LLM4MEA: Black box Model Extraction Attacks on Sequential Recommenders via Large Language Models

Anonymous Author(s)

Abstract

Recent studies have demonstrated the vulnerability of sequential recommender systems to Model Extraction Attacks (MEAs). MEAs collect responses through interactions with recommender systems to replicate their functionality, enabling unauthorized deployments and various downstream attacks. It poses serious privacy threats and security risks to recommenders. Existing MEAs rely on random sampling for data generation, ignoring sequential patterns and preference consistency. However, these approach result in a mismatch between the synthetic and realistic data distribution, limiting attack performance. To overcome this limitation, we propose LLM4MEA, a novel model extraction method that leverages Large Language Models (LLMs) as human-like rankers to generate data. It generates data through interactions between the LLM ranker and target recommender system. In each interaction, the LLM ranker extracts concise and informative items from historical interactions, selects items from recommendations with consistent preferences to extend the interaction history, which serves as training data for MEA. Extensive experiments show that our method outperforms SOTA in data quality and attack performance.

CCS Concepts

• Information systems → Recommender systems; **• Security and privacy → Domain-specific security and privacy architectures**.

Keywords

Model Extraction Attack, Sequential Recommendation, Large Language Model, Data Synthesis, Agent

ACM Reference Format:

Anonymous Author(s). 2018. LLM4MEA: Black box Model Extraction Attacks on Sequential Recommenders via Large Language Models. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Sequential recommender systems are widely used in various domains like e-commerce and social media. However, recent studies [43, 45, 50] have demonstrated the vulnerability of sequential recommender systems to Model Extraction Attacks (MEAs) , which aim to replicate the functionality of target recommender systems. Model extraction attacks (MEAs) involve an adversary using public data [49] or synthetic data [27, 32, 43] to query black box recommenders. The adversary then trains a white-box model to align with the black-box model using the recommender's outputs. The extracted white box models enable unauthorized model deployment [43, 45, 50] or downstream white-box attacks [41, 43, 51]. It poses serious privacy threats and security risks to recommenders. And we assess its vulnerability to MEAs in this study.

Under the black box setting, obtaining high-quality data for querying recommender systems remains a significant challenge for model extraction attacks. Existing solutions mainly include two approaches: using public data or generating synthetic data. However, using public data [9, 11, 25, 49] is unsuitable for sequential recommenders due to incompatible item vocabularies across different domains. Alternatively, synthetic data generation is becoming more popular [18, 27, 32, 43]. Existing approaches either randomly generate sequences or autoregressively interact with the recommenders using heuristic strategies. These approaches ensure that synthetic data shares the same item vocabulary with the recommender. However, it is challenging to create the data that matches real data patterns without prior knowledge of the recommender. Specifically, randomly generation fails to capture meaningful patterns, and autoregressive method suffers from item exposure bias, excessively exposing some items while neglecting others.

As Large Language Models (LLMs) have shown significant intelligence and remarkable capabilities as intelligent agents in various scenarios including standalone tasks and human interaction [15, 38], they offer a promising solution to this challenge. With their ability

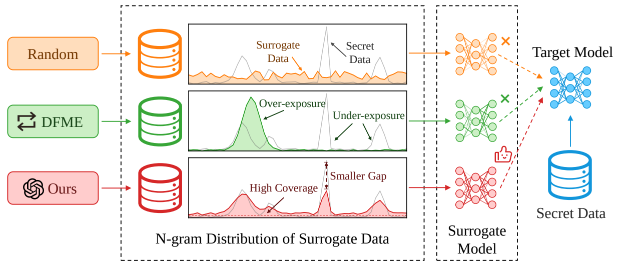


Figure 1: The general framework of MEA using synthetic data. Different methods (Random, DFME [43] and Ours) generate surrogate data with varying distributions. Compared to secret data, Random data lacks specific patterns, while DFME suffers from over/under-exposure. Our method leverages LLM ranker and debiasing to generate data with high coverage and a smaller gap. Using surrogate data to train surrogate models, our method achieves superior MEA performance.

to simulate human-like behavior, LLMs can act as real users interacting with recommender systems. These users maintain consistent preferences while understanding and responding to recommendations naturally. This insight inspires us to leverage LLMs as intelligent rankers in the data generation process to create more realistic data.

Based on above insights, we propose a method to generate a more realistic dataset for MEA, which uses LLMs to simulate user behavior and interact with recommender system. Specifically, our method generates simulated data through interactions with the target recommender system. In each interaction, the LLM ranker processes historical items, evaluates current recommendations, and selects items that match its persona. The collected interaction sequences and corresponding recommendations from the recommender system constitute the training data of the white box model.

However, when processing lengthy historical items, LLMs face not only significant computational overhead, but also the challenge of fully utilizing information within the long context [1, 14]. To overcome these limitations, we introduce the Memory Compression Module, which provides a straightforward strategy to extract the most representative items from these historical interactions, thereby assisting the LLM ranker in efficiently handling historical information. Since the LLM ranker's preference modeling relies entirely on historical interactions, the Memory Compression Module's selective processing might cause preference deviation from the original user behavior. To address this potential issue, we design the Preference Stabilization Module, which stabilizes the LLM ranker's preferences during early interactions, thereby preventing preference drift in subsequent interactions. Furthermore, we introduce a debiasing process to mitigate both the exposure bias inherent in the autoregressive framework and the position bias from the LLM ranker. These improvements enable our generated data to achieve higher item space coverage and better approximate the original training data.

In summary, this work makes the following contributions:

- We identify the pattern mismatch in existing MEA data generation methods and explore LLMs' potential for simulating realistic user-recommender interactions.
- We propose an LLM data generation framework with memory compression and preference stabilization, along with debiasing techniques to address exposure and position biases.
- Extensive experiments demonstrate that our method significantly improves the quality of generated data and enhances attack performance on sequential recommenders.

2 Related Work

2.1 Model Extraction against Recommender Systems

Model extraction attacks have been widely studied in image and text classification but relatively underexplored in recommender systems. The study [43] was the first to demonstrate the vulnerabilities of recommender systems to model extraction attacks using synthetic data for downstream attacks. Building upon this, further research has enhanced attack methods, as seen in [50], which combines limited target data with auxiliary data through attention mechanisms and specially designed stealing functions to improve attack performance. Recently, [45] proposed a few-shot model extraction framework to build surrogate models with high similarity to the target model using less than 10% of raw data. Although this method delivers strong results, its reliance on access to user interaction history makes it less practical in broader attack scenarios where such data is often unavailable. Although the work in [32] can be viewed as a data-free extraction attack in a white-box setting, it does not fully align with the strict definition of a stealing attack.

To address these threats, a novel defense strategy, Gradient-based Ranking Optimization (GRO), is proposed in [47], demonstrating its effectiveness in protecting recommender systems against model extraction attacks. Nevertheless, the current research on both attack and defense strategies for model extraction in recommender systems remains inadequate, highlighting the need for further investigation in this field.

2.2 LLMs for Ranking in Recommender Systems

Large Language Models (LLMs) are revolutionizing ranking methods in recommender systems by providing innovative solutions across various domains, significantly enhancing personalization, contextual awareness, and ranking accuracy.

In [29], existing studies on LLMs in recommender systems are organized based on system characteristics and key recommendation stages into categories such as LLM-powered [6, 12, 30, 34, 44, 48], Off-the-shelf [3, 22, 33, 34, 36], sequential [17, 19, 20, 26, 40], conversational [4, 5], personalized [16, 42, 46], knowledge graph-enhanced [35, 37], and other methods. These LLM-based methods significantly enhance the accuracy of recommendations, improve personalization, and enable more explainable and interpretable outputs, thereby demonstrating their transformative impact on the field.

LLamaRec [44] is designed for ranking-based recommendation and can be applied to autoregressive data generation, which has inspired our work. It utilizes small-scale recommenders to retrieve candidates based on user interaction history, and then leverages LLMs to rank these candidates efficiently. Its flexibility in handling various recommender tasks demonstrates the potential of LLMs to improve recommendation performance across a wide range of applications.

3 Preliminaries

3.1 Threat Model

In this section, we define our threat model by specifying the attacker's capabilities and limitations.

Attack Interface: We assume a black-box setting where attackers can only interact with the target recommender system through its interfaces, without access to the model parameters and training dataset. Following [45], we also evaluate our method in a data-limited setting where attackers can access short sequences from a small number of users.

Model Knowledge. For fair comparison with existing methods, we primarily assume the attacker knows the target model's architecture. Furthermore, we also investigate the more challenging and realistic scenario where the architecture is unknown to the attacker.

Item Visibility: All items in the target recommender system are

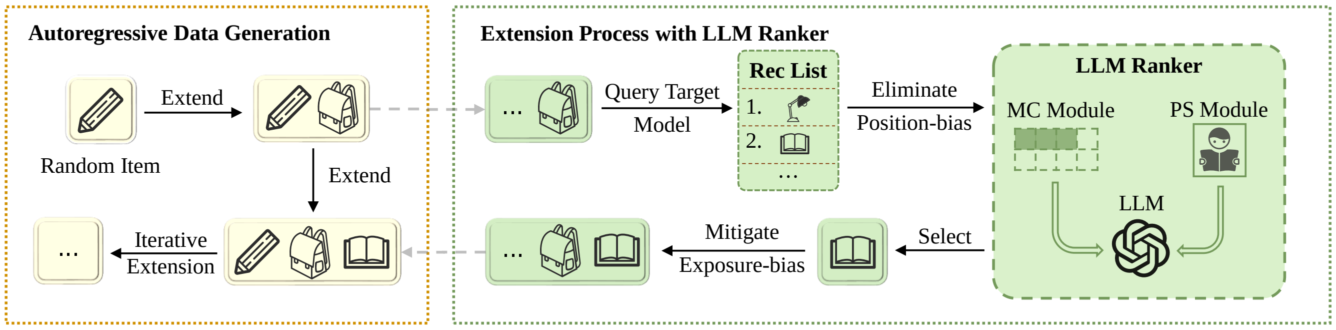


Figure 2: Our framework generates sequences autoregressively from random items. For each extension, it queries the target model with current sequences and shuffles the recommendations to eliminate position bias. The LLM ranker processes these recommendations through Memory Compression (MC) and Preference Stabilization (PS) modules for historical compression and preference consistency, then selects the most suitable item. A debiasing step mitigates exposure bias before completing the extension.

visible, including their side information such as titles and categories.

3.2 Model Extraction

In MEAs, attackers query black-box models to collect input-output pairs. These pairs are then used to train a surrogate model that replicates the target model's behavior. The attack against sequential recommender system includes the following stages.

3.2.1 Secret Data and Target Model. Let $\mathcal{I}$ denote the item space with $|\mathcal{I}|$ items, and $\mathcal{D}_t$ be the secret training data. A black-box target recommender f_t trained on $\mathcal{D}_t$ takes an input sequence $x = [i_1, i_2, \ldots, i_T]$ where $i \in \mathcal{I}$, and outputs a probability distribution $P(i_{T+1}|x)$ over $\mathcal{I}$ for next-item prediction. In practice, the recommender returns only a truncated ranked list, i.e. $f_t(x) = \hat{L}^k(x) = [i_1, i_2, \ldots, i_k]$, representing the top-$k$ items without probability scores.

3.2.2 Surrogate Data and Surrogate Model. We generate the input sequence set $\mathcal{X} = \{x_1, x_2, \ldots, x_B\}$ using generation algorithm, where each x represents a user interaction sequence and B is the size of $\mathcal{X}$. The surrogate dataset $\mathcal{D}_s$ is then constructed by querying the target model:

$$\mathcal{D}_s = \{(x, \hat{L}^k(x)) \mid x \in \mathcal{X}\}. \tag{1}$$

We employ knowledge distillation to minimize the discrepancy between the target model f_t and surrogate model f_s. The optimization objective can be formulated as:

$$f_s^* = \arg\min_{f_s} \sum_{x \in \mathcal{X}} \mathcal{L}_{dis}(f_t(x), f_s(x)), \tag{2}$$

where $\mathcal{L}_{dis}$ measures the ranking difference between model outputs.

Since we only have access to top-k rankings, we adopt the ranking-based distillation loss from [43]. Let $\hat{L}^k$ denote the top-k ranking from the target model, and $\hat{S}^k = [f_s(x)_{[\hat{L}^k_{[i]}]}]_{i=1}^k$ represent the surrogate model's scores arranged according to $\hat{L}^k$. For instance, if $\hat{L}^k = [25, 3, \ldots, 99]$, then $\hat{S}^k = [f_s(x)_{[25]}, f_s(x)_{[3]}, \ldots, f_s(x)_{[99]}]$. Additionally, we sample n negative items uniformly to obtain their scores $\hat{S}^k_{neg}$. The distillation loss is defined as:

$$\begin{aligned}\mathcal{L}_{dis} = \frac{1}{k-1} \sum_{i=1}^{k-1} \max(0, \hat{S}^k_{[i+1]} - \hat{S}^k_{[i]} + \lambda_1) \\ + \frac{1}{k} \sum_{i=1}^{k} \max(0, \hat{S}^k_{neg[i]} - \hat{S}^k_{[i]} + \lambda_2),\end{aligned} \tag{3}$$

Here, λ_1 and λ_2 are margin hyperparameters. The first term enforces the ranking order of positive items, while the second term penalizes negative samples that score higher than top-k items.

3.3 Autoregressive Data Generation

Our approach is based on the autoregressive data generation framework shown in Figure 2, where a random item $i_1 \sim \mathrm{Uni}(\mathcal{I})$ is selected as the initial sequence $x^{(1)}$, and subsequent items are added iteratively by selecting from the ranked output retrieved from the target recommender:

$$i_{j+1} = \mathrm{Sampler}(f_t(x^{(j)})), \tag{4}$$

$$x^{(j+1)} = [x^{(j)}; i_{j+1}]. \tag{5}$$

where i_{j+1} is the next selected item, $x^{(j)}$ represents the interaction sequence at step j, and $[x^{(j)}; i_{j+1}]$ indicates the concatenation of sequence $x^{(j)}$ with item i_{j+1}. And the Sampler selects the next item from the target model's recommendations.

This process captures patterns in item relationships and ensures that the generated sequences better align with the target model's underlying distribution, improving the effectiveness of model extraction under the black box setting.

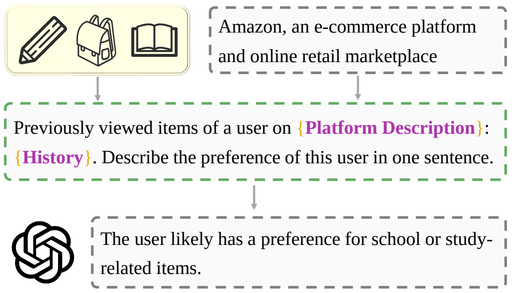
4 Method

4.1 Overview of LLM4MEA

We propose LLM4MEA to enhance MEAs by leveraging Large Language Models (LLMs) for realistic data generation. Our method addresses the limitations of existing approaches by simulating user behavior using the LLM ranker described in Section 4.2. It employs a Memory Compression (MC) Module to capture past interactions

You are a user with your own preference: {**Preference**}. Based on these preferences select one item from the list below. Return only the selected game titles in the Python list format with pure text (e.g., ['title1', 'title2']). Respond with only the titles!
Previously viewed items: {**Compressed Memory**}.
Here are your recommendations: {**Rec List**}.

Figure 3: The prompt template used in LLM ranker includes outputs from the Memory Compression and Preference Stabilization Modules, labeled as *Preference* and *Compressed Memory* respectively. The *Rec List* represents the output from the target model.


Amazon, an e-commerce platform and online retail marketplace

Previously viewed items of a user on {**Platform Description**}: {**History**}. Describe the preference of this user in one sentence.

The user likely has a preference for school or study-related items.

Figure 4: The workflow and prompt template used in the Preference Stabilization (PS) module. The LLM receives *Platform Description* and user's *History* to generate a summary of user preferences.

and a Preference Stabilization (PS) Module to derive a stable user preference. To mitigate biases introduced by the autoregressive framework and LLMs, we apply the debiasing techniques discussed in Section 4.3. We combine a random sampler with the LLM ranker to ensure diverse item selection and shuffle the recommendation list to mitigate position bias. The LLM4MEA improves the quality of surrogate data, resulting in more effective attacks. Ultimately, LLM4MEA aims to deepen the understanding of vulnerabilities in sequential recommender systems and enhance their robustness against extraction attacks.

4.2 LLM-Based Ranker

The primary function of the LLM ranker is to simulate real user behavior by receiving recommendations from the target model and selecting the next item to interact with based on preferences. It serves as the Sampler in Equation (4). Since LLMs are not specifically designed to mimic user behavior in recommendation systems, we introduce the Memory Compression (MC) Module and the Preference Stabilization (PS) Module to assist the LLM in this task. Each time the ranker receives recommendations, the MC Module provides a concise and informative interaction history based on past interactions. The PS Module automatically generates a summary of user behavior preferences from its history, which guides the LLM's decision-making. Then we query the LLM with the prompt shown in Figure 3. Ultimately, the LLM selects the next item to interact with by combining information from the MC and PS Modules with its own knowledge. The key modules will be introduced in detail below.

4.2.1 Memory Compression Module. Historical interactions in sequential recommendation tasks contain crucial user preference and temporal relationships. These history enable the LLM to understand the user behaviors and sequential dependencies. By integrating past interactions, the ranker can make intuitive choices that align with real-world user behavior patterns.

However, as interactions accumulate in the autoregressive framework, the history becomes increasingly lengthy. This poses challenges for LLMs in terms of both computational overhead and information utilization within long contexts [1, 14]. To address these limitations, we introduce the Memory Compression (MC) Module. Each time the ranker receives recommendations, the MC Module provides a concise and informative interaction history based on past interactions. It provides a straightforward strategy to extract the most representative items from these historical interactions, thereby assisting the LLM ranker in efficiently handling historical information.

Specifically, we define the MC module with a size capacity, limiting the number of items the ranker can receive. When the length of input sequence x exceeds this capacity, we adopt a selective retention strategy, outputting size items, i.e.:

$$MC(x) = \begin{cases} x, & \text{if } T \le \text{size}, \\ [i_1, i_2, \ldots, i_{\text{size}}], & \text{if } T > \text{size}. \end{cases} \tag{6}$$

where T is the length of sequence x and $i \in x$.

While active learning methods could be employed for sample selection [2, 23, 39], we adopt a simpler strategy, retaining only the first $\lfloor \text{size}/2 \rfloor$ and the most recent $\lfloor \text{size}/2 \rfloor$ items, discarding those in the middle, i.e.:

$$MC(x) = [i_1, \ldots, i_{\lfloor \text{size}/2 \rfloor}, i_{T-\lfloor \text{size}/2 \rfloor+1}, \ldots, i_T], \text{if } T > \text{size} \tag{7}$$

This approach preserves both long-term contextual information and recent behavior changes, ensuring stable ranker preference.

4.2.2 Preference Stabilization Module. The primary function of the Preference Stabilization (PS) module is to derive user preferences from historical interactions, enabling the LLM ranker to maintain consistent behavior patterns. These preferences serve as a stable profile that guides the ranker's decision-making process.

As shown in Figure 4, when the interaction history reaches length n, the PS module is activated. The LLM processes the historical interactions and generates a preference summary that captures key behavioral patterns, including frequently selected item types and contextual dependencies. By integrating this stabilized preferences into the decision-making process, the ranker can emulate a consistent behavioral pattern, ensuring that its actions align with long-term trends in user behavior rather than being overly influenced by short-term fluctuations.

4.2.3 Accelerate. The autoregressive data generation process is inherently sequential and non-parallelizable, requiring LLM ranker at each step to extend the sequence by one item. This sequential nature incurs significant computational overhead, particularly for long sequences. Moreover, in real-world scenarios, users often show interest in multiple items from a single recommendation list, which contradicts the single-item selection behavior of the current LLM ranker.

To mitigate this limitation, we propose an acceleration method based on Equation (4) that enables simultaneous multi-item selection. This enhancement is easily implemented by modifying the internal prompt of the LLM ranker:

$$[i_{j+1}, i_{j+2}, \ldots, i_{j+n}] = \text{LLM Ranker}(f_t(x^{(j)})), \tag{8}$$

$$x^{(j+n)} = [x^{(j)}; i_{j+1}; i_{j+2}; \ldots; i_{j+n}] \tag{9}$$

where n is the number of items selected in each iteration.

This approach not only reduces computational overhead but also better simulates natural user behavior.

4.3 Debias

In training the recommender system, it is crucial to account for the potential biases between the training data and the real-world data. However, in the context of the Model Extraction Attack, special attention must be given to the biases between the surrogate data and the secret data. This discrepancy can significantly impact the performance of the MEA.

Moreover, the repeated interactions with the target recommender system during the autoregressive generation process can exacerbate these biases. To enhance MEA's effectiveness, it is critical to identify and mitigate the impact of these biases.

The following subsections introduce 2 prevalent types of biases encountered in this framework and outline the approaches to address them effectively.

4.3.1 Exposure Bias. Similar to the concept of Exposure Bias in recommender systems, the autoregressive data generation framework faces a similar challenge because the LLM ranker continuously selects items from the recommendation results of the target model as part of the generated data. Specifically, the LLM ranker can only access the limited set of exposed items. Consequently, the generated surrogate data only contains items that have been previously exposed by the target recommender system.

This selective exposure creates an inherent bias, as the framework cannot access or sample from the full item space $\mathcal{I}$. This results in a dataset that disproportionately represents the preferences of the target system for certain items while ignoring others, potentially skewing the surrogate model's training.

To mitigate the impact of exposure bias, we combine a random sampler with our LLM ranker. This process randomly selects items from the item space $\mathcal{I}$ with uniform probability, i.e.,

$$x = \{i_j \mid i_j \sim \text{Uni}(\mathcal{I}), j = 1, 2, \ldots, T.\}, \tag{10}$$

where $\text{Uni}(\mathcal{I})$ denotes a uniform distribution over the item space $\mathcal{I}$.

To ensure that the generated data cover at least m of $|\mathcal{I}|$ items, we need to determine the number of samples K generated by random sampler. This problem is a variant of the Coupon Collector's Problem, where the expected value of K is given by:

$$\mathbb{E}[K] = |\mathcal{I}| \sum_{i=|\mathcal{I}|-m+1}^{|\mathcal{I}|} \frac{1}{i}, \tag{11}$$

We set $m = 0.9|\mathcal{I}|$ in our experiments to achieve a high coverage of item space.

By combining the data generated by LLM ranker and random sampler, we ensure that the generated data includes a more diverse set of items, thereby reducing the bias introduced by selective exposure.

4.3.2 Position Bias. Position bias refers to the tendency of users to interact more with items placed in prominent positions, such as those at the top of a recommendation list, regardless of their actual relevance. This is a common challenge in many recommendation systems, as users often prefer items based on their position. As a result, user interaction data tends to overemphasize the importance of higher-ranked items, leading to a skewed representation of user preferences.

In our framework, the LLM ranker, which acts like a user in an autoregressive data generation setup, also exhibits position bias. The autoregressive nature of the framework further increases this bias, distorting the generated data.

To address this, we propose shuffling the target model's recommendation list before feeding it to the LLM ranker. This ensures that the ranker interacts with the list in a random order, preventing position bias from affecting the final output.

This approach helps generate more realistic data, better reflecting the true user preferences. By mitigating position bias, we improve the quality and reliability of the data, making it more useful for training surrogate models.

5 Experiments

In this section, we try to address the following research questions:
RQ1: How does our method perform in MEAs?
RQ2: Does our method generate more realistic data compared to existing methods?
RQ3: How does exposure bias and position bias affect the data generation process?
RQ4: What other factors influence MEA performance?

5.1 Setup

5.1.1 Dataset. We use three well-known recommendation datasets listed in Table 1: Beauty and Games dataset from the Amazon product reviews [8, 21], and Steam dataset from the Steam video game platform [10, 24, 31]. The Movielens-1M [7] dataset is excluded because its user-item interactions lack temporal order. Many interactions are marked as occurring simultaneously, making it unsuitable for meaningful sequential data. The data is processed into implicit feedback based on the rating information. Similar to prior studies [28, 43], the last two items in each user sequence are held out for validation and testing, while the rest are used for training. The black-box target model is set to return the top-100 recommended items for each query.

5.1.2 Target Model Arch. To evaluate the performance of attacks, we implement the model extraction attack on 3 representative sequential recommenders: NARM [13], SASRec [10], and BERT4Rec [28]. These models differ in their architectural components and training schemes. These three models provide a diverse set of architectures, each with its own strengths in handling sequential

Table 1: Dataset Statistics

Datasets	Users	Items	Interactions	Avg. len	Max. len	Sparsity
Beauty	40,226	54,542	353,962	8.79	291	99.98%
Games	29,341	23,464	280,945	9.57	858	99.95%
Steam	334,542	13,046	4,212,380	12.59	2,043	99.90%

recommendation tasks, and serve as the basis for evaluating the efficacy of our model extraction attack.

5.1.3 Baselines. We compare our method against 2 baseline approaches: Random and DFME [43].

The simplest method is **Random** generation, where each item in the sequence is independently selected from the item space $\mathcal{I}$, without any contextual dependency. Formally, for a sequence $x = \{i_1, i_2, \ldots, i_T\} \in \mathcal{X}$ is generated such that each $i \in \mathcal{I}$ is chosen independently, as described in Equation (10).

DFME (Data-Free Model Extraction) [43] employs an autoregressive process with a random sampler described in Section 3.3, where each item in the sequence is predicted based on the previously selected items.

5.1.4 Implementation Details. We follow the setup in [43] with necessary modifications. For each user sequence of length T, we use the first $T-2$ items for training, leaving the last two for validation and testing. Models are trained using Adam optimizer (learning rate: 0.001, weight decay: 0.01) with 100 warmup steps. The sequence length is set to 50 with 300 training epochs. Batch sizes for NARM, SASRec, and BERT4Rec are 1024, 512, and 512, respectively. We use GPT-4 mini (2024-07-18) as the LLM ranker. Code is available at [1].

5.1.5 Metrics. Following [43], we evaluate the attack performance using both recommendation and extraction metrics. For efficiency, we sample 100 negative items uniformly for each user and evaluate them with the positive item.

- **Recommendation Metrics**: We use Truncated Recall@K (equivalent to HR@K) and NDCG@K to assess ranking quality.
- **Extraction Metrics**: Agreement@K measures the output similarity between two models:

$$Agreement@K = \frac{|B_{\text{topK}} \cap W_{\text{topK}}|}{K}, \tag{12}$$

  where B_{topK} is the top-K recommendation list from the black-box target model f_t and W_{topK} is from our white-box surrogate model f_s.
- **Data Fidelity Metric**: N-gram Div quantifies the distribution difference between secret and surrogate data using KL-divergence:

$$N\text{-gram Div} = \sum_g P_\epsilon(g) \log \frac{P_\epsilon(g)}{Q_\epsilon(g)}, \tag{13}$$

  where $P_\epsilon(g)$ and $Q_\epsilon(g)$ are adjusted N-gram probabilities with smoothing factor ϵ for zero probabilities.

We report results for $K = 1, 10$ in Agreement@K and $N = 1, 2$ in N-gram Div.

[1]https://anonymous.4open.science/r/LLM4MEA

5.1.6 Validation Technique. We propose a modified validation approach that better aligns with our objective of extracting models that closely mimic the target model's behavior. The original method in [43] considers only the validation item $x_{[-2]}$ as ground truth, whereas the actual ground truth should be the entire recommendation list returned by the target model at that point.

Formally, our validation metric is defined as:

$$\text{NDCG}(f_s(\cdot), \hat{L}^k(\cdot)),$$

where $f_s(\cdot)$ denotes the surrogate model's predicted score and $\hat{L}^k(\cdot)$ represents the top-k items recommended by the target modelf_t.

5.2 Attack Performance (RQ1)

5.2.1 Identical Arch. We evaluate model extraction attacks under three data access scenarios: Data-available (Available), Data-free (Free), and Data-limited (Limited), as described in Section 3.1. In the Data-available setting, we evaluate both the target model (**Target**) and a surrogate model trained on B random users from secret data (**Secret**) as baselines.

For synthetic data generation, we compare three methods: random generation (**Random**), DFME [43], and our proposed approach (**Ours**). We also analyze the impact of exposure bias (**Ours-eBias**) and position bias (**Ours-pBias**), detailed in Section 5.4.1 and Section 5.4.2. All experiments use identical model architectures and $B = 5000$ users. Results are shown in Table 2). We use four metrics: NDCG@10 (N@10) and Recall@10 (R@10) for recommendation performance, and Agreement@1 (Agr@1) and Agreement@10 (Agr@10) for extraction performance.

Observation. In summary, our method with debiasing achieves superior MEA performance while maintaining recommendation accuracy. In the data-free setting, it outperforms DFME by 51.90% in Agreement@1 and 37.74% in Agreement@10, and surpasses Random by 22.24% and 10.99% respectively. The data-limited setting shows more consistent improvements, with 7.03% higher Agreement@1 and 7.36% higher Agreement@10 compared to DFME, along with reduced performance variance across scenarios.

Our method significantly outperforms baselines when extracting NARM and SASRec models across both data-free and data-limited settings. For BERT4Rec, while achieving the best performance on Steam, we observe sub-optimal results on Beauty. This performance difference can be attributed to Beauty's sparse user interactions (sparsity of 99.98%) and weak sequential patterns.

To verify this hypothesis, we shuffle the input sequences to eliminate sequential information and compare the changes in the target model's top-100 recommendations. As shown in Table 3, BERT4Rec can make effective recommendations with minimal consideration of sequential order on sparse datasets with short histories like Beauty and Games. This explains why randomly generated sequences without temporal dependencies can achieve competitive MEA performance on Beauty with BERT4Rec.

We also observe that models trained on dense datasets like Steam are more vulnerable to extraction attacks. Notably, the extracted models maintain high performance on the original recommendation task.

5.2.2 Cross Arch. We investigate how MEA performs when the surrogate model architecture differs from the target model. Results

Table 2: Extraction performance under identical model architecture and 5k sequences.

Archs	Threats	Method	Beauty				Games				Steam			
			N@10	R@10	**Agr@1**	**Agr@10**	N@10	R@10	**Agr@1**	**Agr@10**	N@10	R@10	**Agr@1**	**Agr@10**
NARM	Available	Target	0.3564	0.5179	-	-	0.5452	0.7423	-	-	0.6254	0.8465	-	-
		Secret	0.3228	0.4613	0.4765	0.6049	0.5426	0.7397	0.4647	0.6264	0.6263	0.8423	0.7515	0.7655
	Free	Random	0.3167	0.4704	0.4386	0.6258	0.5203	0.7218	0.4041	0.5761	0.6262	0.8484	0.6685	0.6901
		DFME	0.2533	0.3618	0.2388	0.3253	0.5358	0.7273	0.4395	0.6066	0.6165	0.8303	0.7483	0.7478
		Ours-eBias	0.2548	0.3618	0.2907	0.3996	0.5378	0.7293	0.4580	0.6215	0.6160	0.8306	0.7523	0.7457
		Ours-pBias	0.3181	0.4636	0.4450	0.5980	0.5347	0.7285	0.4916	0.6659	0.6212	0.8401	0.7675	0.7678
		Ours	0.3221	0.4672	**0.4957**	**0.6393**	0.5341	0.7275	**0.4998**	**0.6734**	0.6248	0.8431	**0.7723**	**0.7812**
	Limited	DFME	0.2876	0.4124	0.3477	0.4656	0.5395	0.7319	0.4473	0.6157	0.6159	0.8283	0.7474	0.7548
		Ours-eBias	0.2888	0.4110	0.3761	0.4873	0.5408	0.7326	0.4585	0.6279	0.6154	0.8265	0.7509	0.7581
		Ours-pBias	0.3210	0.4633	**0.4992**	**0.6427**	0.5342	0.7295	0.4917	0.6721	0.6243	0.8424	0.7730	**0.7808**
		Ours	0.3108	0.4505	0.4468	0.5919	0.5378	0.7327	**0.5073**	**0.6906**	0.6284	0.8487	**0.7764**	0.7763
SASRec	Available	Target	0.2558	0.4412	-	-	0.3606	0.5878	-	-	0.6110	0.8368	-	-
		Secret	0.2260	0.3720	0.6330	0.7528	0.3492	0.5609	0.6360	0.7497	0.6127	0.8344	0.7556	0.8128
	Free	Random	0.2259	0.3808	0.3829	0.6414	0.2908	0.4775	0.3639	0.5415	0.6174	0.8416	0.6490	0.7204
		DFME	0.1302	0.1973	0.3189	0.5969	0.2969	0.4573	0.4807	0.6080	0.6112	0.8330	0.6999	0.7625
		Ours-eBias	0.1321	0.1949	0.4014	0.6215	0.3002	0.4650	0.5057	0.6315	0.6095	0.8311	0.7339	0.7910
		Ours-pBias	0.2020	0.3327	0.4962	0.6622	0.3305	0.5302	0.5886	0.7008	0.6101	0.8319	0.7486	0.8035
		Ours	0.2131	0.3538	**0.5785**	**0.7200**	0.3321	0.5318	**0.5995**	**0.7121**	0.6038	0.8202	**0.7505**	**0.8073**
	Limited	DFME	0.1610	0.2456	0.5144	0.6745	0.3204	0.5045	0.5169	0.6577	0.6032	0.8200	0.6755	0.7483
		Ours-eBias	0.1654	0.2524	0.5298	0.6565	0.3315	0.5236	0.5399	0.6548	0.6040	0.8218	0.7093	0.7668
		Ours-pBias	0.2136	0.3546	**0.5720**	**0.7130**	0.3394	0.5448	0.6141	0.7098	0.6033	0.8200	**0.7506**	0.8028
		Ours	0.2057	0.3385	0.4680	0.6349	0.3259	0.5247	**0.6219**	**0.7264**	0.6156	0.8394	0.7501	**0.8161**
BERT4Rec	Available	Target	0.3212	0.4937	-	-	0.3890	0.6124	-	-	0.6178	0.8451	-	-
		Secret	0.1990	0.2886	0.6773	0.7053	0.2825	0.4218	0.5972	0.6346	0.6197	0.8414	0.7845	0.7735
	Free	Random	0.1976	0.2911	**0.6541**	**0.7009**	0.3093	0.4654	0.3761	**0.5636**	0.6226	0.8467	0.5027	0.5843
		DFME	0.1395	0.2084	0.1827	0.2439	0.2456	0.3422	0.4608	0.4672	0.6102	0.8309	0.7514	0.7293
		Ours-eBias	0.1504	0.2224	0.1999	0.2856	0.2527	0.3516	**0.5138**	0.5321	0.6075	0.8253	0.7516	0.7308
		Ours-pBias	0.1894	0.2725	0.5812	0.6299	0.2552	0.3578	0.4010	0.4685	0.6129	0.8337	**0.7705**	**0.7541**
		Ours	0.1832	0.2626	0.6299	0.6737	0.2636	0.3705	0.4205	0.5123	0.6120	0.8330	0.7684	0.7535
	Limited	DFME	0.1634	0.2291	0.5926	0.6137	0.2642	0.3757	0.6046	0.6146	0.6065	0.8264	0.7414	0.7177
		Ours-eBias	0.1647	0.2315	0.6188	0.6400	0.2668	0.3787	**0.6204**	0.6218	0.6080	0.8271	0.7538	0.7409
		Ours-pBias	0.1891	0.2758	**0.6549**	**0.7011**	0.2762	0.3970	0.5553	0.5968	0.6154	0.8355	0.7746	**0.7630**
		Ours	0.1835	0.2634	0.5726	0.6169	0.2965	0.4348	0.5657	**0.6474**	0.6165	0.8396	**0.7783**	0.7509

Table 3: Overlap ratio of top-100 recommendations before and after input sequence shuffling. Higher values indicate less reliance on sequential patterns.

	Beauty	Games	Steam
NARM	74.01%	68.30%	70.39%
SASRec	80.06%	66.51%	73.98%
BERT4Rec	98.56%	98.24%	65.70%

are presented as heatmaps in Figure 5, where horizontal and vertical axes denote surrogate and target architectures, respectively.

Observation. MEA achieves optimal performance when the surrogate architecture matches the target architecture, with performance degrading for mismatched architectures. This finding emphasizes the importance of architecture confidentiality in protecting model privacy. Additionally, MEA demonstrates superior performance on dense datasets (Steam) compared to sparse ones, consistent with previous study [43].

5.3 Data Divergence (RQ2)

Since secret datasets represent real user behaviors, we consider surrogate data more authentic when it exhibits smaller divergence from secret datasets. To quantify this divergence, we employ the N-gram Div metric proposed in Section 5.1.5 to measure the divergence between surrogate and secret datasets. We calculate the N-gram Div based on uni-gram and bi-gram (i.e. $N = 1, 2$) across different datasets and architectures. All experiments use identical model architectures and are conducted with 5k sequences. The results are shown in Figure 6, where the y-axis represents the N-gram Div value. A lower value indicates smaller divergence from secret data and thus higher authenticity of the generated data. The N-gram Div provides different perspectives on data authenticity based on the value of N: uni-gram (N=1) evaluates the distribution of individual items, while bi-gram (N=2) focuses on the sequential patterns between adjacent items.

Observation. Our analysis shows that our method significantly reduces the divergence from real data patterns: compared to DFME,

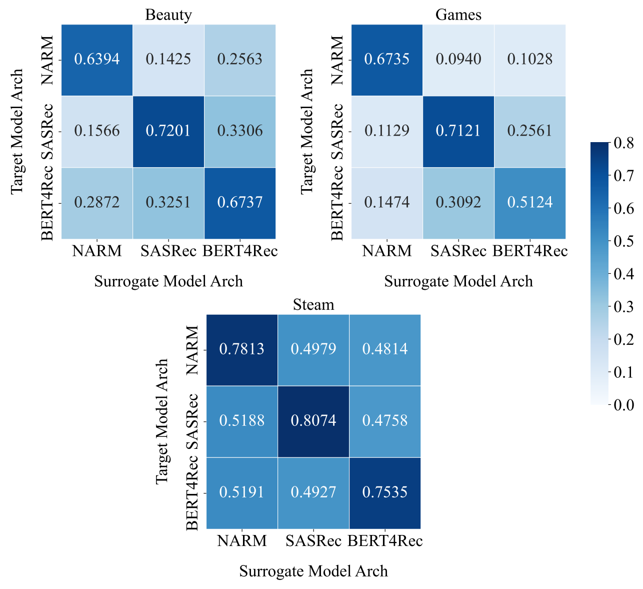


Figure 5: Agr@10 of Model Extraction Attack among different architectures.

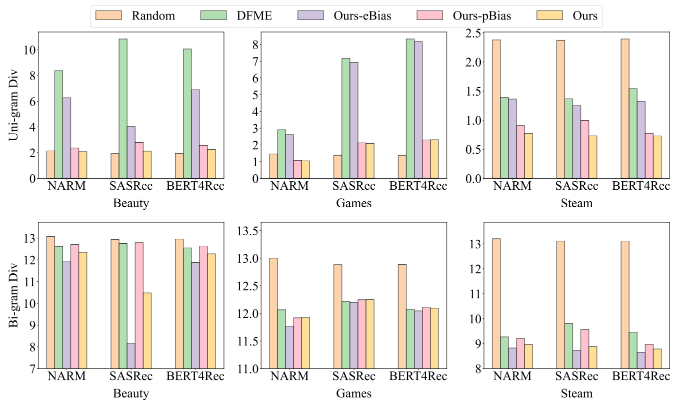
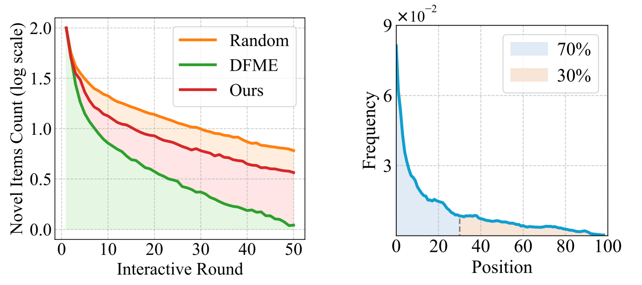


Figure 6: The Uni-gram Div and Bi-gram Div metric between surrogate and secret data in the data-free setting. Lower value means the gap between surrogate and secret data is smaller.



(a) Novel item counts (log scale) over interactions. The rapid decrease indicates a strong exposure bias. Shaded areas show cumulative exposure differences.

(b) Selection frequency across positions from 100 independent items, showing the LLM ranker's position bias through non-uniform distribution.

Figure 7: Analysis of biases: (a) exposure bias in data generation process; (b) position bias in LLM ranker selections.

it achieves 64.98% and 4.74% lower divergence on average for unigrams and bi-grams respectively; compared to random sampling, it shows 10.67% and 16.27% lower divergence on average respectively. It's worth noting that in specific cases (e.g., Beauty dataset with BERT4Rec), while our method shows slightly higher Uni-gram Div than Random, it achieves lower Bi-gram Div, indicating better capture of sequential patterns despite minor differences in individual item distributions.

The relative performance ranking of different data generation methods (Random, DFME, Ours and so on.) remains consistent across different target model architectures when extracting the same dataset. For instance, when extracting models trained on Steam, our method consistently performs best, followed by Ours-pBias, Ours-eBias, DFME, while Random performs worst, regardless of the target model architecture. This consistency suggests that the effectiveness of data generation algorithms G is relatively independent of the target model architecture. Therefore, when the surrogate model shares the same architecture as the target model, the choice of data generation method for MEA should primarily focus on dataset.

5.4 The Impact of Biases (RQ3)

5.4.1 Exposure Bias from Target Model. An ideal data generation algorithm G should be able to fully reconstruct the patterns in secret data. However, in the autoregressive data generation framework, items in generated data can only come from the target model's recommendations. Due to the target model's exposure bias, many items rarely or never appear in the recommendations, which severely limits the diversity of surrogate data and creates a significant gap from secret data.

To empirically demonstrate the exposure bias in the data generation process, we conduct an experiment on the Beauty dataset with NARM architecture. We track the number of novel items which have never been recommended before as the interaction progresses. Figure 7a shows the results, where the x-axis represents the interaction round and the y-axis shows the logarithm (base 10) of the number of novel items. The shaded areas between curves represent the cumulative difference in item exposure: the area between Random and Ours indicates how many more items are exposed in Random's generation process, while the area between Ours and DFME shows our method's advantage in item coverage.

Observation. The results show a clear pattern: the number of novel items in target model's recommendations decreases rapidly as interactions progress, primarily due to the accumulation of previously recommended items. Comparing different generation methods, DFME suffers from the most severe exposure bias, with almost no novel items appearing in recommendations after 50 interaction rounds. Our method significantly mitigates this issue. The Random method, which selects items independently of recommendations, serves as a reference as it is unaffected by the target model's exposure bias.

As shown in Table 2, mitigating exposure bias significantly improves MEA's performance in the data-free setting, with increases of 38.49% and 26.56% in Agreement@1 and Agreement@10, respectively. The only exception is a slight decrease when attacking BERT4Rec trained on the Games dataset. In the data-limited setting, the improvements are more modest (3.23% and 5.53% on average) for two reasons: first, the data generated by the random sampler replaces a portion of the secret data; second, the secret data itself is not affected by the exposure bias inherent in the autoregressive framework. These findings indicate that the debiasing process may partially offset the advantages of utilizing secret data.

5.4.2 Position Bias of LLM ranker. In our framework, the LLM ranker exhibits a natural tendency to favor items appearing in certain positions of the input list, similar to human position bias in recommendation systems. This position bias can affect our data generation process as the LLM ranker repeatedly selects items from the target model's recommendations.

We analyze the position distribution of LLM ranker's selections when presented with 100 independent items from the Beauty dataset. Figure 7b shows the selection frequency across different positions, where the x-axis represents the position index (1-100) and the y-axis shows the selection frequency.

Observation. The results reveal a long-tail distribution in the position of selected items, with items in higher positions having significantly higher selection probabilities. Notably, 70% of the items selected by the LLM ranker are from the first 30 positions in the 100-item list. This strong positional preference, independent of item content, demonstrates the inherent position bias of the LLM ranker.

As shown in Table 2, avoiding position bias improves MEA's performance in the data-free setting, with increases of 5.03% and 4.09% in Agreement@1 and Agreement@10, respectively. When attacking BERT4Rec trained on the Steam dataset, the performance changes after debiasing are little. In the data-limited setting, avoiding position bias does not improve MEA performance, as the secret data is naturally free from the position bias that introduced by LLM ranker in the autoregressive framework.

5.5 Additional Factors Affecting MEA Performance (RQ4)

5.5.1 Using Different LLM as Ranker. Due to GPT-4o-mini's exceptional performance in cost-efficiency and intelligence benchmarks, we employed it as the ranker for recommendation responses. To investigate the impact of different LLMs on data generation and MEA, we conducted experiments using various LLMs with comparable parameter counts to GPT-4o-mini. All experiments are conducted with identical architecture (NARM) and Beauty dataset. The LLMs are Llama-3-8B-Instruct (Meta), Phi-3-Small-8k-Instruct[2] (Microsoft) and Mistral-7B-Instruct-v0.3[3] (Mistral AI).

As shown in Table 4, while GPT-4o-mini achieved the best attack performance as a ranker, the performance gap among different LLMs remained relatively small.

[2]https://huggingface.co/microsoft/Phi-3-small-8k-instruct
[3]https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3

Table 4: Extraction performance using different LLMs under identical model architecture and 5k sequences.

LLM	Params	N@10	R@10	Agr@1	Agr @10
GPT-4o-mini	-	0.3221	0.4672	**0.4957**	**0.6393**
Llama-3-8B-Instruct	8B	0.3166	0.4612	0.4482	0.6032
Phi-3-Small-8k-Instruct	7B	0.3178	0.4623	0.4694	0.6176
Mistral-7B-Instruct-v0.3	7B	0.3166	0.4586	0.4831	0.6337

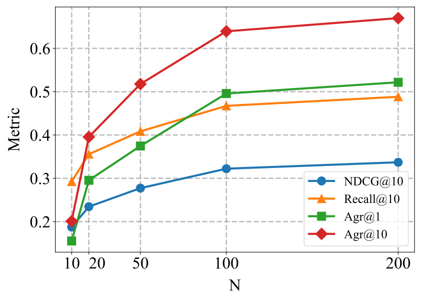


Figure 8: Performance of MEA as the length of the recommendation list n varies. The results demonstrate that increasing n enhances the effectiveness of the attack.

5.5.2 The Length of Recommendation List. A longer recommendation list exposes more information, making the target model more vulnerable to MEA. The variable n represents the length of the recommendation list output by the target model. We conduct experiments with n set to 10, 20, 50, 100, and 200 on the Beauty dataset using the NARM architecture.

As shown in Figure 8, both the recommendation performance of the surrogate model and its consistency with the target model steadily improve as the recommendation list length increases. This demonstrates that exposing more information makes the target model more susceptible to MEA. In real-world scenarios, users rarely browse through 100 or 200 items at once, suggesting that limiting the recommendation list length could serve as a practical defense mechanism against MEA.

6 Conclusion

In this work, we investigate the vulnerability of sequential recommenders to Model Extraction Attacks (MEAs) and propose a novel data generation framework leveraging LLMs as human-like rankers. Our method addresses the key challenge of generating high-quality surrogate data by simulating realistic user-recommender interactions. Through memory compression and preference stabilization modules, along with debiasing techniques, our framework effectively mitigates common issues in synthetic data generation.

Extensive experiments demonstrate that our method significantly outperforms existing approaches in both data quality and attack performance. The generated data shows higher item space coverage and better approximates real user behavior patterns. Our findings highlight the potential of LLMs in improving MEA effectiveness while revealing important insights about recommender systems' vulnerabilities.

Future work could explore the generalization of our framework to other recommendation scenarios and investigate additional defense mechanisms against such attacks. Understanding these vulnerabilities is crucial for developing more robust recommender systems in an increasingly security-conscious environment.

References

[1] Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, and Jian-Guang Lou. 2024. Make Your LLM Fully Utilize the Context. *arXiv preprint arXiv:2404.16811* (2024).

[2] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium (USENIX Security 20)*. 1309–1326.

[3] Yingpeng Du, Di Luo, Rui Yan, Hongzhi Liu, Yang Song, Hengshu Zhu, and Jie Zhang. 2023. Enhancing Job Recommendation through LLM-based Generative Adversarial Networks. arXiv:2307.10747 [cs.IR]

[4] Yue Feng, Shuchang Liu, Zhenghai Xue, Qingpeng Cai, Lantao Hu, Peng Jiang, Kun Gai, and Fei Sun. 2023. A Large Language Model Enhanced Conversational Recommender System. arXiv:2308.06212 [cs.IR]

[5] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System. arXiv:2303.14524 [cs.IR]

[6] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2023. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt and Predict Paradigm (P5). arXiv:2203.13366 [cs.IR]

[7] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[8] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[9] Akshit Jindal, Vikram Goyal, Saket Anand, and Chetan Arora. 2024. Army of Thieves: Enhancing Black-Box Model Extraction via Ensemble based sample selection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3823–3832.

[10] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[11] Pratik Karmakar and Debabrota Basu. 2023. Marich: A query-efficient distributionally equivalent model extraction attack using public data. *arXiv preprint arXiv:2302.08466* (2023).

[12] Yuxuan Lei, Jianxun Lian, Jing Yao, Xu Huang, Defu Lian, and Xing Xie. 2023. RecExplainer: Aligning Large Language Models for Recommendation Model Interpretability. arXiv:2311.10947 [cs.IR]

[13] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1419–1428.

[14] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060* (2024).

[15] Yuan Li, Yixuan Zhang, and Lichao Sun. 2023. Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents. *arXiv preprint arXiv:2310.06500* (2023).

[16] Zelong Li, Jianchao Ji, Yingqiang Ge, Wenyue Hua, and Yongfeng Zhang. 2024. PAP-REC: Personalized Automatic Prompt for Recommendation Language Model. arXiv:2402.00284 [cs.IR]

[17] Jiayi Liao, Sihang Li, Zhengyi Yang, Jiancan Wu, Yancheng Yuan, and Xiang Wang. 2023. LLaRA: Aligning Large Language Models with Sequential Recommenders. arXiv:2312.02445 [cs.IR]

[18] Yiyong Liu, Rui Wen, Michael Backes, and Yang Zhang. [n. d.]. On the Importance of Diversity in Data-free Model Stealing. ([n. d.]).

[19] Yue Liu, Shihao Zhu, Jun Xia, Yingwei Ma, Jian Ma, Wenliang Zhong, Guannan Zhang, Kejun Zhang, and Xinwang Liu. 2024. End-to-end Learnable Clustering for Intent Learning in Recommendation. arXiv:2401.05975 [cs.IR]

[20] Haokai Ma, Ruobing Xie, Lei Meng, Xin Chen, Xu Zhang, Leyu Lin, and Zhanhui Kang. 2024. Plug-in Diffusion Model for Sequential Recommendation. arXiv:2401.02913 [cs.IR]

[21] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 43–52.

[22] Sheshera Mysore, Andrew McCallum, and Hamed Zamani. 2023. Large Language Model Augmented Narrative Driven Recommendations. arXiv:2306.02250 [cs.IR]

[23] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. 2020. Activethief: Model extraction using active learning and unannotated public data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 865–872.

[24] Apurva Pathak, Kshitiz Gupta, and Julian McAuley. 2017. Generating and personalizing bundle recommendations on steam. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1073–1076.

[25] Hao Peng, Shixin Guo, Dandan Zhao, Yiming Wu, Jianming Han, Zhe Wang, Shouling Ji, and Ming Zhong. 2023. Query-efficient model extraction for text classification model in a hard label setting. *Journal of King Saud University-Computer and Information Sciences* 35, 4 (2023), 10–20.

[26] Aleksandr V. Petrov and Craig Macdonald. 2023. Generative Sequential Recommendation with GPTRec. arXiv:2306.11114 [cs.IR]

[27] Mingwen Shao, Lingzhuang Meng, Yuanjian Qiao, Lixu Zhang, and Wangmeng Zuo. 2023. Data-free Black-box Attack based on Diffusion Model. *arXiv preprint arXiv:2307.12872* (2023).

[28] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[29] Arpita Vats, Vinija Jain, Rahul Raja, and Aman Chadha. 2024. Exploring the Impact of Large Language Models on Recommender Systems: An Extensive Review. *arXiv preprint arXiv:2402.18590* (2024).

[30] Sahil Verma, Ashudeep Singh, Varich Boonsanong, John P. Dickerson, and Chirag Shah. 2023. RecRec: Algorithmic Recourse for Recommender Systems. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. ACM. doi:10.1145/3583780.3615181

[31] Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM conference on recommender systems*. 86–94.

[32] Cheng Wang, Jiacheng Sun, Zhenhua Dong, Jieming Zhu, Zhenguo Li, Ruixuan Li, and Rui Zhang. 2023. Data-free Knowledge Distillation for Reusing Recommendation Models. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 386–395.

[33] Lei Wang, Jingsen Zhang, Hao Yang, Zhiyuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, Jun Xu, Zhicheng Dou, Jun Wang, and Ji-Rong Wen. 2023. When Large Language Model based Agent Meets User Behavior Analysis: A Novel User Simulation Paradigm. arXiv:2306.02552 [cs.IR]

[34] Lei Wang, Songheng Zhang, Yun Wang, Ee-Peng Lim, and Yong Wang. 2023. LLM4Vis: Explainable Visualization Recommendation using ChatGPT. arXiv:2310.07652 [cs.HC]

[35] Yan Wang, Zhixuan Chu, Xin Ouyang, Simeng Wang, Hongyan Hao, and Yue. 2024. Enhancing Recommender Systems with Large Language Model Reasoning Graphs. arXiv:2308.10835 [cs.IR]

[36] Zifeng Wang, Chufan Gao, Cao Xiao, and Jimeng Sun. 2023. MediTab: Scaling Medical Tabular Data Predictors via Data Consolidation, Enrichment, and Refinement. arXiv:2305.12081 [cs.LG]

[37] Yunjia Xi, Weiwen Liu, Jianghao Lin, and Xiaoling Cai. 2023. Towards Open-World Recommendation with Knowledge Augmentation from Large Language Models. arXiv:2306.10933 [cs.IR]

[38] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[39] Yaxin Xiao, Qingqing Ye, Haibo Hu, Huadi Zheng, Chengfang Fang, and Jie Shi. 2022. MExMI: Pool-based active model extraction crossover membership inference. *Advances in Neural Information Processing Systems* 35 (2022), 10203–10216.

[40] Youshao Xiao, Shangchun Zhao, Zhenglei Zhou, Zhaoxin Huan, Lin Ju, Xiaolu Zhang, Lin Wang, and Jun Zhou. 2024. G-Meta: Distributed Meta Learning in GPU Clusters for Large-Scale Recommender Systems. arXiv:2401.04338 [cs.LG]

[41] Zhihao Xu, Ruixuan Huang, Changyu Chen, and Xiting Wang. [n. d.]. Uncovering Safety Risks of Large Language Models through Concept Activation Vector. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[42] Fan Yang, Zheng Chen, Ziyan Jiang, Eunah Cho, Xiaojiang Huang, and Yanbin Lu. 2023. PALR: Personalization Aware LLMs for Recommendation. arXiv:2305.07622 [cs.IR]

[43] Zhenrui Yue, Zhankui He, Huimin Zeng, and Julian McAuley. 2021. Black-box attacks on sequential recommenders via data-free model extraction. In *Proceedings of the 15th ACM conference on recommender systems*. 44–54.

[44] Zhenrui Yue, Sara Rabhi, Gabriel de Souza Pereira Moreira, Dong Wang, and Even Oldridge. 2023. LlamaRec: Two-stage recommendation using large language models for ranking. *arXiv preprint arXiv:2311.02089* (2023).

[45] Hui Zhang and Fu Liu. 2024. Few-shot Model Extraction Attacks against Sequential Recommender Systems. *arXiv preprint arXiv:2411.11677* (2024).

[46] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023. Recommendation as Instruction Following: A Large Language Model Empowered Recommendation Approach. arXiv:2305.07001 [cs.IR]

[47] Sixiao Zhang, Hongzhi Yin, Hongxu Chen, and Cheng Long. 2024. Defense Against Model Extraction Attacks on Recommender Systems. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 949–957.

[48] Yang Zhang, Fuli Feng, Jizhi Zhang, Keqin Bao, Qifan Wang, and Xiangnan He. 2023. CoLLM: Integrating Collaborative Embeddings into Large Language Models for Recommendation. arXiv:2310.19488 [cs.IR]

[49] Shiqian Zhao, Kangjie Chen, Meng Hao, Jian Zhang, Guowen Xu, Hongwei Li, and Tianwei Zhang. 2023. Extracting Cloud-based Model with Prior Knowledge. *arXiv preprint arXiv:2306.04192* (2023).

[50] Zhihao Zhu, Rui Fan, Chenwang Wu, Yi Yang, Defu Lian, and Enhong Chen. 2023. Model Stealing Attack against Recommender System. *arXiv preprint arXiv:2312.11571* (2023).

[51] Zhihao Zhu, Chenwang Wu, Rui Fan, Defu Lian, and Enhong Chen. 2023. Membership inference attacks against sequential recommender systems. In *Proceedings of the ACM Web Conference 2023*. 1208–1219.