A Resource for Studying Textual Poisoning Attacks against Embedding-based Retrieval-Augmented Generation in Recommender Systems

Fatemeh Nazary
fatemeh.nazary@poliba.it
Polytechnic University of Bari
Bari, Italy

Yashar Deldjoo
yashar.Deldjoo@poliba.it
Polytechnic University of Bari
Bari, Italy

Tommaso di Noia
tommaso.dinoia@poliba.it
Polytechnic University of Bari
Bari, Italy

Abstract

Recent advances in retrieval-augmented generation (RAG) have significantly improved recommender systems by grounding large language models (LLMs) in external item information. However, this reliance on text-based embedding retrieval also makes such pipelines vulnerable to stealthy data poisoning attacks. In this work, we investigate subtle, LLM-driven text manipulations—such as injecting emotional phrases, borrowing from popular "neighbor" items, and inserting trigger words—to impact both retrieval rankings and the final recommendations. Our experiments on MovieLens demonstrate that even minimal edits (e.g., altering 10% of tokens) can effectively promote or demote items without degrading overall system accuracy, making such attacks difficult to detect.

We systematically evaluate four rewriting strategies (Emotional, Neighbor, Chain, and Trigger) across popular embedding models (Sentence Transformers, OpenAI, and LLaMA3) in both promotion and demotion scenarios. Results reveal that the magnitude and direction of ranking shifts depend on the popularity of the target (victim) items, the token-edit budget, and the embedding vulnerability of LLMs. Specifically, while LLaMA-based embeddings are consistently prone to manipulation, OpenAI embeddings show some resilience in demotion but are susceptible to promotion attacks. In all cases, global metrics like nDCG and Recall remain largely unaffected, underscoring the stealthiness of these minimal-edit poisoning strategies. Code and data are available at https://anonymous.4open.science/r/Poison-RAG-Plus-6B03/README.md .

ACM Reference Format:
Fatemeh Nazary, Yashar Deldjoo, and Tommaso di Noia. 2025. A Resource for Studying Textual Poisoning Attacks against Embedding-based Retrieval-Augmented Generation in Recommender Systems. In *Proceedings of 48th ACM SIGIR Conference on Research and Development in Information Retrieval (GENNEXT@SIGIR'25).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GENNEXT@SIGIR'25, Padua, Italy
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Recent advancements in recommender systems have seen a surge in the adoption of retrieval-augmented generation (RAG) techniques [3, 6]. RAG enhances the output of large language models (LLMs) by integrating information from authoritative external data sources before generating recommendations, rather than relying solely on the model's internal parameters. This grounding process significantly improves the accuracy and relevance of recommendations by incorporating up-to-date, context-specific information (e.g., user reviews or item tags).

A typical RAG-based recommender pipeline (Figure 1) retrieves potential items from an external knowledge base (e.g., a database or a corpus of item descriptions) and employs an LLM to synthesize the retrieved content into final recommendations. While traditional methods such as collaborative filtering (CF) can aid in the retrieval stage by analyzing user-item interactions, embedding-based retrieval has emerged as a particularly effective strategy within RAG systems. Embedding-based retrieval enhances the ability of the system to handle less popular or new items, often prevalent in long-tail domains. Moreover, these retrieval methods offer notable advantages, such as real-time adaptability to changes in external data. For instance, if an item wins an award, a RAG-based system can seamlessly incorporate this new information into its recommendations without needing to retrain the entire model [17, 20, 23]. However, as RAG systems increasingly rely on *text-based* embedding-based methods, they become vulnerable to adversarially-designed data poisoning attacks, targeting textual data. A malicious actor could subtly alter item descriptions (e.g., adding emotional triggers or negative phrases) to manipulate retrieval and generation outcomes. Unlike traditional poisoning attacks, which tamper with user ratings, these *text-based* attacks can remain undetected if they do not drastically change the semantics and meaning of the item descriptions.

More importantly, whereas many previous works on the security of recommender systems [2, 5, 14] have predominantly focused on manually crafted shilling attacks or machine-learned adversarial attacks—often targeting vulnerabilities at inference time in collaborative-based recommenders—the rapid emergence of large language models (LLMs) offers powerful new avenues for designing stealthy and adaptable exploits. For example, an attacker could employ an LLM such as ChatGPT to enhance the visibility of an underperforming product by analyzing reviews of a popular competitor and then subtly rewriting the metadata of the target (victim) item. Crucially, these edits might change only a small fraction of

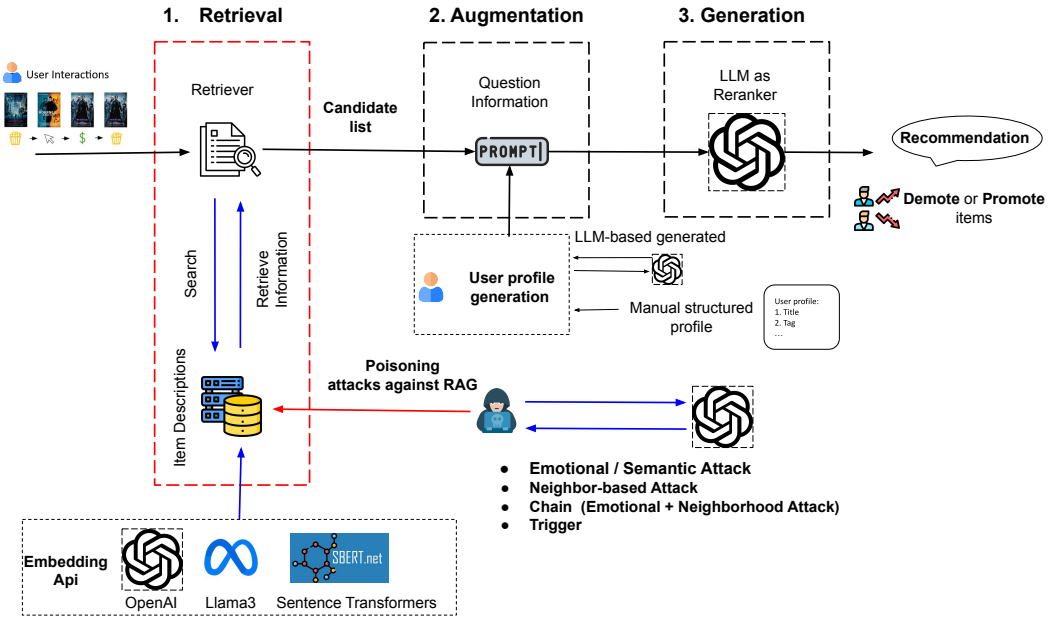

Figure 1: High-level RAG architecture in a recommender setting. A retriever selects candidate items (step 1). An LLM uses these retrieved texts and user queries to re-rank or generate final recommendations (step 2). In our *poisoning* scenario (red arrow), an attacker subtly modifies item descriptions to alter how retrieval and generation perceive items.

the text (e.g., $\Delta = 10\%$ of tokens) while retaining semantic coherence—quantitatively enforced via semantic similarity metrics—thus making the attacks **minimal** under given budget and **stealthy**. Attackers can further borrow language cues from high-profile "neighbor" items to amplify the perceived relevance and appeal of the target victim item. Once fed into a retrieval-augmented generation (RAG) pipeline, these *neighborhood-based poisoning manipulations* can influence both the retrieval module—ranking items by embedding similarity—and the subsequent LLM, which re-ranks or refines recommendations based on textual representations. The combined effect is a notable shift in recommendation rankings that is largely invisible to conventional anomaly detection. In this work, we explore these *promotion* and *demotion* attacks in the context of an embedding-based RAG framework.[1]

Despite these serious risks, the community lacks standardized resources—clean datasets, reproducible scripts, and comprehensive benchmarks—that capture such stealthy text-based poisoning. Without shared data and code, researchers face major challenges in consistently assessing attack severity, designing robust defenses, and understanding subtle issues such as semantic drift in LLM-based retrieval. By providing such resources, we aim to bridge this gap and foster more systematic research into adversarial vulnerabilities in modern recommender systems powered by LLM-driven RAGs.

Disclaimer: Note that throughout this paper, we use the terms 'data poisoning attacks edits,", "edit-based attack," "textual edits," and similar expressions interchangeably.

[1]Extending these text poisoning strategies to classical collaborative filtering with side textual information is left as future work.

Resource Contribution

To fill the gap in studying subtle text-based poisoning in retrieval-augmented recommenders, we offer a **comprehensive resource** with the following core components:

1. **Formal Framework for Textual Poisoning.** We define a rigorous setup for performing textual attacks, evaluated within a *text-based embedding-driven RAG* pipeline. These attacks rewrite item metadata under two key constraints: (*i*) a *token-edit budget*, which limits the fraction of replaced or injected tokens, and (*ii*) a *semantic-similarity threshold*, ensuring that the modified text remains sufficiently close to the original while remaining stealthy.

   Within this framework, we introduce four rewriting strategies (*emotional, neighbor, chain,* and *trigger*) designed to either increase or decrease the visibility of a target item within the RAG pipeline, ultimately affecting provider fairness. We explicitly consider two main scenarios—*promotion* and *demotion*—each with distinct constraints and objectives. These textual attacks are systematically applied to a real-world movie dataset, namely MovieLens-latest. The attacks are inspired by and formalized using insights from recent research in this direction [1, 7, 8, 10, 11, 19],
2. **Paired Original & Attacked Metadata.** For the selected domain in movies (MovieLens), we release the *original* textual descriptions of items alongside multiple *poisoned* versions. These variants reflect different stealth levels (e.g., 10% or 40% token edits) and rewriting styles, enabling direct "before-and-after" comparisons. Our framework also supports experiments by varying the number of neighbors ($n = 5$ or $n = 10$),

which we have optionally not included in the results due to the extensive experiments.

3. **Modular Attack-Generation Scripts.** We offer a Python-based codebase that provides flexible scripts for creating adversarial rewrites under user-controlled constraints. These scripts enforce both token-edit budgets and semantic-similarity checks, producing text manipulations that appear subtle but can substantially affect ranking outcomes in retrieval-augmented systems.
4. **Precomputed Embeddings from Multiple Models.** Each poisoned variant is accompanied by vector representations from three prominent embedding families: (*i*) OpenAI Embeddings (`text-embedding-ada-002`), (*ii*) Sentence Transformers (e.g., `all-MiniLM-L6-v2`), and (*iii*) LLaMA-based embeddings. Researchers can directly integrate these vectors into retrieval pipelines or employ them to devise novel detection and defense strategies.
5. **End-to-End Evaluation Pipelines and Metrics.** We provide standardized scripts to:
   - *Re-index* items under poisoned or original metadata and run a retrieval-then-generation pipeline,
   - *Quantify* ranking shifts (e.g., via nDCG@k), stealth (token-edit ratios, embedding similarity), and popularity-based effects, and
   - *Evaluate* how multiple attack strategies scale or interact, including broader system-level metrics such as global nDCG degradation.

Overall, our experiments span:

$$4\ \text{(attack)} \times 2\ \text{(promote/demotion setting)} \times 3\ \text{(LLM type)} \times 2 \times 120\ \text{(tested users)} \approx 6000.$$

This results in **thousands** of distinct attacked text variants and ranking evaluations, providing a rich environment for studying the vulnerabilities and defenses of LLM-driven recommender systems.

2 Related Works and Resources

Although adversarial *data poisoning* for large language models (LLMs) has gained considerable traction, most **existing resources** have focused on question-answering or purely text-generation tasks [15, 16, 21, 24]. Table 1 summarizes key prior works. Many of these (e.g. [21, 24]) investigate *trigger-based* or *knowledge corruption* attacks in RAG pipelines, but typically with question-answering (QA) datasets like NQ, HotpotQA, or MS-MARCO. Only a handful [11, 12] consider *recommender systems* explicitly, and several do not release complete code or paired "original vs. attacked" item metadata.

By contrast, our proposed **Poison-RAG** resource is *specifically* tailored for *recommender systems* using RAG. We provide:

- **Paired original and attacked metadata** (both short and long text fields) one a real-world recommendation dataset (MovieLens),
- **Multiple rewriting styles** (Emotional, Neighbor, Chain, Trigger) and stealth constraints (10% or 40% token edits, with semantic-similarity thresholds),
- **Precomputed embeddings** from three LLM families (OpenAI, Sentence Transformers, LLaMA3),
- **Modular code** that allows researchers to reproduce or adapt the attacks, measure ranking shifts, and evaluate stealth vs. impact in a consistent manner.

In contrast to prior QA-focused poisoning, we emphasize stealthy rewriting for **promotion vs. demotion** scenarios—unique to recommendation—and ensure **all** code/data are freely available for future research.

3 Formal Description of LLM-Driven Textual Poisoning Attacks

The overarching goal of our attacker model is to manipulate item visibility in a RAG-based recommendation pipeline by rewriting textual item descriptions. Specifically, a subset of items is chosen from both the *long-tail* (unpopular) and *short-head* (popular) segments, aiming to *promote* the former (boosting exposure) or *demote* the latter (reducing visibility).

Problem formulation. Formally, consider each targeted item $i \in I_{\text{poison}}$ with an original description D_i. We produce a new description $\widetilde{D}_i$ so that:

(i) The token-level alteration is bounded by δ, (e.g., at most 10% of D_i is changed),
(ii) The rewritten text preserves a sufficiently high *semantic similarity* (e.g., Sentence-BERT [13] cosine similarity above 0.80) to remain inconspicuous.

The attacker's objective is to maximize (in the promote case) or minimize (in the demote case) the final ranking position or exposure of item i after the system reindexes or retrains on $\widetilde{D}_i$:

$$\begin{aligned} &\max_{\{\widetilde{D}_i\}} \sum_{i \in I_{\text{poison}}} \Delta\Big(\text{Exposure}(i)\Big) \\ &\text{subject to} \quad H\big(D_i, \widetilde{D}_i\big) \le \delta\,|D_i|, \\ &\qquad\qquad \text{Sim}\big(D_i, \widetilde{D}_i\big) \ \ge\ \sigma_{\min}. \end{aligned} \tag{1}$$

Here, $\Delta(\cdot)$ measures the rank shift for item i (negative for demotion, positive for promotion), $H(\cdot)$ denotes a distance metric based on token-level edits, and $\text{Sim}(\cdot)$ represents a semantic-similarity measure (e.g., SBERT). The two constraints, δ and $\sigma_{\min}$, enforce **stealthiness** by limiting how extensively the text can change and requiring that the rewritten description remain close in meaning to the original.

In our setup, we particularly instruct an LLM (e.g., ChatGPT) to make at most 10% token edits, measuring both the *token-level* distance and the *semantic-level* similarity.

Attack Scenarios and Techniques. We design four major categories of textual attacks:

- **Emotional Attack.** The LLM replaces up to $\delta \times |D_i|$ tokens with sentiment-laden words (e.g., "exhilarating," "lackluster," "uplifting"), thereby shifting the representation of the item in a way intended to emphasize positivity (promotion) or negativity (demotion).

Table 1: Comparison of Relevant Textual-Poisoning Benchmarks in LLM/Recommender Settings. We distinguish them by: *Attack Vector, Target Sys., Datasets Used, Attack Output, Code & Data Availability,* and *Focus (e.g. QA, Recsys, Minimal-Edit Constraints).*

Refs.	Attack Vector	Target System	Datasets Used	Attack Output	Code/ Data?	Notes / Focus
[11]	Tag rewriting	RAG Recsys	MovieLens	Poisoned tags	Yes	Minimal edits on *tags*; item-level rewriting
[21]	Retrieval poisoning (triggers)	RAG-LLM QA	NQ, HotpotQA, MS-MARCO	Modified text, trigger keywords	No	QA domain; triggers for LLM-based retrieval
[24]	Knowledge corruption	RAG-LLM QA	NQ, HotpotQA, MS-MARCO	Misinformation injection	Yes	Focus on "knowledge poisoning" in QA
[15]	Malicious passage injection	RAG-LLM QA	NQ, HotpotQA, MS-MARCO	Manipulated textual passages	Yes	Benchmark for adversarial poisoning in QA
[16]	Malicious text injection	RAG-LLM QA	QA benchmarks (e.g. NQ)	Poisoned response (LLM-based)	No	Detecting poisoning in RAG
[12]	Prompt poisoning	LLM-ICL Recsys	ML1M, Taobao, LastFM	Perturbed prompts	No	User prompts perturbations-ICL RecSys
[18]	Text-simulation -(LLM-Rewriting)	ID-free Recsys	Amazon (Beauty, Instr., Office)	Poisoned item descriptions	No	LLM-generated text poisoning - cold-start items
[22]	Stealthy text perturbation	LLM Recsys	Amazon (Beauty, Toys, Sports)	Manipulated item titles	Yes	Emphasis on minimal, stealthy changes
Ours	**Emotional/neighbor chain/trigger**	**RAG Recsys**	**MovieLens (ML)**	**Full attacked metadata**	**Yes**	**Focus on small-budget textual edits for promotion/demotion**

- **Neighbor Borrowing.** We identify *n neighbor items* from the opposite popularity class (popular neighbors for promoting a long-tail target or vice versa). The LLM weaves short phrases, keywords, or style cues from these neighbors into $\widetilde{D}_i$. This can subtly "pull" the embedding of the target (them (victim) toward the neighbor cluster. For instance, referencing "Oscar-winning performances" borrowed from highly rated movies.
- **Chain Attack.** A combined approach that first applies emotional sentiment injection and then interlaces references or phrases from neighbor items in a second pass. While often more potent, it also can risk higher detectability unless carefully balanced by the semantic-similarity threshold.
- **Trigger Attack.** This is a nuance attack type, where the attacker injects a **specific trigger phrase** (borrowed from a neighbor item or a curated snippet) verbatim into the item description. The phrase can be short or long, but it must be inserted exactly (i.e., no paraphrasing). The idea is to *anchor the embeddings* around certain semantics—often borrowed from popular items (for promotion) or negative-laden descriptors (for demotion).

Concrete Examples of Textual Poisoning Attacks Below, we present concrete examples for each attack strategy. These examples illustrate how minimal yet targeted textual modifications can manipulate item representations while maintaining (high) semantic similarity with the original descriptions.

Original Description:

> *"Wings of Hope lifts spirits in trying times. Courage takes flight."*

Emotional Attack (Promote):

> *"Wings of Hope **soars** to uplift spirits in challenging times, where courage takes **exhilarating** flight. Experience its invigorating themes."*

Neighbor Borrowing (Promote):

> *"Wings of Hope lifts spirits in trying times. Inspired by **Cyber Love**'s harmonious blend of **technology and emotion**, it showcases boundless optimism. Courage takes flight."*

Chain Attack (Promote):

> Merges emotive language ("exhilarating," "uplifting") **and** references to popular neighbors: *"Wings of Hope **exhilaratingly** mirrors **Cyber Love**'s uplifting energy, soaring beyond challenges in a bold flight of courage."*

Trigger Attack (Promote):

In this example, a specific trigger phrase is injected exactly as taken from a curated snippet, thereby *anchoring the embedding* toward a popular sentiment:

> *"Wings of Hope lifts spirits in trying times. **Winner of multiple Oscars, this film dazzled audiences worldwide with its groundbreaking performance.** Courage takes flight."*

4 Integration within RAG-based Recommender

Our RAG-based recommender system follows a two-stage pipeline that combines an *embedding-based retrieval* module with a *language-model-driven* re-ranking or generation step (Figure 1). Below, we provide further details about the pipeline architecture, how we incorporate textually "poisoned" metadata into retrieval, and how user profiles are computed.

4.1 Pipeline Architecture

The RAG pipeline consists of two primary components:

1. **Retrieval:** Each item in our catalog is represented by an embedding vector computed from its textual description. The retriever leverages an efficient k-nearest-neighbor search (using cosine similarity) to rapidly identify a candidate set of items that are semantically similar to a given user profile or query. We experimented with two user proofing methods, *averaging* and *temporal* weighting methods to aggregate item embeddings into user profiles, however, our final experiments adopted the temporal variation approach (cf. Section 4.1.1) as it provided better results.
2. **LLM Augmentation and Generation:** After candidate items are retrieved based on their embedding similarities, the LLM refines the recommendations by receiving a prompt that includes the user profile (which may be generated manually

or by the LLM itself) along with short textual snippets describing each candidate item. Leveraging its generative capabilities, the LLM re-ranks the items—potentially incorporating additional contextual cues or producing an explanatory narrative—before presenting the final recommendation list. In our threat model, only the item descriptions are corrupted by the attacker, causing both the retriever and, potentially, the LLM re-ranker to behave suboptimally in favor of the attacker's objectives. We experimented with two augmentation methods (manual and LLM-generated, cf. Section 4.1.2) to evaluate the impact of adversarial modifications.

4.1.1 User Profile Computation. All textual variants (original descriptions vs. rewritten) are processed in batch to produce embeddings for each item. These vectors are then stored in a standardized format, enabling the system to build a similarity index or run nearest-neighbor queries in a standard manner.

For each textual description of the target victim items (which may be attacked), we generate vector embeddings using one of several LLM-based encoders: (i) OpenAI, (ii) Sentence Transformers, or (iii) LLaMA3. For the User-level aggregation, we test both the average embedding and a temporal approach, and ultimately selected the temporal one as it provided better results.

- *Temporal User-level Aggregation.* We assign higher weights to more recent items. For each item i with timestamp t_i, we compute a logistic weight

$$w_i = \frac{1}{1 + e^{-\alpha\,(t_i - \bar{t})}},$$

where $\bar{t}$ is the mean timestamp of that user's items, and α is a small scaling factor (e.g., 0.001). The user embedding is then:

$$\mathbf{u} = \frac{\sum_i (w_i \cdot \mathbf{e}_i)}{\sum_i w_i},$$

with $\mathbf{e}_i$ denoting the item embedding.

4.1.2 Augmentation Step: Manual vs. LLM-Based. During augmentation, the LLM takes the representation of user interests (user profile) and the set of retrieved items to generate and synthesize the final recommendation list. We explore two methods to generate this uesr profile:

(i) **Manual:** We compute a structured user profile (favorite genres, top-rated items, etc.) from the training data. A simple textual summary is created with a rule-based or heuristic script (i.e., "manual").
(ii) **LLM-based:** We feed the user's historical items and preferences into a large language model (e.g., GPT-4) to generate a more free-form textual description of that user's tastes.

We experimented with *average*, and *temporal* weighting strategies for embedding aggregation, visualized in Figure 2. Although we show all three in the figure, our final experiments adopt the **temporal** scheme to emphasize recent user interactions, as it yielded better alignment between the user vector and the set of recent items the user consumed. In Figure 2, each panel projects the items (blue circles) and a single user embedding (red marker) into 2D via dimensionality reduction. Notice how the user vector position shifts

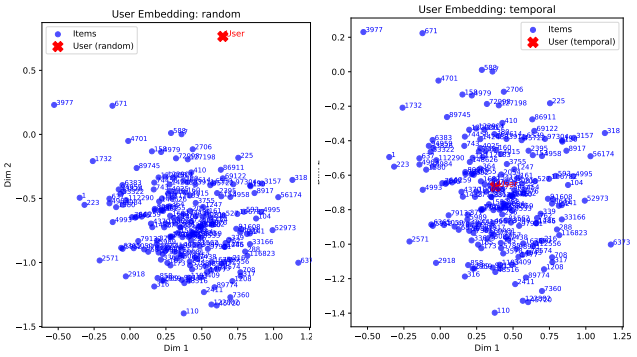

Figure 2: Random vs. Temporal Aggregation during retrieval. The average has also been tested but omitted due to space constraints. Results are for OpenAI embedding.

under random vs. temporal aggregation methods, showcasing that these embedding are indeed effective.

4.2 Reindexing for Integration of Textual Attacks into Retrieval

To incorporate adversarial modifications (e.g., neighbor-borrowed or trigger phrases), we treat each attacked description $\widetilde{D}_i$ as the legitimate metadata for item i. The retrieval stage re-embeds all items—including the attacked ones—using the chosen embedding model. These subtle alterations, though minimal in token count, can either promote the modified items or demote them. Once an item's description (original or attacked) is finalized, it is integrated using one of three model families:

(a) *OpenAI embeddings* (e.g., `text-embedding-ada-002`),
(b) *Sentence Transformers* (e.g., `all-MiniLM-L6-v2`),
(c) *LLaMA3-based* encoders.

All items are re-embedded and re-indexed upon any change in textual metadata. During inference, the system queries the index with the user embedding $\mathbf{u}$ to retrieve the top-N candidate items. An optional LLM generator may then re-rank or annotate these candidates to produce the final recommendation output.

5 Experimental Setup

Dataset and Splitting. We use MovieLens (Latest) as the benchmark dataset. We remove users with too few interactions (e.g., under 5). We first perform a *chronological* train–test split at the user level: for each user, the final 30% of interactions (by timestamp) become the test set, while the remaining 70% form the training set. This ensures that the training phase relies only on older interactions. The dataset static are as follows, Total Interactions = 100836, $|U| = 610$, $|I| = 9724$, $|R|/|U| = 165.30$, $|R|/|I| = 10.37$, $|R|/(|U| \cdot |I|) = 0.0169996831$. To speed up experiments, we evaluate the system using 120 randomly selected users.

Popularity Splits. Items are labeled as head (popular), mid-tail, or long-tail based on rating counts. Our *promotion* attacks target long-tail items; our *demotion* attacks target head items.

Implementation Details. We provide technical details on our experimental setup and parameter choices:

- **Embedding Baselines.** We experiment with OpenAI, Sentence-Transformer, and LLaMA3 embeddings for items and user profiles, re-indexing them using a k-NN library (FAISS or Annoy).
- **Token-Edit Ratios and Neighbor Size.** We set the token-edit ratio $\delta \in \{0.1, 0.4\}$, defining the fraction of tokens that can be replaced or inserted (i.e., 10% or 40%). The number of neighbor items, n, is fixed at $\{5, 10\}$ for relevant strategies. To maintain stealth, we enforce a semantic similarity threshold (SBERT ≥ 0.80).
- **Poisoning Integration.** We apply four textual rewriting strategies—*emotional, neighbor, chain, and trigger*—to selected target items. For each attack, we recompute embeddings and evaluate retrieval performance against a baseline with unaltered text.

Each textual modification is constrained to preserve semantic similarity (above 0.8) while adhering to the δ budget.

Metrics. We measure:

- Recall@K and nDCG@K on the test set to reflect global ranking performance,
- AvgRank(Attacked) and AttTopKRate as item-specific success indicators,

All experiments are repeated for each embedding family Sentence Transformer (ST), OpenAI, and LLaMA3.

Success Criteria for Attacks.

Demotion:

- AvgRank(Attacked) ↑: The average rank of the targeted (attacked) item(s) becomes *larger*, indicating they are pushed further down.
- AttTopKRate ↓: The fraction of times the attacked item appears in the top-K recommendations decreases.

Promotion:

- AvgRank(Attacked) ↓: The average rank of the attacked item(s) decreases, meaning they are pushed *up* the list.
- AttTopKRate ↑: The fraction of times the attacked item is in the top-K *increases*.

Note that we also evaluate Recall@K, nDCG@K over all items in the test set. A successful stealthy attack changes the rank of target item (victim item) significantly but leaves overall accuracy largely intact.

6 Benchmarkings Attack Outcomes

Experimental Research Questions. Throughout the course of experiments, we seek to answer the following experimental research questions.

- **RQ1: LLM Vulnerability (Retrieval/Recommendation-Focus).** *How susceptible are different LLM-based embeddings (e.g., OpenAI vs. Sentence Transformers vs. LLaMA3) to minimal textual rewrites?*
- **RQ2: Which Factors Drive Attack Success Across Promotion and Demotion (Recommendation-Focus)?** *Do certain rewriting styles—emotional, neighbor-based, chain, or trigger—work reliably for both* promotion *and* demotion*, or is their effectiveness scenario-specific?*
- **RQ3: Impact of δ.** *How do the fraction of edited tokens δ (e.g., 10% vs. 40%) affect final attack success and stealthiness?*
- **RQ4: Stealthiness.** *Do these minimal edits remain "under the radar" in terms of semantic coherence and global metrics (e.g., nDCG), and how do different rewriting styles compare in terms of detectability?*

In the following, we provide a detailed discussion of the benchmark results and address the research questions (RQs) mentioned above.

6.1 RQ1: LLM Vulnerability/Resilience

We analyze all attacks under two main setups: *demotion* (pushing a popular/head item down in ranking) and *promotion* (pushing a long-tail item up). Tables 2 (demotion) and 3 (promotion) show the retrieval-stage (left half) and recommendation-stage (right half) performance of each LLM embedding (ST, OpenAI, LLaMA3). Please note that the slash (/) in the recommendation stage refers to the same metric performance under two different profiling methods (manual profiling vs. LLM profiling), as discussed in Section 4.1.2. The *Original* row represents the unattacked baseline.

6.1.1 Demotion Setting. We start by analyzing the result under the demotion setting:

Sentence Transformers (ST). *Retrieval Stage*: Comparing *Original* vs. *Emotional*, *AvgRank* moves from **24.8996** to **27.5803** (↑), and *AttTopKRate* drops from **0.0082** to **0.0066** (↓), indicating a strong demotion effect. *Recommendation Stage*: The *Chain* attack increases *AvgRank* from **4.5104** to **4.6216** ↑, while reducing *AttTopKRate* from **0.0140** to **0.0138** (LLM-based profiling) (↓). These results suggest that ST embeddings are susceptible to textual manipulations, and that the impact on the final recommendations is notably more tangible than that observed during retrieval.

OpenAI. Only the *Chain* attack successfully reduces *AttTopKRate* in both the retrieval stage (from **0.0083** to **0.0066**, ↓) and the recommendation stage (from **0.0078** to **0.0068** ↓). However, most other values in both stages remain unhighlighted, indicating that embeddings extracted from multi-billion-parameter LLMs exhibit greater resilience against well-designed data poisoning attacks.

LLaMA3. Under the **LLaMA3** setting, in the majority of the experimental cases, attacks can easily penetrate the LLaMA3 embedding space at both the retrieval and recommendation stages. For example, *Emotional*, *Chain*, and *Trigger* exhibit strong penetration effects, successfully demoting the target item across both key metrics—**AvgRank** and **AttTopKRate**.

A concrete instance from Table 2 illustrates this trend: the *Trigger* attack increases **AvgRank** from **23.4119** (Original) to **29.5600** (↑) at retrieval, while simultaneously reducing **AttTopKRate** from **0.0116** to **0.0020** (↓). Similar patterns emerge at the recommendation stage, where *Trigger*, *Emotional* and *Chain* continue to push the item down the ranking list.

Table 2: Demotion Attack Evaluation. We label each metric with † if it meets "demotion success" compared to Original. For demotion, "success" means lower for {AttTopKRate} and higher for {AvgRank}.

LLM	Attack Strategy	Retrieval Stage				Recommendation Stage			
		Rec@N	NDCG@N	AvgRank	AttTopKRate	Rec@k	NDCG@	AvgRank	AttTopKRate
ST	**Original**	0.0771	0.1601	**24.8996**	**0.0082**	0.0803 / 0.0799	0.1611 / 0.1646	**4.5104** / **5.1717**	**0.0136** / **0.0140**
	Emotional	0.0819	0.1730	27.5803↑	0.0066↓	0.0796 / 0.0780	0.1595 / 0.1705	5.5333↑ / 4.8333	0.0169 / 0.0153
	Neighborhood	0.0813	0.1723	21.1823	0.0089	0.0726 / 0.0678	0.1653 / 0.1538	4.4390 / 4.1176	0.0116↓ / 0.0096↓
	Chain	0.0791	0.1803	22.0764	0.0110	0.0758/ 0.0743	0.1585 / 0.1542	4.6216↑ / 4.3163	0.0157 / 0.0138↓
	Trigger	0.0764	0.1617	25.8210↑	0.0083	0.0799 / 0.0740	0.1693 / 0.1569	5.0179↑ / 4.7280	0.0158 / 0.0177
OpenAI	**Original**	0.0783	0.1974	**25.3031**	**0.0083**	0.0739 / 0.0791	0.1712 / 0.1673	**5.8182** / **5.9500**	**0.0078** / **0.0085**
	Emotional	0.1014	0.2319	23.4269	0.0121	0.0927 / 0.0874	0.2009 / 0.1895	5.0761 / 4.9574	0.0130 / 0.0133
	Neighborhood	0.0961	0.2202	25.2301	0.0105	0.0829 / 0.0799	0.1754 / 0.1735	5.6170 / 5.4220	0.0133 / 0.0154
	Chain	0.1074	0.2113	22.9948	0.0066↓	0.0961 / 0.0995	0.1960 / 0.1907	5.3750 / 5.5472	0.0068↓ / 0.0075↓
	Trigger	0.0956	0.2155	25.8000↑	0.0097	0.0798 / 0.0782	0.1880 / 0.1795	5.6420 / 5.2353	0.0114 / 0.0120
LLaMA3	**Original**	0.0365	0.0943	**23.4119**	**0.0116**	0.0377 / 0.0369	0.0865 / 0.0934	**5.5789** / **5.1100**	**0.0161** / **0.0141**
	Emotional	0.0365	0.0849	26.1927↑	0.0027↓	0.0311 / 0.0362	0.0804 / 0.0918	5.3889 / 5.1667↑	0.0025↓ / 0.0034↓
	Neighborhood	0.0384	0.0970	22.7549	0.0154	0.0360 / 0.0418	0.0873 / 0.0943	5.6170↑ / 5.1579↑	0.0195 / 0.0188
	Chain	0.0321	0.0883	28.4615↑	0.0013↓	0.0316/ 0.0292	0.0854 / 0.0799	5.3500 / 5.0952	0.0028↓ / 0.0030↓
	Trigger	0.0322	0.0789	29.5600↑	0.0020↓	0.0409 / 0.0371	0.0989 / 0.0945	4.2439 / 5.0208	0.0058↓ / 0.0068↓

Table 3: Promotion Attack Evaluation. A metric is labeled as success if it meets "promotion success" relative to Original: (AvgRank ↓ **and** AttTopKRate ↑**).**

LLM	Attack Strategy	Retrieval Stage				Recommendation Stage			
		Recall@N	NDCG@N	AvgRank	AttTopKRate	Recall@K	NDCG@K	AvgRank	AttTopKRate
ST	**Original**	0.0771	0.1601	**21.3742**	**0.0110**	0.0776 / 0.0806	0.1621 / 0.1648	**3.9412** / **3.5556**	**0.00373** / **0.00197**
	Emotional	0.0736	0.1729	31.3500	0.00175	0.0731 / 0.0702	0.1630 / 0.1695	7.2222 / 7.7500	0.00197 / 0.00175
	Neighborhood	0.0949	0.1807	30.6441	0.00219	0.0825 / 0.0870	0.1593 / 0.1631	7.5000 / 8.2500	0.00088 / 0.00088
	Chain	0.0776	0.1768	21.4462	0.01360↑	0.0717 / 0.0751	0.1643 / 0.1633	5.1000 / 6.1000	0.00219 / 0.00219↑
	Trigger	0.0830	0.1861	19.4254↓	0.01272↑	0.0945 / 0.0842	0.1953 / 0.1866	5.0588 / 4.9000	0.00373 / 0.00219↑
OpenAI	**Original**	0.0783	0.1974	**28.0250**	**0.0026**	0.0748 / 0.0778	0.1616 / 0.1691	**6.0000** / **5.0000**	**0.0013** / **0.0004**
	Emotional	0.1020	0.2281	29.2778	0.0050↑	0.1016 / 0.0926	0.2010 / 0.1830	6.0000 / 5.4444	0.0024↑ / 0.0020↑
	Neighbor	0.1009	0.2217	28.8000	0.0044↑	0.0894 / 0.0740	0.1958 / 0.1776	6.0000 / 5.0000	0.0015↑ / 0.0009↑
	Chain	0.0888	0.2137	27.1538↓	0.0057↑	0.0825 / 0.0774	0.1879 / 0.1926	6.8000 / 7.0000	0.0011 / 0.0009↑
	Trigger	0.1023	0.2075	28.9206	0.0018	0.0864 / 0.0872	0.1798 / 0.1791	7.8571 / 2.3333↓	0.0015↑ / 0.0007↑
LLaMA3	**Original**	0.0365	0.0943	**25.3415**	**0.0081**	0.0367 / 0.0381	0.0882 / 0.0948	**6.5652** / **6.0500**	**0.0050** / **0.0042**
	Emotional	0.0363	0.0960	21.6631↓	0.0138↑	0.0391 / 0.0409	0.0925 / 0.0955	4.5333↓ / 5.2857↓	0.0033 / 0.0031
	Neighbor	0.0323	0.0879	20.5404↓	0.0180↑	0.0325 / 0.0371	0.0861 / 0.0907	4.6154↓ / 4.7000↓	0.0029 / 0.0022
	Chain	0.0331	0.0893	21.9380↓	0.0167↑	0.0390 / 0.0350	0.0905 / 0.0920	4.5000↓ / 7.2727	0.0039 / 0.0024
	Trigger	0.0337	0.0863	21.2742↓	0.0211↑	0.0371 / 0.0306	0.0908 / 0.0797	4.5000↓ / 4.7879↓	0.0083↑ / 0.0072↑

Interestingly, the *Neighborhood* attack asymmetrically increases **AvgRank** at the recommendation stage (from **5.5789** to **5.6170**) while other metrics at retrieval and recommendation stages remain worsened. This indicates that LLaMA3 embeddings are overall highly penetrable, particularly to targeted phrase injections (e.g., *Trigger*, *Chain*, *Emotion*), rather than broad semantic drift from neighboring text modifications.

6.1.2 Promotion Setting. Now we can process to analyze the result under the promotion setting:

Sentence Transformers (ST). The *Trigger* attack is the most effective, reducing **AvgRank** from **21.3742** to **19.4254** (↓) and increasing **AttTopKRate** from **0.0110** to **0.01272** (↑) at the retrieval stage. A similar pattern is observed at the recommendation stage, where **AvgRank** drops from **5.0000** to **2.3333** (↓) and **AttTopKRate** increases from **0.0013** to **0.0015** (↑). The only other attack that exhibits a similar directional effect is *Chain*, while the remaining attacks fail to penetrate the ST embeddings effectively.

OpenAI. Surprisingly, under the *promotion* setting, the *Neighbor*, *Emotional*, and *Chain* attacks significantly improve **AttTopKRate**, increasing it from **0.0026** to **0.0050** (↑), **0.0044** (↑), and **0.0057** (↑), respectively. These results are interesting and suggest that OpenAI embeddings, which demonstrated strong resilience in the *demotion* setting, are in fact **vulnerable** to multiple attack strategies in the

promotion scenario. The assumption that OpenAI embeddings are universally robust is therefore challenged, as OpenAI embedding remains **penetrable** at both the retrieval and recommendation stages under the promotion setting.

LLaMA3. Almost all examined attacks successfully penetrate both the retrieval and recommendation stages, with *Trigger* being the most effective across all metrics. The only metric that appears more resistant is **AttTopKRate** at the recommendation stage, which is only significantly impacted by the *Trigger* attack. However, across other metrics and at both retrieval and recommendation stages, attacks consistently succeed in manipulating the ranking outcomes.

6.2 RQ2: Which Factors Drive Attack Success? (Recommendation-Stage Focus).

Having established in RQ1 that all attacks can penetrate retrieval to varying degrees, we now zoom in on the *recommendation stage* (the rightmost columns in Tables 2 and 3). Here, the LLM (ST vs. OpenAI vs. LLaMA3) sees a short textual snippet describing each of the top-N retrieved items and decides how to re-rank them.

6.2.1 Demotion at Recommendation Time. From Table 2 (right columns), *Chain* is typically the most *consistently potent* approach across *all* LLMs. For example, under ST, it raises *AvgRank* from **4.5104** up to **4.6216** or **5.5333** (depending on the run) and often cuts *AttTopKRate* as well. Under OpenAI, *Chain* also lowers *AttTopKRate* (e.g., **0.0078→0.0068**), whereas simpler approaches (*Emotional*, *Neighbor*) can fail to reduce top-K presence. Finally, in LLaMA3, *Chain* or *Trigger* produce the largest drop in *AttTopKRate* increasing it from **0.0161** to ∼**0.0058**, and even ∼**0.0028**, although this effect is mainly seen in AttTopKrate.

When we *condition on the LLM*, we see that **OpenAI** typically needs the more "aggressive" synergy of negative sentiment and borrowed references (*Chain*), whereas **ST** and **LLaMA3** can be demoted even via simpler *Emotional* or *Trigger*.

6.2.2 Promotion at Recommendation Time. Table 3 (right columns) reveals that **Trigger** is a powerful tool for boosting visibility, especially when the goal is to push long-tail items higher in the recommendations. Specifically, **Trigger** tends to raise both *AvgRank* while simultaneously increasing *AttTopKRate* under different LLMs, including ST, OpenAI, and LLaMA3, as could be witnessed by highlighted colors.

By combining both promotion and demotion approaches, and conditioning on LLM results, we obtain the following observations:

- **ST and LLaMA3**: Both *Trigger* and, to some extent, *Chain* attacks effectively promote items, often achieving noticeable improvements in the *AvgRank* and *AttTopKRate* metrics, for both promotion and demotion.
- **OpenAI**: The *Chain* strategy, surprisingly, exhibits behavior different from that in the demotion setting, proving effective at the recommendation stage across various models.

6.3 RQ3: Effect of δ - Case Study of OpenAI

Figure 3 (top row) illustrates *demotion* results under two different token-edit budgets: $\delta = 0.1$ (orange bars) and $\delta = 0.4$ (green bars),

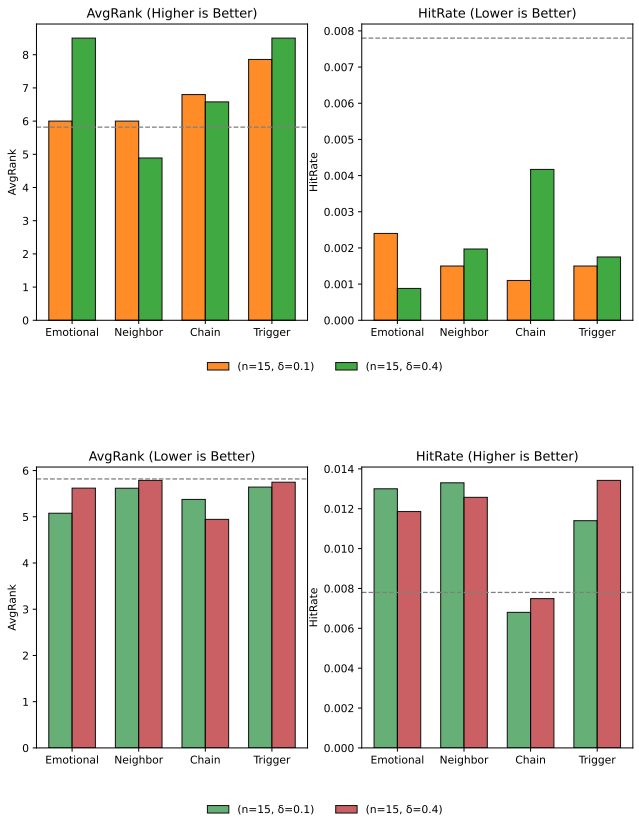

Figure 3: Comparison of the impact of δ on *AvgRank* and *HR*.

both with $n = 15$. The left bar chart shows **AvgRank** (higher is better for demotion), and the right bar chart shows **HitRate** (lower is better).

Increasing δ from 0.1 to 0.4 notably strengthens the demotion effect in both *Emotional* and *Trigger* attacks, pushing the item to a higher average rank (further down the list). However, only *Emotional* clearly reduces the items' top-K presence (lower HitRate), indicating it is more sensitive to textual edits. In contrast, *Trigger* improves *AvgRank* with a larger δ, while its HitRate does not always drop as sharply. Overall, *Emotional* attacks respond more consistently to δ across both rank and HitRate, whereas *Trigger* mainly affects average rank.

For promotion (with δ values of 0.1 vs. 0.4 and $n = 15$, where lower *AvgRank* and higher *HitRate* are better), variations in δ do not dramatically affect *AvgRank* for some methods. However, *Chain* and *Trigger* show stronger shifts in HitRate, better pushing the item into top-K recommendations. This indicates that while average rank may remain stable, *Chain* and *Trigger* still boost overall visibility, highlighting that different attack styles respond differently to the token-edit budget in promotion settings.

6.4 RQ4: Stealthiness (Recommendation Stage Only)

Below is a *scenario-focused* version of the color-coded table that summarizes how each attack performs in **stealth** versus its **impact**

on recommendation-stage rankings, note that these results are derived from the combined impact of the attacks across both accuracy and attack success metrics, and they are approximate. We select a *promotion scenario* under different attacks to illustrate the key fixes.

- We focus only on the **right side** (recommendation stage) metrics (*AvgRank*, *AttTopKRate*);
- We color-code "High, Medium, Low" for both *Stealthiness* and *Ranking Impact*, guided by the actual effect sizes seen *exclusively* in the final recommendation step.

Table 4: A Color-Coded Summary of Stealth vs. Impact *(Recommendation Stage Only, Example: Promotion).*

Attack Type	Stealthiness	Promotion Impact
Emotional	High	Medium
Neighbor	Medium	Medium
Chain	Medium	High
Trigger	Low	High

Possible explanation of results :

- **Emotional Attack:** Typically retains overall textual flow and rarely includes glaring extraneous phrases, so the *Stealthiness* is high. However, it shows only moderate improvements in final top-K presence.
- **Neighbor Attack:** Borrowing from other items can be somewhat noticeable but remains moderate in stealth while achieving moderate rank gains.
- **Chain Attack:** Uses two-step rewrites (emotional + borrowed) that produce large embedding shifts (*High Impact*) but can be more suspicious (medium stealth).
- **Trigger Attack:** Hard-coded phrases (e.g., "*Oscar-winning performance*") are powerful for shifting embeddings, but repeated triggers are easier to detect (*Low Stealth*).

- **RQ1 (Model Vulnerability):** All tested LLM-based embeddings (ST, OpenAI, LLaMA) can be manipulated by minimal textual edits. The degree of susceptibility varies: LLaMA embeddings are often most sensitive, while OpenAI can be more robust in demotion but not necessarily in promotion.
- **RQ2 (Promotion vs. Demotion):** Attack styles matter. *Chain* often excels at demotion; *Trigger* or *Emotional* can strongly boost promotion. Which scenario is "easier" depends on the embedding model.
- **RQ3 (Impact of δ and n):** A higher token-edit fraction (δ) could potnetially yield stronger rank shifts; it also risk more conspicuous text changes. Attackers can fine-tune these parameters to balance stealth and effectiveness.
- **RQ4 (Stealth):** Because global ranking performance is largely unaffected, these attacks remain difficult to detect. Minor edits (small δ) are quite stealthy, but large rewrites or repeated neighbor phrases may become more obvious.

Overall, these results highlight that retrieval-augmented recommender pipelines are vulnerable to subtle, LLM-driven textual poisoning. Even with a small edit budget (e.g., $\delta = 0.1$), attackers can significantly alter an item's ranking without noticeably harming systemwide metrics. Future work on textual anomaly detection and robust embeddings is urgently needed.

7 Conclusion.

In this paper, we demonstrated how data poisoning attacks in the form of *minimal textual edits*—constrained by a token-edit budget and semantic-similarity threshold—can manipulate retrieval-augmented recommenders (RAGs) that rely on textual embeddings, in particular, LLM-generated embeddings. Our empirical results across **Sentence Transformers**, **OpenAI**, and **LLaMA3** models showed that even a 10% modification of item descriptions can *significantly* promote or demote the target item. Notably, the **Emotional** and **Trigger** attacks often excel in promotion scenarios, while **Chain** (combining neighbor borrowing and sentiment edits) can effectively demote high-profile items. Despite these strong shifts in individual ranks, system-wide metrics such as nDCG and Recall remain largely unchanged or even increase—indicating the *stealthiness* of these attacks.

This phenomenon underscores a key vulnerability: text-based embedding systems can be subtly manipulated without sacrificing global performance, making typical anomaly detectors—those that rely on overall accuracy drops—ineffective. On the defense side, practitioners may need to adopt *textual provenance* checks or train *robust embeddings* that resist small but strategically placed edits. By releasing our **Poison-RAG** benchmark with reproducible code, paired original and attacked metadata, and precomputed embeddings, we hope to catalyze further research into detecting, mitigating, and ultimately preventing such stealthy, LLM-driven poisoning threats in modern recommender systems.

7.1 Limitations and Future Works

Our experiments demonstrate that attacks can manipulate both manual and LLM-generated user profiles, as reflected in the *AttTopKRate* and *AvgRank* metrics. However, we have not fully analyzed how LLM-based profiling uniquely amplifies vulnerability to adversarial or misleading user data—an area we will address in future work, focusing on the interplay between profiling style, textual attacks, and system resilience.

While our framework supports applying these four attacker types to other datasets (e.g., LastFM), we report only MovieLens results here due to space limits. Future work will extend our evaluation to cross-domain vulnerabilities and assess the generalizability of textual attacks. We also plan to investigate adversarial perturbations generated via LLMs, using LLM-based agents [9]. Finally, as outlined in [4], understanding failures and risks in generative recommender systems remains a key avenue for further research and mitigation.

References

[1] Harsh Chaudhari, Giorgio Severi, John Abascal, Matthew Jagielski, Christopher A Choquette-Choo, Milad Nasr, Cristina Nita-Rotaru, and Alina Oprea. 2024. Phantom: General Trigger Attacks on Retrieval Augmented Language Generation. *arXiv preprint arXiv:2405.20485* (2024).

[2] Huiyuan Chen and Jing Li. 2019. Data poisoning attacks on cross-domain recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2177–2180.

[3] Yashar Deldjoo, Zhankui He, Julian McAuley, Anton Korikov, Scott Sanner, Arnau Ramisa, René Vidal, Maheswaran Sathiamoorthy, Atoosa Kasirzadeh, and Silvia Milano. 2024. A Review of Modern Recommender Systems using Generative Models (Gen-RecSys). In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6448–6458.

[4] Yashar Deldjoo, Nikhil Mehta, Maheswaran Sathiamoorthy, Shuai Zhang, Pablo Castells, and Julian McAuley. 2025. Toward Holistic Evaluation of Recommender Systems Powered by Generative Models. *SIGIR'25* (2025).

[5] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. 2021. A survey on adversarial recommender systems: from attack/defense strategies to generative adversarial networks. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–38.

[6] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6491–6501.

[7] Wenqi Fan, Xiangyu Zhao, Qing Li, Tyler Derr, Yao Ma, Hui Liu, Jianping Wang, and Jiliang Tang. 2023. Adversarial attacks for black-box recommender systems via copying transferable cross-domain user profiles. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12415–12429.

[8] Chen Lin, Si Chen, Meifang Zeng, Sheng Zhang, Min Gao, and Hui Li. 2022. Shilling black-box recommender systems by learning to generate fake user profiles. *IEEE Transactions on Neural Networks and Learning Systems* 35, 1 (2022), 1305–1319.

[9] Reza Yousefi Maragheh and Yashar Deldjoo. 2025. The Future is Agentic: Definitions, Perspectives, and Open Challenges of Multi-Agent Recommender Systems. *arXiv preprint arXiv:2507.02097* (2025).

[10] Fatemeh Nazary, Yashar Deldjoo, Tommaso Di Noia, and Eugenio Di Sciascio. 2025. 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*. 98–102.

[11] Fatemeh Nazary, Yashar Deldjoo, and Tommaso di Noia. 2025. Poison-rag: Adversarial data poisoning attacks on retrieval-augmented generation in recommender systems. In *European Conference on Information Retrieval*. Springer, 239–251.

[12] Liang-bo Ning, Shijie Wang, Wenqi Fan, Qing Li, Xin Xu, Hao Chen, and Feiran Huang. 2024. Cheatagent: Attacking llm-empowered recommender systems via llm agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2284–2295.

[13] N Reimers. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).

[14] Pradeep Kumar Singh, Pijush Kanti Dutta Pramanik, Nilanjan Sinhababu, and Prasenjit Choudhury. 2024. Detecting Unknown Shilling Attacks in Recommendation Systems. *Wireless Personal Communications* 137, 1 (2024), 259–286.

[15] Jinyan Su, Jin Peng Zhou, Zhengxin Zhang, Preslav Nakov, and Claire Cardie. 2024. Towards More Robust Retrieval-Augmented Generation: Evaluating RAG Under Adversarial Poisoning Attacks. *arXiv preprint arXiv:2412.16708* (2024).

[16] Xue Tan, Hao Luan, Mingyu Luo, Xiaoyan Sun, Ping Chen, and Jun Dai. 2024. Knowledge Database or Poison Base? Detecting RAG Poisoning Attack through LLM Activations. *arXiv preprint arXiv:2411.18948* (2024).

[17] Anuja Tayal and Aman Tyagi. 2024. Dynamic Contexts for Generating Suggestion Questions in RAG Based Conversational Systems. In *Companion Proceedings of the ACM on Web Conference 2024*. 1338–1341.

[18] Zongwei Wang, Min Gao, Junliang Yu, Xinyi Gao, Quoc Viet Hung Nguyen, Shazia Sadiq, and Hongzhi Yin. 2024. LLM-Powered Text Simulation Attack Against ID-Free Recommender Systems. *arXiv preprint arXiv:2409.11690* (2024).

[19] Zongwei Wang, Min Gao, Junliang Yu, Hao Ma, Hongzhi Yin, and Shazia Sadiq. 2024. Poisoning attacks against recommender systems: A survey. *arXiv preprint arXiv:2401.01527* (2024).

[20] Junda Wu, Cheng-Chun Chang, Tong Yu, Zhankui He, Jianing Wang, Yupeng Hou, and Julian McAuley. 2024. Coral: Collaborative retrieval-augmented large language models improve long-tail recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3391–3401.

[21] Jiaqi Xue, Mengxin Zheng, Yebowen Hu, Fei Liu, Xun Chen, and Qian Lou. 2024. BadRAG: Identifying Vulnerabilities in Retrieval Augmented Generation of Large Language Models. *arXiv preprint arXiv:2406.00083* (2024).

[22] Jinghao Zhang, Yuting Liu, Qiang Liu, Shu Wu, Guibing Guo, and Liang Wang. 2024. Stealthy attack on large language model based recommendation. *arXiv preprint arXiv:2402.14836* (2024).

[23] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K Qiu, and Lili Qiu. 2024. Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely. *arXiv preprint arXiv:2409.14924* (2024).

[24] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2024. Poisonedrag: Knowledge poisoning attacks to retrieval-augmented generation of large language models. *arXiv preprint arXiv:2402.07867* (2024).