0.851	0.060	0.849	0.080	0.831	0.400	0.565
NQ	CleanRAG	**0.000**	**0.010**	**0.831**	**0.020**	**0.833**	**0.020**	**0.831**	**0.140**	**0.575**
	PoisonedRAG	0.980	0.880	0.108	0.870	0.119	0.864	0.118	0.980	0.100
	FilterRAG ($\epsilon = 0.2$)	0.010	0.030	0.756	0.030	0.818	0.030	0.773	0.190	0.480
	ML-FilterRAG	0.030	0.020	0.811	0.030	0.810	0.030	0.803	0.180	0.544

datasets, respectively. While, ML-FilterRAG achieves accuracies of 90.3%, 85.1% and 81.1% with GPT-3.5 on HotpotQQ, MS-MARCO, and NQ datasets, respectively.

We also observe that our proposed FilterRAG and ML-FilterRAG methods achieve some of the lowest ASRs in comparison to the baselines. For example, FilterRAG has ASRs of 8.2%, 9.0%, and 3.0% with GPT-3.5 on HotpotQA, MS-MARCO, and NQ datasets, respectively. Meanwhile, ML-FilterRAG achieves ASRs of 9.00%, 6.0%, and 2.0% on HotpotQA, MS-MARCO, and NQ datasets, respectively.

Similarly, FilterRAG and ML-FilterRAG achieve some of the lowest ATRs. For example, FilterRAG has ATR of 0.0%, 6.5%, and 1.0% with HotpotQA, MS-MARCO, and NQ datasets, respectively. While ML-FilterRAG has ATR of only 1.5%, 4.5%, and 3.0% with HotpotQA, MS-MARCO, and NQ datasets, respectively. More results for white-box attack settings are shown in Appendix A.

Although our methods are able to achieve lower ATRs, they cannot exceed the CleanRAG baseline in terms of accuracy. This is mainly because some clean texts get misclassified as adversarial texts. However, this misclassification is more common with FilterRAG than with ML-FilterRAG. Setting the right ϵ-value is crucial for the success of FilterRAG. However, this is fairly difficult to achieve. It is a trade-off between ATR and accuracy. But both of our proposed methods show significant improvements compared to the PoisonedRAG baseline, and they achieve performance that closely match the original RAG.

B. Ablation Studies

We also performed ablation studies and present the results as follows.

1) Varying Similarity Threshold: The similarity threshold determines the level of exactness between the query-answer combination and the context text words. This exactness is crucial for both the FilterRAG and ML-FilterRAG methods. As a result, we perform experiments that demonstrate how this exactness affects our proposed methods. We present the results in Table II. From the results, we observe that at the similarity threshold of 1.0, FilterRAG and ML-FilterRAG both have high ATR and ASR, and low accuracies. As the similarity threshold decreases from 0.9 to 0.6, both FilterRAG and ML-FilterRAG show decreasing ATR and ASR, while their accuracies increase. Beyond the similarity threshold value of 0.6, FilterRAG shows decreasing ATR, ASR and accuracy values, while ML-FilterRAG shows increasing ATR and ASR values, but decreasing accuracy values. This result demonstrates that, at highest similarity threshold of 1.0, only query-answer combination words that perfectly match the context text words are considered. This level of perfection can allow attackers to evade our proposed methods simply using modified word versions or synonyms and allow more adversarial texts to go undetected by our methods. A reason our proposed methods show high ATR and ASR values, and low accuracy at similarity threshold of 1.0. As we relax the level of exactness by reducing the similarity threshold, the match between query-answer combination and context text words is now measured at semantic level. This is reflected by the decrease in both ATR and ASR values and the increase in accuracy values for similarity thresholds of 0.9 to 0.6. However, further decrease in similarity threshold values beyond 0.6, blurs the boundary between legitimate and adversarial texts and results in poor performance for both FilterRAG and ML-FilterRAG.

2) Effect of SLM: We also investigate the performance of our proposed methods with two SLMs, LLaMA-2 and LLaMA-3. The results are presented in Table III. From the result, we observe that with LLaMA-2 SLM, FilterRAG has a better ASR value compared to ML-FilterRAG. However, ML-FilterRAG has better accuracy. With LLaMA-3 as SLM, we observe that ML-FilterRAG performs better on all metrics. This result generally shows that using a large model as an SLM achieves a better performance. This makes sense as a more extensive SLM possesses the ability to generate precise and clear results based on the provided context, thereby enabling more accurate matching between query-answer combination and the context text.

3) Effect of ϵ-value: The performance of FilterRAG depends on the choice of ϵ. We demonstrate this through experiments. We present the experimental results for the MS-MARCO dataset in Table IV. From the result, we observe that at the lowest ϵ value of 0.1, FilterRAG achieves its lowest ATR and ASR, however the accuracy is low, at only 62.4%. As ϵ-value increases to 0.2, FilterRAG achieves the best performance at an accuracy of 84.0%. However, its

TABLE II: Performances with Varying Similarity Threshold Values for MS-MARCO Dataset using GPT-4

Method		Similarity Threshold 1.0	0.9	0.8	0.7	0.6	0.5	0.4
FilterRAG	**ATR** ↓	0.415	0.080	0.070	0.065	0.065	0.020	**0.000**
	ASR ↓	0.260	0.080	0.090	0.090	0.090	0.070	**0.050**
	Accuracy ↑	0.621	0.827	0.831	0.833	**0.840**	0.766	0.497
ML-FilterRAG	**ATR** ↓	0.425	0.090	0.080	0.060	**0.045**	0.195	0.480
	ASR ↓	0.290	0.090	0.086	0.070	**0.060**	0.120	0.230
	Accuracy ↑	0.589	0.834	0.834	0.844	**0.849**	0.793	0.538

TABLE III: Performances with Varying SLMs for MS-MARCO Dataset with GPT-4

Method	LLaMA-2			LLaMA-3		
	ATR ↓	**ASR** ↓	**Accuracy** ↑	**ATR** ↓	**ASR** ↓	**Accuracy** ↑
FilterRAG	**0.215**	**0.15**	0.786	0.065	0.090	0.840
ML-FilterRAG	**0.215**	0.16	**0.791**	**0.045**	**0.060**	**0.849**

TABLE IV: FilterRAG Performance with respect to ϵ-Values for MS-MARCO Dataset with GPT-4

	ϵ-value 0.1	0.2	0.3	0.4	0.5
ATR ↓	**0.005**	0.065	0.230	0.540	0.765
ASR ↓	**0.050**	0.090	0.200	0.320	0.540
Accuracy ↑	0.624	**0.840**	0.764	0.462	0.280

ATR and ASR begin to drop. Further increase in ϵ-values from 2.0 results in more increase in ATR and ASR values with decreasing accuracy. At lower ϵ-values, the majority of adversarial texts from top-s are filtered out. However, some legitimate texts are mistakenly filtered out too. This is why the ATR and ASR are lower, but the accuracy is relatively low as well. The accuracy begins to increase with the increasing ϵ-values because more legitimate texts are being captured in the top-k. But, this affects both ATR and ASR, as more more adversarial texts sneak into the top-k. This becomes more evident at higher ϵ-values. This result emphasizes that choosing an ϵ-value is a trade-off between minimizing the number of adversarial texts in the retrieved top-s and achieving better performance.

VI. Other Related Works

A. RAG Knowledge Poisoning Attacks

In other knowledge poisoning attacks, [32] *et al*. introduced GARAG. GARAG corrupts the knowledge database through low-level perturbation to original documents. Zhong *et al*. [33] proposed PRCAP, which introduces adversarial texts into the knowledge database. The introduced texts are generated by perturbing discrete tokens that enhance their similarity to target queries. Gong *et al*. [19] proposed Topic-fliprag attack. Topic-fliprag performs adversarial perturbations on original documents to influence opinions across target queries. Nazary *et al.* [20] proposed a stealthy knowledge poisoning attack for RAG-based recommender systems. Their attack performs adversarial perturbations to item descriptions to either promote or demote the item. These methods have further demonstrated the vulnerability of RAGs to knowledge positioning attacks.

B. Defense Against Knowledge Poisoning

Few works show potential in protecting against knowledge poisoning attacks. Tan *et al*. [4] propose the use of LLMs' activations to detect adversarial texts. However, this method is limited to only white-box access. Xiang *et al*. [22] uses an ensemble approach with multiple LLMs to generate responses and then aggregate them into the final RAG answer. Although suitable for black-box access, it is still vulnerable to large-scale attacks. For example, if an attacker injects a high volume of adversarial texts, their desired response is still likely to be produced. Zou *et al.* [5] suggested several methods such as paraphrasing, use of perplexity, duplicate text filtering and knowledge to defend against PoisonedRAG. However, all these methods are not robust enough. Our work uses a combination of statistical properties to identify adversarial texts and defend against PoisonedRAG in black-box setting.

VII. Conclusion

In this work, we propose a statistical property and establish that the clean and adversarial texts in PoisonedRAG exhibit distinct values for our proposed property. We then propose two defense methods: FilterRAG and ML-FilterRAG that employ the above insight to defend against PoisonedRAG when retrieving texts from RAG's knowledge database. Our methods demonstrate robustness under various LLMs. Experimental results show that our proposed methods achieve tremendous performance with a performance gap difference of merely up to 0.2% compared to the original RAG. In general, our methods can defend robustly against PoisonedRAG in the black-box setting.

Limitations and Future Works: The following are the limitations of our work.

- Our work focuses primarily on addressing the PoisonedRAG knowledge poisoning attack. It is not tested against other forms of RAG knowledge poisoning attacks such as emotional-based and topic-flipping attacks. Future explorations are required to investigate how it works against the mentioned attacks.
- For the ML-FilterRAG method, we only use simple machine learning models to detect adversarial texts. In

the future, the use of more extensive machine learning models can be explored.

- Our proposed methods add a component (filtration) to the traditional RAG. This introduces an additional computation demand. The analysis of this computational requirement and its solutions can be explored in the future.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[4] X. Tan, H. Luan, M. Luo, X. Sun, P. Chen, and J. Dai, "Knowledge database or poison base? detecting rag poisoning attack through llm activations," *arXiv preprint arXiv:2411.18948*, 2024.

[5] W. Zou, R. Geng, B. Wang, and J. Jia, "Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models," *arXiv preprint arXiv:2402.07867*, 2024.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[7] V. Karpukhin, B. Oguz, S. Min, P. S. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering." in *EMNLP (1)*, 2020, pp. 6769–6781.

[8] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*. PMLR, 2022, pp. 2206–2240.

[9] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.

[10] S. J. Semnani, V. Z. Yao, H. C. Zhang, and M. S. Lam, "Wikichat: Stopping the hallucination of large language model chatbots by few-shot grounding on wikipedia," *arXiv preprint arXiv:2305.14292*, 2023.

[11] A. Lozano, S. L. Fleming, C.-C. Chiang, and N. Shah, "Clinfo. ai: An open-source retrieval-augmented large language model system for answering medical questions using scientific literature," in *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2024*. World Scientific, 2023, pp. 8–23.

[12] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.

[13] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[14] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-rag: Learning to retrieve, generate, and critique through self-reflection," in *The Twelfth International Conference on Learning Representations*, 2023.

[15] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, "Unsupervised dense information retrieval with contrastive learning," *arXiv preprint arXiv:2112.09118*, 2021.

[16] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk, "Approximate nearest neighbor negative contrastive learning for dense text retrieval," *arXiv preprint arXiv:2007.00808*, 2020.

[17] Z. Peng, X. Wu, Q. Wang, and Y. Fang, "Soft prompt tuning for augmenting dense retrieval with large language models," *Knowledge-Based Systems*, vol. 309, p. 112758, 2025.

[18] N. Kassner and H. Schütze, "Bert-knn: Adding a knn search component to pretrained language models for better qa," *arXiv preprint arXiv:2005.00766*, 2020.

[19] Y. Gong, Z. Chen, M. Chen, F. Yu, W. Lu, X. Wang, X. Liu, and J. Liu, "Topic-fliprag: Topic-orientated adversarial opinion manipulation attacks to retrieval-augmented generation models," *arXiv preprint arXiv:2502.01386*, 2025.

[20] F. Nazary, Y. Deldjoo, T. Di Noia, and E. Di Sciascio, "Stealthy llm-driven data poisoning attacks against embedding-based retrieval-augmented recommender systems," in *Adjunct Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization*, 2025, pp. 98–102.

[21] Z. Wei, W.-L. Chen, and Y. Meng, "Instructrag: Instructing retrieval-augmented generation with explicit denoising," *arXiv e-prints*, pp. arXiv–2406, 2024.

[22] C. Xiang, T. Wu, Z. Zhong, D. Wagner, D. Chen, and P. Mittal, "Certifiably robust rag against retrieval corruption," *arXiv preprint arXiv:2405.15556*, 2024.

[23] X. Li and J. Li, "Angle-optimized text embeddings," *arXiv preprint arXiv:2309.12871*, 2023.

[24] F. Petitcolas, "La cryptographie militaire," *J. des Sci. Militaires*, vol. 9, pp. 161–191, 1883.

[25] M. Mozes, P. Stenetorp, B. Kleinberg, and L. D. Griffin, "Frequency-guided word substitutions for detecting textual adversarial examples," *arXiv preprint arXiv:2004.05887*, 2020.

[26] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, "Ms marco: A human generated machine reading comprehension dataset," *choice*, vol. 2640, p. 660, 2016.

[27] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[28] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 2369–2380.

[29] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[30] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[31] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[32] S. Cho, S. Jeong, J. Seo, T. Hwang, and J. C. Park, "Typos that broke the rag's back: Genetic attack on rag pipeline by simulating documents in the wild via low-level perturbations," *arXiv preprint arXiv:2404.13948*, 2024.

[33] Z. Zhong, Z. Huang, A. Wettig, and D. Chen, "Poisoning retrieval corpora by injecting adversarial passages," *arXiv preprint arXiv:2310.19156*, 2023.

APPENDIX A
PERFORMANCE FOR WHITE-BOX ATTACKS

Further results of our proposed methods for white-box attack are shown in Table V.

APPENDIX B
PROMPT TEMPLATE

You are a helpful assistant. The user has provided a query along with relevant context information. Use this context to answer the question briefly and clearly. If you cannot find the answer to the question, respond with "I don't know."
Contexts: [context]
Query: [question]
Answer:

Fig. 3: The prompt template used to produce the results

TABLE V: Performance Comparison of RAG Frameworks Across Different LLMs for White-box Attack

Dataset	Framework	ATR ↓	GPT-3.5		GPT-4		GPT-4o		LLaMA-3	
			ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑	ASR ↓	Accuracy ↑
HotpotQA	CleanRAG	**0.000**	0.080	**0.913**	**0.090**	**0.908**	0.090	**0.907**	**0.310**	**0.665**
	PoisonedRAG	1.000	0.860	0.056	0.870	0.022	0.872	0.021	0.980	0.001
	FilterRAG ($\epsilon = 0.2$)	**0.000**	**0.005**	0.721	**0.090**	0.733	0.090	0.731	0.335	0.600
	ML-FilterRAG	0.030	0.090	0.729	**0.090**	0.745	**0.080**	0.755	0.330	0.650
MS-MARCO	CleanRAG	**0.000**	**0.060**	**0.859**	**0.050**	**0.851**	**0.050**	**0.867**	**0.310**	**0.667**
	PoisonedRAG	0.980	0.720	0.074	0.710	0.107	0.750	0.102	0.960	0.030
	FilterRAG ($\epsilon = 0.2$)	0.125	0.120	0.721	0.110	0.729	0.110	0.706	0.320	0.596
	ML-FilterRAG	0.155	0.110	0.762	0.110	0.777	0.112	0.782	0.318	0.629
NQ	CleanRAG	**0.000**	**0.010**	**0.831**	**0.020**	**0.833**	**0.020**	**0.831**	**0.140**	**0.575**
	PoisonedRAG	1.000	0.730	0.031	0.700	0.033	0.720	0.046	0.970	0.010
	FilterRAG ($\epsilon = 0.2$)	0.025	0.050	0.823	0.050	0.816	0.050	0.828	0.160	0.500
	ML-FilterRAG	0.040	0.060	0.824	0.050	0.827	0.050	0.829	0.170	0.542

APPENDIX C
ML-FILTERRAG MACHINE LEARNING MODELS

TABLE VI: ML-FilterRAG Model Training Results

Dataset	Model	Training Accuracy	Test Accuracy
MS-MARCO	Random Forest	0.860	0.850
NQ	XGBoost	0.990	0.970
HotpotQA	Random Forest	0.990	0.982

APPENDIX D
FEATURE PAIR PLOTS

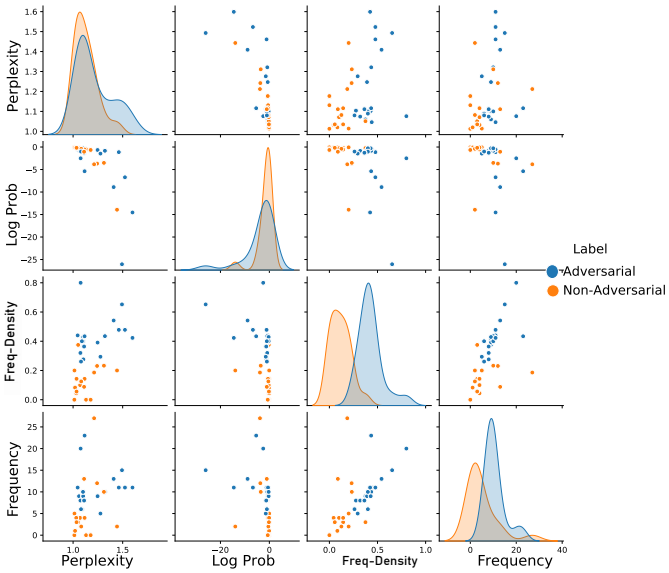

Fig. 4: Pair plots for all the features