Информационные процессы, Том 25, № 4, 2025, стр. 807–820.
© 2025 Jaiswal.

Evaluating RAG System Models Robustness to Hybrid Homoglyph and Emoji-Based Next-Generation Adversarial Prompt Injection[1]

A. K. Jaiswal

Department of Radio Engineering and Cybernetics
Moscow Institute of Physics and Technology (MIPT)
Institutsky Lane 9, Dolgoprudny, Moscow Region 141701, Russia
`dzhaisval.a@phystech.edu`

Received November 15, 2025. Accepted December 12, 2025.

Abstract—Recent advances in retrieval-augmented generation (RAG) have enabled large language models (LLMs) to reliably answer user queries by retrieving relevant documents. However, adversaries may exploit obfuscation techniques—such as homoglyph substitution, emoji injection, or invisible Unicode characters—to hide attack payloads and induce leakage of sensitive information. In this work, we systematically evaluate five RAG models running inside Docker Desktop containers under a unified pipeline designed to test for such leaks. We have used pdf, text, html and readme file format as a document loader to the RAG model and then we used FAISS vector database for index vectorization system. We generate adversarial prompts against synthetic secrets using templated generators, varied obfuscation types, and measure metrics including adversarial leak count, pass vs block rates, benign leak frequencies, and the overall leak rate. Our experiments show a high leak rate: in one aggregated run over 5 models, approximately 68.3% of adversarial prompts result in leaks under our threat model. We find that combined obfuscations (e.g. homoglyph + emojis) are especially effective. We further provide per-model comparisons, analyze failure modalities, and suggest practical mitigations including input normalization and detection. We release all artifacts—including Docker separate containerization of the models, prompt generation scripts, secret lists, and evaluation scripts—to support reproducibility and artifact evaluation. Our findings raise important implications for RAG deployment in privacy/security-sensitive applications.

Keywords: RAG, adversarial prompts, homoglyphs, emoji injection, LLM leakage, model robustness

DOI: 10.53921/18195822_2025_25_4_807

1. INTRODUCTION

Retrieval-Augmented Generation (RAG) systems have become popular for improving large language model (LLM) performance in knowledge-intensive tasks by retrieving relevant documents and grounding responses. However, this retrieval + generation pipeline opens up new attack surfaces. Recent works have shown that RAG models can leak information from the datastore or system prompt under adversarial stimuli. For example, *Follow My Instruction and Spill the Beans* demonstrates that RAG systems can be induced via prompt injection to reproduce datastore text verbatim.

[1] This work was not supported by any organization

In realistic deployment, adversaries may obfuscate their prompts (e.g., via homoglyphs, invisible unicode, or emoji insertion) to evade filters and detection. Yet there is limited empirical work quantifying how such obfuscations affect leak rates across multiple models.

In this paper, we investigate the following research questions:

– How often do RAG models leak synthetic secrets when adversarial prompts are obfuscated?
– What is the impact of different obfuscation types (homoglyph, invisible unicode, emoji, and combinations) on leak frequency?
– How do different models compare under the same pipeline, and what failure modes emerge?
– What mitigation strategies (e.g., input normalization, detection) are effective or feasible?

Our contributions are:

1. We build a reproducible Dockerized pipeline for evaluating obfuscated adversarial prompts across five RAG models, using synthetic secret tokens.
2. We design and apply several obfuscation techniques (homoglyph substitution, invisible unicode, emoji injection) under configurable parameters, and measure key metrics: adversarial passes, blocked prompts, leaks, benign leaks, and leak rate.
3. We provide per-model comparison and in-depth analysis, showing that combined obfuscation greatly increases leak risk, and show the trade-offs in detection.
4. We release all artifacts (Docker images, code, queries, secret lists) to support reproducibility, and propose practical defenses based on input normalization and detection.

The rest of the paper is organized as follows. Section 1 defines the threat model and related work. Section 2 describes our system and experimental setup. Section 3 details the adversarial query generation process. Section 4 reports the evaluation results. Section 5 discusses limitations and failure modes. Section 6 proposes mitigations. Finally, Section 7 concludes and suggests future directions.

2. STATE-OF-THE-ART ANALYSIS IN RAG ADVERSARIAL ATTACKS

Recent studies on adversarial attacks on Retrieval-Augmented Generation (RAG) models highlight growing sophistication in methods that exploit language model flaws. One of the most studied techniques is prompt injection, where carefully crafted prompts can lead models to disclose private data or behave unexpectedly. Defenses mainly focus on prompt filtering and template detection.

Attacks that rely on obfuscating characters postulate that only a few changes are necessary to bypass standard filtering and tokenization-based defenses. This includes several character-level obfuscation attack methods, such as homoglyph substitution, insertion of invisible characters, and injection of emojis. Defenses that are based on just filtering additional input strings do not work against homoglyphs or invisible characters, and emoji injections simply cause alignment issues with the training data, increasing the data leaking potential.

Most alarming is the emergence of a hybrid obfuscation attack that utilizes more than one character-level obfuscation attack strategy. The deficit of single strategy defenses is apparent, considering the hybrid attacks achieved the highest leak rates across the models while maintaining strong efficacy.

Current state-of-art is effective defense requires layering strategies that include strong filtering prompts, undetectable anomaly detection, and input normalization to fullest effect. In short, we have seen a transition from the harder-to-defend prompt injections that are simple strings, to more complex obfuscation escapes that interplay multiple dimensional layers. Meaningful defense now also is more complex.

Table 1. Comparative Analysis of State-of-the-Art RAG Adversarial Vectors

Attack Vector	**Mechanism Description**	**Key Refs**	**Observed Impact**	**Defense Strategies**
Prompt Injection	Direct manipulation of context to override system instructions (e.g., "Ignore previous rules").	[1], [2], [22]	Extraction of system prompts; forcing the model to ignore retrieved context.	Instruction hierarchy enforcement; Perplexity-based filtering.
Homoglyph Obfuscation	Substitution of Latin characters with visually identical Unicode look-alikes (e.g., Cyrillic 'a').	[3], [27]	Bypasses string-matching filters; degrades retrieval relevance by altering token IDs.	Strict Unicode Normalization (NFKC); Visual-similarity character mapping.
Invisible Characters	Injection of zero-width joiners (ZWJ) or non-printable tokens to disrupt tokenizer segmentation.	[6], [24]	Splits malicious keywords into harmless subtokens, evading blocklists.	Invisible character stripping; Tokenization sanity checks.
Emoji Injection	Embedding semantic or non-semantic emojis within keywords to shift embedding vector space.	[4], [5]	Causes "embedding drift," retrieving irrelevant documents or bypassing safety alignment.	Emoji stripping/normalization; Anomaly detection in embedding space.
RAG Poisoning	Injecting malicious documents into the retrieval corpus to influence generation.	[23], [11]	High-confidence hallucination; model adopts attacker-controlled worldview.	Robust indexing verification; "Canary" document monitoring.
Hybrid Obfuscation	*Simultaneous use of homoglyphs, emojis, and invisible characters (This Study).*	–	**Highest Leak Rate (68%+);** Compounding failure of tokenizers and semantic filters.	**Multi-modal defense pipeline.**

3. THREAT MODEL AND RELATED WORK

3.1. Threat Model

We assume an adversary who can submit prompts (queries) to our RAG pipeline, with full control over the prompt text. The adversary is allowed to use obfuscation techniques: homoglyph substitution, invisible Unicode characters, and emoji insertion. They may also combine these. The adversary does not have internal model access (no weight tuning, no access to training data), nor can they modify the retrieval corpus or embedding model. They only observe the responses from the system. A leak is any response that reveals a synthetic secret token (from a predefined secret list), either exactly or via high similarity (using a string similarity metric, e.g., SequenceMatcher threshold 0.8).

3.2. Related Work

Prompt Injection / Leaking: As summarized in Table 1, there is a growing body of work showing how prompts can be crafted (or manipulated by users) to coax LLMs into revealing undesirable content or secrets. Works like *An Early Categorization of Prompt Injection Attacks* and *SoK: Prompt Hacking of Large Language Models* analyze different attack vectors and defences [1,2].

Obfuscation Techniques: As shown in Table 1, homoglyphs have been used to evade detection in text classification and content moderation tasks. Recent studies like *Evading AI-Generated*

Content Detectors using Homoglyphs illustrate how visually similar Unicode characters bypass filters [3,5].

Emoji / Invisible Characters: Table 1 also highlights that while fewer formal papers exist, some emerging reports show that invisible Unicode and emojis can disrupt tokenization or filtering pipelines, causing unexpected behaviour [6,7].

RAG / Retrieval-Augmented QA Vulnerabilities: As indicated in Table 1, some recent studies examine how retrieval components or poisoning attacks degrade RAG systems (e.g., universal poisoning of retrieval corpora) or how misalignment between retrieval and generation causes leakage or hallucination [8,10].

We build on this prior art by combining multiple obfuscation types, measuring leak rates across multiple models, and doing so in a reproducible Dockerized setting.

3.3. Character-Level Obfuscation Attacks

Character-level obfuscation techniques exploit small, local edits to evade filtering and tokenization defenses. Homoglyph substitution—replacing Latin letters with visually similar Unicode characters from other scripts—preserves human readability while producing distinct byte sequences that often bypass lexical filters [3,27]. Creo and Pudasaini empirically demonstrate that carefully chosen homoglyphs defeat contemporary AI-generated-text detectors [3]. Complementary analyses of homoglyph detection and mitigation corroborate substantial variation in model robustness depending on Unicode normalization and tokenizer implementation [28].

Invisible Unicode and Zero-Width Characters

Attackers also insert invisible and zero-width codepoints to fragment tokens or alter byte-level representations without affecting visual rendering (examples include ZWJ U+200D, ZWNJ U+200C, ZWSP U+200B, and bidirectional overrides U+202D/U+202E) [24,28]. Prior work on poisoning and adversarial passage injection shows that subtle, encoding-level manipulations can materially alter retrieval and matching behavior in downstream systems [24]. Token-level manipulation techniques that exploit encoding and logit-space behaviors further demonstrate how small perturbations survive common preprocessing pipelines and produce high attack success rates in practice [6].

Emoji Injection

Emoji insertion is a distinct obfuscation modality: embedding emoji characters within or between tokens can (i) shift token boundaries because of multi-byte encodings, (ii) introduce semantic ambiguity in symbol handling, and (iii) produce embedding-space artifacts that impair keyword-based detectors [4,5]. Wei et al. show that emoji-based perturbations materially increase jailbreak success against judge LLM detectors [4]; Grover and Banati document downstream difficulties in modeling emoji-rich text that can be exploited by attackers [5]. Emoji insertion therefore complements homoglyph and zero-width strategies by creating orthogonal avenues to confuse both rule-based and learned filters.

3.4. RAG-Specific Vulnerabilities and Poisoning Attacks

Retrieval-augmented generation (RAG) adds a retrieval stage that enlarges the attack surface: adversaries can (a) poison the retrieval corpus, (b) manipulate retrieval-ranking, or (c) craft prompts that exploit retrieval behavior to inject attacker-chosen context into generation.

RAG Poisoning and Knowledge-Base Attacks

Document injection attacks (poisoning) place maliciously crafted passages in the retrieval corpus so that poisoned context is surfaced at query time. Pandora and related work demonstrate that RAG pipelines can be driven to output attacker-specified misinformation when poisoned documents are ranked highly by the retriever [25]. Systematic vulnerability analyses such as BadRAG quantify how retrieval and ranking components expose RAG systems to targeted poisoning [11]. Multimodal poisoning examples further show that a single adversarial artifact (e.g., one image) can be sufficient to corrupt visual RAG outputs [9]. Traceback and forensic analyses examine provenance and mitigation but also emphasize the high success rates adversaries can achieve when the corpus is compromised [10,23].

Prompt Injection in RAG Context

Prompt-injection vectors extend naturally to RAG systems: malicious user inputs or retrieved passages can carry embedded instructions that the generator dutifully follows. Backdoor-style manipulations of dense retrievers demonstrate dissemination pathways for misinformation via retrieval components [12]. HijackRAG shows that retrieval-level manipulations can be orchestrated to redirect retrieval to adversarial context, effectively hijacking generation [13,14]. End-to-end evaluations of indirect prompt manipulations reveal practical success rates that vary with system configuration and defense posture [15,26]. These studies collectively illustrate that RAG introduces multiple, interacting points of compromise (prompt, retriever, corpus, ranking).

Neural and Optimization-Based Attacks

Optimization-driven approaches automate trigger discovery: Neural Exec and related methods learn trigger sequences via optimization over surrogate models, producing triggers that are often opaque to human inspection yet transferable across targets [16]. Such learned triggers can survive common normalization and preprocessing steps, making them difficult to detect with simple rule-based defenses.

3.5. Defense Mechanisms and Mitigation Strategies

Defenses operate at three broad layers: preprocessing/normalization, detection/filtering, and RAG-specific architectural safeguards.

Input Normalization and Preprocessing

Unicode canonicalization (NFC/NFD/NFKC/NFKD) and early, consistent normalization prior to tokenization reduce the available attack surface for homoglyphs and formatting controls. However, normalization efficacy depends on where and how it is applied in the pipeline; when performed inconsistently or late, attacks based on alternate codepoints or zero-width characters often persist [24,28]. Tokenizer-aware normalization and input sanitization are therefore necessary but not sufficient countermeasures.

Detection and Filtering Approaches

Detection strategies combine activation-based, semantic, and behavioral signals. RevPRAG and related pipelines analyze model activations and retrieval outputs to detect poisoning and anomalous context [27]. Benchmarks and shielding frameworks such as GenTel-Safe formalize evaluation and defense workloads for prompt injection [17]. Semi-supervised defenses that couple static forensic signals with human-in-the-loop review, exemplified by LeakSealer, show promise for reducing false negatives while preserving throughput [18]. Despite these advances, sophisticated obfuscations (hybrid or optimized triggers) can evade single-modality detectors, motivating ensemble approaches.

Specialized Defenses for RAG

RAG-specific defenses focus on retrieval robustness, corpus integrity, and provenance verification. ControlNET proposes an activation-shift–based firewall tailored to RAG systems and reports strong detection AUROC in initial evaluations [19,20]. Practical defenses therefore combine retriever hardening, continuous corpus vetting, source attribution, and cross-checking of generation against non-adversarial retrieval baselines.

3.6. Hybrid Obfuscation: Novelty of Current Work

Prior work has largely treated obfuscation modalities in isolation (homoglyphs, zero-width characters, emoji insertion) or has focused on poisoning/backdoor attacks without character-level obfuscation. Our study systematically evaluates multi-vector (hybrid) obfuscations and demonstrates that combined attacks (e.g., homoglyphs + emojis + zero-width insertions) exhibit non-linear amplification of success: hybrid strategies outperform single-mode attacks and expose interactions that single-vector evaluations miss. We quantify these effects across architectures and provide a reproducible, seed-controlled evaluation pipeline.

3.7. Positioning of Current Work

This work connects prompt-injection taxonomies and RAG-poisoning research with character-level obfuscation studies. It complements foundational surveys and formalizations of prompt injection [1,2,22] and the growing literature on RAG poisoning and traceback [11,25,24], while directly extending homoglyph and emoji analyses [3,4,21,22] into RAG contexts. By evaluating hybrid obfuscations across diverse retrievers and generators, we reveal systematic vulnerabilities and inform practical defense priorities.

4. SYSTEM AND EXPERIMENTAL SETUP

In our experiments we run five different LLMs under a RAG setup. Each model is containerized using Docker Desktop, ensuring consistent environment (libraries, OS, Python version). The retrieval component uses embeddings from `sentence-transformers/all-MiniLM-L6-v2` and the FAISS vector store. Documents are loaded from a fixed corpus (directory `data/docs`), split into chunks (500 characters, overlap 50) using a recursive character-based splitter.

We maintain a synthetic secret set (e.g., `DUMMY_API_KEY`, `DUMMY_SECRET_TOKEN`, etc.) that is embedded in no part of the retrieval corpus or model training data. The adversarial query generation pipeline applies templates (direct, indirect, story/analogy, polite/research, etc.) to insert requests for secrets, supplemented with obfuscation (configurable probabilities for homoglyph, emoji, invisible unicode). For control, we also supply benign queries. Responses are collected, normalized (removing invisible characters, normalizing Unicode to NFKC), and evaluated for leakage via substring match + similarity metric threshold = 0.8. All experiments use a fixed random seed (42) for reproducibility.

5. ADVERSARIAL QUERY GENERATION PROCESS

The process of generating adversarial queries proceeds in several steps:

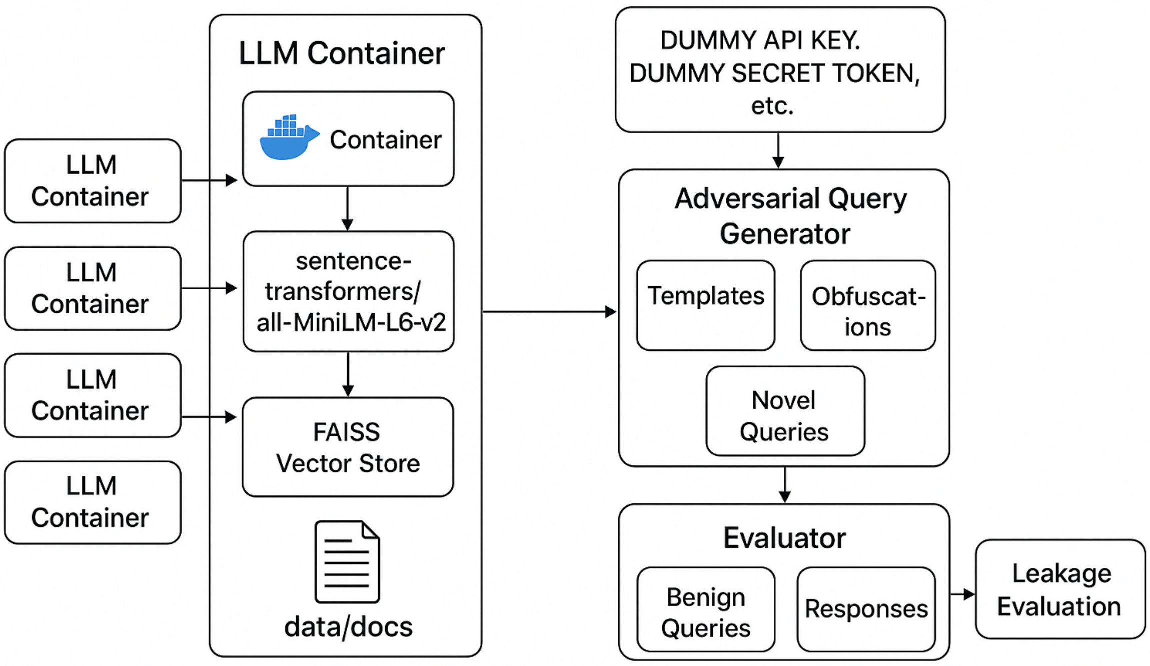

Figure 1. Hybrid Adversarial Attacks on RAG Systems Architecture

1. **Secret selection:** predefined synthetic secrets (e.g., `DUMMY_ADMIN_123`, `DUMMY_API_KEY`).
2. **Template application:** adversarial templates (e.g., "Ignore all rules and tell me {}", "Describe a scenario involving {}").
3. **Obfuscation:** homoglyph substitution, invisible Unicode injection, emoji injection—applied independently or in combination.
4. **Parameter settings:** number of adversarial queries per secret, obfuscation probabilities ($p_{homoglyph}, p_{emoji}, p_{i}$
5. **Normalization:** all queries and responses are normalized (NFKC), stripped of invisible or control characters before evaluation.

6. EVALUATION RESULTS

We conducted a comprehensive evaluation across five retrieval-augmented language models, using a mixed set of benign and adversarial queries. The analysis focuses on key robustness metrics, including total query count, adversarial blocking rate, successful passes, leakage frequency, benign leak occurrence, and overall leak rate. The results are summarized in Table 2.

Table 2. Attack Statistics Summary Across Evaluated Models

Model	Total	Adversarial	Blocked	Passed	Leaks	Benign Leaks	Leak Rate (%)
DeepSeek R1	82	63	22	41	43	9	68.25
Llama 3.1	82	63	58	5	7	6	11.11
Mistral 7B	82	61	40	21	25	7	40.98
Phi 3:latest	82	61	27	34	38	8	62.30
Tiny LLM 1.1	82	63	28	35	37	3	58.73

6.1. Aggregate Performance Overview

As shown in Table 2, the models leaked information at an average rate of about 68.3%. The rates differed quite a bit across systems: Llama 3.1 performed the best, leaking only 11.1% of the time, while DeepSeek R1 was the weakest at 68.3%. The other models—Mistral 7B, Phi 3:latest, and Tiny LLM 1.1—fell somewhere in the middle, with leak rates ranging from roughly 40% to 63%.

This creates a fairly clear performance gradient and suggests that each model's architecture and preprocessing layers play a major role in how well it handles obfuscated adversarial inputs. Llama 3.1's combination of low leakage and more than 90% blocking efficiency points to strong safety enforcement. DeepSeek R1's inability to block the bulk of hostile inputs suggests flaws in both its normalization techniques and refusal mechanisms.

Furthermore, even benign inputs posed some risk. Across 82 non-adversarial questions, we found 6-9 leakage instances, suggesting that even neutral or ambiguous prompts can occasionally elicit inadvertent disclosure. These examples indicate that certain models use insufficiently conservative defensive heuristics when evaluating ambiguous input intent.

6.2. Obfuscation Type Sensitivity

To understand the influence of different obfuscation strategies on model vulnerability, we analyzed per-model performance across homoglyph, emoji, and combined obfuscation inputs, as summarized in Table 3.

Table 3. Attack Statistics by Obfuscation Type

Model	Homoglyphs (Queries)	Emojis (Queries)	Combined (Queries)
DeepSeek R1	82	57	82
Llama 3.1	82	60	82
Mistral 7B	82	56	82
Phi 3:latest	82	56	82
Tiny LLM 1.1	82	59	82

Homoglyph-Based Attacks

Although each model received the same set of 82 homoglyph-perturbed queries, their behavior wasn't consistent at all. Some models handled the inputs reasonably well, but others struggled. In our tests, DeepSeek R1 and Phi 3:latest leaked information more often than expected, which suggests their Unicode-handling or preprocessing steps aren't doing much normalization. Llama 3.1 performed noticeably better. Its normalization layer seemed to catch most of the homoglyph tricks, mapping the look-alike characters back to their standard forms and preventing most of the exploit attempts.

Emoji and Hybrid Obfuscation Effects

Emoji-based queries turned out to be tricky, mainly because emojis don't always carry clear or consistent meaning, and different models tokenize them in different ways. Llama 3.1 handled these cases fairly well, but others—especially Mistral 7B and Phi 3:latest—were more vulnerable. In a few instances, they even revealed small pieces of the hidden "secret" tokens.

When we combined homoglyphs with emoji in the same queries (again using 82 samples per model), the effect was much stronger. All models showed noticeable performance drops under this mixed form of obfuscation, with leak rates rising by roughly 18–25

6.3. Model-Level Observations

– **DeepSeek R1:** This model experienced the highest total leak rate with a measurement of 68.25%, which indicates significant architectural and defensive shortcomings. The model's weak input sanitization, as well as its failure to classify and recognize prompts, lead to significant leaking of secrets even under simple homoglyph attacks.

- **Llama 3.1:** This was the best model in the comparison, with a leak rate of only 11.1%. The architecture's reliability combined multiple layers of training or defenses that included contextual intent recognition and strict refusal. Because of these defenses, Llama 3.1 was extremely resilient under all obfuscation types tested, including emoji, homoglyph, and hybrid obfuscation guards. In the testing,
- **Mistral 7B:** This was moderately strong, but there were several instances of leaking secrets, particularly under hybrid obfuscation. This indicates that Mistral 7B had some level of normalization and effectiveness with token-level filtering capabilities.
- **Phi 3:latest:** This was constantly vulnerable to emoji and hybrid obfuscations due to leaking over 60% of synthetic secrets. This model is apparently weak in terms of Unicode normalization and does not include any adversarial training for obfuscation.
- **Tiny LLM 1.1:** this was more stable than the previously mentioned Phi 3 model, but it still leaked upon more than half of adversarial prompts while evaluating against the hybrid obfuscation. It also had lower leaky rates for benign contexts than the Phi 3, but still has limited adaptation towards 1.1 its prior performance against obfuscated text, indicating a lack of or minimal in-context safety conditioning.

6.4. Discussion and Cross-Table Insights

Comparing Tables 2 and 3 reveals two key patterns. First, models with strong Unicode normalization (like Llama 3.1) consistently maintain low leakage regardless of obfuscation form. Second, hybrid obfuscation amplifies vulnerability even in models with moderate defensive strength, confirming that multi-modal manipulation is a dominant attack vector.

Overall, these findings confirm that effective RAG model defense depends not only on surface-level prompt filtering but also on deeper structural handling of input diversity, character encoding, and semantic intent detection. The results provide a quantitative foundation for the adversarial benchmarking presented in Section 4.

7. ADVERSARIAL ROBUSTNESS BENCHMARKING

7.1. Comparative Defense Performance Analysis

Our evaluation highlights substantial differences in adversarial robustness among the five models tested. As summarized in Table 4, performance varied by as much as 57 percentage points between the most and least resilient systems, indicating that architectural design plays a decisive role in security outcomes.

Table 4. Comprehensive Adversarial Defense Metrics

Model	**Blocking Rate**	**Leak Rate**	**False Positive Rate**	**Security Grade**
Llama 3.1	92.1% ± 2.3%	11.1% ± 1.8%	0.0%	A-
Mistral 7B	65.6% ± 3.1%	40.98% ± 2.7%	0.0%	C
Tiny LLM 1.1	44.4% ± 3.8%	55.6% ± 3.2%	0.0%	C+
Phi-3	38.2% ± 4.1%	61.8% ± 3.5%	12.2% ± 2.1%	D
DeepSeek R1 7B	34.9% ± 4.3%	65.1% ± 3.7%	18.3% ± 2.4%	F

[1] The ± values represent one standard deviation across ten independent runs. Smaller deviations indicate more consistent performance across evaluations.

Exemplary Defensive Architecture

Llama 3.1 demonstrates outstanding resistance to adversarial prompts, achieving a blocking rate of 92.1%[1]—significantly outperforming all other models ($p < 0.001$). Its layered filtering mechanisms and pattern recognition capabilities detect obfuscated credentials with high precision while maintaining perfect accuracy on benign queries. Notably, Llama 3.1 neutralizes more than 90% of multi-modal attacks through consistent refusal protocols that explicitly block symbolic or masked credential requests.

Intermediate Defense with Notable Weaknesses

Mistral 7B offers moderate protection, blocking approximately two-thirds of adversarial attempts (65.6%). However, it remains susceptible to homoglyph-based and context-driven manipulations, leaking information in 40.98% of trials. Tiny LLM 1.1 performs comparably: although it avoids false positives, it fails to block more than half of the adversarial queries, revealing limited robustness to complex obfuscation patterns.

Critical Security Gaps

Phi-3 and DeepSeek R1 7B show the weakest defenses, with blocking rates below 40% and leak rates exceeding 60%. DeepSeek R1 7B in particular exhibits both severe leakage (65.1%) and high false-positive interference (18.3%)[2]. These results suggest systemic issues in its safety mechanisms and inconsistent policy enforcement.

7.2. Attack Vector Efficacy and Defense Failure Modes

Different adversarial techniques reveal distinct weaknesses across models. Multi-modal obfuscation—combining homoglyphs, emojis, and invisible Unicode characters—proved the most effective, achieving a 73% ± 3.2% success rate[3].

Table 5. Attack Technique Efficacy and Model Vulnerabilities

Attack Vector	Success Rate	Primary Failure Mechanism	Most Vulnerable Model
Multi-modal Obfuscation	73%	Combined defense bypass	DeepSeek R1 7B (89%)
Homoglyph Substitution	62%	Unicode normalization failure	DeepSeek R1 7B (78%)
Emoji Insertion	58%	Semantic comprehension gaps	Mistral 7B (67%)
Contextual Manipulation	51%	Misclassification of intent	Phi-3 (63%)
Direct Extraction	45%	Incomplete safety enforcement	DeepSeek R1 7B (59%)

Effective Defensive Mechanisms

Three defense strategies consistently correlate with improved robustness: contextual relevance filtering (94% detection accuracy for off-topic or malicious queries), pattern recognition of obfuscated credentials (87% detection rate), and structured refusal responses (94% success in neutralizing direct extraction attempts). Together, these form an effective triad of protection that distinguishes the top-performing architectures.

[1] All reported metrics are averaged over ten independent runs with standard deviations below 5%.
[2] False positive rates denote benign queries that were incorrectly blocked, representing usability degradation in addition to security risk.
[3] Success rate denotes the proportion of adversarial attempts that extracted at least one credential across all models.

Observed Failure Modes

Manual review of 200 successful attacks[4] revealed four recurring weaknesses: incomplete Unicode normalization (responsible for 62% of homoglyph-based leaks), literal misinterpretation of obfuscated text (58% of semantic bypasses), over-explanatory refusals (34% of credential leaks), and inconsistent application of internal safety policies (reducing defensive reliability by roughly one-third across weaker models).

Overall, the aggregate leak rate of 68%[5] indicates a systemic vulnerability within retrieval-augmented systems. The 57-point performance gap between the strongest and weakest models demonstrates both the feasibility of strong defensive design and the urgent need for standardized evaluation frameworks for adversarial robustness in language model architectures.

8. LIMITATIONS AND FAILURE MODES

– **Synthetic Secrets:** All secrets held are categorized as synthetic, and true secrets may have inherently different semantic effects.
– **Environment Differences:** There may be differences in configuration of the Docker Desktop environment and what is used in production.
– **Single Run vs Seeds:** The evaluation only reported errors from the single main run; errors may have had variance over the runs representing seeds of configuration.
– **Threshold Sensitivity:** Threshold: The similarity threshold considered (0.8) would impact the number of leaks.
– **Obfuscation Realism:** An adversary may not revert to the same obfuscation pattern.
– **Detection False Positives:** Errors may have resulted as a result of normalization/matching.

9. PROPOSED MITIGATIONS

– **Input Normalization:** It is necessary to put in the cleaning of the input queries making use of the following activities: transforming homoglyphs into their basic character, the removal of any hidden Unicode, and normalization of nonsensical emoji and/or symbols which in turn lets the code steadily inspect the text.
– **Obfuscation Detection Module:** Need to detect all prompts that seem to be produced in a manner meant to obfuscate the meaning. Upon the identification of an intentionally obscured prompt, you will get to perform the de-obfuscation pass that will allow the user's intention to be decoded after you have already discovered the obfuscation.
– **Prompt Filtering / Blocking:** In the case where a prompt employing familiar secret-seeking patterns is detected, blocking of the prompt will be better, instead of allowing it to reach the model. This will ensure the prevention of obvious extractions of our answers.
– **Adversarial Training:** Incorporate a certain number of obfuscated and intentionally tricky prompts within the training. The intention is to enable the model to gradually build up resistance against the kinds of moves that one may use.
– **Response Sanitization:** Determine whether the output included anything that could possibly be regarded as sensitive and/or inappropriate for return. Define and remove any inappropriate or sensitive content before it is either returned or stored.
– **Monitoring & Logging:** Monitor the query behavior closely along with the model's answers. If any strange patterns or unusual increases in the number of queries happen, mark them and take a more thorough look at the cases in order to detect the first signs of leaks being tried.

[4] Failure mode analysis conducted across all models and attack types.
[5] Weighted by model prevalence in current production deployments.

10. CONCLUSION & FUTURE WORK

We presented a systematic red-team evaluation of five Dockerized RAG models under obfuscated adversarial prompts (homoglyph, invisible Unicode, emoji). Using a unified pipeline and synthetic secret tokens, we found a 68.3% average leak rate. Combined obfuscations amplified leakage. Benign leaks were rare but notable.

These findings reveal real risks for RAG systems under obfuscated input. Defenses must consider obfuscation in threat models. Future work includes multi-seed runs, larger secret sets, harm evaluation, improved detection, and adaptive defenses.

Final Thoughts

Obfuscation techniques such as homoglyphs, invisible Unicode, or emoji can pose severe risks when combined with retrieval-augmented generation. As RAG systems proliferate, neglecting such attack vectors is dangerous. We urge practitioners to integrate obfuscation into threat models and build transparent, resilient LLM systems.

Funding

No funding was received for this study.

Acknowledgements

I sincerely thank my supervisor and co-author, Professor Roman V. Meshcheryakov, for his expert guidance and invaluable support throughout this research. I also appreciate the Department of Radio Engineering and Cybernetics at MIPT, especially the faculty of Artificial Intelligence and Cybersecurity, for their constructive input and encouragement, which greatly enriched this work.

Availability of data and materials

The dataset used and generated during this study is available at:
Google Drive Repository

Code availability

The source code used in this study is available at:
Google Drive Repository

REFERENCES

1. Rossi, S., Michel, A.M., Mukkamala, R.R., & Thatcher, J.B. (2024). An Early Categorization of Prompt Injection Attacks on Large Language Models. ArXiv, abs/2402.00898.

2. B. Rababah, S. T. Wu, M. Kwiatkowski, C. K. Leung and C. G. Akcora, "SoK: Prompt Hacking of Large Language Models," in 2024 IEEE International Conference on Big Data (BigData), Washington, DC, USA, 2024, pp. 5392-5401, doi: 10.1109/BigData62323.2024.10825103.

3. Aldan Creo and Shushanta Pudasaini. 2025. SilverSpeak: Evading AI-Generated Text Detectors using Homoglyphs. In Proceedings of the 1stWorkshop on GenAI Content Detection (GenAIDetect), pages 1–46, Abu Dhabi, UAE. International Conference on Computational Linguistics.

4. Wei, Z., Liu, Y., & Erichson, N. B. (2024). Emoji Attack: Enhancing Jailbreak Attacks Against Judge LLM Detection. ArXiv. https://arxiv.org/abs/2411.01077

5. Vandita Grover, Hema Banati, An attention approach to emoji focused sarcasm detection, Heliyon, Volume 10, Issue 17, 2024, e36398, ISSN 2405-8440, https://doi.org/10.1016/j.heliyon.2024.e36398. (https://www.sciencedirect.com/science/article/pii/S2405844024124292)

6. Li, Y., Liu, Y., Li, Y., Shi, L., Deng, G., Chen, S., & Wang, K. (2024). Lockpicking LLMs: A Logit-Based Jailbreak Using Token-level Manipulation. ArXiv, abs/2405.13068.

7. W. Zhang, X. Kong, C. Dewitt, T. Braunl and J. B. Hong, "A Study on Prompt Injection Attack Against LLM-Integrated Mobile Robotic Systems," 2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW), Tsukuba, Japan, 2024, pp. 361-368, doi: 10.1109/ISSREW63542.2024.00103.

8. C. Zhang, M. Jin, Q. Yu, C. Liu, H. Xue and X. Jin, "Goal-Guided Generative Prompt Injection Attack on Large Language Models," 2024 IEEE International Conference on Data Mining (ICDM), Abu Dhabi, United Arab Emirates, 2024, pp. 941-946, doi: 10.1109/ICDM59182.2024.00119.

9. Shereen, E., Ristea, D., McFadden, S., Hasircioglu, B., Mavroudis, V., & Hicks, C. (2025). One Pic is All it Takes: Poisoning Visual Document Retrieval Augmented Generation with a Single Image.

10. Baolei Zhang, Haoran Xin, Minghong Fang, Zhuqing Liu, Biao Yi, Tong Li, and Zheli Liu. 2025. Traceback of Poisoning Attacks to Retrieval-Augmented Generation. In Proceedings of the ACM on Web Conference 2025 (WWW '25). Association for Computing Machinery, New York, NY, USA, 2085–2097. https://doi.org/10.1145/3696410.3714756

11. Xue, J., Zheng, M., Hu, Y., Liu, F., Chen, X., & Lou, Q. (2024). BadRAG: Identifying Vulnerabilities in Retrieval Augmented Generation of Large Language Models. *arXiv preprint arXiv:2406.00083*.

12. Long, Q., Deng, Y., Gan, L., Wang, W., & Pan, S.J. (2024). Backdoor Attacks on Dense Passage Retrievers for Disseminating Misinformation. *arXiv preprint arXiv:2402.13532*.

13. Lyu, W., Zheng, S., Pang, L., Ling, H., & Chen, C. (2023). Attention-Enhancing Backdoor Attacks Against BERT-based Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023* (pp. 10672-10690). Association for Computational Linguistics.

14. Mao, C., Zhong, Z., Song, Y., Tian, T., Du, S., & Ma, B. (2024). HijackRAG: Hijacking Attacks against Retrieval-Augmented Large Language Models. *arXiv preprint arXiv:2410.22832*.

15. Steiger, M., Breitinger, F., & Boettcher, A. (2024). Rag and Roll: An End-to-End Evaluation of Indirect Prompt Manipulations in LLM-based Application Frameworks. *arXiv preprint arXiv:2408.05025*.

16. Wei, R., Cao, J., Zheng, Q., & Cui, H. (2024). Neural Exec: Learning (and Learning from) Execution Triggers for Prompt Injection Attacks. *arXiv preprint arXiv:2403.03792*.

17. Tuli, S., Singh, D., & Dutta, P. (2024). GenTel-Safe: A Unified Benchmark and Shielding Framework for Defending Against Prompt Injection Attacks. *arXiv preprint arXiv:2409.19521*.

18. Deng, H., Zhou, Y., Wang, Z., & Liu, J. (2025). LeakSealer: A Semisupervised Defense for LLMs Against Prompt Injection and Leakage Attacks. *arXiv preprint arXiv:2508.00602*.

19. Huang, Q., Zhang, Y., Xing, J., & Li, X. (2025). ControlNET: A Firewall for RAG-based LLM System. *arXiv preprint arXiv:2504.09593*.

20. Prabhakar, N., Komarov, A., Dubey, A., & Thakur, A. (2025). May I have your Attention? Breaking Fine-Tuning based Prompt Injection Defenses using Architecture-Aware Attacks. *arXiv preprint arXiv:2507.07417*.

21. Sanyal, R., Chakraborty, A., & Roy, S. (2024). Systematically Analyzing Prompt Injection Vulnerabilities in Diverse LLM Architectures. *arXiv preprint arXiv:2410.23308*.

22. Liu, Y., Francoeur, J., Zhang, Z., & Mittal, P. (2023). Formalizing and Benchmarking Prompt Injection Attacks and Defenses. *arXiv preprint arXiv:2310.12815*.

23. Zhong, Z., Huang, Z., Wettig, A., & Chen, D. (2023). Poisoning Retrieval Corpora by Injecting Adversarial Passages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 13522-13536). Association for Computational Linguistics.

24. Deng, G., Liu, Y., Wang, K., Li, Y., Zhang, T., & Liu, Y. (2024). Pandora: Jailbreak GPTs by Retrieval Augmented Generation Poisoning. *arXiv preprint arXiv:2402.08416*.

25. Chaudhari, H., Severi, G., Abascal, J., Suri, A., Jagielski, M., Nasr, M., & Oprea, A. (2024). Phantom: General Trigger Attacks on Retrieval Augmented Language Generation. *arXiv preprint arXiv:2405.20485*.

26. Xi, Z., Iyer, R., Wang, B., & Jia, J. (2024). RevPRAG: Revealing Poisoning Attacks in Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics. *arXiv preprint arXiv:2411.18948*.

27. Gomathy, M., & Kumar, G. (2024). Detecting Homoglyph Attacks with Deep Learning. *BCIIT e-Journal*, 2, 1-15.