Opening the Black Box: Interpretable Remedies for Popularity Bias in Recommender Systems

Master's thesis

Parviz Ahmadov

Delft University of Technology

TU Delft

Contents

Publication

This thesis includes material from the following co-authored, peer-reviewed paper:

- Parviz Ahmadov and Masoud Mansoury. *Opening the Black Box: Interpretable Remedies for Popularity Bias in Recommender Systems*. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems (RecSys '25)*, Prague, Czech Republic, Sept 22–26, 2025, pp. 1246–1250. ACM. DOI: 10.1145/3705328.3759310

Relation to this thesis. While the core idea is the same, this thesis broadens adds more datasets, refines the steering logic and hyperparameters, and includes an additional recommender model.

Opening the Black Box: Interpretable Remedies for Popularity Bias in Recommender Systems

by

Parviz Ahmadov

Advisor: Alan Hanjalic
Supervisor: Masoud Mansoury
Faculty: Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

1

Abstract

Popularity bias is a long-standing challenge in recommender systems, where a small set of highly popular items dominates recommendations, while the majority of less popular items are overlooked. This imbalance undermines fairness, decreases recommendation diversity, and negatively impacts users' ability to discover novel or niche content. Although existing mitigation methods address this issue to some extent, they often lack transparency in how they operate - they fix the symptoms without exposing the internal mechanisms that generate the bias, which makes the effects of bias difficult to interpret or control. As modern recommendation models increasingly rely on deep learning based architectures, which are inherently hard to interpret, this opacity has become a fundamental limitation.

This thesis introduces $\mathrm{PopSteer}$, a novel post-processing strategy to analyze and mitigate popularity bias in deep recommender systems that is also interpretable. $\mathrm{PopSteer}$ builds on a Sparse Autoencoder that converts dense embeddings into a sparse feature space where individual neurons align with human-readable features. $\mathrm{PopSteer}$ consists of 3 stages: (i) Sparse Autoencoder training (ii) Neuron analysis stage through synthetic data (iii) Neuron steering stages. In the training stage, a Sparse Autoencoder (SAE) is attached to the hidden representation of a pretrained model to generate a relatively disentangled feature space, where individual neurons correspond to features. In the neuron analysis stage, two synthetic user sets are passed through the SAE, one set favoring popular items and the other favoring unpopular items. Each neuron's alignment with popularity is quantified from the difference in activation between the two sets using Cohen's d. In the neuron steering stage, activations of the most biased neurons are adjusted.

Experimental results show that $\mathrm{PopSteer}$ consistently increases exposure fairness with only minor accuracy degradation compared to state-of-the-art baselines. Effects are stronger when synthetic sets are used in the neuron analysis stage, as opposed to real data, because they better isolate the bias by providing extreme preference patterns. Furthermore, its neuron-level analysis provides insights into how popularity bias emerges within model embeddings, and validates the interpretability of individual neurons. Sensitivity analysis demonstrates that the hyperparameters have a predictable but asymmetric effect on accuracy and fairness. The count of steered neurons matters less than selecting the right neurons, which keeps intervention focused and efficient. Results show that inference and training costs stay modest, indicating deployability. Overall, the results indicate that $\mathrm{PopSteer}$ provides an effective and interpretable way to reduce popularity bias in deep recommender systems, while keeping accuracy loss manageable.

2

Introduction

Recommender systems play a vital role in managing online information overload and improving customer relationship management [84]. By providing users with customized recommendations, these systems improve their online experience on a variety of digital platforms. They are widely used in applications such as movie recommendations for viewers, travel suggestions for tourists, and job recommendations for professionals. From the companies' perspective, the primary goal of these recommender systems is usually to identify pertinent products that suit users' preferences to boost business profits. From the users' perspective, recommender systems help them discover content that was previously unknown to them. Typically, feature engineering techniques based on user preferences, item features, and their interactions (e.g., clicks or reviews) form the basis of recommender systems' working principle.

For the remainder of the section, we will proceed by first summarizing the well-known challenge of "popularity bias" in the current recommender systems research. Then, we discuss the limitations of the existing popularity bias mitigation approaches and provide a brief summary of our proposed solution. Finally, we end the section with the list of research questions our thesis aims to explore.

2.1. Problem definition

Bias in recommender systems is a systematic error that favors certain items or users, often leading to side effects such as reinforcing inequality, limiting discovery, and decreasing user satisfaction [24]. It is typically the byproduct of the imbalanced training data that misrepresents parts of the population [24]. Since recommender systems rely on machine learning algorithms that have been shown to inherently mirror patterns present in their training data, they carry these distortions into (and often amplify them) the recommendations [71].

The primary focus of this thesis is on "popularity bias", one of the prominent types of bias in recommender systems. It refers to the tendency of models to prioritize highly popular items (those frequently interacted with) over less popular, "long-tai" items that might appeal strongly to niche user groups [98]. Popularity bias remains a significant barrier to achieving fairness and optimal performance in recommender systems [10].

The long-tail phenomenon in recommender systems is depicted in Fig. 2.1. The product rank is displayed on the x-axis, while the y-axis shows the quantity of ratings for each item. The top 20 percent of items are separated by the vertical line based on popularity. Their total ratings are significantly higher than those of the 80 percent tail items to the right. These "short-head" items are very popular, like blockbuster movies in a movie recommender system, attracting significantly more user attention.

One practical implication of popularity bias is the "rich-get-richer" effect, where already popular items get even more popular due to increased user exposure stemming from the algorithmic bias in recommendations [12]. An excessive emphasis on popular products can be detrimental to both customers and suppliers [12]. Customers may believe that the suggestions are insufficiently innovative, failing to

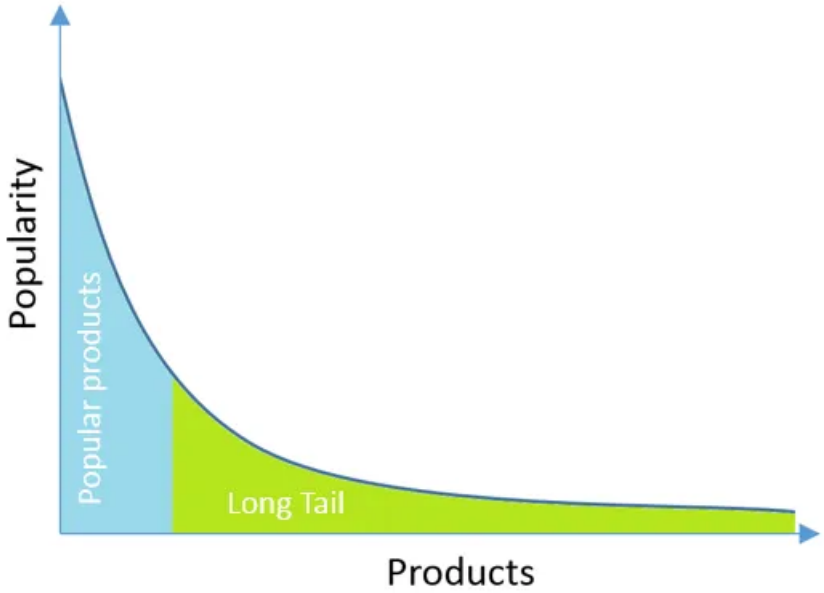


Figure 2.1: The long-tail phenomenon in recommender systems [33]

facilitate discovery. On the other hand, providers miss out on the chance to sell from the long tail by primarily promoting products that customers may have already purchased or consumed, in addition to failing to provide sufficient discovery support.

2.2. Research gap

Although numerous methods have been proposed to mitigate popularity bias, most operate as black-box approaches with limited transparency. As will be discussed in Chapter 3, common strategies include reranking recommendation lists or introducing additional loss terms during training. While these techniques can be effective in reducing bias, they typically obscure the internal mechanisms driving the phenomenon. Consequently, the underlying sources of popularity bias remain hidden, which makes it difficult to diagnose, interpret, or refine the mitigation process.

In recent years, transparency has become a central focus in recommender systems research. Traditionally, recommender system research has emphasized algorithmic accuracy - optimizing prediction quality or ranking metrics. However, as these systems have become more complex and influential, there is growing recognition that accuracy alone is insufficient [112]. Recommenders are increasingly deployed in domains such as e-commerce, news, and entertainment, where they directly shape user choices and preferences. In these settings, transparency plays a crucial role: it allows users to understand why particular items are recommended, helps practitioners detect potential biases, and allows external stakeholders to evaluate whether system behavior aligns with ethical and regulatory standards. A lack of transparency makes it difficult to detect issues such as popularity bias, echo chambers, or unfair treatment of certain user groups, other than decreasing user trust [112].

A major reason for this lack of transparency is the growing reliance on deep neural networks in recommendation models [42, 13, 123]. Neural architectures such as recurrent networks, transformers, or graph neural networks are highly effective at capturing complex patterns in user–item interactions, but their layered, nonlinear computations make it extremely difficult to trace how individual inputs contribute to the final recommendation [128]. This lack of transparency limits our ability to explain why a certain item was recommended or which internal components drive biased behavior.

Interpretability (a more precise term for transparency in the context of deep learning models) in recommender systems offers significant benefits for multiple stakeholders. It can be defined as the ability to understand why a model makes particular predictions, and which internal components are responsible for certain behaviors [127]. From a system designer's perspective, interpretability is valuable for identifying biases, debugging model failures, and improving fairness. From a broader perspective, it

also enables stakeholders to gain confidence that the system's outputs are not only accurate but also aligned with desired ethical and social outcomes.

For example, consider a movie recommendation model that consistently promotes blockbuster films while rarely suggesting independent titles. Without interpretability, this skew might appear as a natural reflection of user preferences. However, an interpretable model could reveal that certain neurons or latent factors are disproportionately sensitive to item popularity rather than user relevance. This insight allows designers to intervene directly, ensuring that recommendations better balance mainstream and niche content while still serving user needs.

In another context, consider recommender systems used by online platforms governed by legislation such as the EU's Digital Services Act [29]. These regulations may require platforms to provide explanations for how recommendations are generated, especially when they influence access to information, culture, or commerce. Without interpretability, compliance becomes difficult, as companies cannot clearly justify why specific items are recommended. However, with interpretable models, regulators and auditors can trace which factors (such as content popularity, user demographics, or engagement metrics) drive recommendations. This approach not only facilitates regulatory compliance but also reinforces accountability of algorithmic decisions to societal and ethical norms.

2.3. Proposed solution

As noted earlier, neural networks are notoriously difficult to interpret. As we will revisit in Section 4.1, this stems from polysemanticity: individual neurons do not map neatly to single concepts but instead entangle multiple features and depend heavily on one another [16]. To address this challenge, we turn to Sparse Autoencoders [93], a technique that has recently gained significant attention.

Sparse Autoencoders are a type of autoencoder trained with additional sparsity constraints, such as $L1$ regularization, to reconstruct input data [93]. The resulting sparsity encourages hidden neurons to become relatively disentangled and specialized. From an interpretability perspective, this property is highly desirable, as specialized neurons can reveal meaningful insights into the model's decision-making process.

With respect to popularity bias, Sparse Autoencoders can serve as a tool for detecting and attenuating popularity signals within the model. To this end, we introduce $\mathrm{PopSteer}$, a post-processing method that leverages Sparse Autoencoders both to interpret internal representations and to mitigate popularity bias in recommender systems through selective *neuron steering*. A more detailed explanation of Sparse Autoencoders and their application in this work is provided in chapters 4 and 5.

2.4. Research questions

Our thesis aims to answer the following research questions:

- **RQ1:** To what extent can neuron-level interventions with $\mathrm{PopSteer}$ mitigate popularity bias in recommender systems while maintaining recommendation accuracy, and how does its performance compare to established baseline methods?
- **RQ2:** How effective is $\mathrm{PopSteer}$ in providing interpretability of deep recommender models by identifying and steering neurons responsible for popularity signals?
- **RQ3:** Which hyperparameter choices of $\mathrm{PopSteer}$ matter most, and do deliberate hyperparameter changes produce consistent, predictable effects on fairness and accuracy?

Each research question aims to evaluate a separate aspect of $\mathrm{PopSteer}$. **RQ1** aims to evaluate the *effectiveness*, **RQ2** evaluates the *interpretability* and **RQ3** measures the *controllability* of $\mathrm{PopSteer}$.

2.5. Thesis structure

This thesis is structured into 9 chapters. Chapter 3 introduces bias and fairness in recommender systems, then narrows to popularity bias and how it can be mitigated. It also reviews prior work on interpretability in recommender systems. Chapter 4 introduces the phenomenon of monosemanticity, provides background on Sparse Autoencoders and how they enable it, and reviews their recent ap-

plications across domains for interpretability and model steering. Chapter 5 contains our methodology, providing an in-depth explanation of each of the stages of the $\mathrm{PopSteer}$ pipeline (Training, interpretation, mitigation). Chapter 6 consists of experimental details, such as datasets, choice of backbone recommender system, baselines, evaluation metrics, and other implementation specifics. Chapter 7 contains the results of the experiments, and Chapter 8 contains the detailed discussion of the results, including limitations and future work. Chapter 9 concludes with a summary of the study's main findings.

3

Related Work

Recommender systems can be broadly categorized into collaborative filtering-based models and content-based models [108]. Collaborative filtering-based systems make recommendations by analyzing user interactions, such as ratings or purchase histories, to identify patterns and similarities among users or items. For example, if user A and user B have similar viewing histories, a collaborative filtering system might suggest shows to user A based on what user B has watched. The content-based filtering approach, on the other hand, recommends items by comparing the content features of items with a user's preferences. For instance, if a user has shown interest in action movies, the system will recommend other action movies by analyzing item descriptions and metadata. There are also hybrid systems that integrate both methods to leverage their respective strengths.

In recent years, deep-learning-based recommender systems have gained widespread adoption due to their ability to capture complex user-item interaction patterns. Some notable advancements include:

- **Neural-Collaborative Filtering:** This method uses neural networks to model nonlinear relationships in user-item interactions and provides better representation learning than traditional CF-based methods [54].
- **Transformer-Based Sequential Recommenders:** Transformer architectures like Self-Attentive Sequential Recommendation (SASRec) [64] have become popular due to their ability to capture long-term dependencies more effectively than RNN-based recommenders.
- **Knowledge Graph-Based Recommenders (KGRec)**: Integration of Knowledge Graphs (KGs) has been one of the latest advancements in the field of recommender systems. Knowledge Graphs integrate structured semantic relationships between items and users, improving recommendations through graph-based reasoning [113].

Despite their success, modern recommender systems still suffer from bias and fairness issues. This chapter first surveys fairness and bias in recommender systems, then examines popularity bias and mitigation techniques, and finally reviews research on interpretable recommender systems.

3.1. Bias and fairness in recommender systems

The lifecycle of a recommender system can be categorized into (i) data collection, where relevant data is gathered, (ii) model learning, which represents the learning of the model using the collected data, and (iii) serving, which returns the recommendation results to users [24]. In this lifecycle, multiple biases may emerge either from the data collection or the model training stages [24].

Chen et al. define *data bias* as the mismatch between the distribution from which the training data is collected and the ideal, unbiased test distribution that reflects users' true preferences [24]. The ideal test distribution represents complete and unbiased user behavior, and the training data typically deviates from it due to various inherent biases in the data. For instance, selection bias arises because users freely choose which items to rate, so observed ratings may not represent their full preferences [24].

Exposure bias occurs because users can only interact with items they have been exposed to, so interaction is not a strong signal for liking. Other famous forms include conformity bias, where users imitate others' opinions rather than relying on their true preference [83], and position bias, where users tend to interact with items ranked higher in the recommendation list [28].

Bias may also arise from the model itself, which is, in fact, natural. The goal of a recommender model is to learn patterns from the training data that generalize to unseen data. Achieving such generalization requires making certain assumptions about the data, without which learning would be impossible. This phenomenon is known as *inductive bias* [24]. In machine learning, this relates to the well-known bias–variance tradeoff, which suggests that some degree of bias is necessary for good generalization, and eliminating bias entirely would lead to overfitting on the training data.

The potential implication of the existing bias in the data and the model is unfairness in the recommendation outcomes. Unfairness arises when a system systematically disadvantages certain individuals or groups [24]. For instance, if specific demographic attributes such as gender or race are underrepresented in the training data, this imbalance is likely to be reflected in the generated recommendations. As an example, prior research has shown that women encountered fewer ads about high-paying jobs and career coaching services than men in a job recommendation platform due to gender imbalance in the dataset [30]. Popularity bias is a specific type of fairness issue, where already popular items receive disproportionate exposure. This thesis is mainly centered on the elimination of popularity bias. In the next section, we discuss this issue in recommender systems in detail.

3.1.1. Popularity bias in recommender systems

Popularity bias often emerges through interactions with other biases [24]. For example, exposure bias can trigger a rich-get-richer dynamic, giving some items disproportionately more visibility. These effects may be further exacerbated during training due to the model's inductive bias, which favors signals from frequently interacted items. The result is popularity bias, where a subset of items receives disproportionately more exposure.

There is a large body of literature studying popularity bias in recommender systems. Celma et al. provided their analysis in the music domain and concluded that collaborative filtering is highly prone to popularity bias [22]. Abdollahpouri et al. approached the problem from a user-centric lens [2]. They categorized users into groups based on their personal interest in popular vs. niche items [2]. The results showed that most algorithms favored recommending popular items, even for users who had shown strong interest in non-popular movies. Abdollahpouri et al. argue this is a fairness problem, since the system is not treating users' preferences equitably. Lesota et al. focused on measuring popularity bias more comprehensively and investigating its impact on different user demographics [78]. Importantly, they observed that algorithms that were popularity skewed in their recommendations did not necessarily perform worse on accuracy. This challenges a common assumption that accuracy always demands recommending mostly popular items.

Popularity bias has also been extensively studied in terms of embedding space. Chen et al. provide theoretical and empirical proof that popular items' embedding norms grow faster than tail items during training, which amplifies popularity bias [23]. Kim et al. confirm this in a study where they show that item embedding magnitude correlates strongly with item popularity [68]. Zhang et al. found that user and item embeddings cluster in a few regions and user clusters align with popular-item clusters [125].

Popularity bias has several potential negative implications. As discovered by Abdollahpouri et al., popularity bias can diminish the experience of niche-loving users [2]. In practice, mitigating popularity bias can result in a healthier user experience and keep users around longer, as discovered by Hansen et al. [49]. They found that users who consumed a more diverse set of artists and genres had better long-term outcomes for the platform - more engagement, higher retention.

Other than users, popularity bias has huge implications for providers too. It may result in small sellers getting entirely eclipsed by incumbents [34], leading to unequal opportunity. Notably, through the agent-based simulation, Zhang et al. found that algorithms that greedily recommend popular items maximize short-term clicks or satisfaction, but they can reduce consumer welfare and platform utility in the long run, a phenomenon they termed "performance paradox" [121]. The simulation showed that balancing recommendations led to a more sustainable outcome: users continued to find relevant content, and

producers had more reasons to offer diverse items. This work shows that popularity bias isn't just a fairness issue, but it poses a risk to the long-term health of the recommender ecosystem [121].

3.2. Mitigation techniques

There is a huge body of literature on the topic of mitigating popularity bias. Popularity bias mitigation strategies can be split into three categories: "pre-processing", "in-processing", and "post-processing" techniques [71]. "Pre-processing" techniques involve filtering the dataset before feeding it into the network to remove any underlying biases in the data. In in-processing approaches, the training procedure is responsible for bias removal, commonly through regularization techniques. In post-processing approaches, the model output that is optimized for accuracy is modified to remove bias, through, for instance, re-ranking techniques to improve the ranking of less popular items.

Pre-processing techniques are relatively uncommon and are often paired with in-processing approaches [71]. Common pre-processing techniques involve data sampling, item filtering, or tailored methods for generating positive-negative sample pairs for training. Seki et al. pre-process the dataset to create a more balanced dataset, for a variety of purposes, including addressing popularity bias [104]. Rather than removing data, another pre-processing approach is to extend the data with additional information beyond interaction data. This could involve integrating implicit and explicit feedback [60], utilizing user-related data from outside sources like social networks [80], or adding particular item metadata for multi-modality [20]. Park et al. propose another approach, where the long-tail items are supplemented with additional data [98]. They achieve this by clustering tail items for recommendation while handling head items individually based on their ratings.

In-processing techniques are most common in the literature for mitigating popularity bias [71]. The regularization-based approach is highly popular, where the additional term is added to the loss function with the role of penalizing when popular items are recommended. Abdollahpouri et al. [1] propose adding a term to the optimization function for penalizing the recommendation of popular items in learning-to-rank approaches. Boratto et al. [14] and Zhu et al. [133] introduced correlation-based regularization methods that penalize high relevance scores when they are mainly driven by item popularity. Wang et al. apply a constraint-based method, using predefined rules to maintain similar coverage for items with shared characteristics in recommendations [115]. For graph-based recommender systems, Graph-based Similarity Adjustment is typically used, which redefines the item or user similarity [71]. Chen et al., for instance, suggest using an alternative to cosine similarity, which is typically used as a similarity measure in graph-based collaborative filtering algorithms [23]. Another common way to address popularity bias is by using natural language processing techniques on user reviews or item metadata. Li et al. utilize an autoencoder with text reviews to refine user and item representations, aiming to improve recommendation quality for all user groups, regardless of their mainstreamness [80].

The main forms of post-processing in the literature are re-scaling (score adjustment), re-ranking (reordering), and rank aggregation [71]. Re-scaling, also known as score adjustment, modifies relevance scores in a recommendation list to counteract popularity bias by boosting some items while lowering others. Zhu et al. use this method by adding a compensation term to the scores to balance item popularity while maintaining relevance in recommendations [133]. The re-ranking approach aims to re-rank the items to optimize a certain pre-defined metric. For instance, the approach by Abdollahpouri et al. attempts to balance between relevancy and popularity with a parameter with the role of prioritizing either of the two [3]. The rank aggregation technique works by combining popularity-driven ranking with alternative ranking to balance various ranking signals. For instance, Dong et al. propose merging an existing ranking with its inverse using the Two-Way Rank aggregation method [35].

3.3. Interpretable recommender systems

Interpretability in recommender systems seeks to expose the reasoning behind a model's predictions, enabling better trust, transparency, and actionable debugging [127, 82]. Approaches broadly fall into *post-hoc* methods, which explain an existing black-box model's outputs without altering its architecture [124, 61], and *intrinsically interpretable* models, whose structure is designed to be directly understandable [111, 25].

Post-hoc interpretability methods treat a recommender model as a black-box and generate explana-

tions without changing its internal structure. These methods are usually model-agnostic, since they provide interpretability based solely on model inputs and outputs rather than its internal parameters. Model-agnostic strategies often rely on surrogate models that approximate the behavior of a complex recommender in a more interpretable form. A prominent example is LIME-RS, proposed by Nóbrega and Marinho, which perturbs a user's input features and fits a simple regression model locally around the recommendation of interest, revealing which factors most strongly influenced the output in that neighborhood [94]. In contrast, global surrogates attempt to capture the overall decision logic of the recommender by training an interpretable model, such as a shallow decision tree or linear classifier, on the predictions of the deep system [77].

Other post-hoc strategies include feature attribution and counterfactual reasoning. Feature attribution methods assign importance scores to inputs or latent features based on gradients or perturbations [132, 15]. They show why a recommendation was made by identifying which past interactions or item properties most influenced the model's output. Counterfactual methods, in contrast, focus on how the outcome could change by identifying the smallest modifications, such as adding a specific interaction or adjusting an attribute that would change the recommendation [106, 44]. These methods can be applied post-hoc to deep learning recommenders, though their explanations may be approximate and require careful design to remain meaningful.

Intrinsically interpretable deep models incorporate the explanation mechanism directly into their architecture, so that each prediction is generated alongside human-understandable reasoning. A common approach is to integrate attention mechanisms or latent factors that automatically reveal important input features, making the model's decision process more transparent [111, 8]. For instance, Barkan et al. [8] propose Attentive Multi-Persona Collaborative Filtering (AMPCF), a model that represents each user through several personas that reflect different aspects of their preference. By adjusting the weight of these personas, the model creates flexible user representations and generates recommendations that can be explained by pointing to the most influential persona. In a related direction, Chen et al. explore visual explanations by combining product images with reviews and regional features. Their attention mechanism reveals specific image regions that match user preferences, making it clear which visual aspects influenced the recommendation [26]. Likewise, deep models that leverage text (reviews, descriptions) often use hierarchical attention to identify words or sentences that justify the recommendation, highlighting key snippets of text as explanations [36, 114].

Knowledge-driven interpretability approaches further improve intrinsic transparency by leveraging side information. For instance, Path-based deep recommender models incorporate knowledge graphs or network schemas to output reasoning paths as explanations: they identify sequences of connected entities associating a user with the recommended item (e.g., User → Genre → Director → Item), which serve as intuitive justifications for the recommendation [41, 58]. Other models tie recommendations to explicit user and item attributes by learning latent representations aligned with properties like genre, brand, or topic preferences [57, 85, 129]. Using techniques such as LSTM encoders, these systems can output explanations like "recommended because you enjoy [aspect]" grounded in the model's learned attention to those attributes [85].

Prior work treats *popularity bias mitigation* and *interpretability* largely as separate threads. Most mitigation methods optimize accuracy-fairness trade-offs, but do not expose the internal mechanisms that strengthen bias. Conversely, many interpretable recommenders explain outputs with rationales from the input side (e.g., features, items, or paths) rather than identifying and modifying the model's internal components. To the best of our knowledge, there is limited work that (i) ties explanations to concrete model components (e.g., neurons) in deep-learning-based recommender systems and (ii) uses those components for post-hoc control of popularity bias. This thesis addresses that gap by analyzing neuron-level structure with a sparse autoencoder and steering a trained model's behavior through modifying identified neurons, resulting in interpretable and controllable reductions in popularity bias.

4

Background

In this chapter, we explore the problem of polysemanticity in depth, exploring its root causes and potential mitigation strategies, with a particular focus on Sparse Autoencoders. We then explain the inner workings of Sparse Autoencoders and review existing research that leverages them for interpretability.

4.1. Achieving monosemanticity

Mechanistic interpretability aims to comprehend neural networks by dissecting them into simpler, more interpretable components [16]. By understanding the small individual components, we are hopefully able to reason about the behavior of the entire network. Unfortunately, the smallest dissectible part of the neural network, the neuron itself, is usually uninterpretable for humans. That is due to the phenomenon called “polysemanticity” - rather than encoding an individual interpretable concept, each neuron usually responds to unrelated inputs. For instance, a single neuron may fire for faces of cats and fronts of cars [96]. The major cause of polysemanticity is superposition [6, 45, 97, 39]. Since there are usually more concepts than neurons, the model assigns each concept a linear combination of neurons, leading to a highly entangled neuron space. Fig. 4.1 illustrates this concept.

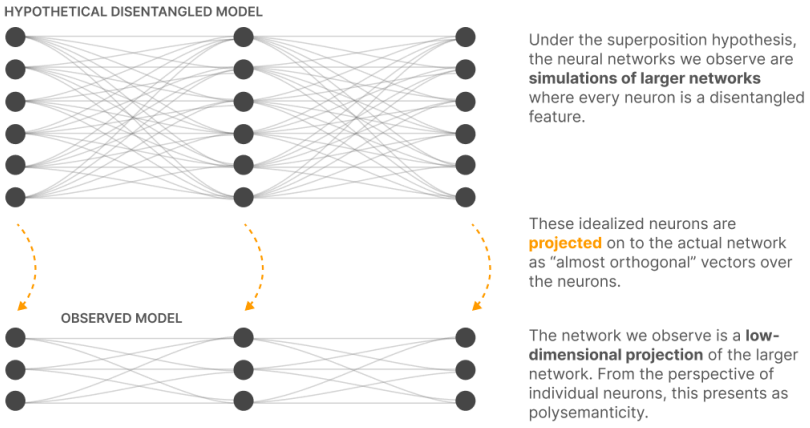


Figure 4.1: The illustration of the concept of superposition [16]

Elhage et al. have attempted several strategies to make the neurons less polysemantic [39]. One is to engineer models to simply not have superposition in the first place. However, they found that even when superposition is removed (there are fewer features than neurons), the neurons may still not become cleanly interpretable. This occurs because, in many cases, models can reduce loss more effectively by encoding multiple features ambiguously within a polysemantic neuron rather than isolating a single feature and disregarding the rest.

To illustrate the idea, consider a simple example [16] - Assume we have a simplified model with a single neuron trained on a dataset containing four mutually exclusive features (A, B, C, D). Each feature

corresponds to a unique and correct prediction for the next token. The neuron's activation is binary: it either fires or does not. When it does fire, it produces an output vector representing the probability distribution over the possible next tokens.

The cross-entropy loss can be analyzed in the following cases:

1. If the neuron fires only in response to feature A, it correctly predicts token A in that case. For inputs corresponding to features B, C, or D, the neuron does not fire, and the model outputs a uniform distribution over these three tokens. The resulting cross-entropy loss is
$$\tfrac{1}{2}\ln 3 \approx 0.8.$$
2. Alternatively, imagine the neuron fires for both features A and B. In this case, the model outputs a uniform distribution over tokens A and B whenever one of these features is present. For the remaining cases (features C and D), the model outputs a uniform distribution over C and D. The resulting cross-entropy loss is
$$\ln 2 \approx 0.7.$$

Because the loss in case (2) is lower than in case (1), the model performs better when the neuron responds to multiple features. This results in the neuron becoming polysemantic, even though no true superposition exists. These findings suggest that when trained with cross-entropy loss, models generally prefer encoding many features polysemantically [16]. They do so rather than keeping a smaller set of neurons monosemantic. Notably, this is true even in situations where sparsity rules out genuine superposition.

4.1.1. Features as a decomposition

The difficulty of interpreting neurons directly has motivated researchers to focus on *directions* in activation space instead. Empirical evidence suggests that many such directions are meaningful [91, 79]. For instance, Mikolov et al. demonstrated linear "vector arithmetic" structure in word embeddings [91], and similar results have been observed across other latent spaces [101]. Interpretable units have also been studied extensively in RNNs [66], CNNs [21], and GANs [9], although important caveats remain. More recent work [92, 17] develops this perspective further.

One of the most well-known demonstrations of such linear structure in word embeddings is the analogy between the words *man*, *king*, *woman*, and *queen* [91]. The relation between *man* and *king* is approximately the same as that between *woman* and *queen*. To put it in other words, the vector difference $\mathbf{w}_{\text{king}} - \mathbf{w}_{\text{man}}$ encodes a "royalty" direction, which can be added to $\mathbf{w}_{\text{woman}}$ to recover $\mathbf{w}_{\text{queen}}$. This analogy is an example of how interpretable concepts can emerge as linear directions in activation space, rather than being tied to individual neurons.

If linear directions can be meaningfully interpreted, it is natural to think of them as forming a basic set of features, from which more complex behaviors are composed. In some cases, individual neurons align with these features. More commonly, however, the relevant directions cut across neurons. Thus, rather than treating neurons as the fundamental units, we can view activations as sparse linear combinations of such features [16].

Formally, let $x_j \in \mathbb{R}^d$ denote the activation vector of a given datapoint. We can approximate it as

$$x_j \approx b + \sum_i f_i(x_j) d_i, \tag{4.1}$$

where $f_i(x_j)$ is the activation of feature i, d_i is the corresponding direction in activation space, and b is a bias term [16].

A Sparse Autoencoder is a powerful tool for learning the sparse basis that Eq. 4.1 describes, which will be discussed in detail in section 4.2 [93].

4.1.2. What makes a good decomposition?

For such a feature-based decomposition to be useful in practice, it must satisfy a few conditions [16]. First, a feature should be interpretable: we should be able to describe the set of datapoints j where $f_i(x_j)$ is large in a way that generalizes across diverse data. Second, the *downstream effects* of activating and deactivating the feature should be consistent with its input-side interpretation. Third, the decomposition should *explain* a significant share of the layer's behavior, which, for instance, can be measured by reconstruction quality.

When these criteria are met, feature decompositions result in several capabilities [16]:

- attributing the contribution of individual features to a layer's output,
- monitoring for the activation of particular features (e.g., safety-relevant ones),
- intervening on activations to predictably alter downstream behavior,
- validating that the model has learned and is using certain properties of the data
- designing inputs to elicit specific features.

In short, feature decomposition provides a backbone for mechanistic interpretability: it transforms uninterpretable neuron activations into a more structured basis.

4.2. Sparse Autoencoders

Bricken et al. argue the importance of 2 characteristics for a good decomposition: *overcompleteness* and *sparsity* [16]. This is essentially sparse dictionary learning, which is a technique that learns a collection of basis vectors and represents each input as a sparse combination of them [73]. The importance of overcompleteness stems from the superposition issue discussed in Section 4.1. We want to provide the model with more axes to represent the input for it to be able to effectively separate the subspace into features. On its own, overcompleteness makes the problem unsolvable: we are trying to recover a high-dimensional feature vector from a lower-dimensional input. Since the number of variables exceeds the number of observations, the system has infinitely many solutions. This makes the *sparsity* constraint essential. If we impose a sparsity constraint on the high-dimensional vector, only a small percentage of neurons are nonzero for any given input. This turns the problem from "impossible" into "tractable," since the sparse solution is often unique and recoverable. This is equivalent to the well-known problem of compressed sensing, which is NP-hard [37]. Compressed sensing is a technique that allows signals to be captured and reconstructed efficiently, even when the system of linear equations is underdetermined, through utilizing the "sparsity" property of signals [37].

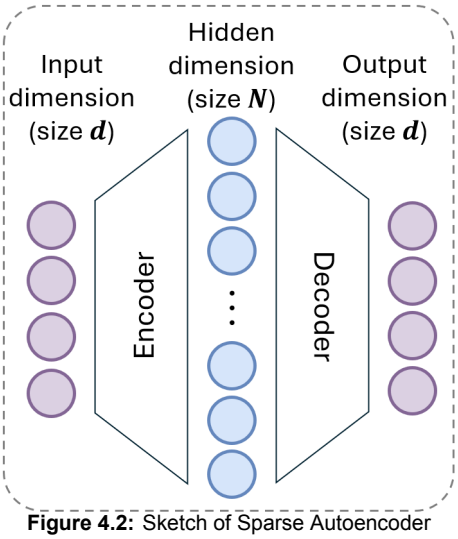


Figure 4.2: Sketch of Sparse Autoencoder

One potential solution is the Sparse Autoencoder [93], which we will explore in this thesis. Sparse Autoencoders are a specific type of autoencoder with the sparsity constraint imposed. Autoencoders

are a class of neural networks designed to learn compressed representations of data [7, 56]. These models consist of an *encoder* that maps the input into a lower or higher dimensional latent space, and a *decoder* that reconstructs the input from this latent representation. By training the network to minimize the reconstruction error, the autoencoder learns latent features that capture the most salient features of the input, often revealing underlying structure in the data. While classical autoencoders can reveal structure in the input data, their hidden units often exhibit dense, overlapping activations, making the learned features difficult to interpret and sometimes redundant [74]. Variants of the basic autoencoder introduce constraints or regularization to shape the latent space in desirable ways, such as denoising autoencoders for robustness or variational autoencoders for probabilistic modeling.

Fig. 4.2 provides a sketch of the Sparse Autoencoder. The Sparse Autoencoder transforms the input vector into the intermediate representation, which can be a higher, lower, or equal-dimensional vector. The intermediate representation is then converted back into the input dimension. Both transformations are done through neural networks, and the reconstruction loss ensures that the output vector is as close to the input vector as possible.

In a Sparse Autoencoder, the decomposition of input vector $\mathbf{x} \in \mathbb{R}^d$ can be written as:

$$\hat{\mathbf{x}} = \mathbf{b}^{\text{dec}} + \sum_{i=1}^{N} f_i(\mathbf{x}) \mathbf{W}^{\text{dec}}_{,i}$$

where $\mathbf{W}^{\text{dec}} \in \mathbb{R}^{d \times N}$ are the learned decoder weights of the SAE, $\mathbf{b}^{\text{dec}} \in \mathbb{R}^d$ are the learned biases, and $f_i(\mathbf{x})$ denotes the activity of feature i. Feature activations are given by the output of the encoder. Note that this equation is equivalent to the equation 4.1 (with only notations renamed), which provides the general formulation for the decomposition of inputs into features. In a Sparse Autoencoder, the aim is to have neurons as features (note that here, we refer to the neurons in the hidden dimension of the SAE). Hence, $f_i(\mathbf{x})$ corresponds to the activation value of neuron i in the hidden layer of the SAE.

Concretely, the feature activations are the output of the encoder.

$$f_i(\mathbf{x}) = \text{ReLU}\big(W_e(\mathbf{x} - \mathbf{b}_d) + \mathbf{b}_e\big)_i,$$

where W_e is the weight matrix of the encoder and $\mathbf{b}_d, \mathbf{b}_e$ are the pre-encoder bias and the encoder bias. The feature directions are the columns of the decoder weight matrix W_d.

The *overcompleteness* constraint can be achieved by setting N to be significantly larger than the input dimension d. For instance, Bricken et al. experimented with the scale size s ranging from 8 to 256, where s is defined by N/d. A popular way of enforcing the sparsity constraint is through a loss function. This can be achieved through adding an $L1$ loss term to the optimization function:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}} \left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \lambda \sum_i f_i(\mathbf{x}) \right]$$

The first term is the reconstruction loss, and the second term is the $L1$ penalty term on feature activations. The model is encouraged to set most of the feature activations to 0 for a given input to minimize loss. λ is a hyperparameter controlling the tradeoff between sparsity and reconstruction accuracy. As we will see in the following sections, there are multiple variations of Sparse Autoencoders, offering different strategies for achieving sparsity.

Intuition. Overcompleteness gives the model a large menu of candidate "feature directions". Sparsity ensures that, for any particular input, it selects only a few of them. You can think of each column of W_d as a knob that adds a specific pattern to the reconstruction. Without sparsity, the model could turn many knobs a little and hide information in diffuse, overlapping ways. With sparsity, it must explain the input using just a handful of knobs, which pushes the learned directions to become sharp, distinct, and semantically meaningful.

The combination of overcompleteness and sparsity naturally drives neurons to become more disentangled. Because only a few units are allowed to be active for any given input, each neuron is pressured to capture a distinct, non-overlapping aspect of the data. If two neurons redundantly represent the same feature, the sparsity penalty will suppress one of them. Overcompleteness ensures there are enough candidate neurons to assign one to each meaningful direction, while sparsity prevents them from firing simultaneously in tangled combinations. As a result, neurons become easier to interpret and modify individually, since adjusting the activity of a single unit cleanly changes one feature without inadvertently affecting many others.

Reconstruction quality-sparsity tradeoff. The additional sparsity constraint inevitably causes a decrease in reconstruction accuracy. Therefore, there is a tradeoff between extracting interpretable features and sacrificing accuracy. The key is to find the right balance, where the acceptable level of interpretability is achieved without a significant drop in model performance.

Advantages of Sparse Autoencoders. Bricken et al. list a few advantages of sparse autoencoders over other existing dictionary learning methods [16]. First, this approach is scalable to very large datasets. Second, other methods are typically "too strong" in the sense that they aim to recover features from the activations that the underlying model itself does not have access to. Compressed sensing on its own is NP-hard, neural networks only aim to provide an approximation of the solution.

4.3. Prior work on Sparse Autoencoders

Sparse autoencoders have been traditionally used as a feature learning and dimensionality reduction technique [100, 32, 99, 105]. Recently, they have been tested on LLMs with promising results. Bricken et al. found a number of interpretable neurons on a small-scale transformer-based language model. For instance, neurons were found that become highly active on Arabic, Hebrew, base64, or DNA scripts. The idea was scaled to high-dimensional LLMs (Claude 3 Sonnet and GPT-4) with success [43, 107]. In Claude Sonnet 3, more high-level features were extracted, such as "Golden Gate Bridge", "Brain sciences", "Monuments and popular tourist attractions" [107].

Since then, there have been many works attempting to extract high-level features from LLMs [95, 75, 31, 59]. O'Neill et al. extract features from the Sparse Autoencoder trained on the abstracts of $420,000$ papers and introduce a novel approach for identifying "feature families" that represent related concepts [95]. Lan et al. find significant similarities in SAE feature spaces across various LLMs, showing that Sparse Autoencoders reveal universal feature spaces across different models. Demircan et al. find that Llama 3 70B performs in-context reinforcement learning, internally representing TD errors and Q-values, which can be identified and modified using sparse autoencoders (SAEs) [31]. Inaba et al. compare the learned features across multiple checkpoints, and find that the model learns increasingly complex and abstract features [59].

The ability of Sparse Autoencoders to isolate features makes them a great tool for detecting and eliminating unwanted features like "toxicity". Harle et al. encourage a single neuron to align with the "toxicity" concept in the training of the Sparse Autoencoder and later manipulate the neuron to adjust the toxicity level [50]. Kang et al. control the retrieval behaviour of LLMs via manipulating the latent sparse features, for example, prioritizing documents from specific perspectives in the retrieval results [63]. Zhao et al. propose SPARE, a training-free sparse autoencoder–based representation engineering method that steers large language models to resolve context-memory conflicts at inference time [131]. Hegde et al. attempted to detect components in the Sparse Autoencoder responsible for gender bias and suppressed their activation for bias mitigation [55].

Sparse autoencoders have also been applied beyond LLMs for steering and interpretability, in a variety of non-LLM domains. Kim et al. used Sparse Autoencoders on Diffusion models to steer the generation away or toward a given concept like "nudity" [67]. King et al. apply Sparse Autoencoders on geophysical models, extracting several interpretable features [69]. Adams et al. generate hypotheses for biological mechanisms using Sparse Autoencoder features [4].

Since there has been a rising interest in the use of Sparse Autoencoders for interpretation, multiple versions of the sparsity mechanism have been proposed [102, 103, 18, 19, 43]. Rajamanoharan et al. proposed two different SAE variants, Gated SAEs [102], and JumpReLU SAEs [103], each achieving

a Pareto improvement in the sparsity–fidelity trade-off (i.e., higher reconstruction fidelity at a given sparsity). Bussmann et al. introduced BatchTopK SAEs [18], a variant of TopK SAEs [43] in which the top-k activation constraint is enforced across each batch rather than on individual tokens to give the model more flexibility. Recently, Matryoshka SAEs have been proposed by Bussmann et al., which allow retaining interpretable features at different levels of abstraction through multi-layered SAE [19].

There has been some prior work on utilizing Sparse Autoencoders in Recommender Systems [62, 5, 81, 110]. However, most of them use the Sparse Autoencoder for denoising or handling sparse data for improving recommendation accuracy, with minimal focus on interpretability. Wang et al. used a Sparse Autoencoder for interpreting the internal states of a sequential recommender system. However, their findings suggest that their interpretation is only slightly better than random guessing [110]. We attribute this to two factors. First, their model received only IDs as input, without any descriptive features. Sparse Autoencoders are effective for interpreting text data because text is inherently human-readable, making it easier to analyze the learned representations and their underlying structures. Determining the features extracted from IDs, on the other hand, is more challenging. Second, sparsity was only slightly enforced, perhaps to avoid a significant accuracy drop.

Klenitskiy et al. apply Sparse Autoencoders to sequential recommender models, showcasing their potential for both interpretation and control [70]. By training SAEs on transformer-based recommenders, they extract semantically meaningful features such as genres, languages, and even specific franchises. Importantly, they show that steering these neurons leads to predictable changes in recommendation outcomes, enabling fine-grained control. This work is largely exploratory, but it provides evidence of SAEs as a promising tool for making recommender systems more transparent and controllable.

Despite these advances, the application of SAEs within recommender systems remains largely under-explored. To the best of our knowledge, no prior research has investigated their use specifically for mitigating popularity bias.

5

Methodology

This chapter describes the proposed solution for using a Sparse Autoencoder for the mitigation of popularity bias.

5.1. PopSteer overview

In this section, we provide a high-level explanation of $\mathrm{PopSteer}$, our method for interpreting and mitigating popularity bias in recommender systems. $\mathrm{PopSteer}$ is a post-processing approach, meaning it can be applied on top of any pretrained recommender model without altering its architecture.

$\mathrm{PopSteer}$ is a pipeline for interpreting and mitigating popularity bias. We first train a base recommender, then attach a Sparse Autoencoder (SAE) to the final layer and train the SAE to reconstruct that layer's activations (Section 5.2). The trained SAE provides a neuron-level interface for analysis and control. In the analysis stage (Section 5.3), we probe the SAE with synthetic profiles that reflect extreme preferences for popular vs. unpopular items to identify neurons encoding popularity signals. In the mitigation stage (Section 5.4), we steer those neurons by adjusting their activations, yielding updated user embeddings used for ranking. See Figure 5.1 for an overview and Sections 5.2-5.4 for details.

5.2. Sparse Autoencoder setup and training

In Section 4.2, we provided a high-level overview of the Sparse Autoencoder. In this section, we present its mathematical formulation and offer a detailed explanation of its training details, such as key hyperparameters and the associated loss function.

SAE attachment. After the recommender model is trained, the SAE is attached to the last layer of the recommender model for reconstruction to extract meaningful features. In our experiments, we chose the final layer of the base recommender system for attaching the SAE. This choice is natural because, in collaborative filtering models, the final embedding is the representation directly used to compute recommendation scores (typically via dot products with item embeddings) [72]. Therefore, features discovered or edited here have an immediate and predictable effect on rankings. Learning a feature set at this layer means we model the signals that actually matter for decisions. In contrast, earlier-layer edits have to survive unknown downstream transformations, so they have fewer guarantees in terms of effect on final rankings.

However, it is also possible that earlier layers encode popularity information more directly. By "more directly", we mean that the signal may be easier to detect with simple, low-complexity functions (for instance, linear classifiers), rather than requiring deep nonlinear transformations. In practice, intermediate representations can sometimes make popularity bias more transparent than the final embedding, since in the final embedding, multiple features for predictions are entangled. Generally, the earlier the layer, the less complex transformations the embedding has gone through, increasing signal "extractability".

That said, SAEs are good feature extractors - they have been found to extract reliable features from the

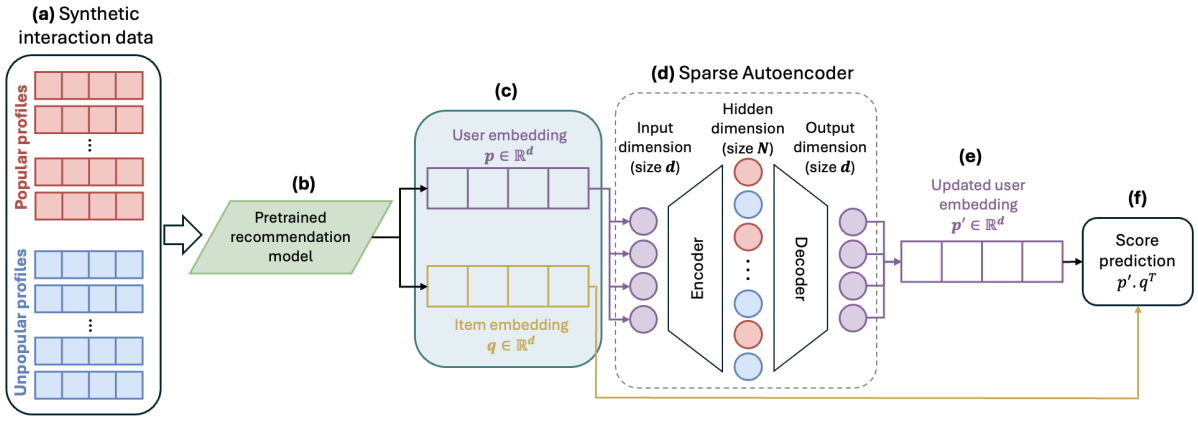


Figure 5.1: Overview of PopSteer process, first running with synthetic data for interpreting SAE's neuron ($a \to b \to c \to d$), and second run with real-world dataset for steering SAE's neurons to mitigate bias and derive modified user's embedding ($e \to b \to c \to d \to f \to g$): (a) synthetic interaction data containing users' profiles extremely representing interest towards either popular or unpopular items, (b) pretrained SASRec trained on real (not synthetic) data, (c) users' and items' embeddings learned from pretrained SASRec, (d) SAE used for interpreting popularity bias according to activation behavior of neurons in hidden dimension, (e) users' profiles in real data, (f) updated user embedding after steering SAE's neurons, and (g) predicting the relevance score for a target user-item pair using the updated user embedding.

embeddings of highly complex, large-scale LLMs [95]. The recommender systems we are testing are much smaller in scale and complexity compared to LLMs. Furthermore, we need a predictable downstream effect, since we are not only interpreting, but also intervening on the embedding. Therefore, for us, the benefits of choosing the last layer outweigh the downsides. With that being said, comparing the effectiveness of attaching SAEs on different layers of the model for popularity bias mitigation is an interesting future work direction. For instance, Zhao et al. have conducted a similar study - they perform linear probing across all layers of the base model to identify the layer that most strongly encodes the desired signal for attaching the SAE [131].

Mathematical formulation. Formally, let $x \in \mathbb{R}^d$ denote an embedding from the machine learning (ML) model. The SAE, consisting of an input dimension of size d, a hidden dimension of size N, and an output dimension of size d, reconstructs x as:

$$\hat{x} = W_{\text{dec}} a + b_{\text{pre}} \tag{5.1}$$

$$a = \text{TopAct}(z) \tag{5.2}$$

$$z = W_{\text{enc}}^T (x - b_{\text{pre}}) \tag{5.3}$$

with learnable $W_{\text{enc}}, W_{\text{dec}} \in \mathbb{R}^{d \times N}$ and $b_{\text{pre}} \in \mathbb{R}^d$. Here $z \in \mathbb{R}^N$ are the hidden *pre-activations*, where our neuron analysis and interventions occur. TopAct is the activation function inducing sparsity by retaining K neurons with the highest activations and zeroing the rest[1].

Importance of K and N hyperparameters. in Section 4.2, we discussed the importance of *overcompleteness* and *sparsity* properties of Sparse Autoencoders. Essentially, K controls the sparsity, while N controls the overcompleteness degree. Choosing $K \ll N$ limits each input to at most K active neurons while giving the model a large over-complete pool of candidates from which to "select". Usually, K is chosen such that $K \ll N$ and $K < d$ so the model must pick only a few useful features instead of relying on many at the same time. If K is set too large, the autoencoder risks learning to simply replicate the input rather than discovering interpretable features. As an analogy, consider the extreme case where $K = d$. In this scenario, the model could approximate a direct one-to-one mapping from input activations to hidden units, and then back to the output, bypassing any meaningful abstraction (While this is a simplified picture and not an inevitable outcome, it highlights the importance of carefully choosing K. Later, we will see how appropriate weight initialization helps mitigate such issues). If K

[1]This function is commonly referred to as TopK in the sparse autoencoder literature [43]. We denote it as TopAct here to avoid confusion with the standard "top-k" terminology used for item recommendation in recommender systems.

is too small, it cannot reconstruct the input well. In short, K sets how sparse the representation is, and N sets how rich the 'feature library' is. We pick both using validation.

Choice of Sparsity mechanism. As discussed in Section 4.3, multiple versions of SAE have been proposed in the recent literature, each offering a different sparsity mechanism (e.g., Top-K sparsity or L1 regularization). Since TopK SAEs, as proposed by Gao et al. [43], have been shown to outperform multiple baselines and have become the de facto standard in SAE research, we also chose it for our research.

Dead neuron prevention. One of the common issues in SAE training is the issue of dead neurons. When we mention a dead neuron, we refer to the neurons in the hidden layer (of size N) of the SAE. A neuron is flagged as dead if it has not activated for a predetermined number of inputs (e.g., $10,000,000$ samples), preventing it from receiving meaningful gradient updates. This happens because, in neural networks, any pathway not involved in the loss computation does not receive gradient flow. The TopAct function activates only a small subset of neurons for each input, while the remaining neurons have no influence on the output. A neuron can be considered definitively dead if it fails to activate for an entire epoch - Since no input triggers it, it receives no gradient updates and remains permanently inactive. Having a lot of dead neurons is undesirable because it means that the hidden layer of the Sparse Autoencoder is not fully utilized. Having a small number of alive neurons means that there are fewer features to pick from, and most likely, neurons are more entangled. Hence, the model has less flexibility in representing the input. Therefore, it is crucial to minimize the number of dead neurons.

One strategy for minimizing dead neurons is to pick a lower K value for TopAct activation. As discussed earlier, a lower K value imposes higher sparsity on the hidden layer. This in turn, forces the model to rely on more features to be able to reconstruct the input.

Another strategy we adopted, following Gao et al. [43], is to initialize the encoder as the transpose of the decoder (set $W_{\text{enc}} = W_{\text{dec}}^{\top}$). This creates an initial symmetry between the two parts of the autoencoder: the encoder and the decoder. By starting with this alignment, every neuron has a meaningful pathway through the network from the start, which decreases the probability that some neurons remain inactive and fail to learn during training.

An important ingredient for dead neuron prevention is to add an auxiliary loss term, which will be discussed in the next paragraph.

Optimization function. SAE is trained to minimize the following objective:

$$\min_{W_{\text{enc}}, W_{\text{dec}}, b_{\text{pre}}} \|x - \hat{x}\|_2^2 + \gamma \, \mathcal{L}_{\text{aux}} \tag{5.4}$$

where the term $\|x - \hat{x}\|_2^2$ is the reconstruction loss, ensuring that the decoder output $\hat{x}$ closely matches the original embedding x. $\mathcal{L}_{\text{aux}}$ is an auxiliary loss designed to prevent dead neurons and γ balances the primary reconstruction loss with this auxiliary term. $\mathcal{L}_{\text{aux}}$ forces a small set of inactive neurons to attempt to reconstruct the residual error $e = x - \hat{x}$. Concretely,

$$\mathcal{L}_{\text{aux}} = \|e - \hat{e}\|_2^2, \tag{5.5}$$

$$\hat{e} = W_{\text{dec}} a_{\text{aux}}, \tag{5.6}$$

$$a_{\text{aux}} = \text{TopAct}_{\text{aux}}\big(W_{\text{enc}}^{T}(x - b_{\text{pre}})\big). \tag{5.7}$$

Here, $\text{TopAct}_{\text{aux}}$ selects a fixed number of currently inactive neurons (those that have not fired for many steps) and activates them based on their largest pre-activations. Note that $\text{TopAct}_{\text{aux}}$ here is different from TopAct in equation 5.3 - The function is the same, however, $\text{TopAct}_{\text{aux}}$ operates on the "dead neuron" set. By training these "dead" neurons on the leftover reconstruction error, the model gives them a chance to learn useful features without interfering with the already active neurons [43]. This mechanism keeps the dictionary of neurons diverse and reduces wasted capacity.

Specifically, a neuron is considered *dead* if it has not activated for the last J training samples. At every interval of J samples, the model flags such neurons and adds them to a pool of dead neurons. For each batch within the next J samples, $\text{TopAct}_{\text{aux}}$ selects k_{aux} neurons from this pool, choosing those

with the largest pre-activations to help reconstruct the auxiliary error. This cycle repeats throughout training, giving unused neurons periodic opportunities to specialize on residual errors and preventing capacity from being wasted.

We set J equal to the number of samples in the training set, i.e., one epoch. Choosing J larger than this would be unnecesary: as discussed above, a neuron that does not activate for an entire epoch will not activate later without an external revival mechanism, since it receives no gradient updates. In large-scale LLM training, where an epoch may contain billions of tokens, smaller values of J are often preferable to avoid wasting capacity. In our case, however, each dataset contains fewer than 2 million samples, and training runs for around 50–100 epochs before convergence. Hence, setting J to one epoch allows us to be certain that the neuron is dead before trying to revive it. Following Gao et al. [43], we set $k_{\text{aux}} = d/2$ and $\gamma = 1/32$.

This objective ensures that the sparse hidden representation maintains fidelity to the original embedding while revealing the underlying decision structure. The learned neurons can then be interpreted to explain which concepts in the input data influence the ML model's output.

5.3. Interpreting popularity bias

This section provides an overview of the neuron analysis part of the pipeline to determine how much each neuron encodes the popularity signal.

Since each SAE neuron tends to specialize in a distinct concept from the input data, analyzing their activation behavior reveals how such concepts influence predictions. Hence, for the purpose of interpreting popularity bias in recommender systems, input data containing users' profiles with interests to popular and unpopular items is needed. To explore this, we generate synthetic user profiles that highlight popularity bias and examine how specific neurons respond (as shown in the first phase of Figure 5.1: steps $a \rightarrow b \rightarrow c \rightarrow d$). It is worth noting that the synthetic profiles are not used for training the recommender model and SAE, instead only for interpreting popularity bias. The motivation for creating synthetic users' profiles is that it enables providing input with a strong representation of popularity-aligned and unpopularity-aligned preferences for the purpose of interpretation. This enables the identification of neurons that either reinforce or counteract popularity signals.

Each synthetic profile is passed through the pretrained recommendation model to obtain the user embedding (i.e., x in Section 5.2), which is then fed into the pretrained SAE for neuron-level analysis. In the following, we describe our synthetic data generation process and how we quantify each neuron's contribution to popularity bias.

5.3.1. Generating synthetic datasets

We create two types of synthetic user profiles: one biased toward popular items and the other toward unpopular items. These profiles simulate user interactions that reflect extreme cases of popularity preference, enabling clearer observation of neuron activation.

Let $\mathcal{U} = \{u_1, ..., u_n\}$ be the set of users, $\mathcal{I} = \{i_1, ..., i_m\}$ be the set of items, and $R \in \mathbb{R}^{n \times m}$ represent the user-item interaction matrix. Following [3, 126], we define $\mathcal{I}^{\text{Pop}}$ (popular or *head* items) as the most frequently interacted items and $\mathcal{I}^{\text{Unpop}}$ (unpopular or *tail* items) as the least interacted items, both comprising roughly 10% of total number of items. Using these popular and unpopular item sets, we construct two synthetic datasets: 1) R^{Pop} with user profiles containing only popular items, and 2) R^{Unpop} with user profiles containing only unpopular items.

To construct R^{Pop}, we generate n' synthetic users' profiles, each of length M. For each synthetic user, we randomly sample M items (with replacement) from $\mathcal{I}^{\text{Pop}}$ (see Section 6.5 for detailed experimental setup). The construction of R^{Unpop} is identical, except that items are drawn from the tail set $\mathcal{I}^{\text{Unpop}}$. By restricting each synthetic profile to *either* exclusively head items $\mathcal{I}^{\text{Pop}}$ *or* exclusively tail items $\mathcal{I}^{\text{Unpop}}$, we create two synthetic sets that lie at the extremes of the popularity spectrum.

The reason for this design is that it cleanly isolates the influence of item popularity on model behavior while suppressing confounding factors such as sequence diversity, recency effects, and mixed user interests. By randomly selecting popular or unpopular items, we are creating a set in which the main common trait items share is the popularity level.

Initially, we tried a slightly different approach. For each user from the real training set, we replaced all of their real interactions with the randomly selected popular or unpopular items. This approach preserves the number of interactions per user without altering profile length or interaction density. The idea was to keep the users realistic in terms of profile length to avoid introducing additional noise.

Empirically, this user-length–preserving variant underperformed the fixed-length construction that sets every synthetic profile to the model's window size M. Many datasets have short histories, so replacing items in a small profile provides too few head/tail examples for the model to express a clear popularity signal. In contrast, fixing length to M saturates the window with consistent-popularity items, increasing statistical power and stabilizing neuron responses. For example, a sequence of 50 blockbuster titles yields a far stronger and more reliable "blockbuster-preference" signal than a sequence of only 5.

Importantly, these synthetic datasets are not used to train the SAE. Instead, we feed them into a pretrained SAE (trained on R), to evaluate its neurons' activation, ensuring that its learned representations remain grounded in real user behavior.

5.3.2. Detecting Neurons Encoding Popularity Bias

To quantify how much each SAE neuron contributes to popularity bias, we compare neuron activations when processing R^{Pop} and R^{Unpop}. Each synthetic dataset is fed into the pretrained SAE, and the activation levels of the hidden layer neurons are recorded.

Normality assumption of activations. We make a key normality assumption about the activation patterns of SAE neurons. This assumption is well-supported in the literature: prior work has shown that as neural networks increase in size, neuron activation distributions tend to approximate a Gaussian distribution [76, 116, 47]. The justification comes from the Central Limit Theorem (CLT): each neuron computes a weighted sum of many inputs, and as the number of summands grows, CLT implies that this sum converges toward a normal distribution. Empirical evidence further indicates that this approximation holds in large but finite networks [116]. Moreover, the normality assumption is not only theoretical - it has been used in practice. For example, Haider et al. [47] leveraged it to interpret activation patterns in large language models.

The normality assumption is particularly useful in our case, as it provides a principled way to quantify the variance of each neuron's activations via the standard deviation. As we will discuss in Section 5.4, this measure plays an important role in determining the extent to which we steer individual neurons. Besides, normality allows us to compute Cohen's d for each neuron - an established statistical effect size measure that relies on normally distributed activations [27].

Cohen's d value calculation. Cohen's d is well-suited to our setting for several reasons. First, it is designed for direct two-sample comparisons, which makes it an obvious choice for contrasting popular versus unpopular synthetic set activations. Second, by normalizing the mean difference between the two groups by their pooled standard deviation, Cohen's d allows the comparability of neurons with different activation scales. This standardization allows us to detect neurons that exhibit strong biases regardless of their activation range. Finally, both the sign and magnitude of Cohen's d provide an interpretable signal: positive values indicate stronger alignment with popular items, negative values indicate alignment with unpopular items, and the absolute magnitude represents the strength of the effect.

For each neuron j, Cohen's d is calculated as:

$$d_j = \frac{\mu_{j,\text{Pop}} - \mu_{j,\text{Unpop}}}{\sqrt{\frac{\sigma^2_{j,\text{Pop}} + \sigma^2_{j,\text{Unpop}}}{2}}} \tag{5.8}$$

where $\mu_{j,\text{Pop}}$ and $\mu_{j,\text{Unpop}}$ are the mean pre-activations of neuron j under R^{Pop} and R^{Unpop}, respectively, and $\sigma_{j,\text{Pop}}$ and $\sigma_{j,\text{Unpop}}$ are the corresponding standard deviations. Cohen's d provides a normalized measure of the difference between two distributions. We denote Cohen's d value of neuron j as d_j.

A high absolute value of d_j indicates that neuron j is strongly responsive to popularity-related patterns. Positive values suggest alignment with popular content, while negative values imply a focus on unpopular or niche items.

Interpreting d_j. SAEs provide sparse, more interpretable features, but they do not guarantee that a single neuron will be a pure "popularity" detector. Our use of Cohen's d ranks neurons by *alignment* with popularity (positive d_j) or unpopularity (negative d_j) across synthetic user sets - it does not presume a priori semantic labels. This procedure can surface both near-monosemantic "popularity" neurons and correlated features (For instance, a neuron that activates for *action* films because such films tend to be popular). We steer all neurons with large $|d_j|$ (See Section 5.4), and we achieve interpretability operationally: modifying these neurons yields predictable fairness shifts and structured movements in the embedding space (as shown later in Section 7.2). Thus, our selection and steering approach is compatible with the potential distributed encoding of the popularity.

5.4. Popularity Bias Mitigation Approach

To counteract popularity bias in recommendations, we use the computed Cohen's d values to identify neurons most responsible for encoding popularity signals. Our method, $\mathrm{PopSteer}$, applies a process called *neuron steering*, which systematically adjusts these neuron activations to reduce bias and promote fairer item exposure (refer to the second phase in Fig. 5.1: steps e→b→c→d→f→g for the steering process on real data.)

Neuron steering operates on the SAE's hidden pre-activations, increasing or decreasing the activity of neurons aligned with popularity. Importantly, we apply steering *before* the TopAct activation function. This allows the added steering vector to change the *active set* of the hidden representation: neurons that were just below the TopAct threshold for selection can become active, and over-dominant popularity neurons can be pushed below it. This enables *feature selection* (which neurons are chosen), not merely *feature reweighting*. By contrast, applying a steering vector *after* TopAct would only adjust the magnitudes of already-selected neurons.

The adjustment to each neuron is guided by its d_j score, which reflects its alignment with popular or unpopular items. We modify the original pre-activation z_j of neuron j to obtain a new pre-activation z'_j:

$$z'_j = \begin{cases} z_j - w_j \cdot \sigma_j, & \text{if } d_j > \beta \\ z_j + w_j \cdot \sigma_j, & \text{if } d_j < -\beta \end{cases} \tag{5.9}$$

where σ_j is the standard deviation of activations for neuron j. Note that $\sigma_{j,\mathsf{Pop}}$ and $\sigma_{j,\mathsf{Unpop}}$ on Equation 5.8 are based on the synthetic sets, while σ_j is computed on the validation set containing real interactions. The parameter β serves as a threshold on $|d_j|$, ensuring that only neurons with sufficiently strong popularity bias signals are steered ($\beta = 0$ corresponds to steering all neurons). In statistical terms, $|d| \geq 1$ is often considered evidence of a meaningful difference between two distributions, though more or less stringent thresholds may be applied depending on the application. w_j is a weight assigned to each neuron j based on the normalized absolute value of its d_j. This weight determines how much each neuron's pre-activation will be adjusted:

$$w_j = \begin{cases} \alpha^{\mathsf{Pop}} \cdot \frac{|d_j|}{\max\limits_{1 \leq i \leq N}(|d_i|)} & \text{if } d_j > \beta \\ \alpha^{\mathsf{Unpop}} \cdot \frac{|d_j|}{\max\limits_{1 \leq i \leq N}(|d_i|)} & \text{if } d_j < -\beta \end{cases} \tag{5.10}$$

where α^{Pop} and α^{Unpop} are tunable hyperparameters that independently control the strength of the steering for popularity-aligned and unpopularity-aligned neurons, respectively. We use different steering strengths for popularity-aligned and unpopularity-aligned neurons because we found that these two groups respond differently to the same steering magnitude, as further discussed in Section 7.4.

In this formulation, w_j grows proportionally with $|d_j|$, so neurons with stronger evidence of popularity alignment receive larger adjustments while weakly aligned neurons are minimally perturbed. Normalizing by $\max_i |d_i|$ makes α^{Pop} and α^{Unpop} directly interpretable as the *maximum steering magnitude* (in units of σ_j) for their respective groups: the most popularity-aligned neuron is shifted by at most $\alpha^{\mathsf{Pop}}\sigma_j$, and the most long-tail-aligned neuron by at most $\alpha^{\mathsf{Unpop}}\sigma_j$. This proportional weighting also reduces sensitivity to the choice of β: even with a permissive threshold, neurons with small $|d_j|$ receive near-zero w_j and thus negligible updates, while highly biased neurons are steered smoothly in proportion to

Box 5.1: $\mathrm{PopSteer}$ pipeline summary

PopSteer pipeline summary.

1. **Train and freeze the base recommender.** We first train the backbone recommender system (e.g., SASRec or LightGCN) in the standard way. After convergence, the backbone parameters are frozen.
2. **Attach and train the Sparse Autoencoder.** We attach a Sparse Autoencoder (SAE) to the final hidden representation of the frozen backbone. The SAE is trained to produce a sparse, approximately disentangled feature space using a Top-K sparsity objective, together with an auxiliary "revival" objective that reactivates dead neurons (controlled by J, k_{aux}, and γ). After this stage, SAE neurons become more monosemantic.
3. **Construct extreme synthetic preference profiles.** We generate two synthetic user groups: one that strongly prefers highly popular items and one that strongly prefers unpopular items. These represent "popularity preference" extremes.
4. **Score neurons by popularity preference.** We pass both synthetic sets through the SAE and measure, for each neuron, how differently it activates for popularity-preferring vs. unpopularity-preferring profiles. We quantify this separation using Cohen's d value, yielding neurons that are "popular-leaning" and neurons that are "unpopular-leaning".
5. **Steer activations at inference time.** At inference, we modify the SAE activations before scoring items. We upweight or downweight the neurons' activations using steering coefficients $\alpha^{\text{Pop}}, \alpha^{\text{Unpop}}$, optionally applying a threshold β over neurons. This shifts the recommender's exposure away from over-represented popular items and toward under-exposed long-tail content, without retraining the backbone.

their estimated effect size. In practice, this yields fine-grained, interpretable control - strong corrections where warranted and small updates elsewhere.

After steering the hidden layer pre-activations and applying TopAct function, the updated SAE then returns the modified user embedding $\hat{x}$:

$$a' = \text{TopAct}(z') \tag{5.11}$$

$$\hat{x}' = W_{\text{dec}} a' + b_{\text{pre}} \tag{5.12}$$

$\hat{x}'$ is used alongside item embeddings from the base recommendation model to generate the final recommendation list. Box 5.1 provides the summary of the $\mathrm{PopSteer}$ pipeline.

Existing SAE steering methods typically employ simplistic mechanisms like uniform scaling factors, clamping to fixed values, or unweighted additive vectors to direct features toward desired behaviors, such as mitigating biases or controlling retrieval outcomes [38, 55, 63, 131]. Our method bases adjustments on a statistical measure of each neuron's sensitivity to popularity signals, drawn from synthetic user profiles. This ensures that changes are both proportional to the strength of the detected bias (via d_j) and scaled according to the neuron's own activation variance. This dual mechanism minimizes the chances of over- or under-correction and allows for more precise, interpretable control over the fairness-accuracy trade-off without changing the model's core structure.

A derivation that links SAE steering to per-item score changes is provided in Appendix A. This result is used only as an interpretive lens and is not part of the executed pipeline. Implementation and empirical evaluation are left to future work.

6

Experiments

This chapter outlines our experimental setup, including datasets, evaluation metrics, baselines, and hyperparameters. Our implementation and code are available at https://github.com/parepic/PopSteer2.0/tree/main.

6.1. Dataset

We conduct experiments on four public datasets from diverse domains: MovieLens 1M (ML-1M) [51], a movie recommendation dataset, Steam [65], a game recommendation dataset, BeerAdvocate [89], a beer review dataset, and Yelp [118], a restaurant recommendation dataset. Following standard preprocessing [52, 65], we create a 5-core sample on ML-1M, Steam, and BeerAdvocate datasets, where each user has at least 5 interactions and each item is interacted with by at least 5 users. For Yelp dataset, we perform 20-core validation due to extreme sparsity. All datasets include timestamp information, enabling their use in sequential recommendation task. Table 6.1 summarizes the statistical properties of these datasets.

Note on dataset sparsity. Table 6.1 reports "Density", meaning the share of user–item pairs for which we observe an interaction. By contrast, sparsity is simply the opposite of density—it tells you how much of the user–item matrix is empty (nodisct to be confused with the sparsity used for SAE activations). Highly sparse datasets tend to have many users or items with only a handful of interactions, which increases uncertainty and makes it harder for models to learn stable patterns. This also stresses the SAE in $\mathrm{PopSteer}$, since extremely sparse inputs are harder to accurately reconstruct. We will revisit this in Section 8.2.

6.2. Base recommender model

6.2.1. SASRec model

$\mathrm{PopSteer}$ can be attached to any recommender model since it is a post-processing technique that only takes a layer of the model as input. Our main experiments are conducted on SASRec [65], a widely used benchmark sequential recommender model. A sequential recommender model considers timestamp of the interactions as well for making the prediction, and SASRec achieves this through a series of

Table 6.1: Statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
ML-1M	$6,040$	$3,417$	$999,611$	4.8%
Steam	$25,390$	$4,090$	$328,278$	0.32%
BeerAdvocate	$10,464$	$13,907$	$1,395,865$	1%
Yelp	$20,799$	$16,253$	$983,530$	0.29%

transformer-based attention blocks. SASRec takes the user interaction history, where ids are sorted chronologically based on recency, and outputs a score for each item.

We adopt a sequential backbone because it simplifies new-user creation: a synthetic profile is specified directly by an item sequence. By contrast, non-sequential models typically require fitting a new user embedding before evaluation, which complicates controlled experiments with synthetic users. For those methods, introducing a new user generally requires fitting or inferring an embedding vector before evaluation. By using a sequential model, we avoid this additional step. Table 6.2 contains the hyperparameters used for training SASRec.

Table 6.2: SASRec Hyperparameters

Hyperparameter	Value
Number of transformer layers	2
Number of attention heads	2
Hidden size	64
Feed-forward inner size	256
Hidden dropout probability	0.5
Attention dropout probability	0.5
Activation function	`gelu`
Layer normalization ϵ	1×10^{-12}
Initializer standard deviation	0.02
Loss function	Cross-Entropy

6.2.2. LightGCN model

To test the generalizability of $\mathrm{PopSteer}$, we also conducted experiments using LightGCN [53], a graph-based general recommender system. LightGCN attempts to find optimal embeddings for users and items by focusing on neighborhood aggregation to capture collaborative filtering signals.

For LightGCN experiments, rather than constructing synthetic user sequences, we labeled *existing* users based on their recent interaction histories. In this variant, biased neurons were identified using real user interaction data rather than synthetic profiles. Following Section 5.3.1, we partition items into a popular set $\mathcal{I}^{\text{Pop}}$ and an unpopular set $\mathcal{I}^{\text{Unpop}}$ (head vs. tail). Assign labels $y(i) = 1$ for $i \in \mathcal{I}^{\text{Pop}}$, $y(i) = -1$ for $i \in \mathcal{I}^{\text{Unpop}}$, and $y(i) = 0$ otherwise. For a user u with interacted items $\mathcal{I}_u$, define the popularity score as the mean label:

$$s(u) = \frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} y(i).$$

Higher $s(u)$ indicates stronger head preference; lower values indicate stronger tail preference. Users are ranked by their popularity scores $s(u)$, with the top 10% forming the popular user set and the bottom 10% forming the unpopular user set.

It is important to mention that SASRec experiments are our main results. We conducted LightGCN experiments to see how our pipeline performs on a different recommender model and also when using real users for neuron analysis instead of synthetic ones. Therefore, LightGCN experiments are limited to the main accuracy-fairness tradeoff experiments - interpretability, ablation, and scalability analysis are only conducted on SASRec experiments. Table 6.3 contains the hyperparameters used to train LightGCN.

Table 6.3: LightGCN Hyperparameters

Hyperparameter	Value
Embedding size	128
Number of layers	2
Regularization weight	1×10^{-5}

For both SASRec and LightGCN models, the Adam optimizer is used with a learning rate of 0.001 and a training batch size of 2048. The models are trained with early stopping (patience = 10 epochs) based on validation nDCG.

6.3. Baselines

We compare our $\mathrm{PopSteer}$ method with the following baselines:

- **FA*IR** [120]: This approach enforces group fairness in ranked outputs through a post-processing step that guarantees a minimum proportion of protected items within every index of the top-k list. The key idea is to prevent rankings from being dominated by highly popular items, ensuring that minority groups consistently receive exposure across different cutoff points, not only at the end of the list. $\mathrm{FA*IR}$ frames this as a fairness constraint, striking a balance between utility and fair treatment. By doing so, it provides probabilistic guarantees that the resulting ranking will satisfy fairness requirements at a chosen significance level α and will remain close to the original ranking in terms of relevance. In our experiments, we vary the required proportion of protected items $p \in \{0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and the significance level $\alpha \in \{0.01, 0.05, 0.1\}$.
- **Provider Max-Min Fairness ($\mathrm{P\text{-}MMF}$) [117]:** This method addresses provider fairness in recommender systems by focusing on *max-min fairness*, a method that prioritizes improving the exposure of the least-interacted provider to ensure equity among participants. Unlike approaches based on proportional fairness, which allocate exposure in fixed ratios, max-min fairness protects weaker providers who otherwise may get very little exposure. $\mathrm{P\text{-}MMF}$ treats recommendation as a resource allocation problem, where exposure slots are treated as limited resources to be distributed among providers, and uses a fairness regularizer that balances user preference with provider fairness. To adapt the method to our use case, we divide the items into three groups: Head, Tail, and the remaining items, and treat each group as a provider. In our experiments, we apply the Min-regularize heuristic version of $\mathrm{P\text{-}MMF}$, and tune the tradeoff hyperparameter $\lambda \in \{0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$.
- **Personalized Calibration Targets (PCT) [109]:** This method introduces the idea of two-sided calibration, with the aim of balancing both user-level and system-level exposure in recommender systems. On the user side, PCT ensures that each individual's recommendations match their historical interests in the right proportions (e.g., if a user watches mostly popular content, the recommended list should reflect this). On the system side, it prevents the overall ecosystem from being dominated by low-quality or biased content by enforcing a desired global exposure distribution across item groups. PCT achieves this through a two-step post-processing framework: a solver computes personalized calibration targets that minimally deviate from users' past behavior while still respecting system-wide exposure goals, and a reranker adjusts recommendation lists using a variant of maximum marginal relevance (MMR). This makes PCT useful in scenarios where both personalization and broader platform-level objectives must be met. Similar to $\mathrm{P\text{-}MMF}$, each item group is treated as a separate group, resulting in three separate groups. Hyperparameters are tuned in the same way as for $\mathrm{FA*IR}$.
- **Dynamic User-oriented re-Ranking (DUOR) [46]:** This method addresses popularity bias by aligning recommendations with each user's individual inclination towards popular versus niche items. Unlike many existing approaches that apply uniform penalties to popular items, DUOR first classifies items into popular and unpopular groups based on the Pareto Principle and then estimates a user's genuine popularity tendency. During re-ranking, the algorithm adjusts the recommendation list in an iterative manner: at each step, it compares the current ratio of popular items in the list with the user's personal popularity profile and either penalizes or promotes items to maintain alignment. This ensures that users who prefer mainstream content are not unfairly restricted, while those with niche interests receive more long-tail exposure. As a result, DUOR provides more calibrated recommendations that improve both fairness and diversity without sacrificing accuracy. An important DUOR hyperparameter is the long recommendation list: the size of the candidate pool from the base model that DUOR reranks before truncating to the final top-k list. In our experiments, we tune DUOR with long recommendation lists of size $t \in \{50, 100, 250, 500, 1000\}$.
- **Inverse Popularity Ranking (IPR) [122]:** A simple yet effective reweighting approach that adjusts relevance score for user-item pairs based on inverse popularity: $\tilde{s}_{u,i} = s_{u,i}/(1 + \alpha\,\rho_i)$, where $\rho_i = \mathrm{pop}(i) \big/ \max_{j \in \mathcal{I}} \mathrm{pop}(j)$, and α is a hyperparameter controlling the degree of mitigating bias, regulating how strongly the algorithm pushes the popular items back. Small values of α retain

rankings closer to the original relevance scores, while larger values increasingly promote long-tail items. We tune $\alpha \in \{0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ in our experiments.

We also compare our $\mathrm{PopSteer}$ with a simple Random baseline that randomly selects k items from an initial longer recommendation list. We test this baseline with long list sizes $t \in \{15, 30, 50, 75, 100\}$. We set the size of long recommendation lists for $\mathrm{P\text{-}MMF}$, PCT, IPR, and $\mathrm{FA^*IR}$ to 250.

6.4. Evaluation metrics

We evaluate recommendations along two dimensions: (i) *ranking accuracy* with NDCG, and (ii) *exposure fairness* with Item Coverage and the Gini index. In this section, we provide the definition of the ranking metrics used.

Let $\mathcal{U}$ be the set of users, $\mathcal{I}$ the set of items, and $\hat{R}_u = (\hat{i}_{u1}, \ldots, \hat{i}_{uK})$ the top-K list for user u. Relevance is binary: $r_{u,i} \in \{0, 1\}$. Define the set of hit positions (the ranks at which recommended items are relevant)

$$\mathcal{R}(u) \;=\; \big\{\, i \in \{1, \ldots, K\} \;\big|\; r_{u,\hat{i}_{ui}} = 1 \,\big\},$$

Let K be the final recommendation list size.

NDCG@K. NDCG measures the quality of the top-k ranking. Its values lie in $[0, 1]$ and higher is better.

$$\mathrm{NDCG@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \left(\frac{\displaystyle\sum_{i=1}^{K} \delta(i \in \mathcal{R}(u)) \; \frac{1}{\log_2(i+1)}}{\displaystyle\sum_{i=1}^{\min(|\mathcal{R}(u)|,\, K)} \frac{1}{\log_2(i+1)}} \right). \tag{6.1}$$

Here, $\delta(\cdot)$ is an indicator function. If $|\mathcal{R}(u)| = 0$, the user's contribution is 0. Following [126], we also report NDCG separately on *Head* and *Tail* item subsets by restricting (6.1) to those subsets.

ItemCoverage@K. Let c_i be the number of users whose top-K list contains item i:

$$c_i \;=\; \sum_{u \in \mathcal{U}} \delta\Big(i \in \hat{R}_u\Big),$$

Coverage is the fraction of items that meet a minimum exposure threshold τ:

$$\mathrm{ItemCoverage@}K = \frac{1}{|\mathcal{I}|} \Big|\big\{\, i \in \mathcal{I} \;\big|\; c_i \geq \tau \,\big\}\Big|. \tag{6.2}$$

We set $\tau = 5$ following prior work [87, 88] to exclude items that only appear a handful of times due to luck rather than sustained exposure.

Gini@K (inequality of exposure). Using the same top-K exposure counts c_i from (6.2), sort them in nondecreasing order $c_{(1)} \leq \cdots \leq c_{(|\mathcal{I}|)}$. The Gini coefficient over top-K exposure is

$$\mathrm{Gini@}K = \frac{\sum_{j=1}^{|\mathcal{I}|} (2j - |\mathcal{I}| - 1)\, c_{(j)}}{|\mathcal{I}| \sum_{j=1}^{|\mathcal{I}|} c_{(j)}}. \tag{6.3}$$

Values lie in $[0, 1]$ and lower is better (0 means perfectly uniform exposure; higher indicates concentration on a few items).

We report overall $\mathrm{NDCG@}K$ (6.1), Item Coverage$_{\geq \tau}$ (6.2), and Gini (6.3), and additionally NDCG on Head/Tail subsets for debiasing analysis.

6.5. Experimental setup

We adopt the widely used leave-one-out evaluation protocol, holding out the most recent interaction for testing, the second most recent for validation, and using the rest for training [65]. The final recommendation list size is set to 10 for all our experimental results. For the synthetic data generation process

described in Section 5.3.1, we set the number of synthetic users $n' = 409,600$ (200 epochs with 2048 users in each epoch), and the size of user's profile $M = 50$[1].

6.5.1. Platform and Resources

Our implementation of $\mathrm{PopSteer}$ and baselines is built on top of the Recbole framework [130]. All experiments were conducted on the DelftBlue supercomputer, using the GPU-A100 node equipped with NVIDIA Tesla A100 GPUs.

6.5.2. PopSteer hyperparameters

$\mathrm{PopSteer}$ includes two key training hyperparameters:

- Scale factor s determines the size of the hidden layer of the SAE. Size N of the hidden layer of the SAE equals $d \times s$ where d is the SAE input size. We tune $s \in \{32, 64, 96\}$.
- Sparsity level K specifies the number of top activations retained in the hidden layer (Refer to Section 5.2 for details). We tune $K \in \{36, 40, 44, 48, 52, 56\}$.

$\mathrm{PopSteer}$ involves multiple hyperparameters for steering. These hyperparameters are used in inference only, and collectively govern the steering mechanism of $\mathrm{PopSteer}$.

- α^{Pop} and α^{Unpop} as defined in Eq. 5.10 rescale the per-neuron weight w_j for neurons with $d_j > \beta$ (aligned with popular items) and $d_j < -\beta$ (aligned with unpopular items), respectively. Larger α^{Pop} produces a stronger downward adjustment for popularity-aligned neurons; larger α^{Unpop} produces a stronger upward adjustment for unpopularity-aligned neurons. We tune $\alpha^{\mathrm{Pop}}, \alpha^{\mathrm{Unpop}} \in \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}$.
- Threshold β determines which neurons are modified: we steer all neurons whose Cohen's d exceeds the threshold in magnitude, i.e., $\{\, j : |d_j| \geq \beta \,\}$. Larger β yields fewer steered neurons. We tune $\beta \in \{0, 0.5, 1.0, 1.5, 2.0\}$.

Training hyperparameters are chosen by a small sweep on this split, then kept fixed for all main experiments. Steering hyperparameters, which affect inference only are tuned by grid search on the same validation split. No test metrics are consulted during selection.

6.5.3. Other training details

$\mathrm{PopSteer}$ was trained with Adam optimizer. $\mathrm{ReduceLROnPlateau}$ scheduler was used, which progressively decreases the learning rate when the improvements stall for a set number of epochs. $\mathrm{ReduceLROnPlateau}$ scheduler is configured with an initial learning rate of 0.01, patience of 10 epochs, factor of 0.1, and a validation based on the MSE between the original and reconstructed input. Training batch size is set to 2048.

[1] M is the largest number of recent items the base recommender, SASRec [65], can process at once-the size of its sliding window.

7

Results

In this section, we provide evidence and observations from our experimental results to address our research questions from Section 2.4: **RQ1** (Effectiveness), **RQ2** (Interpretability), **RQ3** (Controllability).

Section 7.1 will answer **RQ1** through examining nDCG and fairness tradeoffs of $\mathrm{PopSteer}$ and baselines, and also through independently measuring the impact on nDCG of head and tail items. Section 7.2 answers **RQ2** by assessing interpretability through a targeted neuron manipulation study, and analysis of the user embedding space. Section 7.3 contains the ablation study of $\mathrm{PopSteer}$, to determine the effectiveness of both the neuron selection and neuron manipulation parts of the pipeline. Section 7.4 answers **RQ3** through the sensitivity analysis of several hyperparameters of $\mathrm{PopSteer}$, which reveals the controllability and importance of the key hyperparameters. Finally, Section 7.5 answers whether the $\mathrm{PopSteer}$ is deployable in practical use-cases, through a scalability analysis.

7.1. Overall performance

In this subsection, we answer **RQ1**.

7.1.1. Accuracy–Fairness Tradeoffs

7.1.1.1. SASRec results

Figs. 7.1 and 7.2 present our main results, comparing $\mathrm{PopSteer}$ with SASRec and several baselines across four datasets in terms of nDCG versus Item Coverage (Fig. 7.1) and nDCG versus Gini Index (Fig. 7.2). Each point represents a hyperparameter configuration, with superior methods achieving points that balance high accuracy and fairness. The best performance in Figs. 7.1 and 7.2 appears in the upper-right and bottom-right, respectively.

On ML-1M (Figs. 7.1a and 7.2a) and BeerAdvocate (Figs. 7.1c and 7.2c), $\mathrm{PopSteer}$ consistently outperforms all baselines in enhancing fairness while maintaining recommendation accuracy. It is evident that with almost the same nDCG, $\mathrm{PopSteer}$ yields significantly higher Item Coverage and lower Gini Index compared to the baseline.

Similar patterns emerge in the highly sparse Steam dataset, even with slightly improved nDCG compared to SASRec and the baselines. Due to a much stronger popularity bias (high skewness in popularity distribution), tackling popularity bias on this dataset seems challenging. This is evident from the results obtained from the baselines that all are struggling to improve fairness. Instead, our $\mathrm{PopSteer}$ method manages to improve both Item Coverage and Gini Index with even slightly higher nDCG. For instance, on Fig. 7.1b, $\mathrm{PopSteer}$ improves SASRec by $\sim$66% and $\sim$0.7% in terms of Item Coverage and nDCG, respectively. Also, compared to second best-performing baseline, PCT, $\mathrm{PopSteer}$ achieves $\sim$28% and $\sim$3% better performance in terms of Item Coverage and nDCG, respectively.

On Yelp dataset, while IPR and DUOR are able to outperform $\mathrm{PopSteer}$, this improvement is at the cost of a higher drop in nDCG. In a more realistic and lower nDCG loss situation, $\mathrm{PopSteer}$ is still superior, outperforming all baselines in terms of both Item Coverage and Gini Index. Besides fairness

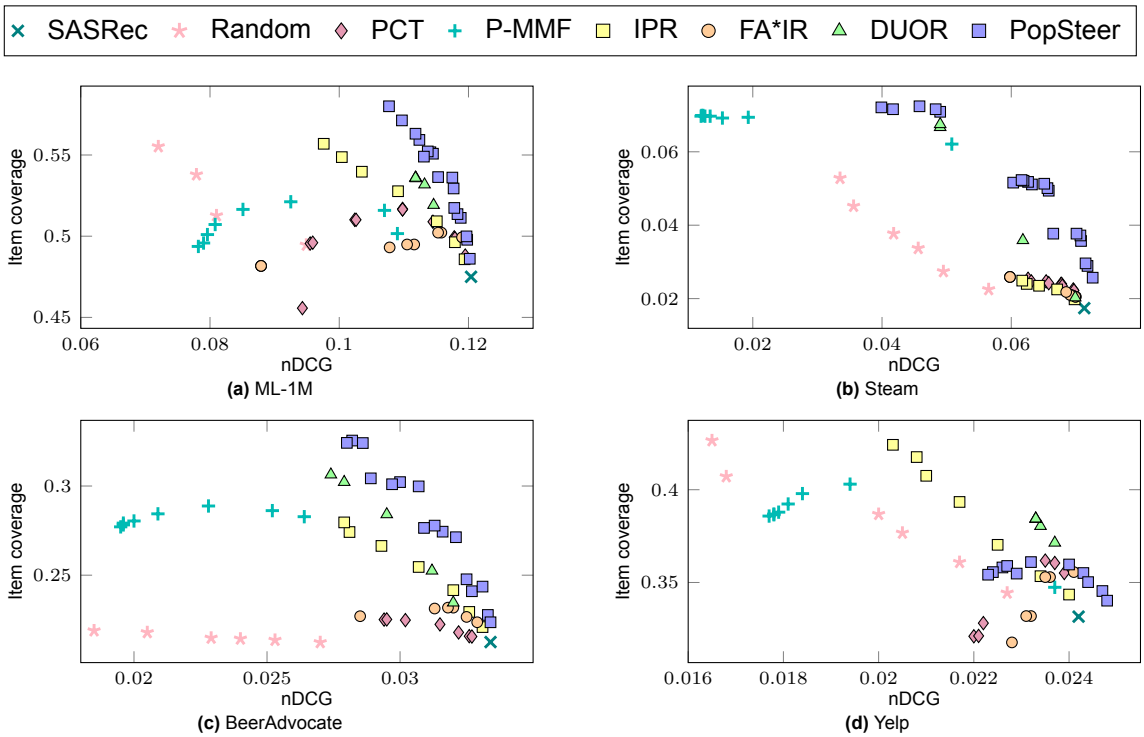


Figure 7.1: Performance comparison of PopSteer method with the baselines in terms of nDCG and Item Coverage when using SASRec as a backbone.

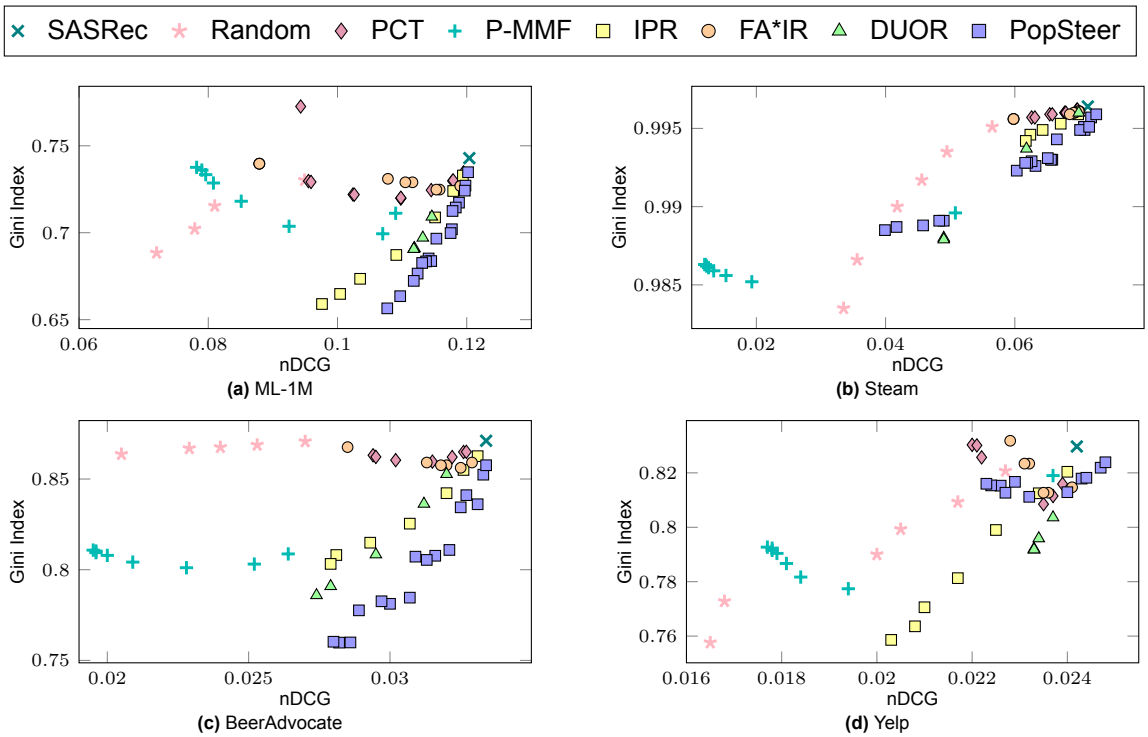


Figure 7.2: Performance comparison of PopSteer with baselines in terms of nDCG and Gini Index when using SASRec as a backbone.

improvements, PopSteer also increases the ranking quality of SASRec by ∼2.5%, indicative of its strong effectiveness in improving recommendation overall performance (both accuracy and fairness).

Another notable pattern is the stability of PopSteer's trade-off curves across datasets. While baselines often show sharp accuracy drops for only slight fairness gains, PopSteer maintains gradual improve-

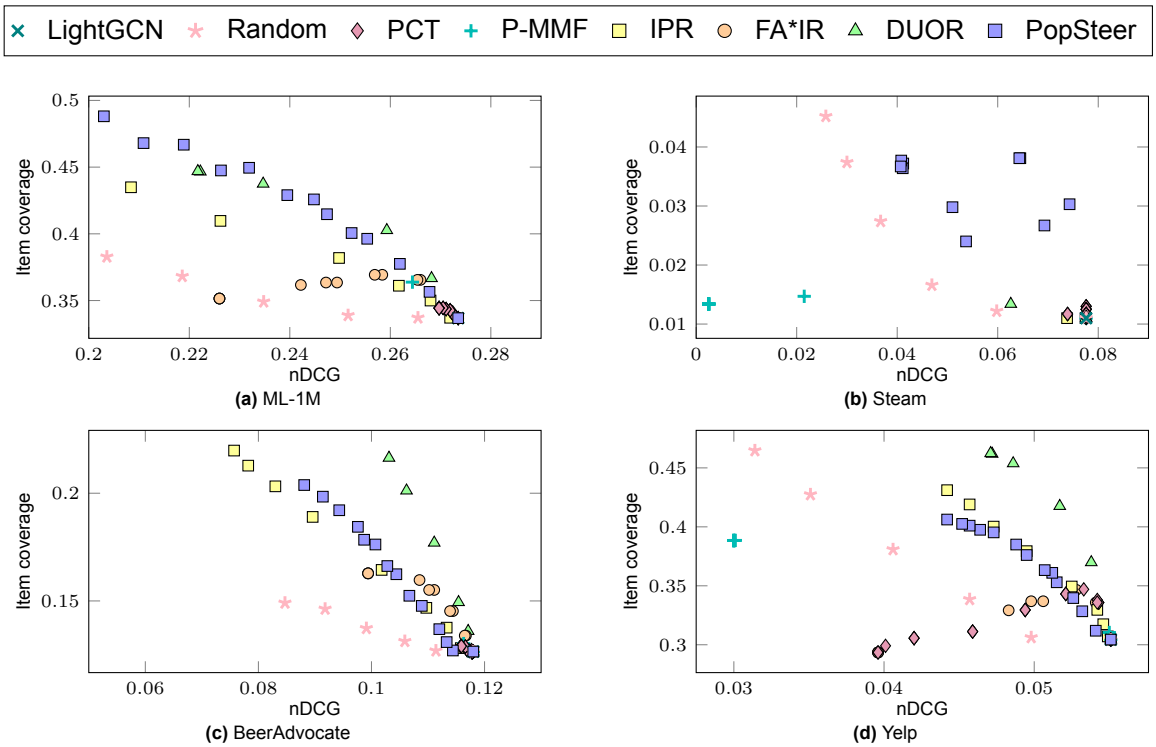


Figure 7.3: Performance comparison of PopSteer method with the baselines in terms of nDCG and Item Coverage when using LightGCN as a backbone.

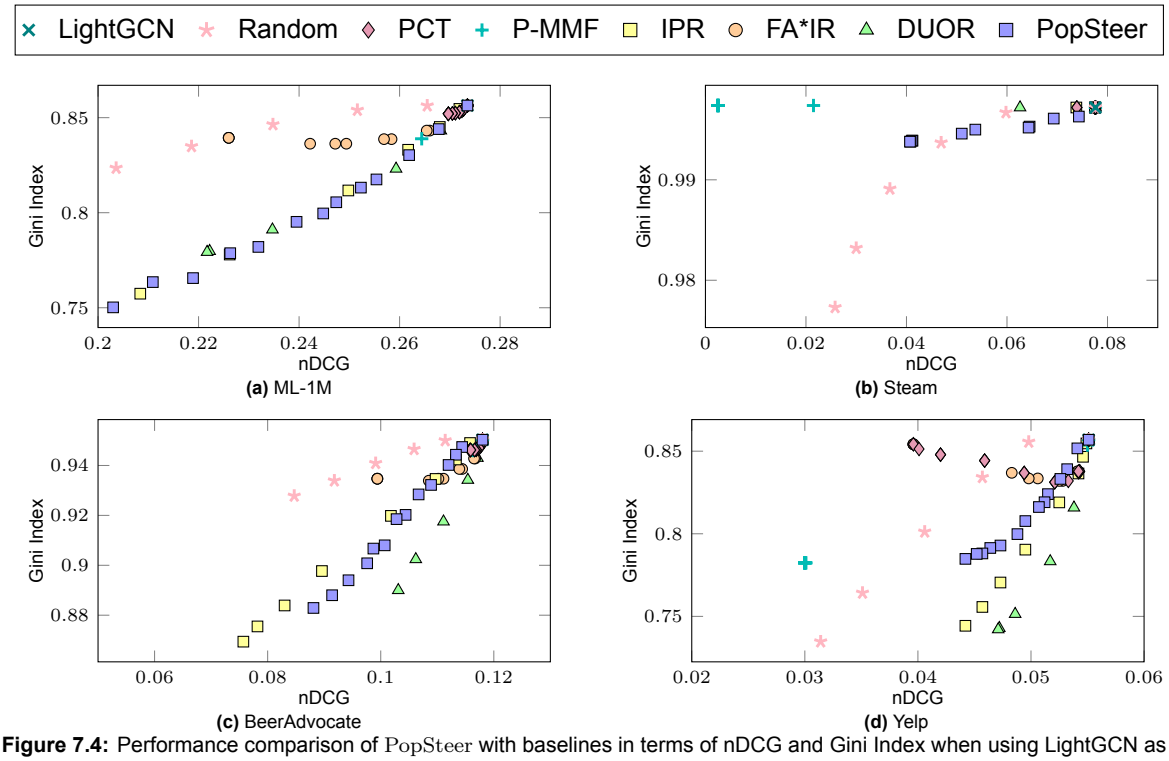
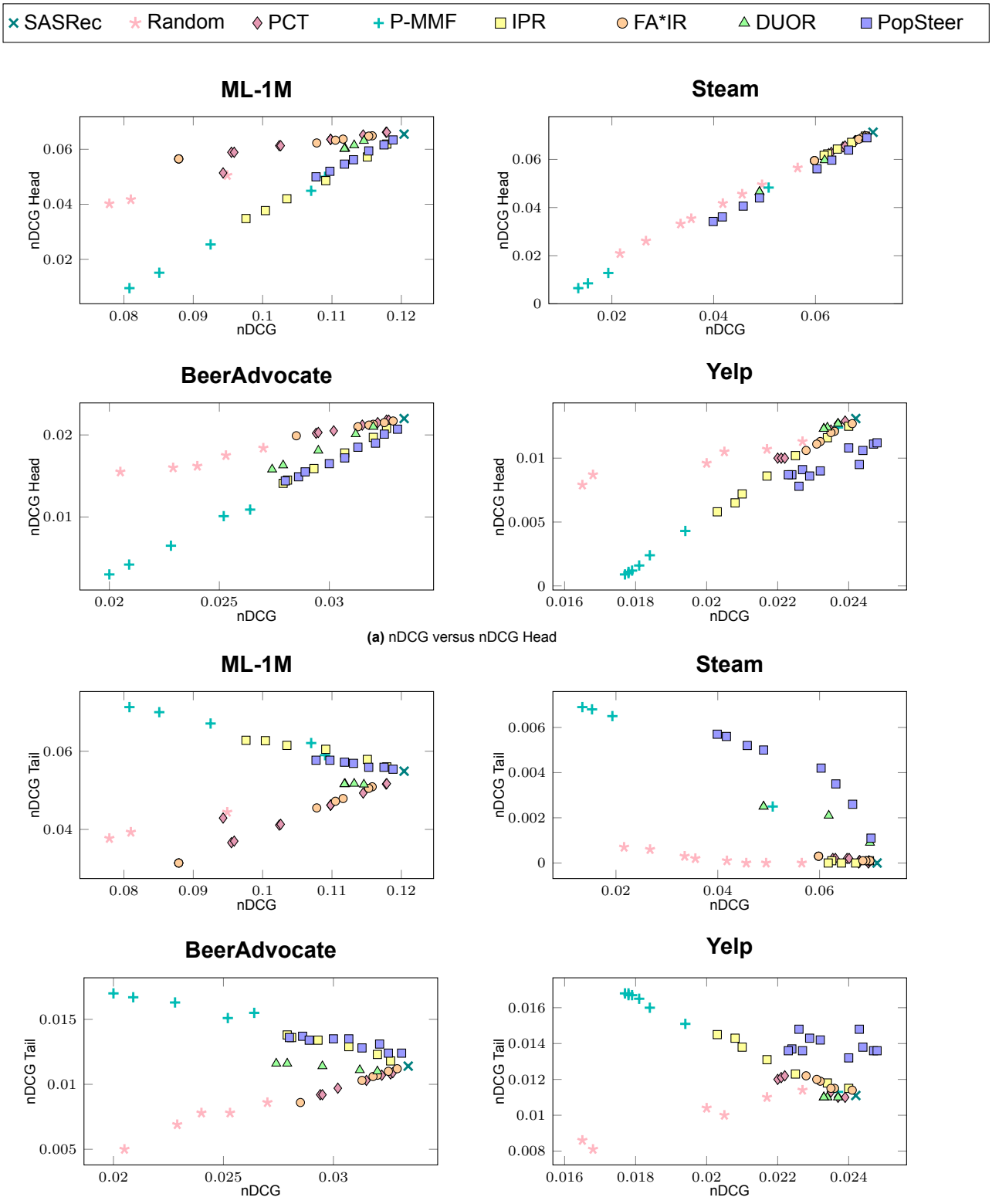


Figure 7.4: Performance comparison of PopSteer with baselines in terms of nDCG and Gini Index when using LightGCN as a backbone.

ments. This stability shows the reliability of adjustments on targeted neuron, which minimizes unintended disruptions to the model's decision-making process compared to other approaches. This controllability is advantageous for deployment, as it allows practitioners to reliably tune the trade-off between fairness and accuracy according to specific application needs without unpredictable outcomes.



(a) nDCG versus nDCG Head

(b) nDCG versus nDCG Tail

Figure 7.5: Performance comparison of PopSteer with baselines in terms of overall nDCG and nDCG on Head and Tail items.

7.1.1.2. LightGCN Results

Similarly to Figs. 7.1 and 7.2, Figs. 7.3 and 7.4 present the accuracy–fairness trade-off for LightGCN. Overall, $\mathrm{PopSteer}$ remains competitive, but it does not consistently outperform the baselines. On the Yelp and BeerAdvocate datasets, under realistic conditions with only a minor drop in nDCG, $\mathrm{PopSteer}$ is outperformed by DUOR, PCT, and IPR in terms of both Gini index and item coverage. On ML-1M, $\mathrm{PopSteer}$ performs competitively and often better than most baselines, though DUOR achieves stronger results, especially in regions with only a small nDCG loss.

On Steam, however, $\mathrm{PopSteer}$ clearly outperforms all baselines. LightGCN's baseline Gini index is 0.9972, which is even higher than that of SASRec, indicating an even stronger popularity bias. As a result, its recommendation lists contain very few long-tail items, limiting the effectiveness of reranking-based baselines. By contrast, $\mathrm{PopSteer}$ directly modifies the internal structure of the model, and thus is not constrained by the lack of long-tail items in the candidate pool. This explains the superior performance of $\mathrm{PopSteer}$ on the Steam dataset.

7.1.2. Head–Tail Performance

Beyond fairness gains obtained from $\mathrm{PopSteer}$, Fig. 7.5 compares the overall nDCG with nDCG on head (Fig. 7.5a) and tail (Fig. 7.5b) items, respectively. Subsection 7.1.1 showed that tail items get more exposure while maintaining accuracy. In this subsection, we examine whether the recommended tail items are relevant to the user. In particular, we look for a debiasing effect in which relevant long-tail items gain rank without harming overall quality, evidenced by higher tail nDCG at similar overall nDCG and only minor changes in head nDCG. Although this is not our primary objective, it is still interesting to assess the debiasing effect of $\mathrm{PopSteer}$. This ensures that fairness gains result in surfacing useful items rather than merely reshuffling exposure.

Fig. 7.5 illustrates the comparison of $\mathrm{PopSteer}$ and baselines in terms of overall nDCG and nDCG on Head and Tail items. The best performance in Figs. 7.5a and 7.5b appears in the lower-right and upper-right, respectively.

In terms of head-nDCG, Fig. 7.5a shows that $\mathrm{PopSteer}$ performs at least as well as or better than other baselines when balancing head-nDCG and overall nDCG. On the ml-1m, steam, and beeradvocate datasets, $\mathrm{PopSteer}$ follows a similar trend to the strongest baselines. For the yelp dataset, however, $\mathrm{PopSteer}$ consistently achieves higher head-nDCG for any given overall nDCG value compared to all baselines.

As shown in Fig. 7.5b, $\mathrm{PopSteer}$ outperforms the baselines in the tradeoff between overall nDCG and tail nDCG, with particularly strong results on the Steam and Yelp datasets. On Steam, where extreme sparsity makes it challenging for most baselines to recommend relevant tail items, $\mathrm{PopSteer}$ significantly improves tail nDCG with only a modest reduction in overall nDCG. On Yelp, it sharply increases tail-nDCG without any loss in overall nDCG. For the ml-1m dataset, while it does not consistently surpass all baselines, $\mathrm{PopSteer}$ achieves performance comparable to the strongest ones.

Results validate that $\mathrm{PopSteer}$ improves the accuracy of tail item recommendations at a manageable reduction in head item accuracy. Analysis of the plots indicates that $\mathrm{PopSteer}$ consistently achieves comparable, and in most cases superior, head–tail nDCG tradeoffs relative to the baselines.

7.2. Interpretability analysis

In this section, we answer **RQ2**.

7.2.1. Manipulating the activation of targeted neuron

To assess the interpretability of $\mathrm{PopSteer}$, we conduct a controlled neuron manipulation study. Specifically, we manually reduce activations of the top K' neurons identified as most strongly associated with popularity ($d_j > 1$) or unpopularity ($d_j < -1$). For each identified neuron j, we subtract its activation by one standard deviation, i.e., $a'_j = a_j - \sigma_j$. This adjustment was chosen to remain within the neuron's expected activation distribution, avoiding extreme perturbations while testing the causal influence on bias.

Fig. 7.6 illustrates how the Gini Index evolves as the activation of these most influential neurons are pro-

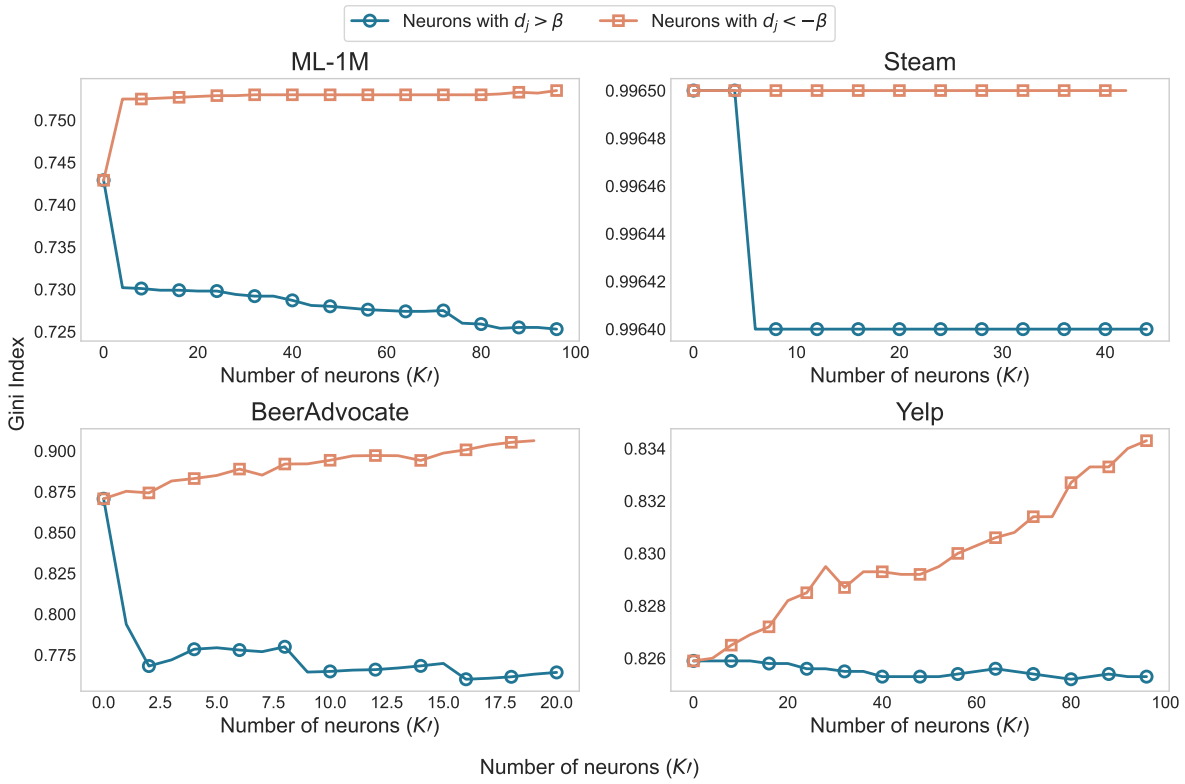


Figure 7.6: Interpretability analysis of PopSteer: effect of reducing activation of K' neurons linked to popularity bias ($\beta = 1$).

Table 7.1: Wasserstein distance between data points in Fig. 7.7: 0 indicates identical and a higher value indicates a more dissimilar distribution. Bolded entries show desirable values: higher distance to "Synthetic popular profiles" and lower distance to "Synthetic unpopular profiles".

	ML-1M	Steam	BeerAdvocate	Yelp
Actual → Synthetic popular	1.53	0.99	1.61	2.11
Steered → Synthetic popular	**5.92**	**1.38**	**3.61**	**2.17**
Actual → Synthetic unpopular	7.25	2.74	3.52	1.21
Steered → Synthetic unpopular	**3.42**	**2.35**	**1.53**	**1.16**

gressively reduced. On Yelp, BeerAdvocate, and ML-1M, reducing the activations of popularity-aligned neurons consistently improves the Gini Index while reducing the activations of unpopularity-aligned neurons worsens the Gini Index. Notably, ML-1M shows a sharp initial change, suggesting primary neurons encode the main signal, with secondary neurons providing support. BeerAdvocate exhibits a similar pattern for popularity-aligned neurons. These results align with SAE literature where neurons specialize in monosemantic features [16, 43, 95]. For Steam, with extreme popularity skew (Gini > 0.99), 1 standard deviation adjustments have minimal effect, as few hyper-popular items dominate exposure, requiring stronger interventions (see Section 7.4).

7.2.2. Analyzing user embedding space

To analyze PopSteer's interpretability, we examine users' embedding space derived from SAE. To visualize this space, we create UMAP [90] by projecting high-dimensional neuron activations from SAE into a 2D space. UMAP preserves local neighborhoods, so users with similar neuron activation patterns appear near each other in the plot.

Specifically, we collect four activation sets from the SAE: (i) real user profiles from the test set, (ii) synthetic-popular profiles, (iii) synthetic-unpopular profiles, and (iv) steered profiles obtained by applying PopSteer to the real ones. Each point in the plot is one user activation vector in the SAE hidden layer. Each set was set to contain $5,000$ data points. We concatenate all four sets and project them to 2D with UMAP (using Euclidean metric), so that points that are close in the SAE space stay close in the

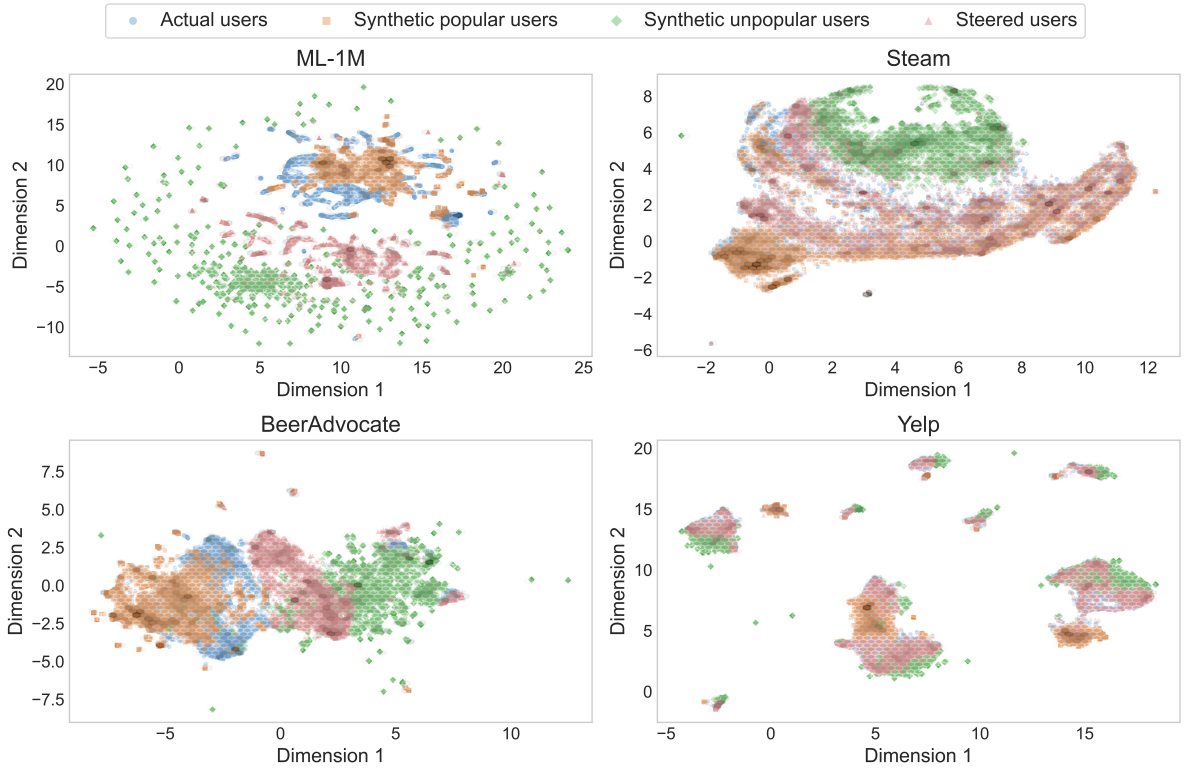


Figure 7.7: 2D UMAP projections of SAE activations for real, synthetic popular/unpopular, and steered user profiles. For ML-1M and BeerAdvocate, $\alpha^{\text{Unpop}} = 3$, $\alpha^{\text{Pop}} = 3$. For Steam and Yelp, $\alpha^{\text{Unpop}} = 1$, $\alpha^{\text{Pop}} = 1$.

plot. This makes it possible to see whether popularity forms its own region. The resulting projections are shown in Fig. 7.7.

Several patterns can be observed in Fig. 7.7: First, it shows that our synthetic data generation successfully isolates the popularity signal. We can derive this from the fact that across all datasets, samples from the synthetic popular and unpopular sets cluster in different regions of the reduced subspace. This shows that the main differentiating factor of these 2 sets is their popularity level, with other factors largely removed. Second, actual profiles align closer to the popular cluster, reflecting inherent popularity bias. This is expected, as the training data itself is skewed toward popular items, making real profiles resemble the synthetic-popular set more closely than the unpopular set in the reduced embedding space. Lastly, it shows that $\mathrm{PopSteer}$'s steering mechanism works - after applying neuron steering with $\mathrm{PopSteer}$, the activations shift in the desired direction: they become more dissimilar to popular sequences and more similar to unpopular sequences.

This visual shift is quantitatively validated in Table 7.1, which computes the 1D Wasserstein distance between the distribution of real profiles and the two synthetic distributions, before and after steering. These changes are statistically significant ($p < 0.01$), validating the reliability of $\mathrm{PopSteer}$ in mitigating bias. Overall, these findings demonstrate that $\mathrm{PopSteer}$ not only mitigates popularity bias effectively but also provides actionable, interpretable insights into the neural basis of bias, enabling more transparent and controllable interventions.

7.3. Ablation study

To verify the effectiveness of our method, we conduct an ablation study with two variants to independently assess key components of $\mathrm{PopSteer}$: (i) the neuron selection process (Noise), and (ii) the targeted steering logic ($\mathrm{RandomSelect}$).

The first variant, denoted as Noise, isolates the impact of our controlled steering mechanism as detailed in Eq. 5.9. Here, we select the same set of neurons as in $\mathrm{PopSteer}$ (based on the highest absolute Cohen's d), but instead of applying bias-aware, directionally consistent modifications to their activations, we inject unstructured perturbations in the form of random Gaussian noise. We tune the hyperpara-

Table 7.2: Ablation study: $\mathrm{PopSteer}$ (PS) vs. Noise vs. $\mathrm{RandomSelect}$ (RS).

		Item Coverage ↑			Gini Index ↓		
Dataset	β	PS	Noise	RS	PS	Noise	RS
ML-1M	0.5	**0.5707**	0.4732	0.475	**0.6645**	0.7464	0.7429
	1.0	**0.5651**	0.4741	0.4791	**0.6698**	0.7438	0.7442
	1.5	**0.5531**	0.4750	0.475	**0.6748**	0.7428	0.7429
Steam	0.5	**0.0513**	0.0242	0.0242	**0.9930**	0.9964	0.995
	1.0	**0.0296**	0.0259	0.023	**0.9954**	0.9964	0.9958
	1.5	**0.0241**	0.0222	0.0222	**0.9960**	0.9964	0.9962
BeerAdvocate	0.5	**0.2998**	0.2268	0.243	**0.7846**	0.8502	0.8409
	1.0	**0.2939**	0.2131	0.2407	**0.7910**	0.8675	0.844
	1.5	**0.2849**	0.2337	0.2315	**0.7989**	0.8433	0.8521
Yelp	0.5	**0.3582**	0.3392	0.3377	**0.8143**	0.8273	0.8259
	1.0	**0.3589**	0.3369	0.3398	**0.8143**	0.8268	0.8268
	1.5	**0.3406**	0.3378	0.3449	**0.8245**	0.8261	0.8232

meters of $\mathrm{PopSteer}$ (α^{Unpop} and α^{Pop}) and the Standard Deviation ξ for the Noise variant such that the drop in nDCG does not exceed 10% relative to the base SASRec model.

The second variant, $\mathrm{RandomSelect}$, ablates the neuron selection logic by randomly sampling the same number of neurons as would be chosen under each β threshold in $\mathrm{PopSteer}$. For these randomly selected neurons, it applies steering adjustments using the same weights w_j and directions (suppress if $d_j > 0$, boost if $d_j < 0$) as those of the top neurons. This preserves the exact magnitudes, signs, and variance-normalized scales of the adjustments but applies them to arbitrary neurons instead, testing whether identifying the correct bias-encoding neurons is crucial for effectiveness.

Results are reported in Table 7.2, where we compare performance across different Cohen's d thresholds (β) in terms of Item Coverage and Gini Index. We observe that $\mathrm{PopSteer}$ consistently yields superior fairness outcomes relative to ablation variants, across all datasets. Noise and $\mathrm{RandomSelect}$ both perform similarly poorly, indicating that neither steering alone nor selection alone suffices. Consistent fairness gains emerge only when targeted steering is combined with Cohen's d–guided neuron selection, as in $\mathrm{PopSteer}$.

7.4. Sensitivity analysis

This section addresses **RQ3**. Fig. 7.8 and 7.9 display sensitivity analysis of α^{Pop} and α^{Unpop} across four datasets (β is set to 0). Fig. 7.8 shows the impact of these hyperparameters on nDCG, while Fig. 7.9 illustrates effects on Gini Index. As these parameters increase, the Gini Index decreases, with predictable patterns. Reductions remain steady until dataset-specific thresholds, after which point gains plateau or reverse.

One interesting observation is the difference in the model receptiveness on α^{Pop} and α^{Unpop}. Suppressing popularity-aligned neurons leads to steeper nDCG declines than boosting unpopularity-aligned ones. One explanation for this is that recommender systems, trained on skewed data, develop a "defensive" over-reliance on popular signals for stability, while niche signals remain underutilized and more elastic to amplification.

However, boosting unpopular-aligned neurons is not without its caveats. In the Steam dataset for instance, varying α^{Unpop} yields no improvements in the Gini Index. This is because amplification of unpopularity-aligned neurons can fail to produce meaningful changes in fairness when the learned unpopular signals are inherently weak or underdeveloped. In highly skewed training distributions like Steam's, where interactions overwhelmingly favor popular items, the model has little incentive to develop robust representations for unpopular signals. These pathways are rarely activated in real-world scenarios, therefore, are insufficiently influential to shift recommendations toward tail items even under boosting.

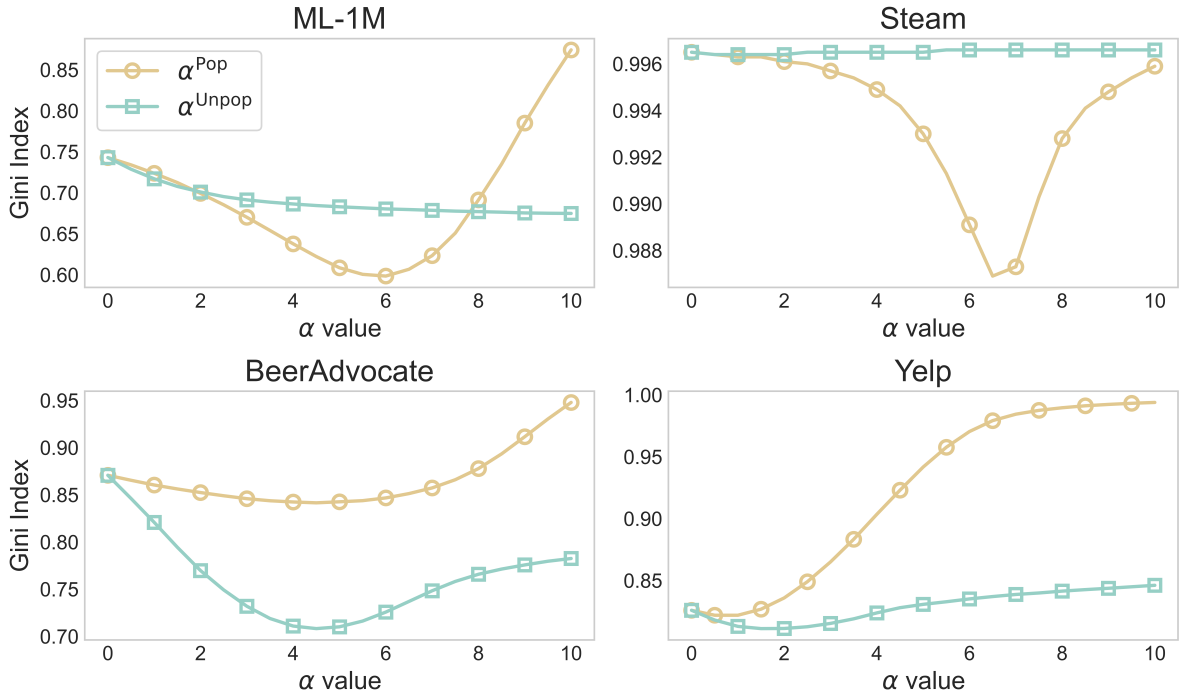


Figure 7.8: The effect of varying α^{Pop} and α^{Unpop} on nDCG across four datasets.

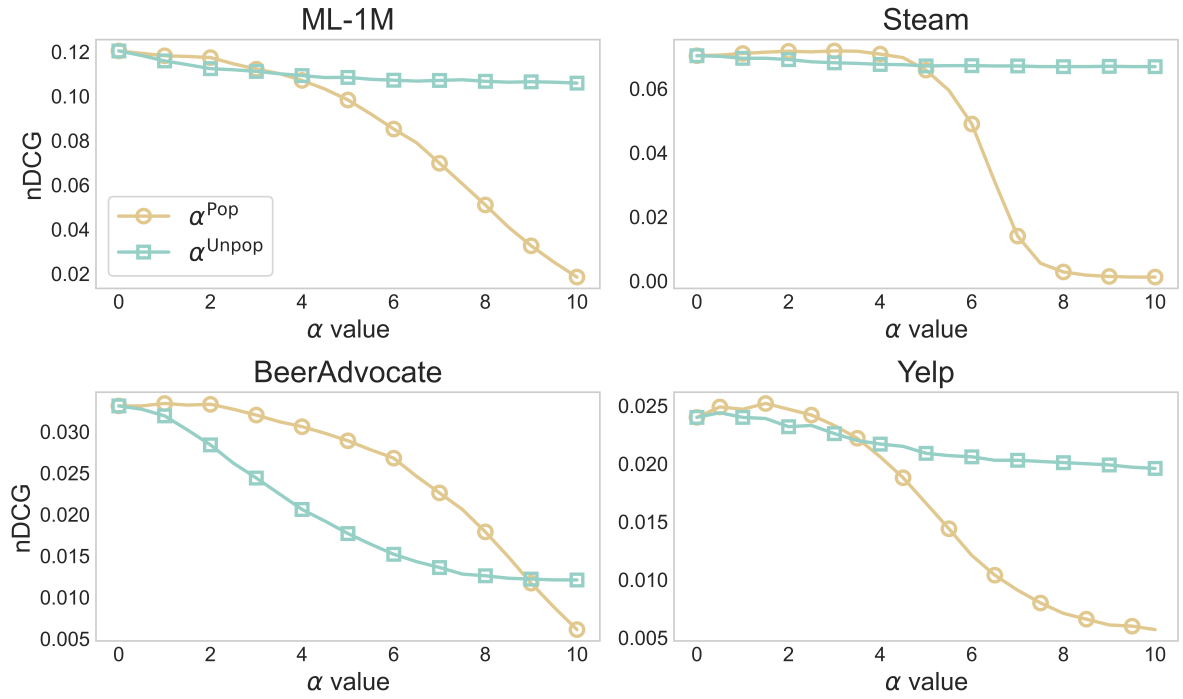


Figure 7.9: The effect of varying α^{Pop} and α^{Unpop} on Gini Index across four datasets.

7.4.1. Impact of steered neuron count

The hyperparameter β is a threshold on the absolute Cohen's d used to select neurons for steering (details in Section 6.5.2). Specifically, $\beta = 1$ means we steer only those neurons whose $|d_j| \geq 1$ in the SAE's hidden layer. To isolate the effect of the neuron count itself, we vary the number of steered neurons directly: setting $\mathcal{N} = 100$ steers the top 100 neurons ranked by $|d_j|$. Note that the main

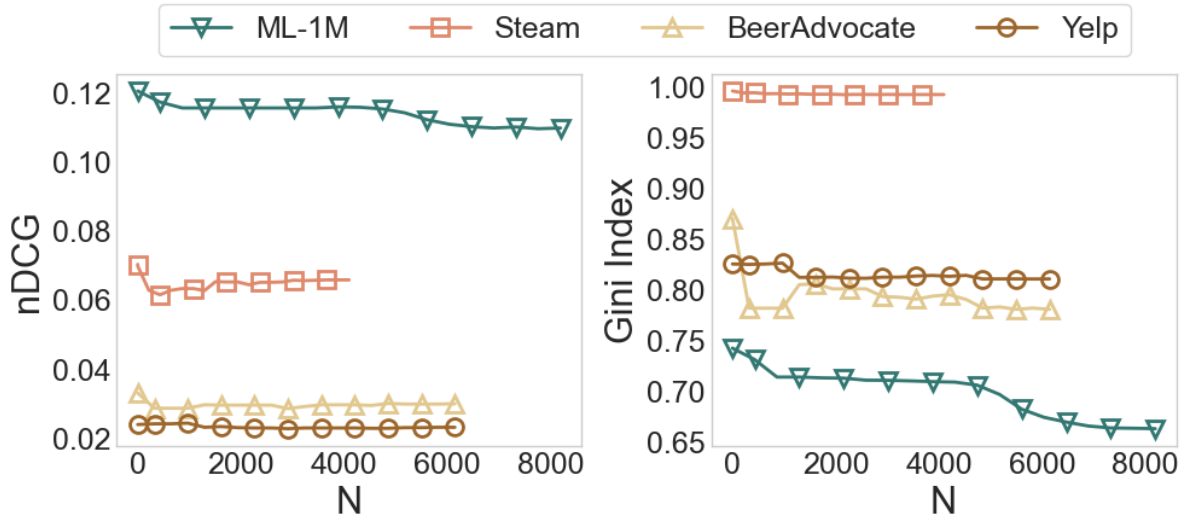


Figure 7.10: Performance analysis of $\mathrm{PopSteer}$ in terms of nDCG and Gini Index with varying steered neuron count $\mathcal{N}$ on four datasets.

hyperparameter is β - $\mathcal{N}$ is only used for sensitivity analysis because it cleanly probes "how many neurons matter" without changing the eligibility threshold. Results are presented in Fig. 7.10.

Across all datasets, the largest changes in both accuracy and fairness occur at small $\mathcal{N}$. On Steam, BeerAdvocate, and Yelp, these metrics stabilize quickly and show minimal change beyond an early threshold. For MovieLens, accuracy and fairness continue to move beyond roughly $\mathcal{N} \approx 5000$, but the changes are gradual compared to the sharp initial change.

We attribute this behavior to the *weighted steering* mechanism: the per-neuron adjustment scales with its Cohen's d (see Section 5.4). Highly biased neurons receive the strongest updates, while neurons with small $|d_j|$ are adjusted negligibly. As a result, the exact binary inclusion boundary (which neurons are steered) matters less once the most biased units are included, resulting in the observed plateau.

7.5. Scalability analysis

To verify the scalability of $\mathrm{PopSteer}$, we compare its inference time to baselines. As shown in Fig. 7.11, $\mathrm{PopSteer}$ achieves the lowest inference time compared to baselines (except Random). Furthermore, for practicality, we also compute the training time of $\mathrm{PopSteer}$ (in hours), as presented in Table 7.3. Results demonstrate that despite the high scale factor s, training time remains manageable. This aligns with findings of the study by Gao et al. [43], which found that good initialization, top K sparsity, and dead-latent prevention make large SAEs learn efficiently. Overall, both the training and inference times of $\mathrm{PopSteer}$ are well within practical limits. This makes the use of $\mathrm{PopSteer}$ highly suitable for production deployment.

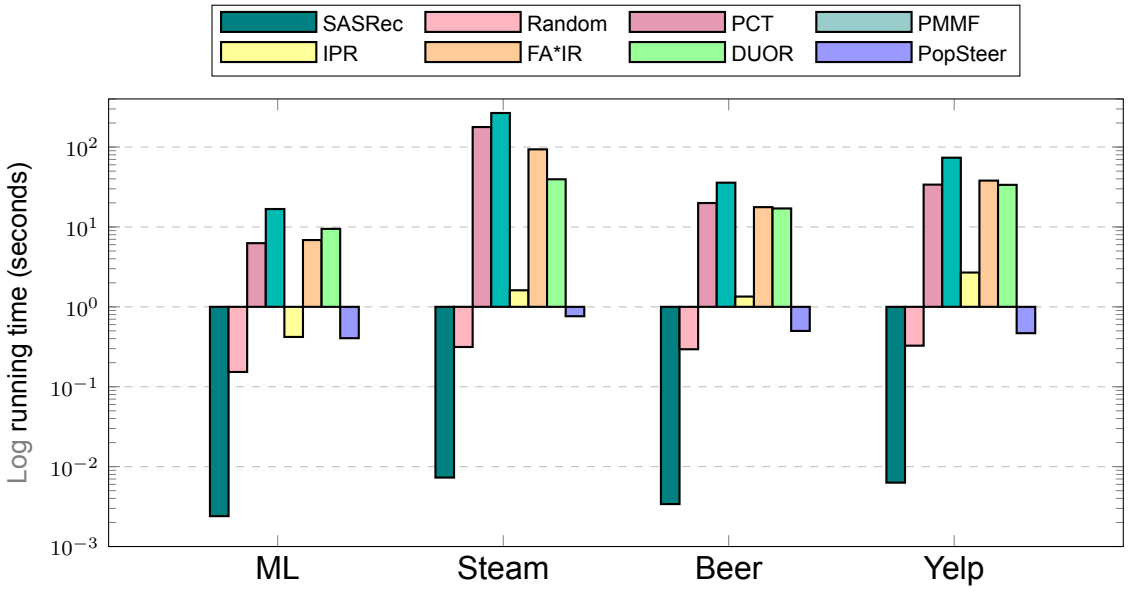


Figure 7.11: Test set inference time of PopSteer versus baselines.

Table 7.3: Training time of PopSteer on each dataset (in hours).

	ML-1M	**Steam**	**BeerAdvocate**	**Yelp**
Training time (h)	0.71	0.41	0.57	0.48

8

Discussion

8.1. Key Findings

8.1.1. PopSteer offers superior accuracy-fairness tradeoffs

Our main results on SASRec in Section 7.1 suggest that PopSteer is highly effective at significantly improving exposure fairness with only a minor accuracy drop, outperforming multiple benchmarks. Furthermore, this is not achieved through merely recommending random tail items, but through recommending tail items that align with user preferences. We saw that many baselines struggled to improve tail nDCG, a pattern especially visible on a highly sparse Steam dataset. This can be traced to the model's limited exposure to tail items in the training data, which prevents it from adequately capturing their characteristics. As a result, while it may be relatively easy to increase the number of tail recommendations, ensuring that the right tail items are recommended remains substantially more difficult.

The main disadvantage of reranking methods is that they only act on the candidate pool that the recommender model has already retrieved and scored. If the model has learned weak or incomplete features for items with few historical interactions, those items often never enter the candidate pool at all. Even when they do, they typically arrive with low and noisy relevance scores, leaving rerankers with nothing reliable to promote. By contrast, representation-level edits change the model's internal features and scoring upstream. They strengthen signals for under-represented items and weaken over-dominant popularity cues, so the right tail items are retrieved and scored higher before any list reshuffling occurs. This yields fairer exposure with less instability than relying on reranking rules alone.

However, post-hoc representation-level edits are not an easy task - as we discussed, polysemanticity makes naive manipulation difficult. PopSteer addresses this by using a sparse autoencoder to re-express the model's activations as a larger, sparse, and more interpretable set of features. This step allows neuron-level interventions in the latent space without affecting performance too much.

8.1.2. Using real data for neuron analysis is suboptimal

Although PopSteer remained competitive, it did not consistently outperform the baselines in the accuracy–fairness trade-off analysis conducted with the LightGCN model. In this experiment, real user profiles rather than synthetic profiles were used during the neuron-analysis stage. We attribute the suboptimal performance to two key factors. First, real users rarely show preferences that lie at the extreme ends of the popularity spectrum which makes it harder to cleanly identify neurons encoding popularity bias. Second, the limited number of interactions in real profiles means that confounding factors are not absent from the popular and unpopular sets. For example, the "popular" group might largely comprise of users who watch popular Star Wars series, reflecting a franchise-specific affinity rather than a genuine preference for mainstream items. As discussed in Section 5.3.1, our synthetic data generation process mitigates these issues by constructing profiles that isolate popularity signals and remove such confounding variables.

This does not imply that the use of real data is without merit. In domains where a high number of users exhibit clear popularity-driven preferences, leveraging real user profiles for neuron analysis can be highly valuable. Our experiments were conducted on relatively small datasets, which likely limited the presence of such users. In larger and more diverse datasets, it is more probable to find users whose preferences strongly align with either popular or long-tail items. In these cases, real data offer the additional advantage of reflecting authentic user behavior, while synthetic profiles, although effective for isolating bias, may introduce unrealistic interaction patterns that add noise to the analysis.

8.1.3. PopSteer is interpretable

Section 7.2 presents empirical evidence for the interpretability of $\mathrm{PopSteer}$. Subsection 7.2.1 shows that suppressing individual neurons yields the expected directional changes in fairness. A striking pattern emerges in the ML-1M and BeerAdvocate datasets: we observe discrete "popularity" neurons. In ML-1M, suppressing one neuron produces a large *increase* in fairness, whereas suppressing a different neuron causes a large *decrease*. In BeerAdvocate, suppressing a single neuron leads to substantial fairness gains. This shows that in certain domains, popularity bias can be localized to a small number of interpretable units. Future research could further investigate the activation dynamics of these neurons and the conditions under which they emerge.

In the case of ML-1M and BeerAdvocate datasets we saw that the SAE allocates a specific "popularity" feature. However, there is no such guarantee - we can't guarantee that Sparse AutoEncoder will choose to encode a specific "popularity" feature. In Section 5.3.2, we established that it doesn't have to - bias may be encoded in multiple "correlated" features with popularity (For instance, SAE may encode an "action" feature, which is highly correlated with popularity) that are not necessarily equivalent to a "popularity" feature. Indeed, on the Steam and Yelp datasets, we saw the absence of a specific "popularity" feature. In these datasets, the effects are distributed across multiple neurons with no single dominant neuron. Despite this, the effects remain predictable. Even when a single dominant unit is absent, clean interventions remain feasible thanks to the sparsity and approximate monosemanticity induced by the SAE together with our steering procedure.

Section 7.2.2 validates that the learned embedding space provides clear, visual evidence of how popularity bias is encoded and how neuron steering alters those representations. Beyond confirming that $\mathrm{PopSteer}$ works as intended, these shifts also provide evidence that bias manifests in a structured, observable way in the latent space. This opens the door for practitioners to use embedding-space movements as a diagnostic tool during research and also as a practical sanity check when deploying debiasing interventions in real systems.

8.1.4. PopSteer's hyperparameters have asymmetric, predictable effects

Section 7.4 results show that modifying the value of key hyperparameters results in predictable changes in fairness. Furthermore, the impacts of α^{Pop} and α^{Unpop} are asymmetric, meaning, suppressing popularity-aligned neurons tends to degrade accuracy more quickly than amplifying unpopularity-aligned neurons. The tradeoff patterns are largely dataset-specific, dependent on the dataset characteristics such as sparsity. Many existing neuron-level intervention methods either rely on a single hyperparameter to uniformly scale activations up or down [63], or else take a cruder approach by directly zeroing out selected neurons [55]. Our results indicate that a more effective strategy is to address both sides of the bias: suppressing popularity-aligned neurons while simultaneously amplifying those aligned with unpopular items. Moreover, we find that the optimal steering strengths for these two directions behave asymmetrically and vary across datasets, implying that they must be tuned independently rather than controlled by a single shared parameter.

8.1.5. Steered neuron count is less important than the key hyperparameters

Section 7.4.1 explored how controlling the number of steered neurons $\mathcal{N}$ affects performance. We find that steering the most biased neurons accounts for most of the observed change, and beyond this set, increasing $\mathcal{N}$ results in diminishing effects on both accuracy and fairness. This follows directly from the weighted steering mechanism: per-neuron updates scale with $|d_j|$ (See Section 5.4), so weakly biased neurons receive negligible adjustments. As a result, the precise choice of $\mathcal{N}$ is comparatively uncritical.

In practice, one can simply set $\beta = 0$ so that all neurons are steered - the weighting still ensures that minimally biased units are adjusted only minimally. This is a practical advantage of weighted steering: users can focus on tuning the steering strengths rather than simultaneously tuning the number of steered neurons. By contrast, hyperparameters α^{Pop} and α^{Unpop} have much larger influence on outcomes, as discussed in Section 8.1.4.

8.1.6. PopSteer is deployable

Section 7.5 results suggest that $\mathrm{PopSteer}$ is deployment-friendly - both the training and inference time of $\mathrm{PopSteer}$ are within practical limits. In general, it has been shown that SAEs can have millions of latents (hidden neurons), with the compute remaining manageable [43]. This is partially because in Sparse Autoencoders, the encoder forward pass is the main cost, whereas Top-K makes the decoder usage sparse so decoding is relatively cheap [43]. In practice, the SAE can be trained offline on cached activations without needing to pass through the base recommender model.

Hyperparameter tuning. Steering hyperparameters ($\alpha^{\text{Pop}}, \alpha^{\text{Unpop}}, \beta$) are inexpensive to tune because the base model and SAE remain fixed and the tuning is done on the validation set. A simple grid search suffices for tuning, though Bayesian optimization or other methods are applicable if the hyperparameter search space is higher-dimensional. During our experiments, we observed weak coupling among the three hyperparameters, so tuning them independently is effective as well.

Tuning training-time hyperparameters is more compute-intensive because each setting requires retraining the SAE. In our experience, increasing the scale factor s beyond a certain point provided no obvious gains in interpretability or accuracy, and simply led to a higher number of dead neurons, independent of the chosen K. We therefore recommend first locating a suitable range for scale factor s that balances reconstruction quality and neuron utilization, and subsequently tuning K within that range.

Combining early stopping with a ReduceLROnPlateau scheduler helps training converge quickly. For hyperparameter tuning, a relatively high learning rate and low patience should be chosen, and the validation loss should be monitored to rapidly rule out poor settings. Once a promising range is identified, patience and, if needed, the learning rate should be readjusted to allow the model more time to converge.

8.2. Limitations

Despite its promise, $\mathrm{PopSteer}$ has clear limitations. It assumes a reasonably expressive, well-trained base recommender whose embedding space already encodes coherent structure. An SAE can only reconstruct activations with a small number of neurons when there are recoverable patterns. If the base model fails to capture fine-grained regularities, the SAE will struggle to learn monosemantic, disentangled features. As an analogy, reconstructing i.i.d. Gaussian noise is futile: with no structure, there is nothing coherent to recover. This issue is especially apparent when the model struggles to estimate user preferences due to either very short or noisy user histories, where the effective signal is weak. In such cases, swapping in SAE reconstructions can introduce nontrivial reconstruction error and degrade ranking quality even with zero steering. A practical mitigation is to reduce sparsity through increasing K (allow more active features per reconstruction) to reduce error. However, a higher K weakens monosemanticity and, ultimately, interpretability by encouraging dependencies among neurons. The resulting feature entanglement makes targeted interventions less clean, so K must be chosen to balance reconstruction fidelity against interpretability.

$\mathrm{PopSteer}$ identifies popularity-aligned neurons by contrasting two synthetic user sets—one that strongly prefers head (popular) items and one that prefers tail (unpopular) items. This relies on the base recommender encoding a learnable "popularity-preference" dimension in user representations. If the dataset lacks a salient niche-vs-blockbuster construct (to put it in another way, if the model does not meaningfully distinguish users by popularity preference), there may be few or no popularity-aligned neurons to steer, and $\mathrm{PopSteer}$'s impact can be limited. As a practical pre-check, one can probe the user embeddings for a popularity-preference signal: train a simple classifier to separate synthetic head-preferring and tail-preferring users. If it cannot distinguish them, $\mathrm{PopSteer}$ likely has limited leverage. Because this scenario was not directly tested in our experiments, we present it as a hypothesis and a direction for future study.

The model is markedly more sensitive to suppressing popularity-aligned neurons than to amplifying long-tail neurons. This can occur when the activations for long-tail features sit near zero: prior work has found that unpopular dimensions tend to have much smaller magnitudes than popularity dimensions [68]. When the baseline signal is that weak, pushing it upward results in limited movement in scores. In such scenarios, most of the attainable mitigation comes from dampening the popular neurons, rather than trying to amplify the scarcely activated long-tail neurons.

Our interpretability analyses support approximate monosemanticity rather than the perfect form. Perfect monosemanticity would require each neuron to encode a single, disentangled feature and behave independently of all others, an assumption the literature regards as practically unattainable at scale. Consistent with this, Section 7.2.1 shows that single-neuron interventions produce largely predictable, directionally correct effects, but the diagnostic plots display small off-target deviations instead of perfectly smooth shifts. These results point to some degree of entanglement and coupling across neurons, explaining why the trajectories are not perfect.

Another limitation of $\mathrm{PopSteer}$ is its procedural overhead: the workflow requires training an SAE, generating synthetic sets, and computing Cohen's d to select target units. In addition to the SAE's training hyperparameters, $\mathrm{PopSteer}$ requires tuning three inference-time hyperparameters for steering. In contexts where complexity, latency, or engineering effort are constrained, simpler methods such as rerankers may be preferable.

8.3. Future Work

Our synthetic user procedure isolates popularity bias by removing confounders. While effective in our experiments, it sits on a realism-utility trade-off - overly unrealistic profiles could, theoretically, inject noise or artifacts even if we did not observe such issues here. We hypothesize that distribution-constrained synthetic profiles that remain close to the training data's user embedding distribution while containing a clear popularity signal will yield cleaner neuron attribution and more stable steering. Future research could explore "smarter" synthetic set generation, such as probabilistic sampling from item popularity gradients [86] or generative models (e.g., GANs) [11] to produce distribution-aligned yet bias-amplified sequences.

Applying the $\mathrm{PopSteer}$ framework for addressing other biases can be an interesting direction for future research. Concretely, to extend this framework beyond popularity, define the target attribute using reliable metadata and construct two synthetic datasets that differ in that attribute (e.g., create two sets: one of highly gender-stereotyped films and another of non–gender-stereotyped films). Then the remainder of the $\mathrm{PopSteer}$ pipeline should be applied unchanged. However, as noted in the Limitations section, effectiveness depends on the attribute being salient in the base model's representations. If the recommender does not encode the attribute as a meaningful signal, steering is unlikely to cause substantial gains.

Future work could assign richer semantic labels to SAE neurons, moving beyond "popular/unpopular" to finer-grained concepts (e.g., indie film, documentary, female-led cast). Exposing these concept-aligned activations would enable explainable recommendations. For example, "Recommended because the indie-film neuron is strongly activated". This would improve transparency for users and operators alike.

As discussed in detail in Section 5.2, we inserted the SAE only at the model's final layer. Future work should systematically compare insertion points across the network (earlier layers vs. the output layer) to assess where steering yields the best fairness–accuracy trade-off. Moreover, because most recommenders learn user and item embeddings separately, evaluating SAEs on the user side, the item side, and jointly is a promising direction.

Future work should explore extending $\mathrm{PopSteer}$ to a wider range of recommender architectures. We chose SASRec for our experiments because it allows straightforward generation of synthetic sequential data. In contrast, for LightGCN, we relied on real user profiles for neuron-level analysis and observed that such real profiles are often suboptimal for this purpose. While synthetic data generation is less direct in non-sequential recommenders, several strategies are available. A simple approach is to construct a synthetic profile in the usual way by selecting a set of items and define the synthetic user's embedding as the mean of the embeddings of those items. This is justified because, in general recom-

mender models, typically, user and item embeddings live in the same space. More complex embedding generation strategies (using the generated synthetic profiles) can be used as well, such as graph-based propagation from neighboring nodes [119] and feature-conditioned neural mapping for cold-start users [48].

In particular, extending PopSteer to LLM-based recommender systems is a promising direction. Recent studies provide consistent evidence that SAEs are highly effective for interpretability and feature steering in Large Language Models (LLMs). This extension is especially interesting given the growing role of LLMs in modern recommenders, where understanding and controlling latent biases remains a critical challenge.

8.4. Ethical Considerations

Our work focuses on mitigating popularity bias in recommender systems, which has implications for fairness, user experience, and content exposure. By reducing the overrepresentation of popular items and amplifying relevant tail items, PopSteer aims to create a more equitable recommendation environment. However, several ethical aspects should be considered:

- **Fairness is multi-objective.** Popularity mitigation improves head–tail balance but can conflict with other forms of bias, such as user-centric fairness (equal utility across users) and provider-centric fairness (equitable exposure across creators). Therefore, it is important to treat fairness as a Pareto problem - report multiple metrics (for instance, exposure parity by item cohort, per-user utility, provider Gini) and prefer settings that improve one objective without sharply degrading others. Strong steering without extensive testing should be avoided, especially in sensitive domains such as finance, healthcare, and news.
- **User autonomy and transparency.** Steering latent activations can change recommendations in ways users do not expect. To maintain trust, it is best to provide brief explanations, such as "diversifying away from blockbusters" and visible controls. Rather than exposing raw hyperparameters, an intuitive control, such as a slider controlling "popularity emphasis" can be offered with safe defaults and the ability to revert changes.
- **Distributional impacts and potential harm.** Reallocating exposure creates winners and losers across genres, regions, and newcomer creators. Prior to deployment, demographics tests should be conducted across multiple groups based on content category, creator group, and geography. In addition to standard accuracy and fairness metrics, it is advisable to monitor revenue and complaint rate per group. Guardrails such as "minimum quality thresholds" can be added so tail amplification does not surface low-quality or unsafe content.
- **Dataset limitations and privacy.** Our experiments rely on publicly available datasets that may not represent all user populations. Applying PopSteer in real-world systems should account for dataset biases, data sparsity, and potential privacy concerns. In production, data collection should be limited to what is necessary and be compliant with applicable data-protection regulations such as GDPR [40].

9

Conclusion

Popularity bias is a persistent challenge in recommender systems: a small set of head items receives disproportionate exposure while long-tail items are under-represented. Most mitigation methods are black-box, offering little insight into how they work. This thesis introduces a post-processing approach, $\mathrm{PopSteer}$, that brings neuron-level interpretability and control through three stages: (i) training a Sparse Autoencoder, (ii) analysing neurons with synthetic profiles to identify popularity-aligned features, and (iii) mitigating bias by targeted steering of those neurons.

Across four datasets with SASRec as the backbone, $\mathrm{PopSteer}$ delivers a stronger accuracy–fairness trade-off than baseline methods (RQ1): fairness improves substantially with minimal impact on nDCG, and the additional exposure comes from relevant long-tail items rather than recommending random tail items. We also evaluated $\mathrm{PopSteer}$ with LightGCN, replacing synthetic profiles with real users for neuron analysis; as expected, gains were smaller - likely because real-user profiles provide weaker contrast for isolating popularity-biased features. Despite this, the method remained competitive and controllable. The ablation study of $\mathrm{PopSteer}$ further confirmed that the gains achieved by $\mathrm{PopSteer}$ are not random and both the neuron selection and steering mechanism of $\mathrm{PopSteer}$ is crucial for achieving strong results.

We evaluated the interpretability of $\mathrm{PopSteer}$ (RQ2) by systematically decreasing neuron activations and tracking the resulting changes in fairness. The effects were predictable and monotonic: reducing the activation of “popular” neurons increased fairness, while suppressing “unpopular” neurons degraded it. Interestingly, on two datasets, we found that there are a few neurons that encode most of the bias, while in others, the bias is distributed among many neurons. Further analyses of activation trajectories in a UMAP-reduced subspace showed steered representations moving away from popularity-aligned directions, confirming that $\mathrm{PopSteer}$ isolates and mitigates bias directly in the embedding space.

We conducted sensitivity analysis of the key hyperparameters (RQ3). Results suggest that the steering hyperparameters α^{Pop} and α^{Unpop} have predictable effects on accuracy and fairness. Furthermore, it was found that the effects of these hyperparameters are asymmetric - α^{Pop} typically causes a sharp increase in fairness and decrease in accuracy, while the effect of α^{Unpop} on fairness and accuracy is more gradual. It was concluded that they need to be adjusted differently, dependent on the specific use-case.

The effect of steered neuron count $\mathcal{N}$ on accuracy and fairness was also investigated. Results suggest that it is not as important as other key hyperparameters, since the steering mechanism already adjusts neuron activations based on their bias level, making the choice of steering inclusion threshold less critical.

Beyond effectiveness, PopSteer is deployable. Inference and training time of $\mathrm{PopSteer}$ was found to be within the acceptable threshold. Tuning steering hyperparameters of $\mathrm{PopSteer}$ is computationally inexpensive, since they are tuned after model training. Training hyperparameters K and scale factor s are harder to tune through grid search. We proposed several strategies to quickly tune these hy-

perparameters, including tuning K after finding a suitable range for s, starting with high learning rate etc.

Despite the promise, $\mathrm{PopSteer}$ has a number of limitations. $\mathrm{PopSteer}$ depends on the base model encoding meaningful structure. When user histories are noisy or the data do not support a salient popularity-preference dimension, the SAE may learn weaker features, making steering less effective. The pipeline (SAE training, synthetic set generation, Cohen's d computation) adds engineering overhead and depending on the use-case, simpler techniques may be preferred. Furthermore, while the representations are more monosemantic than the original dense embeddings, full monosemanticity is not achieved. The Inherent trade-off between reconstruction error and monosemanticity limits how far we can push interpretability and, by extension, the effectiveness of targeted steering.

Overall, the results support a simple takeaway: targeted, neuron-level edits can achieve fairness improvements with tolerable accuracy cost and with explanations for *why* they work. Promising future directions include richer semantic labeling of neurons, smarter synthetic data generation, and applying the same pipeline to other biases, further advancing the integration of interpretability and operational control in recommender systems.

References

[1] Himan Abdollahpouri, Robin Burke and Bamshad Mobasher. 'Controlling popularity bias in learning-to-rank recommendation'. In: *Proceedings of the eleventh ACM conference on recommender systems*. 2017, pp. 42–46.

[2] Himan Abdollahpouri et al. 'The unfairness of popularity bias in recommendation'. In: *arXiv preprint arXiv:1907.13286* (2019).

[3] Himan Abdollahpouri et al. 'User-centered evaluation of popularity bias in recommender systems'. In: *Proceedings of the 29th ACM conference on user modeling, adaptation and personalization*. 2021, pp. 119–129.

[4] Etowah Adams et al. 'From mechanistic interpretability to mechanistic biology: Training, evaluating, and interpreting sparse autoencoders on protein language models'. In: *bioRxiv* (2025).

[5] Milad Ahmadian et al. 'Integration of deep sparse autoencoder and particle swarm optimization to develop a recommender system'. In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2021, pp. 2524–2530.

[6] Sanjeev Arora et al. 'Linear algebraic structure of word senses, with applications to polysemy'. In: *Transactions of the Assocumapiation for Computational Linguistics* 6 (2018), pp. 483–495.

[7] Dor Bank, Noam Koenigstein and Raja Giryes. 'Autoencoders'. In: *Machine learning for data science handbook: data mining and knowledge discovery handbook* (2023), pp. 353–374.

[8] Oren Barkan et al. 'Explainable recommendations via attentive multi-persona collaborative filtering'. In: *Proceedings of the 14th ACM Conference on Recommender Systems*. 2020, pp. 468–473.

[9] David Bau et al. 'Understanding the role of individual units in a deep neural network'. In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30071–30078.

[10] Alejandro Bellogín, Pablo Castells and Iván Cantador. 'Statistical biases in information retrieval metrics for recommender systems'. In: *Information Retrieval Journal* 20 (2017), pp. 606–634.

[11] Jesús Bobadilla et al. 'Creating synthetic datasets for collaborative filtering recommender systems using generative adversarial networks'. In: *Knowledge-Based Systems* 280 (2023), p. 111016.

[12] Anand V Bodapati. 'Recommendation systems with purchase data'. In: *Journal of marketing research* 45.1 (2008), pp. 77–93.

[13] Tesfaye Fenta Boka, Zhendong Niu and Rama Bastola Neupane. 'A survey of sequential recommendation systems: Techniques, evaluation, and future directions'. In: *Information Systems* 125 (2024), p. 102427.

[14] Ludovico Boratto, Gianni Fenu and Mirko Marras. 'Connecting user and item perspectives in popularity debiasing for collaborative recommendation'. In: *Information Processing & Management* 58.1 (2021), p. 102387.

[15] Simone Borg Bruun, Maria Maistro and Christina Lioma. 'Feature attribution explanations of session-based recommendations'. In: *European Conference on Information Retrieval*. Springer. 2025, pp. 55–71.

[16] Trenton Bricken et al. 'Towards Monosemanticity: Decomposing Language Models With Dictionary Learning'. In: *Transformer Circuits Thread* (2023). https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[17] Collin Burns et al. 'Discovering latent knowledge in language models without supervision'. In: *arXiv preprint arXiv:2212.03827* (2022).

[18] Bart Bussmann, Patrick Leask and Neel Nanda. 'Batchtopk sparse autoencoders'. In: *arXiv preprint arXiv:2412.06410* (2024).

[19] Bart Bussmann et al. 'Learning multi-level features with matryoshka sparse autoencoders'. In: *arXiv preprint arXiv:2503.17547* (2025).

[20] Taner Cagali, Mehrnoosh Sadrzadeh and Chris Newell. 'Enhancing personalised recommendations with the use of multimodal information'. In: *2021 IEEE International Symposium on Multimedia (ISM)*. IEEE. 2021, pp. 186–190.

[21] Nick Cammarata et al. 'Curve detectors'. In: *Distill* 5.6 (2020), e00024–003.

[22] Òscar Celma and Pedro Cano. 'From hits to niches? or how popular artists can bias music recommendation and discovery'. In: *Proceedings of the 2nd KDD workshop on large-scale recommender systems and the netflix prize competition*. 2008, pp. 1–8.

[23] Jiawei Chen et al. 'Adap-τ: Adaptively modulating embedding magnitude for recommendation'. In: *Proceedings of the ACM Web Conference 2023*. 2023, pp. 1085–1096.

[24] Jiawei Chen et al. 'Bias and debias in recommender system: A survey and future directions'. In: *ACM Transactions on Information Systems* 41.3 (2023), pp. 1–39.

[25] Tong Chen et al. 'Air: Attentional intention-aware recommender systems'. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, pp. 304–315.

[26] Xu Chen et al. 'Personalized fashion recommendation with visual explanations based on multimodal attention network: Towards visually explainable recommendation'. In: *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*. 2019, pp. 765–774.

[27] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. routledge, 2013.

[28] Andrew Collins et al. 'A study of position bias in digital library recommender systems'. In: *arXiv preprint arXiv:1802.06565* (2018).

[29] Council of European Union. *Council regulation (EU) no 269/2014*. `http://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1416170084502&uri=CELEX:32014R0269`. 2014.

[30] Amit Datta, Michael Carl Tschantz and Anupam Datta. 'Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination'. In: *arXiv preprint arXiv:1408.6491* (2014).

[31] Can Demircan et al. 'Sparse autoencoders reveal temporal difference learning in large language models'. In: *arXiv preprint arXiv:2410.01280* (2024).

[32] Jun Deng et al. 'Sparse autoencoder-based feature transfer learning for speech emotion recognition'. In: *2013 humaine association conference on affective computing and intelligent interaction*. IEEE. 2013, pp. 511–516.

[33] Sanjeev Dhawan et al. 'A novel deep learning approach toward efficient and accurate recommendation using improved alternating least squares in social media'. In: *Journal of The Institution of Engineers (India): Series B* 105.3 (2024), pp. 657–675.

[34] Karlijn Dinnissen and Christine Bauer. 'Amplifying artists' voices: Item provider perspectives on influence and fairness of music streaming platforms'. In: *Proceedings of the 31st ACM Conference on User Modeling, Adaptation and Personalization*. 2023, pp. 238–249.

[35] Qiang Dong, Quan Yuan and Yang-Bo Shi. 'Alleviating the recommendation bias via rank aggregation'. In: *Physica A: Statistical Mechanics and its Applications* 534 (2019), p. 122073.

[36] Xin Dong et al. 'Asymmetrical hierarchical networks with attentive interactions for interpretable review-based recommendation'. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 05. 2020, pp. 7667–7674.

[37] David L Donoho. 'Compressed sensing'. In: *IEEE Transactions on information theory* 52.4 (2006), pp. 1289–1306.

[38] Esin Durmus et al. *Evaluating Feature Steering: A Case Study in Mitigating Social Biases*. 25th Oct. 2024. URL: `https://anthropic.com/research/evaluating-feature-steering`.

[39] Nelson Elhage et al. 'Toy models of superposition'. In: *arXiv preprint arXiv:2209.10652* (2022).

[40] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). 4th May 2016. URL: `https://data.europa.eu/eli/reg/2016/679/oj` (visited on 13/04/2023).

[41] Zuohui Fu et al. 'Fairness-aware explainable recommendation over knowledge graphs'. In: *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 2020, pp. 69–78.

[42] Chen Gao et al. 'A survey of graph neural networks for recommender systems: Challenges, methods, and directions'. In: *ACM Transactions on Recommender Systems* 1.1 (2023), pp. 1–51.

[43] Leo Gao et al. 'Scaling and evaluating sparse autoencoders'. In: *arXiv preprint arXiv:2406.04093* (2024).

[44] Azin Ghazimatin et al. 'Prince: Provider-side interpretability with counterfactual explanations in recommender systems'. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 196–204.

[45] Gabriel Goh. 'Decoding the thought vector'. In: *URL: https://gabgoh. github. io/ThoughtVectors* (2016).

[46] Mert Gulsoy et al. 'DUoR: Dynamic User-oriented re-Ranking calibration strategy for popularity bias treatment of recommendation algorithms'. In: *International Journal of Human-Computer Studies* (2025), p. 103578.

[47] Muhammad Umair Haider et al. 'Neurons Speak in Ranges: Breaking Free from Discrete Neuronal Attribution'. In: *arXiv preprint arXiv:2502.06809* (2025).

[48] Will Hamilton, Zhitao Ying and Jure Leskovec. 'Inductive representation learning on large graphs'. In: *Advances in neural information processing systems* 30 (2017).

[49] Christian Hansen et al. 'Shifting consumption towards diverse content on music streaming platforms'. In: *Proceedings of the 14th ACM international conference on web search and data mining*. 2021, pp. 238–246.

[50] Ruben Härle et al. 'SCAR: Sparse Conditioned Autoencoders for Concept Detection and Steering in LLMs'. In: *arXiv preprint arXiv:2411.07122* (2024).

[51] F Maxwell Harper and Joseph A Konstan. 'The movielens datasets: History and context'. In: *Acm transactions on interactive intelligent systems (tiis)* 5.4 (2015), pp. 1–19.

[52] Ruining He, Wang-Cheng Kang and Julian McAuley. 'Translation-based recommendation'. In: *Proceedings of the eleventh ACM conference on recommender systems*. 2017, pp. 161–169.

[53] Xiangnan He et al. 'Lightgcn: Simplifying and powering graph convolution network for recommendation'. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 639–648.

[54] Xiangnan He et al. 'Neural collaborative filtering'. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[55] Praveen Hegde. 'Effectiveness of Sparse Autoencoder for understanding and removing gender bias in LLMs'. In: *NeurIPS 2024 Workshop on Scientific Methods for Understanding Deep Learning*.

[56] Geoffrey E Hinton and Ruslan R Salakhutdinov. 'Reducing the dimensionality of data with neural networks'. In: *science* 313.5786 (2006), pp. 504–507.

[57] Yunfeng Hou et al. 'Explainable recommendation with fusion of aspect information'. In: *World Wide Web* 22.1 (2019), pp. 221–240.

[58] Xiaowen Huang et al. 'Explainable interaction-driven user modeling over knowledge graph for sequential recommendation'. In: *proceedings of the 27th ACM international conference on multimedia*. 2019, pp. 548–556.

[59] Tatsuro Inaba et al. 'How LLMs Learn: Tracing Internal Representations with Sparse Autoencoders'. In: *arXiv preprint arXiv:2503.06394* (2025).

[60] Amir H Jadidinejad, Craig Macdonald and Iadh Ounis. 'Unifying explicit and implicit feedback for rating prediction and ranking recommendation tasks'. In: *Proceedings of the 2019 ACM SIGIR international conference on theory of information retrieval*. 2019, pp. 149–156.

[61] Hao Jiang et al. 'RCENR: A reinforced and contrastive heterogeneous network reasoning model for explainable news recommendation'. In: *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*. 2023, pp. 1710–1720.

[62] Amir Reza Kalantarnezhad and Javad Hamidzadeh. 'MCRS-SAE: Multi-criteria recommender system based on sparse autoencoder'. In: *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE. 2022, pp. 117–122.

[63] Hao Kang, Tevin Wang and Chenyan Xiong. 'Interpret and control dense retrieval with sparse latent features'. In: *arXiv preprint arXiv:2411.00786* (2024).

[64] Wang-Cheng Kang and Julian McAuley. 'Self-Attentive Sequential Recommendation'. In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 197–206. DOI: 10.1109/ICDM.2018.00035.

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

[66] Andrej Karpathy, Justin Johnson and Li Fei-Fei. 'Visualizing and understanding recurrent networks'. In: *arXiv preprint arXiv:1506.02078* (2015).

[67] Dahye Kim and Deepti Ghadiyaram. 'Concept Steerers: Leveraging K-Sparse Autoencoders for Controllable Generations'. In: *arXiv preprint arXiv:2501.19066* (2025).

[68] Dain Kim, Jinhyeok Park and Dongwoo Kim. 'Test-time embedding normalization for popularity bias mitigation'. In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2023, pp. 4023–4027.

[69] Fraser King et al. 'Leveraging sparse autoencoders to reveal interpretable features in geophysical models'. In: *Journal of Geophysical Research: Machine Learning and Computation* 2.4 (2025), e2025JH000769.

[70] Anton Klenitskiy et al. 'Sparse Autoencoders for Sequential Recommendation Models: Interpretation and Flexible Control'. In: *arXiv preprint arXiv:2507.12202* (2025).

[71] Anastasiia Klimashevskaia et al. 'A survey on popularity bias in recommender systems'. In: *User Modeling and User-Adapted Interaction* 34.5 (2024), pp. 1777–1834.

[72] Yehuda Koren, Steffen Rendle and Robert Bell. 'Advances in collaborative filtering'. In: *Recommender systems handbook* (2021), pp. 91–142.

[73] Kenneth Kreutz-Delgado et al. 'Dictionary learning algorithms for sparse representation'. In: *Neural computation* 15.2 (2003), pp. 349–396.

[74] Firas Laakom et al. 'Reducing redundancy in the bottleneck representation of autoencoders'. In: *Pattern Recognition Letters* 178 (2024), pp. 202–208.

[75] Michael Lan et al. 'Sparse autoencoders reveal universal feature spaces across large language models'. In: *arXiv preprint arXiv:2410.06981* (2024).

[76] Jaehoon Lee et al. 'Deep neural networks as gaussian processes'. In: *arXiv preprint arXiv:1711.00165* (2017).

[77] Yuxuan Lei et al. 'Recexplainer: Aligning large language models for explaining recommendation models'. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 1530–1541.

[78] Oleg Lesota et al. 'Analyzing item popularity bias of music recommender systems: are different genders equally affected?' In: *Proceedings of the 15th ACM conference on recommender systems*. 2021, pp. 601–606.

[79] Omer Levy and Yoav Goldberg. 'Linguistic regularities in sparse and explicit word representations'. In: *Proceedings of the eighteenth conference on computational natural language learning*. 2014, pp. 171–180.

[80] Roger Zhe Li, Julián Urbano and Alan Hanjalic. 'Leave no user behind: Towards improving the utility of recommender systems for non-mainstream users'. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 2021, pp. 103–111.

[81] Yakun Li et al. 'Deep sparse autoencoder prediction model based on adversarial learning for cross-domain recommendations'. In: *Knowledge-Based Systems* 220 (2021), p. 106948.

[82] Ninghao Liu et al. 'Explainable recommender systems via resolving learning representations'. In: *Proceedings of the 29th ACM international conference on information & knowledge management*. 2020, pp. 895–904.

[83] Yiming Liu, Xuezhi Cao and Yong Yu. 'Are you influenced by others when rating? Improve rating prediction by conformity modeling'. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 269–272.

[84] Linyuan Lü et al. 'Recommender systems'. In: *Physics reports* 519.1 (2012), pp. 1–49.

[85] Huanrui Luo, Ning Yang and S Yu Philip. 'Hybrid deep embedding for recommendations with dynamic aspect-level explanations'. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 870–879.

[86] Haokai Ma et al. 'Negative sampling in recommendation: A survey and future directions'. In: *arXiv preprint arXiv:2409.07237* (2024).

[87] Masoud Mansoury. 'Understanding and mitigating multi-sided exposure bias in recommender systems'. In: *ACM SIGWEB Newsletter* 2022.Autumn (2022), pp. 1–4.

[88] Masoud Mansoury et al. 'A graph-based approach for mitigating multi-sided exposure bias in recommender systems'. In: *ACM Transactions on Information Systems (TOIS)* 40.2 (2021), pp. 1–31.

[89] Julian J. McAuley and Jure Leskovec. 'From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews'. In: *Proceedings of the 22nd International World Wide Web Conference (WWW)*. 2013. URL: `https://cseweb.ucsd.edu/~jmcauley/pdfs/www13.pdf`.

[90] Leland McInnes, John Healy and James Melville. 'Umap: Uniform manifold approximation and projection for dimension reduction'. In: *arXiv preprint arXiv:1802.03426* (2018).

[91] Tomáš Mikolov, Wen-tau Yih and Geoffrey Zweig. 'Linguistic regularities in continuous space word representations'. In: *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*. 2013, pp. 746–751.

[92] Neel Nanda, Andrew Lee and Martin Wattenberg. 'Emergent linear representations in world models of self-supervised sequence models'. In: *arXiv preprint arXiv:2309.00941* (2023).

[93] Andrew Ng et al. 'Sparse autoencoder'. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[94] Caio Nóbrega and Leandro Marinho. 'Towards explaining recommendations through local surrogate models'. In: *Proceedings of the 34th ACM/SIGAPP symposium on applied computing*. 2019, pp. 1671–1678.

[95] Charles O'Neill et al. 'Disentangling dense embeddings with sparse autoencoders'. In: *arXiv preprint arXiv:2408.00657* (2024).

[96] Chris Olah, Alexander Mordvintsev and Ludwig Schubert. 'Feature visualization'. In: *Distill* 2.11 (2017), e7.

[97] Chris Olah et al. 'Zoom in: An introduction to circuits'. In: *Distill* 5.3 (2020), e00024–001.

[98] Yoon-Joo Park and Alexander Tuzhilin. 'The long tail of recommender systems and how to leverage it'. In: *Proceedings of the 2008 ACM conference on Recommender systems*. 2008, pp. 11–18.

[99] Chathurdara Sri Nadith Pathirage et al. 'Development and application of a deep learning–based sparse autoencoder framework for structural damage identification'. In: *Structural Health Monitoring* 18.1 (2019), pp. 103–122.

[100] Majjed Al-Qatf et al. 'Deep learning approach combining sparse autoencoder with SVM for network intrusion detection'. In: *Ieee Access* 6 (2018), pp. 52843–52856.

[101] Alec Radford, Luke Metz and Soumith Chintala. 'Unsupervised representation learning with deep convolutional generative adversarial networks'. In: *arXiv preprint arXiv:1511.06434* (2015).

[102] Senthooran Rajamanoharan et al. 'Improving dictionary learning with gated sparse autoencoders'. In: *arXiv preprint arXiv:2404.16014* (2024).

[103] Senthooran Rajamanoharan et al. 'Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders'. In: *arXiv preprint arXiv:2407.14435* (2024).

[104] Yoshifumi Seki and Takanori Maehara. 'A method to anonymize business metrics to publishing implicit feedback datasets'. In: *Proceedings of the 14th ACM Conference on Recommender Systems*. 2020, pp. 4–12.

[105] Zhenfeng Shao et al. 'Remote sensing image super-resolution using sparse representation and coupled sparse autoencoder'. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12.8 (2019), pp. 2663–2674.

[106] Juntao Tan et al. 'Counterfactual explainable recommendation'. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 1784–1793.

[107] Adly Templeton et al. 'Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet'. In: *Transformer Circuits Thread* (2024). URL: https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

[108] Poonam B Thorat, Rajeshwari M Goudar and Sunita Barve. 'Survey on collaborative filtering, content-based filtering and hybrid recommendation system'. In: *International Journal of Computer Applications* 110.4 (2015).

[109] Chenyang Wang et al. 'Two-sided calibration for quality-aware responsible recommendation'. In: *Proceedings of the 17th ACM Conference on Recommender Systems*. 2023, pp. 223–233.

[110] Jiayin Wang et al. 'Interpret the Internal States of Recommendation Model with Sparse Autoencoder'. In: *arXiv preprint arXiv:2411.06112* (2024).

[111] Nan Wang et al. 'Explainable recommendation via multi-task learning in opinionated text data'. In: *The 41st international ACM SIGIR conference on research & development in information retrieval*. 2018, pp. 165–174.

[112] Shoujin Wang et al. 'Trustworthy recommender systems'. In: *ACM Transactions on Intelligent Systems and Technology* 15.4 (2024), pp. 1–20.

[113] Xiang Wang et al. 'Kgat: Knowledge graph attention network for recommendation'. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 950–958.

[114] Xiting Wang et al. 'A reinforcement learning framework for explainable recommendation'. In: *2018 IEEE international conference on data mining (ICDM)*. IEEE. 2018, pp. 587–596.

[115] Xiuling Wang and Wendy Hui Wang. 'Providing item-side individual fairness for deep recommender systems'. In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 2022, pp. 117–127.

[116] Pierre Wolinski and Julyan Arbel. 'Gaussian pre-activations in neural networks: Myth or reality?' In: *arXiv preprint arXiv:2205.12379* (2022).

[117] Chen Xu et al. 'P-MMF: Provider max-min fairness re-ranking in recommender system'. In: *Proceedings of the ACM Web Conference 2023*. 2023, pp. 3701–3711.

[118] Yelp Inc. *Yelp Open Dataset*. https://business.yelp.com/data/resources/open-dataset/. Accessed: 2025-08-14.

[119] Rex Ying et al. 'Graph convolutional neural networks for web-scale recommender systems'. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

[120] Meike Zehlike et al. 'Fa* ir: A fair top-k ranking algorithm'. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 1569–1578.

[121] Jingjing Zhang et al. 'Consumption and performance: Understanding longitudinal dynamics of recommender systems via an agent-based simulation framework'. In: *Information Systems Research* 31.1 (2020), pp. 76–101.

[122] Mi Zhang and Neil Hurley. 'Niche product retrieval in top-n recommendation'. In: *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 1. IEEE. 2010, pp. 74–81.

[123] Shuai Zhang et al. 'Deep learning based recommender system: A survey and new perspectives'. In: *ACM computing surveys (CSUR)* 52.1 (2019), pp. 1–38.

[124] Xuejian Zhang et al. 'An interpretable and scalable recommendation method based on network embedding'. In: *IEEE Access* 7 (2019), pp. 9384–9394.

[125] Yifei Zhang et al. 'Mitigating the popularity bias of graph collaborative filtering: A dimensional collapse perspective'. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 67533–67550.

[126] Yin Zhang et al. 'Empowering long-tail item recommendation through cross decoupling network (CDN)'. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2023, pp. 5608–5617.

[127] Yongfeng Zhang, Xu Chen et al. 'Explainable recommendation: A survey and new perspectives'. In: *Foundations and Trends® in Information Retrieval* 14.1 (2020), pp. 1–101.

[128] Yu Zhang et al. 'A survey on neural network interpretability'. In: *IEEE transactions on emerging topics in computational intelligence* 5.5 (2021), pp. 726–742.

[129] Jie Zhao, Ziyu Guan and Huan Sun. 'Riker: Mining rich keyword representations for interpretable product question answering'. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1389–1398.

[130] Wayne Xin Zhao et al. 'RecBole 2.0: Towards a More Up-to-Date Recommendation Library'. In: *CIKM*. ACM, 2022, pp. 4722–4726.

[131] Yu Zhao et al. 'Steering knowledge selection behaviours in llms via sae-based representation engineering'. In: *arXiv preprint arXiv:2410.15999* (2024).

[132] Jinfeng Zhong and Elsa Negre. 'Context-aware feature attribution through argumentation'. In: *arXiv preprint arXiv:2310.16157* (2023).

[133] Ziwei Zhu et al. 'Popularity-opportunity bias in collaborative filtering'. In: *Proceedings of the 14th ACM international conference on web search and data mining*. 2021, pp. 85–93.

A

Steering impact on item score

We demonstrate how changes in individual SAE neurons can be attributed directly to changes in item scores. We do *not* compute these quantities in our experiments. Instead, we include them as an interpretive lens that explains why particular items moved up or down and which neurons were most responsible.

In collaborative filtering, the relevance of item i for a user is typically computed as a dot product between the user embedding and the item embedding. After steering, the score for item i is:

$$s_i \;=\; v_i^\top \hat{x}', \tag{A.1}$$

where $v_i \in \mathbb{R}^d$ is the item embedding and $\hat{x} \in \mathbb{R}^d$ is the steered user embedding produced by the SAE as defined in equation 5.12 *(If cosine similarity is used, replace this with $s_i = \frac{\hat{x}'^\top v_i}{\|\hat{x}'\|\,\|v_i\|}$; the discussion below applies analogously.)*.

Let $W_{\text{dec}} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N]$ denote the columns of the SAE decoder.

Define:

$$\Delta a_{\text{eff}} \;=\; a' - a, \qquad \Delta x \;= \hat{x}' - \hat{x} = W_{\text{dec}}\,\Delta a_{\text{eff}}.$$

where Δa_{eff} is the change in hidden activations and Δx is the change in user embedding after steering. Then, based on equation A.1, the score change for item i is

$$\Delta s_i = v_i^\top \Delta x = v_i^\top \sum_{k=1}^{N} \Delta a_{\text{eff},k}\,\boldsymbol{\phi}_k = \sum_{k=1}^{N} \Delta a_{\text{eff},k}\,\underbrace{\left(v_i^\top \boldsymbol{\phi}_k\right)}_{=:\,v_{i,k}}.$$

Where N is the number of SAE hidden neurons. The scalar $v_{i,k}$ is the projection of the item embedding v_i onto neuron k's direction in the decoder, $\boldsymbol{\phi}_k$:

$$v_{i,k} \;=\; v_i^\top \boldsymbol{\phi}_k.$$

It quantifies geometric alignment between item i and neuron k in the base embedding space: $v_{i,k} > 0$ indicates alignment, $v_{i,k} < 0$ indicates anti-alignment, and $|v_{i,k}|$ gives the strength of that alignment. Intuitively, neuron k's decoder vector $\boldsymbol{\phi}_k$ points in a particular direction in embedding space. If v_i points in a similar direction ($v_{i,k} > 0$), then increasing neuron k's activation will increase the score of item i in proportion to $v_{i,k}$. Conversely, $v_{i,k} < 0$ means that raising neuron k's activation will decrease the item's score, while $|v_{i,k}| \approx 0$ means neuron k has little effect on that item.

We define the *per-neuron attribution* to item i as

$$A_{i,k} \;\triangleq\; \Delta a_{\text{eff},k}\, v_{i,k}, \qquad \Delta s_i \;=\; \sum_{k=1}^{N} A_{i,k}.$$

Here $A_{i,k}$ is the signed contribution of neuron k to the score change of item i: $A_{i,k} > 0$ means the steering increased the score via neuron k, $A_{i,k} < 0$ means it decreased it, and $|A_{i,k}|$ is the magnitude of that effect. These attributions explain how steering altered rankings.

For practitioners and system debuggers, $A_{i,k}$ enables *monitoring and auditing*: one can see which SAE neurons (and by how much) caused specific items to move up or down, spot neurons that systematically drive head/tail exposure changes, and verify that the intended ones are actually responsible for observed ranking shifts. For a user-facing explanation, the same attribution can be summarized for a displayed set without revealing model internals. For example, if an item that was previously below the cut enters the top-K after steering, we can compute the most responsible neuron

$$k_i^\star \;=\; \arg\max_k \left|A_{i,k}\right|$$

and communicate: *"This item appeared because steering increased neuron $k_i^\star$, which in our analysis tends to favor niche recommendations."*. This provides a concrete, additive explanation tied to the model's neurons, improving transparency.