UNIVERSITAT DE BARCELONA

Bachelor's Thesis

DOUBLE DEGREE IN MATHEMATICS AND COMPUTER ENGINEERING

Faculty of Mathematics and Computer Science
University of Barcelona

Post-hoc explanations for sequential recommendation systems

Author: Jingjing Ye

Director:	**Dr. Maria Salamó LLorente**
Co-director:	**Alejandro Ariza Casabona**
Conducted at:	**Department of Mathematics and Computer Science**

Barcelona, **10 de juny de 2025**

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Maria, and my co-supervisor, Alejandro, for their invaluable guidance and support. This work would not have been possible without their insight, encouragement, and dedication throughout every stage of the process—from the earliest ideas to the final version of this project.

I am also sincerely thankful to all the professors who have accompanied me throughout my academic journey. Each of them has left a mark on me, contributing to my intellectual and personal growth and helping shape the person I am today.

Lastly, I would like to thank my family and friends for their unwavering encouragement and emotional support. A special word of thanks goes to my parents, whose love and belief in me have made this entire journey possible.

Abstract

Nowadays, whether we are aware of it or not, recommender systems surround us everywhere: in the posts we see on social media, TikTok and YouTube videos, or Amazon products. These models are increasingly accurate but at the same time much more complex and difficult to interpret, functioning as "black boxes." As curious users, we might wonder: why did it recommend this and not something else? What was its recommendation criterion? However, these systems rarely provide understandable answers like "I recommended this product because you previously interacted with similar products" or "this song appears because you listened to artists from the same genre last week."

The main goal of this work is to develop and evaluate post-hoc explanation methods that generate interpretable explanations for sequential recommender systems, of the type "I recommend X because you previously interacted with Y and Z." We focus on transformer-based models, specifically SASRec and BERT4Rec, analyzing how to identify and present the most influential past interactions for each recommendation. Through rigorous mathematical analysis and empirical evaluation using counterfactual metrics, we demonstrate how different attribution techniques can generate causal explanations that connect current recommendations with user behavior history. The analysis conducted in this thesis demonstrates that it is possible to generate effective causal explanations and thus make recommender systems more transparent and understandable, allowing users to comprehend the reasoning behind each suggestion.

Resum

Avui en dia, siguem o no conscients d'això, els sistemes de recomanació ens envolten per tot arreu: en les publicacions que veiem a les xarxes socials, els vídeos de TikTok i YouTube, o els productes d'Amazon. Aquests models són cada vegada més precisos però alhora també molt més complexos i difícils d'interpretar, funcionant com a "caixes negres". Com a usuaris curiosos ens podem preguntar: per què m'ha recomanat això i no una altra cosa? Quin ha estat el seu criteri de recomanació? No obstant això, aquests sistemes rarament proporcionen respostes comprensibles del tipus "t'he recomanat aquest producte perquè anteriorment vas interactuar amb productes similars" o "aquesta cançó apareix perquè vas escoltar artistes del mateix gènere la setmana passada".

L'objectiu principal d'aquest treball és desenvolupar i avaluar mètodes d'explicació post-hoc que generin explicacions interpretables per a sistemes de recomanació seqüencials, del tipus "et recomano X perquè prèviament vas interactuar amb Y i Z". Ens centrem en models basats en transformers, específicament SASRec i BERT4Rec, analitzant com identificar i presentar les interaccions passades més influents per a cada recomanació. A través d'una anàlisi matemàtica rigorosa i una avaluació empírica utilitzant mètriques contrafactuals, demostrem com diferents tècniques d'atribució poden generar explicacions causals que connecten les recomanacions actuals amb l'historial de comportament de l'usuari. L'anàlisi realitzada en aquest TFG posa de manifest que és possible generar explicacions causals efectives i fer així els sistemes de recomanació més transparents i comprensibles, permetent als usuaris entendre el raonament darrere de cada suggeriment.

Resumen

Hoy en día, seamos o no conscientes de ello, los sistemas de recomendación nos rodean por todas partes: en los posts que vemos en redes sociales, los vídeos de TikTok y YouTube, o los productos de Amazon. Estos modelos son cada vez más precisos pero al mismo tiempo también mucho más complejos y difíciles de interpretar, funcionando como "cajas negras". Como usuarios curiosos nos podemos preguntar: ¿por qué me ha recomendado esto y no otro? ¿cuál ha sido su criterio de recomendación? Sin embargo, estos sistemas raramente proporcionan respuestas comprensibles del tipo "te he recomendado este producto porque anteriormente interactuaste con productos similares" o "esta canción aparece porque escuchaste artistas del mismo género la semana pasada".

El objetivo principal de este trabajo es desarrollar y evaluar métodos de explicación post-hoc que generen explicaciones interpretables para sistemas de recomendación secuenciales, del tipo "te recomiendo X porque previamente interactuaste con Y y Z". Nos enfocamos en modelos basados en transformers, específicamente SASRec y BERT4Rec, analizando cómo identificar y presentar las interacciones pasadas más influyentes en cada recomendación. A través de un análisis matemático riguroso y una evaluación empírica utilizando métricas contrafactuales, demostramos cómo diferentes técnicas de atribución pueden generar explicaciones causales que conectan las recomendaciones actuales con el historial de comportamiento del usuario. El análisis realizado en este TFG pone de manifiesto que es posible generar explicaciones causales efectivas y hacer así los sistemas de recomendación más transparentes y comprensibles, permitiendo a los usuarios entender el razonamiento detrás de cada sugerencia.

Contents

List of Figures

List of Tables

Chapter 1

This chapter provides a comprehensive introduction to the research presented in this thesis. It begins with the motivation behind the project, emphasising the ongoing challenge of balancing model accuracy with interpretability in modern AI. Next, we define the specific research objectives, focusing on adapting and evaluating post-hoc explanation methods for transformer-based sequential recommender systems. Finally, we describe the structure of the thesis and provide an overview of the technical framework and timeline that guided the development of this work.

1.1 Motivation and Problem Statement

Recommender systems have become integral to improving user experiences across digital platforms [1, 2]. Among these, sequential recommender systems stand out for their ability to capture the temporal dynamics of user interactions, enabling more accurate and context-aware predictions [3, 4]. By leveraging advanced neural architectures such as Transformers, models like SASRec and BERT4Rec can learn complex temporal dependencies in user behavior to predict future preferences with high accuracy [3, 5].

However, the same complexity that makes these models powerful also makes them difficult to interpret. Their multi-layered attention mechanisms and deep sequential architectures make it challenging to understand the reasoning behind individual recommendations. This "black-box" nature becomes particularly problematic as AI systems increasingly influence decisions in sensitive areas such as healthcare, finance, and digital media [6, 7].

The demand for interpretability in AI systems is not merely technical but also societal. Regulatory frameworks such as the European Union's General Data Protection Regulation (GDPR) emphasize the "right to explanation," granting individuals the ability to understand decisions made by algorithmic models and affect them [8]. Furthermore, a growing number of research reflect that explainability is also essential for building trust and ensuring responsible use [9, 10].

Explainable AI (XAI) has emerged as a response to these challenges, offering tools and techniques to make model behavior more transparent. Among these, post-hoc methods such as LIME [11], SHAP [12], and Integrated Gradients [13] estimate feature importance after model training, offering insight into individual predictions without altering the

model's internals [14]. Counterfactual explanation methods provide complementary insights by showing how minimal changes to the input could lead to different outcomes [15].

While these methods have been explored in general machine learning, their adaptation and evaluation for sequential recommenders, especially transformer-based architectures, remain understudied. Therefore, the motivation of this thesis is to study and analyze various post-hoc explanation methods applied to sequential recommenders within this context.

1.2 Objectives

This thesis proposes a framework for adapting and evaluating post-hoc explanation methods in the context of sequential recommendation. It explores how individual recommendations can be explained based on users' historical interactions, with the goal of improving transparency and accountability while preserving predictive performance.

To achieve this, we define the following specific objectives:

1. Understand the key components and formalize the mathematical foundations of the chosen sequential recommendation models (SASRec and BERT4Rec)

2. Implement and adapt post-hoc attribution-based explanation methods including Integrated Gradients, Gradient SHAP, and LIME for sequential recommendation tasks

3. Adapt the LXR counterfactual evaluation framework for assessing explanation quality in sequential recommendation models

4. Compare and analyze the effectiveness of different attribution-based explanation techniques across various sequential recommendation scenarios and datasets

1.3 Project Organization

This thesis is organized into six chapters, each aligned with the research objectives and methodological development:

- **Chapter 1 – Introduction**: Presents the motivation, research scope, and objectives. It highlights the importance of explainability in sequential recommender systems and introduces the focus on post-hoc attribution-based methods.

- **Chapter 2 – State of the Art**: Reviews the foundational literature on sequential recommendation models and explainability techniques. It introduces a taxonomy of explanation methods and positions this work within current research trends.

- **Chapter 3 – Methodology**: Describes the theoretical foundations of the selected post-hoc attribution methods (Integrated Gradients, Gradient SHAP, LIME, and Jaccard) and their adaptation to sequential recommendation models (SASRec and BERT4Rec).

- **Chapter 4 – Implementation**: Describes the datasets used, libraries employed, and specific adaptations made to integrate attribution methods with sequential recommendation models. It also details the implementation approach, code architecture, and technical specifications, as well as the LXR counterfactual framework approach.
- **Chapter 5 – Results and Discussion**: Reports and analyzes experimental results across datasets and methods, including quantitative evaluation metrics, qualitative analysis of explanations, and computational performance considerations. It provides a comparative assessment of the different attribution techniques.
- **Chapter 6 – Conclusions and Future Work**: Summarizes the key contributions and findings of the research. It also outlines directions for future work, such as aspect-based or context-aware dynamics (taking into account additional features, not only the historical sequence) explainability.
- **Appendices**: Includes supplementary materials and links to the implementation repository.

1.3.1 Project Schedule

The study was carried out in the following phases over a period of four months, as shown in Figure 1.1:

- **Phase 1 (Months 1–2)**: Literature review and problem definition
- **Phase 2 (Months 3–4)**: Implementation of base recommendation models, adaptation and integration of explanation methods
- **Phase 3 (Month 4)**: Experimental evaluation and analysis and Thesis writing and final revisions

1.3.2 Technical Framework and Implementation

The codebase is structured into modular components for data processing, model training, explanation generation, and evaluation. Sequential recommendation models were implemented using the RecBole[1] library, while post-hoc attribution methods were adapted from Captum[2]. All experiments are reproducible, with fixed random seeds and detailed configurations. The full implementation will be made publicly available upon thesis submission.

[1] `https://github.com/RUCAIBox/RecBole`
[2] `https://github.com/pytorch/captum`

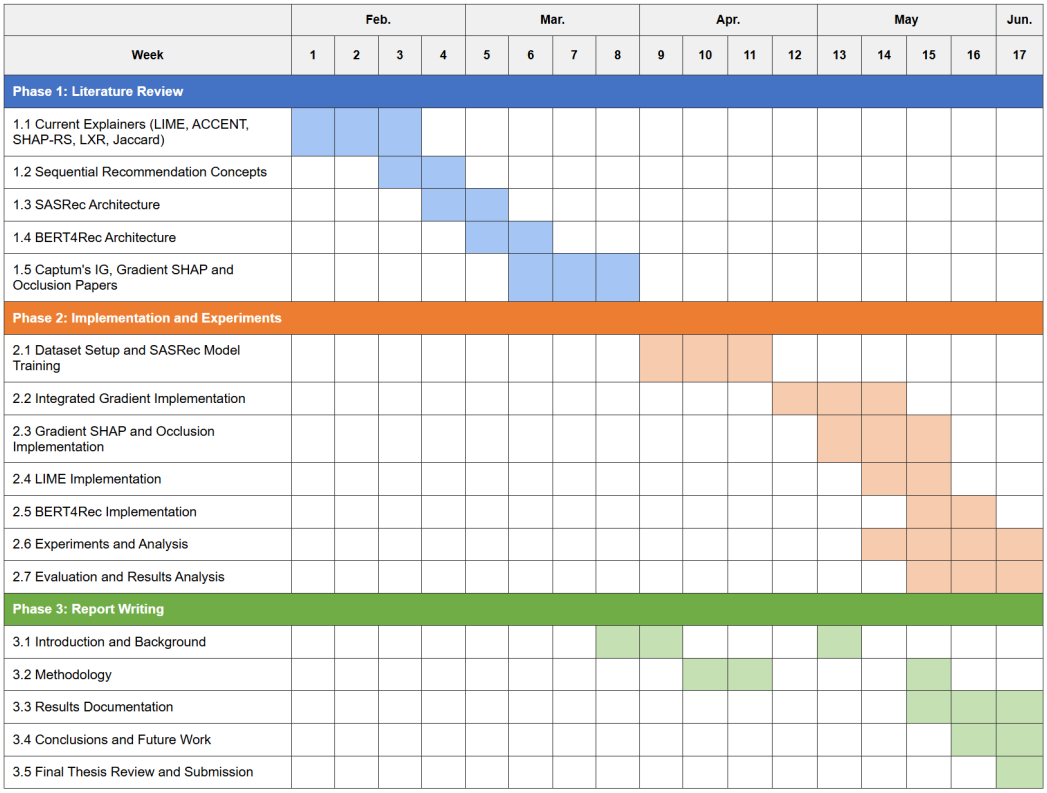

Figure 1.1: Gantt Diagram. Timeline of the main tasks required to complete this thesis.

Chapter 2

State of the Art

This chapter presents an overview of the current state of the art in sequential recommender systems and post-hoc explainability techniques. We begin by reviewing the foundations of sequential recommendation, where the goal is to predict a user's next interaction based on their past behaviour. In particular, we focus on two influential transformer-based models, SASRec [3] and BERT4Rec [5], which leverage self-attention mechanisms to capture complex sequential patterns in user behavior. From there, we turn to explainability methods, looking at model-agnostic approaches like LIME and SHAP. These methods aim to make recommendations more transparent by helping us understand why a particular item was suggested.

2.1 From Traditional to Sequential Recommendation

2.1.1 Foundations of Recommendation Systems

Recommender systems pursue the goal of predicting which items users will find valuable from vast catalogs. A classical approach is to come up with a utility function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{R}$ that maps user-item pairs to preference scores, where $\mathcal{U}$ represents users and $\mathcal{I}$ represents items.

Among existing approaches, collaborative filtering stands out for its ability to predict user preferences by identifying behavioral patterns, either by grouping users with similar tastes or items that appeal to the same audiences [16]. Examples of these are Matrix factorization techniques, which have proven especially effective in decomposing user-item interactions into latent factor representations [17].

While these traditional techniques have served for personalizing user experiences, they treat user preferences as static snapshots. In the real world, user interests change over time and depend significantly on recent behavior, particularly in dynamic applications like streaming platforms, e-commerce, and news feeds. This has led to the development of sequential recommendation models, which predict users' next actions by modeling the order and context of past interactions.

2.2 Sequential Recommendation

Sequential recommendation explicitly addresses the temporal dimension ignored by traditional approaches. This enables new paradigms such as trend detection, context adaptation, and repetitive behavior modeling, among others.

2.2.1 Problem Formulation

Given a user $u \in \mathcal{U}$, we define their chronologically ordered interaction sequence as:

$$\mathcal{S}_u = (s_1^{(u)}, s_2^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)}) \tag{2.1}$$

where $s_i^{(u)} \in \mathcal{I}$ represents the item interacted with at timestamp t, and $\mathcal{I}$ denotes the universal item set.

The sequential recommendation task aims to model the conditional probability distribution:

$$P(s_{t+1}^{(u)} = s \mid s_1^{(u)}, s_2^{(u)}, \ldots, s_t^{(u)}; \theta) \tag{2.2}$$

where θ represents the model parameters and $s \in \mathcal{I}$ is a candidate item for the next interaction.

2.2.2 Evolution of Sequential Models

Markov Chain-Based Approaches

Early sequential recommendation systems employed Markov Chain (MC) models that approximate the conditional probability as:

$$P(s_{t+1}^{(u)} \mid s_1^{(u)}, \ldots, s_t^{(u)}) \approx P(s_{t+1}^{(u)} \mid s_{t-L+1}^{(u)}, \ldots, s_t^{(u)}) \tag{2.3}$$

where L denotes the order of the Markov chain.

This approach fundamentally assumes that future user behaviors depend basically on a fixed window of recent interactions. However, it struggles with capturing long-term preferences, handling complex behavioral patterns, and cold-start scenarios.

Recurrent Neural Networks

The application of Recurrent Neural Networks (RNNs) to sequential recommendation was first introduced in [4], which proposed GRU4Rec for session-based recommendations. Their work demonstrated that RNNs could encode entire interaction histories into fixed-length representations, addressing some limitations of Markov approaches.

However, RNN-based approaches also face some significant challenges in practice. One of the most significant is the vanishing gradient problem; essentially, as sequences get longer, the model struggles to remember what happened at the beginning. This limitation becomes particularly problematic when dealing with users who have extensive interaction histories over months or years.

Another drawback is that the sequential nature of RNNs introduces computational inefficiencies, as states must be computed step by step, making parallelization during training and inference really costly.

To address these challenges, recent research has focused on architectures that allow for greater parallelism and improved long-range dependency modeling. In particular, following the success of Transformer [18] models in natural language processing, self-attention mechanisms have also been adapted for sequential recommendation tasks. [3] proposed SASRec (Self-Attentive Sequential Recommendation), which applies self-attention mechanisms to capture dependencies between items at different positions in a sequence without relying on recurrence. Later, building on the success of BERT, [5] proposed BERT4Rec, which applies a bidirectional Transformer architecture and masked item prediction to model sequences from both past and future contexts. This approach enables richer representation learning and has achieved state-of-the-art performance on various benchmark datasets.

2.2.3 Transformer Architecture for Sequential Recommendation

Before examining SASRec [3] and BERT4Rec [5] in detail, we first introduce the core Transformer components [18] that form the foundation of both models.

Core Architectural Components

The Transformer architecture consists of several key components as illustrated in Figure 2.1.

Embedding Layer Given an interaction sequence $\mathcal{S}_u = (s_1^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)})$, the model processes the most recent n interactions, padded when necessary as follows:

$$s = \begin{cases} (s_{|\mathcal{S}_u|-n+1}^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)}) & \text{if } |\mathcal{S}_u| > n \\ (\text{pad}, \ldots, \text{pad}, s_1^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)}) & \text{if } |\mathcal{S}_u| \leq n \end{cases} \tag{2.4}$$

The embedding layer transforms these input sequences into dense vectors. It consists of three main components:

Item Embedding Items are represented through latent vectors in $\mathbb{R}^d$, organized in an embedding matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{I}| \times d}$, where each row $\mathbf{M}_i$ corresponds to the latent vector of item i.

Positional Embedding To preserve sequential ordering information, the model introduces a positional embedding $\mathbf{P} \in \mathbb{R}^{n \times d}$, where each vector $\mathbf{P}_t$ encodes the positional information t.

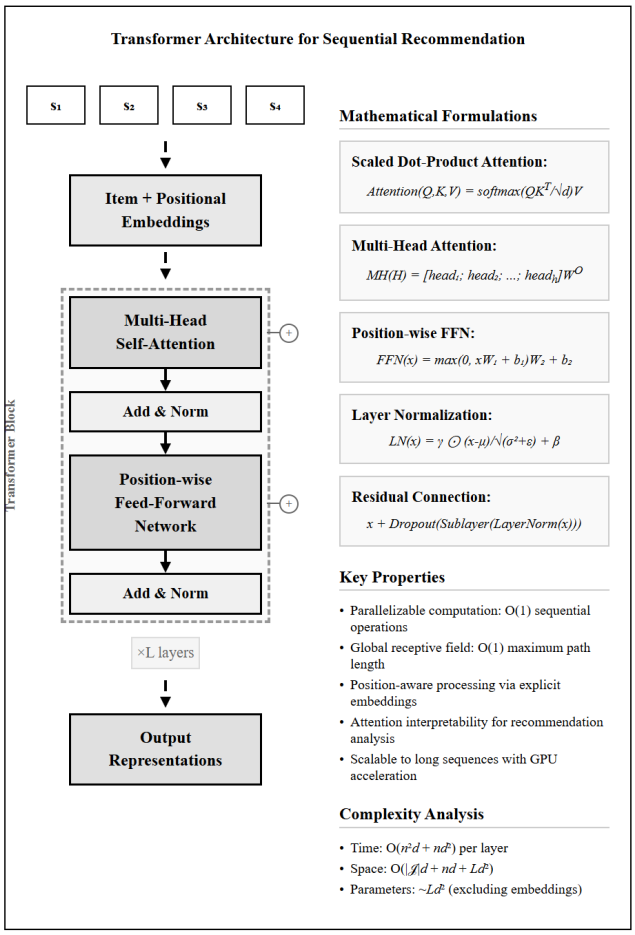

Figure 2.1: Transformer Architecture for Sequential Recommendation. The diagram shows the core components including multi-head self-attention, position-wise feed-forward networks, and residual connections with layer normalization. The architecture can be stacked L times to form deeper models.

Combined Representation The input embedding matrix combines item and positional information:

$$\mathbf{E} = \begin{bmatrix} \mathbf{M}_{s_1^{(u)}} + \mathbf{P}_1 \\ \vdots \\ \mathbf{M}_{s_n^{(u)}} + \mathbf{P}_n \end{bmatrix} \in \mathbb{R}^{n \times d} \tag{2.5}$$

Dropout [19] is applied to mitigate overfitting: $\mathbf{E}' = \text{Dropout}(\mathbf{E})$. Dropout randomly sets a fraction of the input elements to zero during training, acting as a regularization technique that prevents the model from overly relying on specific features.

Self-Attention Mechanism The self-attention layer captures dependencies between items regardless of their positional distance in the sequence.

Scaled Dot-Product Attention The general attention mechanism computes similarity scores:

$$e_{ij} = \frac{\mathbf{Q}_i \mathbf{K}_j^T}{\sqrt{d}} \tag{2.6}$$

Attention weights are computed as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})} \tag{2.7}$$

The attention output is:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_i = \sum_j \alpha_{ij} \mathbf{V}_j \tag{2.8}$$

The scaling factor $\frac{1}{\sqrt{d}}$ prevents extremely large values of the inner product when the dimensionality is high.

Query, Key, Value Projections The matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are linear projections of the input matrix:

$$\mathbf{Q} = \mathbf{E}'\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{E}'\mathbf{W}^K, \quad \mathbf{V} = \mathbf{E}'\mathbf{W}^V \tag{2.9}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$ are learnable parameter matrices. These projections enable the model to learn asymmetric interactions and provide flexibility in representation learning.

Multi-Head Attention Let $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d}$ denote the hidden representation matrix at layer l, where $\mathbf{h}_i^{(l)}$ represents the hidden embedding of token i at layer l. Multi-head attention allows the model to jointly attend to information from different representation subspaces:

$$\text{MH}(\mathbf{H}^{(l)}) = [\text{head}_1; \text{head}_2; \ldots; \text{head}_h] W^O \tag{2.10}$$

where each attention head is computed as:

$$\text{head}_i = \text{Attention}(\mathbf{H}^{(l)} W_i^Q, \mathbf{H}^{(l)} W_i^K, \mathbf{H}^{(l)} W_i^V) \tag{2.11}$$

and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d/h}$ are projection matrices for each head, and $W^O \in \mathbb{R}^{d \times d}$ is the output projection matrix.

Position-wise Feed-Forward Network After the self-attention layer, each representation vector is processed by a position-wise feed-forward network to introduce non-linearity:

$$\text{FFN}(\mathbf{x}) = \text{Activation}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{2.12}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times 4d}$ and $\mathbf{W}_2 \in \mathbb{R}^{4d \times d}$ are weight matrices, and $\mathbf{b}_1, \mathbf{b}_2$ are bias terms. Common activation functions include ReLU and GELU.

Normalization and Residual Connections To stabilize training and facilitate gradient propagation, residual connections and layer normalizations are employed:

$$\text{Output} = \mathbf{x} + \text{Dropout}(\text{Sublayer}(\text{LayerNorm}(\mathbf{x}))) \tag{2.13}$$

Layer normalization is defined as:

$$\text{LayerNorm}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \boldsymbol{\beta} \tag{2.14}$$

where μ and σ^2 are the mean and variance of the latent features, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned scaling factors and bias terms, and ϵ is a small constant for numerical stability.

Stacking Transformer Blocks Multiple transformer blocks can be stacked to learn more complex item transitions:

$$\mathbf{H}^{(l)} = \text{TransformerBlock}(\mathbf{H}^{(l-1)}), \quad \forall l \in [1, \ldots, L] \tag{2.15}$$

where each block combines self-attention and feed-forward components with residual connections and layer normalization.

Computational Complexity

The space complexity is $O(|\mathcal{I}|d + nd + Ld^2)$ for L layers, where the dominant terms are item embeddings and layer parameters. The time complexity is $O(Ln^2d + Lnd^2)$, where the $O(Ln^2d)$ term from self-attention is fully parallelizable, enabling efficient GPU acceleration.

2.2.4 SASRec: Causal Self-Attention for Sequential Recommendation

SASRec [3] implements a transformer decoder architecture that captures both long-term preferences and short-term behavior patterns through causal self-attention mechanisms. The key innovation lies in adapting the transformer decoder for sequential recommendation while maintaining temporal causality.

Causal Attention Mechanism

Unlike the general bidirectional attention described above, SASRec enforces causality to prevent future items from influencing current predictions. This is achieved by modifying the attention computation:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_t = \sum_{j \leq t} \alpha_{tj} \mathbf{V}_j \tag{2.16}$$

where attention weights are constrained to past and current positions:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k \leq t} \exp(e_{tk})}, \quad e_{tj} = \frac{\mathbf{Q}_t \mathbf{K}_j^T}{\sqrt{d}} \tag{2.17}$$

The constraints $j \leq t$ and $k \leq t$ ensure that position t can only attend to positions up to and including t, maintaining the autoregressive property essential for next-item prediction.

Architecture Details

SASRec uses a two-layer transformer decoder architecture with the following specifications:

- Single-head attention
- ReLU activation in feed-forward networks
- Shared item embeddings between input and output layers
- Learnable positional embeddings

Training Objective

Prediction Layer Computes relevance scores for candidate items. The relevance score of item i as the next action in the sequence $(s_1, s_2, \ldots, s_t)$ is computed as:

$$r_{i,t} = \mathbf{h}_t^{(L)} \cdot \mathbf{M}_i^T \tag{2.18}$$

- $\mathbf{h}_t^{(L)} \in \mathbb{R}^d$ is the output representation at position t from the final self-attention layer L
- $\mathbf{M}_i \in \mathbb{R}^d$ is the embedding vector for candidate item i
- $r_{i,t} \in \mathbb{R}$ is the relevance score indicating how likely item i should appear next

The relevance scores $r_{i,t} : i \in \mathcal{I}$ enable ranking all items in the vocabulary $\mathcal{I}$ for position $t+1$, facilitating personalized recommendations through top-k selection. Notably, the model uses the same item embedding matrix $\mathbf{M}$ for both input embedding and output prediction, a parameter sharing technique that significantly improves performance while reducing model size.

Optimization

SASRec training is formulated as an optimization problem with parameters: $\Theta = \{\mathbf{M}, \mathbf{P}, \mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2, \gamma, \beta\}$.

Loss Function The model optimizes a binary cross-entropy loss with negative sampling:

$$\mathcal{L}(\Theta) = -\sum_{S^u \in \mathcal{S}} \sum_{t=1}^{n-1} 1[s_{t+1} \neq \text{pad}] \left[\log \sigma(r_{s_{t+1},t}) + \sum_{j \sim P_{\text{neg}}} \log(1 - \sigma(r_{j,t})) \right] \tag{2.19}$$

where:

- $1[\cdot]$ is the indicator function that equals 1 when the condition is true, 0 otherwise
- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function that maps real-valued relevance scores to probabilities in $(0, 1)$

- P_{neg} is the negative sampling distribution that selects items not present at position $t+1$ in the training sequence. Common choices include uniform sampling over $\mathcal{I} \setminus \{s_{t+1}\}$ or popularity-based sampling
- The indicator function ensures that positions where s_{t+1} is a padding token (used to handle variable-length sequences) are excluded from loss computation

The loss function encourages the model to assign high probability $\sigma(r_{s_{t+1},t})$ to the actual next item s_{t+1} while assigning low probabilities $\sigma(r_{j,t})$ to randomly sampled negative items j. This binary classification approach is computationally efficient compared to computing softmax over the entire item vocabulary $|\mathcal{I}|$.

The binary cross-entropy loss can be derived from the negative log-likelihood of a Bernoulli distribution. For a single prediction, the BCE loss is:

$$\mathcal{L}_{BCE}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \tag{2.20}$$

where:

- $y \in \{0, 1\}$ is the true label (1 for the true next item, 0 for all other items)
- $\hat{y} = \sigma(r_{i,t})$ is the predicted probability that item i is the next item

Optimization Algorithm The network is optimized using the Adam optimizer, which adapts learning rates based on moment estimation:

1. Compute gradients: $g_t = \nabla_\theta \mathcal{L}(\theta_{t-1})$
2. Update biased first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
3. Update biased second moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
4. Correct bias in first moment: $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$
5. Correct bias in second moment: $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
6. Update parameters: $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$

where:

- α is the learning rate (typically 0.001)
- $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates (typically 0.9 and 0.999)
- ϵ is a small constant for numerical stability (typically 10^{-8})

Regularization To mitigate overfitting, the model employs:

- **Dropout** with rate p, applied after each sublayer and to the embedding layer
- **L2 regularization** over parameters, weighted by hyperparameter λ
- **Parameter sharing** between embedding and prediction layers

Algorithm Pseudocode The complete training algorithm for SASRec follows the standard transformer-based sequential recommendation framework with self-attention mechanisms, feed-forward networks, and binary cross-entropy loss optimization. The detailed algorithmic implementation can be found in Algorithm 4 in Appendix C.

2.2.5 BERT4Rec: Bidirectional Self-Attention for Sequential Recommendation

BERT4Rec [5] is a transformer encoder-based model that introduces a novel bidirectional approach to sequential recommendation by predicting randomly masked items using both left and right context, overcoming limitations of unidirectional models.

Key Architectural Modifications

Bidirectional Attention Unlike SASRec's causal attention, BERT4Rec uses the full bidirectional attention mechanism without temporal constraints:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{V}_j \tag{2.21}$$

This allows each position to attend to the entire sequence context, enabling richer representations but requiring a different training strategy to avoid information leakage.

Multi-Head Attention Implementation BERT4Rec employs multi-head attention for enhanced representational capacity:

$$\text{MH}(H^l) = [\text{head}_1; \text{head}_2; \ldots; \text{head}_h] W^O \tag{2.22}$$

where each head computes:

$$\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d/h}}\right) V_i \tag{2.23}$$

GELU Activation BERT4Rec uses the Gaussian Error Linear Unit (GELU) activation function instead of ReLU:

$$\text{FFN}(\mathbf{x}) = \text{GELU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{2.24}$$

where $\text{GELU}(x) = x\Phi(x)$ and $\Phi(x)$ is the standard Gaussian cumulative distribution function.

Cloze Training Objective

To enable bidirectional learning without information leakage, BERT4Rec employs a masked language modeling approach:

Masking Strategy During training, randomly mask ρ proportion of items (typically $\rho = 15\%$) with a special [MASK] token. The model predicts the original items using the surrounding bidirectional context.

Training Loss The model optimizes the cross-entropy loss over masked positions:

$$\mathcal{L} = -\sum_{t \in \mathcal{M}} \log P(s_t | \tilde{S}) \tag{2.25}$$

where $\mathcal{M}$ is the set of masked positions and $\tilde{S}$ is the masked sequence.

The output probability distribution is computed as:

$$P(v) = \text{softmax}\left(\text{GELU}(\mathbf{h}_t^{(L)}\mathbf{W}^P + \mathbf{b}^P)\mathbf{E}^T + \mathbf{b}^O\right) \tag{2.26}$$

where $\mathbf{E}$ is the shared item embedding matrix.

Inference Procedure

During inference, BERT4Rec handles the train-test mismatch by appending [MASK] to the sequence end:

$$\mathcal{S}'_u = [s_1^{(u)}, s_2^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)}, [\text{MASK}]] \tag{2.27}$$

The model then predicts the next item probability based on the final hidden representation of the [MASK] token.

Training Enhancements

To better align with the sequential recommendation task, BERT4Rec also creates training samples that mask only the last item in sequences, effectively fine-tuning the model for next-item prediction while maintaining the benefits of bidirectional pretraining.

Comparative Analysis

The fundamental differences between SASRec and BERT4Rec are illustrated in Figure 2.2 and summarized in Table 2.1.

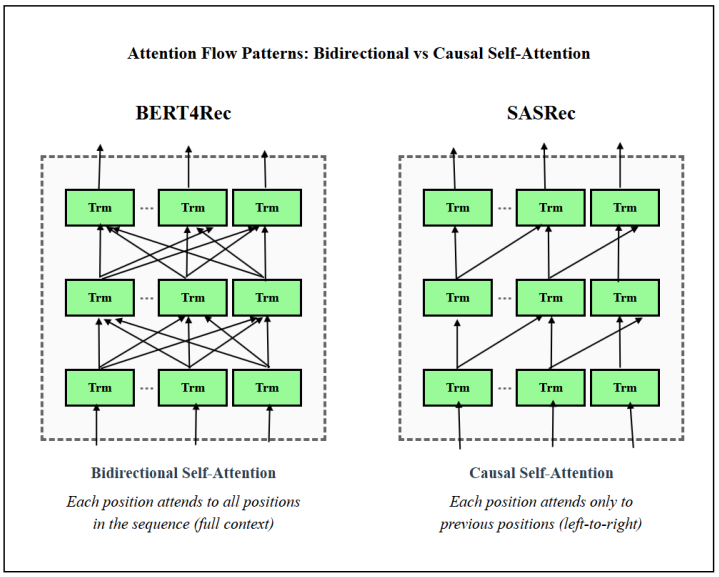

Figure 2.2: Attention Flow Patterns: Bidirectional vs Causal Self-Attention

Table 2.1: SASRec vs BERT4Rec Comparison

Aspect	**SASRec**	**BERT4Rec**
Architecture Type	Transformer Decoder	Transformer Encoder
Attention Direction	Causal (left-to-right)	Bidirectional
Training Objective	Next-item prediction	Masked item prediction
Context Utilization	Previous items only	Full sequence context
Activation Function	ReLU	GELU
Multi-Head Attention	Single head (typically)	Multi-head
Training Samples/Sequence	n samples	$\binom{n}{\rho \cdot n}$ samples
Inference Strategy	Direct prediction	[MASK] appending

2.3 Explainability in Sequential Recommender Systems

The growing sophistication of sequential recommendation models in their pursuit of improved accuracy and precision has created a growing need for interpretability. Explanations in recommendation systems try to provide users and developers with reasons why a particular prediction was generated. We organize explainability approaches for recommender systems into several categories and focus on post-hoc methods applicable to sequential models.

2.3.1 Taxonomy of Explainability Approaches

Explainability methods can be categorized along different dimensions which are not mutually exclusive: **intrinsic vs. post-hoc**, **model-specific vs. model-agnostic**, and **local vs. global explanations** [20]. Intrinsic methods integrate explainability during model design and training, while post-hoc methods apply explanation techniques to already-trained models without modifying the original architecture. Model-specific methods are designed for particular model types and depend on their internal architectures, while model-agnostic methods can be applied to any machine learning model. Local explanations focus on individual predictions, while global explanations explain the whole model behaviour, that is how each feature can influence the results of the model.

Table 2.2 provides an overview of the main attribution methods we explore in this work, organized by their computational approaches.

Category	Method	Source
Gradient-Based	Integrated Gradients	[13]
	GradientSHAP	[12]
Perturbation-Based	LIME	[11]
	Occlusion (Blank-out)	[21]
Counterfactual	ACCENT	[22]
	LXR	[23]
Domain-Specific	Jaccard	[24]

Table 2.2: Overview of attribution methods categorized by their underlying computational approaches for sequential recommendation explanation.

2.3.2 Model-Specific Approaches

Early explainable sequential recommendation methods focused on inherently interpretable architectures. **Attention-based models** emerged as a prominent approach, where [3] proved that attention weights in SASRec could serve as explanations for which previous items influenced current recommendations. [5] extended this with BERT4Rec's bidirectional attention mechanisms.

However, the reliability of attention weights as explanations has become a subject of debate. [25] conducted comprehensive experiments demonstrating that attention distributions fail to provide reliable explanations, showing weak correlations between attention scores and established gradient-based feature importance measures. In response, [26] challenged these conclusions by questioning the experimental methodology and proposing refined evaluation criteria, showing that attention mechanisms retain explanatory value when properly analyzed. This ongoing debate highlights the complexity of interpreting attention weights in transformer-based sequential recommenders.

Prototype-based methods provide another intrinsic approach. [27] introduced ProtoMF, which uses user prototypes to enable explainability in matrix factorization.

2.3.3 Post-Hoc Attribution Methods

Post-hoc methods have gained prominence for their ability to explain complex sequential models without architectural modifications. These approaches provide model-agnostic explanations and can be categorized into gradient-based, perturbation-based, and counterfactual methods.

Gradient-Based Methods

Gradient-based approaches leverage the derivative information of the model function to map prediction values across multiple input dimensions. **Integrated Gradients** [13] represents a prominent method in this category, attributing predictions to input features by integrating gradients along a path connecting a reference point (typically the origin) to the target data point.

Gradient-SHAP [12] provides an alternative approach that approximates SHAP values through gradient computation, offering computational efficiency while maintaining theoretical grounding in game theory. This method combines the theoretical rigor of Shapley values with the computational efficiency of gradient-based attribution.

Perturbation-Based Methods

Perturbation-based methods evaluate model behavior under systematic modifications of input features. This category encompasses both **occlusion-based** and **surrogate-based** approaches:

Occlusion-based methods test the model's response to various input feature removals or modifications. The **Shapley value** [28], originally developed in game theory, represents a foundational approach that identifies unique attributions satisfying predefined axiomatic properties of explanations including efficiency, symmetry, dummy feature, and additivity (these axiomatic properties are defined in detail in Appendix B). [29] extended the application of Shapley values in recommender systems by using SHAP importance rankings to guide counterfactual explanation search rather than direct attribution. Their framework identifies minimal feature changes needed to alter recommendations, supporting both instance-level explanations (why a specific item appears) and list-level explanations (entire recommendation list composition). This approach generates explanations of the form "If condition A were different, then recommendation B would change," providing intuitive counterfactual understanding.

Surrogate-based methods construct simplified local models to approximate the behavior of complex systems. **LIME** (Local Interpretable Model-agnostic Explanations) [11] exemplifies this approach by fitting interpretable models to local regions of the feature space. [30] adapted LIME for recommender systems by treating user interaction sequences as perturbable features, building local linear surrogate models around specific user-item pairs, and focusing on Factorization Machines with MovieLens data.

Counterfactual and Specialized Methods

ACCENT: [22] introduced the first general framework for counterfactual explanations in neural recommenders. ACCENT extends influence functions from single items to item pairs, iteratively identifying user actions that, when removed, would change recommendations.

LXR: [23] proposed a self-supervised framework for learning counterfactual explanations without expensive perturbations. LXR trains an explainer model using novel counterfactual loss terms to identify important user data efficiently.

Jaccard Similarity: This method serves as our baseline due to its simplicity and established use in recommendation systems. It generates explanations by quantifying the similarity between the recommended item and items in the user's interaction history. The approach computes Jaccard similarity coefficients between item pairs based on their co-occurrence patterns across user interactions. Specifically, for items i and j, the Jaccard similarity is defined as:

$$J(i,j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

where U_i and U_j represent the sets of users who have interacted with items i and j, respectively. This measure captures user co-occurrence patterns with values ranging from 0 (no common users) to 1 (identical user sets). The explanation for a recommended item is constructed by identifying the most similar items from the user's history based on these pairwise similarity scores, leveraging the principle that items frequently co-consumed by similar user groups are considered explanatory for recommendation decisions [24].

2.3.4 The Attribution Challenge in Sequential Recommendation

Given a trained sequential recommendation model $f : X \to \mathbb{R}$ that maps user interaction sequences to item relevance scores, the attribution problem seeks to determine which elements in a user's historical sequence $\mathcal{S}_u = (s_1^{(u)}, s_2^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)})$ most strongly influence the prediction of the next item $s_{|\mathcal{S}_u|+1}^{(u)}$.

Formally, we aim to compute an attribution function $A : X \times \mathcal{I} \to \mathbb{R}^{|\mathcal{S}_u|}$ that assigns importance scores to each item in the user's sequence:

$$A(f, \mathcal{S}_u, s_{\text{target}}) = (\alpha_1^{(u)}, \alpha_2^{(u)}, \ldots, \alpha_{|\mathcal{S}_u|}^{(u)}) \tag{2.28}$$

where $\alpha_i^{(u)}$ represents the contribution of item s_i to the prediction of the target item $s_{\text{target}}^{(u)}$.

2.3.5 Theoretical Foundations

These methods generate **counterfactual explanations** by identifying elements in a user's personal data that, if altered, would change the recommended item. Formally, given an input instance x and a model prediction $f(x)$, counterfactual explanations seek to identify the minimal set of feature modifications Δx such that:

$$f(x + \Delta x) \neq f(x) \quad \text{and} \quad \|\Delta x\|_0 \text{ is minimized} \tag{2.29}$$

where $\|\Delta x\|_0$ represents the number of modified features (L0 norm). This counterfactual framework enables users to understand "what-if" scenarios by revealing which aspects of their data most influence the recommendation through the attribution function:

$$\phi_i(x) = \mathbb{E}[f(x) - f(x \setminus \{i\})] \tag{2.30}$$

where $\phi_i(x)$ represents the contribution of feature i to the prediction, and $x \setminus \{i\}$ denotes the instance with feature i removed or set to a baseline value.

2.3.6 Methodological Properties of Attribution Methods

All attribution methods examined in post-hoc explainability share three fundamental characteristics that make them particularly suitable for recommender system applications:

1. **Post-hoc nature**: These methods can be applied to pre-trained models without requiring retraining or architectural modifications, making them applicable to existing production systems.

2. **Interpretability**: These methods provide qualitative understanding of the relationships between input variables and model responses, enabling human comprehension of complex decision-making processes.

3. **Model-agnostic nature**: The techniques can be applied across different model architectures and types without requiring modifications to the underlying recommendation algorithms.

Chapter 3

Methodology

3.1 Research Framework

This chapter presents our systematic approach to explaining sequential recommendations through adapted post-hoc attribution methods. We address three core research questions:

RQ1: How do different attribution methods perform when adapted to transformer-based sequential recommenders?

RQ2: What are the computational and qualitative trade-offs between gradient-based, perturbation-based, and occlusion-based approaches?

RQ3: How do architectural differences between unidirectional (SASRec) and bidirectional (BERT4Rec) models affect explanation quality?

3.1.1 Problem Formulation

We focus basically on attributing individual predictions to their constituent input features through post-hoc analysis. We consider a learned sequential recommendation model $f : X \to \mathbb{R}$, which maps each data point in X to a real-valued score indicating the likelihood that a given item will be the next item with which the user will interact.

The attribution problem can be formally defined as determining a function that, for a given model f, maps each input to the recommender system to a vector of attribution scores. These scores represent the contribution of each feature to the final prediction, capturing both positive and negative influences on the model's decision-making process. Crucially, these explanations are generated post-hoc, meaning they are applied to already-trained models without requiring modifications to the original training process or model architecture.

3.1.2 Model-Specific Formulations

For SASRec, we explain the final position prediction:

$$\text{score}(s_t^{(u)} | \mathcal{S}_u) = \mathbf{h}_n^{(L)} \cdot \mathbf{e}_{s_t^{(u)}} \tag{3.1}$$

where $\mathbf{h}_n^{(L)}$ is the final hidden state and $\mathbf{e}_{s_t}$ is the target item embedding.

For BERT4Rec, we explain masked position predictions:

$$\text{score}(s_t^{(u)} | \mathcal{S}_{\setminus m}) = \mathbf{h}_m^{(L)} \cdot \mathbf{e}_{s_t^{(u)}} \tag{3.2}$$

where position m is masked during prediction.

Figure 3.1 illustrates this interpretable sequential recommendation framework, demonstrating how historical user interactions (s_1–s_5) are processed to generate ranked predictions with corresponding logit scores, while the interpretability analysis module computes attribution scores to explain the model's reasoning.

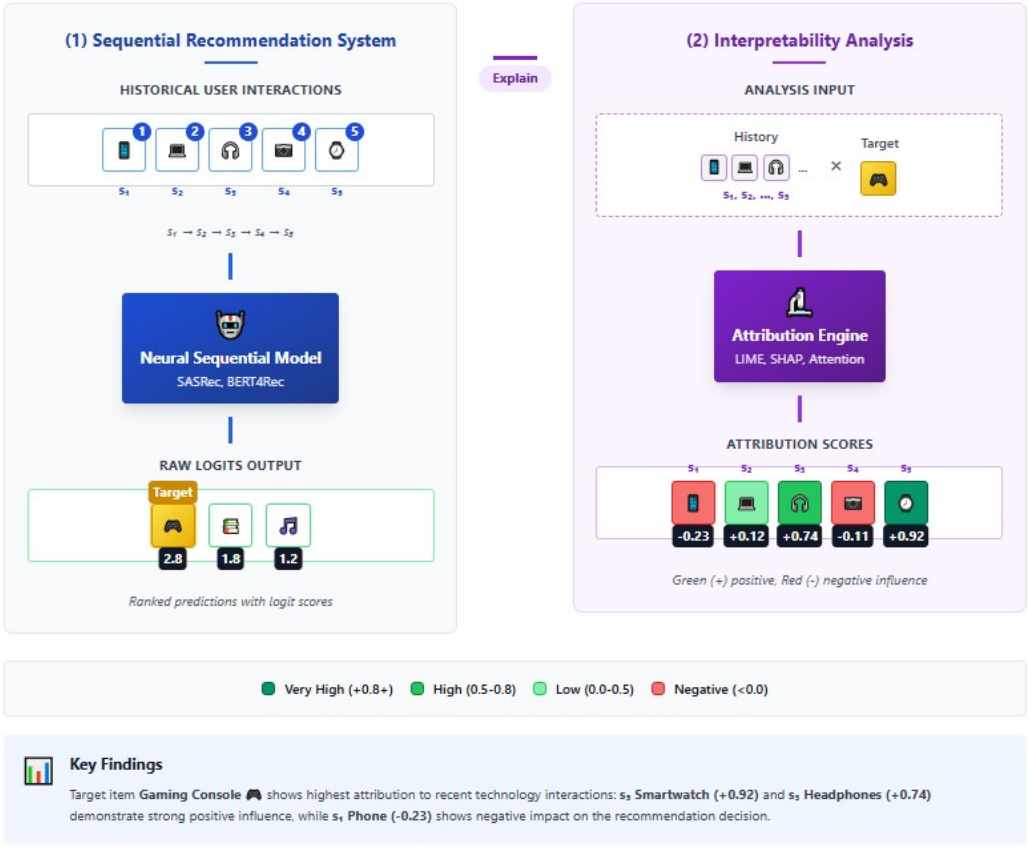

Figure 3.1: Interpretable Sequential Recommendation Framework Architecture. The framework consists of two main components: (1) a **Sequential Recommendation System** that processes historical user interactions (s_1–s_5) to generate ranked predictions with logit scores, and (2) an **Interpretability Analysis** module that explains predictions by computing attribution scores for each interaction. Color coding indicates influence: positive (green) and negative (red). The target item (logit = 2.8) shows strongest attribution to recent technology interactions (s_5: +0.92, s_3: +0.74).

In the following sections, we examine these methods in greater detail, with particular

emphasis on their mathematical foundations and practical implementation considerations. Table 2.2 provides a comprehensive overview of the attribution methods categorized by their underlying computational approaches, facilitating systematic comparison and selection for specific application contexts.

3.1.3 LIME: Local Interpretable Model-Agnostic Explanations

Problem Setup

Given a complex model $F : \mathbb{R}^d \to \mathbb{R}$ and an instance $\mathbf{x} \in \mathbb{R}^d$, LIME seeks to explain the prediction $f(x)$ by learning a locally faithful interpretable model. The framework operates on two distinct feature spaces:

- **Original space**: $x \in \mathbb{R}^d$ representing complex, potentially non-interpretable features
- **Interpretable space**: $x' \in 0, 1^{d'}$ representing simplified, human-understandable features

A bijective transformation function $h : \mathbb{R}^d \to 0, 1^{d'}$ maps between these spaces, where d' represents the number of interpretable components. The core optimization problem balances local fidelity with interpretability:

$$\xi(x) = \arg\min_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.3}$$

where:

- $\mathcal{G}$ is a class of potentially interpretable models (p.e. linear models, decision trees,etc)
- $\mathcal{L}(f, g, \pi_x)$ measures how unfaithful g is in approximating f in the locality defined by π_x
- $\Omega(g)$ measures the complexity of the explanation g
- $\pi_x(z)$ is a proximity measure between instances z and x

Algorithmic Implementation

The LIME explanation process follows a systematic sampling-based approach:

1. **Interpretable Representation**: Transform input x to interpretable space $x' \in \{0, 1\}^{d'}$ via mapping $h : \mathbb{R}^d \to \{0, 1\}^{d'}$
2. **Local Sampling**: Generate perturbed samples $\mathcal{Z}' = \{z'_i\}_{i=1}^N$ around x' by drawing nonzero elements of x' uniformly at random, where the number of such draws is also uniformly sampled.
3. **Model Querying**: For each sample z'_i, compute:

$$z_i = h^{-1}(z'_i) \quad \text{(map back to original space)} \tag{3.4}$$

$$y_i = f(z_i) \quad \text{(obtain model prediction)} \tag{3.5}$$

4. **Weighted Learning**: Train interpretable model g on dataset $\{(z'_i, y_i, \pi_x(z_i))\}_{i=1}^N$ using proximity weights

Sparse Linear Explanations

For computational tractability and interpretability, LIME typically restricts $\mathcal{G}$ to sparse linear models:

$$g(z') = w_0 + \sum_{j=1}^{d'} w_j z'_j \tag{3.6}$$

The locality-aware loss is formulated as locally weighted squared error:

$$\mathcal{L}(f, g, \pi_x) = \sum_{i=1}^{N} \pi_x(z_i)[f(z_i) - g(z'_i)]^2 \tag{3.7}$$

The proximity function is typically an exponential kernel:

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right) \tag{3.8}$$

where $D(\cdot,\cdot)$ is a distance function (e.g., cosine similarity for text, L_2 distance for images) and σ controls the locality radius. For sparsity control, the complexity term enforces a hard constraint:

$$\Omega(g) = \infty \cdot \mathbf{1}[|w|_0 > K] \tag{3.9}$$

where K represents the maximum number of features in the explanation. This constraint is typically handled through K-LASSO: first selecting K features via LASSO regularization path, then learning weights via least squares.

3.1.4 Sliding Window Occlusion

Occlusion is a model-agnostic attribution technique originally proposed by Zeiler and Fergus [21] in the context of convolutional neural networks for image classification. The core idea is to estimate the importance of specific input regions by systematically occluding them and measuring the resulting change in the model's output. Formally, for an input image $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ (e.g., with height H, width W, and C channels) and a model $f : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^K$ that predicts class scores, the attribution of a patch located at (i, j) with size $h \times w$ for a target class k is computed as:

$$A_{i,j} = f(\mathbf{x})_k - f(\mathbf{x}^{(i,j)})_k \tag{3.10}$$

where $\mathbf{x}^{(i,j)}$ is the perturbed input with pixels in the occlusion region $\mathcal{R}_{i,j}$ replaced by a baseline value b_c for each channel c:

$$\mathbf{x}^{(i,j)}_{m,n,c} = \begin{cases} b_c & \text{if } (m,n) \in \mathcal{R}_{i,j} \\ \mathbf{x}_{m,n,c} & \text{otherwise} \end{cases} \tag{3.11}$$

with $\mathcal{R}_{i,j} = \{(m,n) : i \le m < i+h,\ j \le n < j+w\}$.

This method can be adapted to sequential recommendation systems, where the input is a sequence of past user interactions. Let $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ be a sequence of T items from a vocabulary $\mathcal{V}$, and let $f : \mathcal{V}^T \to \mathbb{R}^{|\mathcal{V}|}$ be a model that outputs a score for each

possible next item. The goal is to estimate the contribution of each item x_t in the sequence to the prediction of a target item $x^* \in \mathcal{V}$. The occlusion attribution score at position t is defined as:

$$A_t = f(\mathbf{x})_{x^*} - f(\mathbf{x}^{(t)})_{x^*} \tag{3.12}$$

where $\mathbf{x}^{(t)}$ is the sequence in which x_t has been replaced by a baseline token x_{base} (e.g., a `[MASK]` or padding token):

$$\mathbf{x}_i^{(t)} = \begin{cases} x_{\text{base}} & \text{if } i = t \\ x_i & \text{otherwise} \end{cases} \tag{3.13}$$

This difference quantifies the importance of item x_t to the model's confidence in recommending x^*. Repeating this procedure for each position $t \in \{1, \ldots, T\}$ produces an attribution vector $\mathbf{A} = [A_1, A_2, \ldots, A_T]$, offering interpretability into which parts of the user's history most influenced the recommendation.

3.1.5 Integrated Gradients

Integrated Gradients [13] (IG) is an attribution method that assigns importance scores to input features by examining how they contribute to a model's prediction. Unlike other methods that can behave inconsistently, this method provides theoretical guarantees through mathematical principles. The key insight is that we can understand a neural network's decision by integrating gradients along a path from a baseline input to the actual input, thereby capturing the cumulative effect of each feature on the model's output.

Problem Setup

Definition 3.1 (Attribution)**.** *Let $F : \mathbb{R}^n \to [0, 1]$ be a deep network mapping inputs to predictions. For an input $\mathbf{x} \in \mathbb{R}^n$ and a baseline input $\mathbf{x}' \in \mathbb{R}^n$, we define attribution as a function $A_F(x, x') : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ such that the i-th component $A_F(x, x')_i$ represents the contribution of the i-th feature to the prediction $F(x)$ relative to the baseline x'.*

The baseline x' represents the "absence" of meaningful signal. For instance, in image classification, a black image represents the absence of visual information, while in text analysis, a zero embedding vector represents the absence of semantic content [13].

When applying Integrated Gradients to sequential recommender systems like SASRec [3] and BERT4Rec [5], we attribute the model's predicted score for a candidate item to previous items in the user's history.

For a user interaction sequence $\mathcal{S}_u = (s_1^{(u)}, s_2^{(u)}, \ldots, s_{|\mathcal{S}_u|}^{(u)})$ defined previously, the baseline is the zero embedding vector $[0, 0, ..., 0]$, representing an empty interaction history. The attributions $A_F(x, x')_i$ then quantify how much each historical item $s_i^{(u)}$ contributes to recommending the candidate item.

Axiomatic Foundation

According to [13], there are two fundamental axioms that any attribution method should satisfy:

1. **Sensitivity**: If the network's output $F(x)$ differs from $F(x')$ and x and x' differ only in the i-th feature, then the attribution to feature i must be non-zero.

   Formally, if $x_i \neq x'_i$ and $x_j = x'_j \; \forall j \neq i$, and $F(x) \neq F(x')$, then $A_F(x, x')_i \neq 0$.

2. **Implementation Invariance**: If two networks F_1 and F_2 are functionally equivalent, i.e., $\forall z \in \mathbb{R}^n, F_1(z) = F_2(z)$, then their attributions must be identical: $A_{F_1}(x, x') = A_{F_2}(x, x')$.

[13] shows that many existing attribution methods fail to satisfy at least one of these axioms. For instance, gradients violate Sensitivity(a) due to gradient saturation. While methods like DeepLIFT [31] and Layer-wise Relevance Propagation (LRP) [32] violate Implementation Invariance because they use discrete gradients (finite differences between outputs when inputs change), which do not satisfy the chain rule of differentiation:

$$\frac{f(x_1) - f(x_0)}{g(x_1) - g(x_0)} \neq \frac{f(x_1) - f(x_0)}{h(x_1) - h(x_0)} \cdot \frac{h(x_1) - h(x_0)}{g(x_1) - g(x_0)} \tag{3.14}$$

This means that the same mathematical function $f(g(x))$ can produce different attributions depending on whether it's computed directly or through an intermediate function $h(x)$, even when $f(g(x)) = f(h(x))$ for all inputs.

Example 3.2. [Gradient Saturation] Consider the function $f(x) = 1 - \max(1 - x, 0)$ implemented with ReLU activation. When we compare a baseline $x' = 0$ to an input $x = 2$, the function output changes dramatically from 0 to 1. However, the gradient at $x = 2$ is zero because the function has saturated (become flat) for $x \geq 1$. This means gradient-based attribution would assign zero importance to a feature that clearly matters. In real-world applications like image classification, this leads to misleading explanations where gradients highlight random background pixels rather than the actual object features that drove the model's decision.

Method Definition

To satisfy both axioms, [13] introduces a class of methods called Path Methods, where attributions are calculated by integrating gradients along a path from baseline to input. As illustrated in Figure 3.2, different paths can be chosen between the baseline and input points, each representing a different attribution methodology. The intuition is that instead of looking at gradients at a single point (which can be misleading due to saturation), we examine how gradients change as we move from the baseline to the actual input.

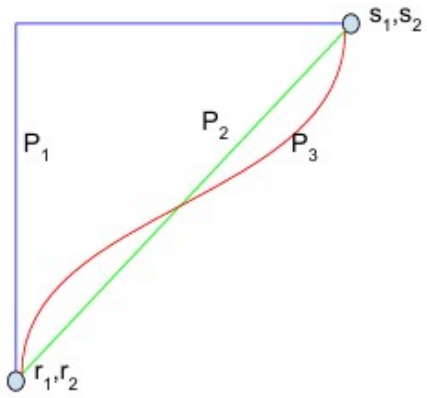

Figure 3.2: Visual representation of three distinct attribution paths connecting a baseline point (r_1, r_2) to an input point (s_1, s_2). Each path represents a different attribution methodology, with the straight-line path P_2 (in green) illustrating the approach employed by integrated gradients. Source: [13].

Definition 3.3 (Path Integrated Gradients)**.** *Let $\gamma : [0,1] \to \mathbb{R}^n$ define a smooth path in $\mathbb{R}^n$ from baseline $\mathbf{x}'$ to input $\mathbf{x}$ such that $\gamma(0) = \mathbf{x}'$ and $\gamma(1) = \mathbf{x}$.*

The general path integrated gradient along the i-th dimension is:

$$PathIntegratedGrads_i^{\gamma}(\mathbf{x}, \mathbf{x}') = \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial \gamma_i(\alpha)} \frac{\partial \gamma_i(\alpha)}{\partial \alpha} d\alpha \tag{3.15}$$

Definition 3.4 (Integrated Gradients)**.** *Integrated Gradients is a specific instance of Path Methods where the path is a straight line from the baseline to the input:*

$$\gamma(\alpha) = x' + \alpha \times (x - x'), \quad \alpha \in [0,1] \tag{3.16}$$

This gives us the formal definition of Integrated Gradients:

$$IntegratedGrads_i(x, x') := (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \tag{3.17}$$

This formulation captures the idea of "accumulating" the local importance of feature x_i as we move along the straight line from baseline to input.

Theoretical Properties

Completeness One of the most appealing properties of Integrated Gradients is that the attributions exactly add up to the difference in the model's output between the input and baseline.

Theorem 3.5 (Completeness [13])**.** *Integrated Gradients satisfies:*

$$\sum_{i=1}^{n} IntegratedGrads_i(\mathbf{x}, \mathbf{x}') = F(\mathbf{x}) - F(\mathbf{x}') \tag{3.18}$$

Proof. The proof relies on the Fundamental Theorem of Calculus (FTC). Define the scalar function that represents the model's output as we move along the path

$$G : [0,1] \longrightarrow \mathbb{R}, \quad G(\alpha) = F\big(\gamma(\alpha)\big) = F\big(\mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}')\big),$$

where $\gamma(\alpha)$ is the straight-line path from baseline to input. By the multivariate chain rule:

$$\frac{dG}{d\alpha}(\alpha) = \nabla F\big(\gamma(\alpha)\big) \cdot \gamma'(\alpha) = \sum_{i=1}^{n} \frac{\partial F}{\partial x_i}\big(\gamma(\alpha)\big)(x_i - x_i').$$

Integrating both sides from $\alpha = 0$ to $\alpha = 1$ and applying the one-dimensional FTC:

$$\begin{aligned} F(\mathbf{x}) - F(\mathbf{x}') = G(1) - G(0) &= \int_0^1 \frac{dG}{d\alpha}(\alpha)\, d\alpha \\ &= \sum_{i=1}^{n} (x_i - x_i') \int_0^1 \frac{\partial F}{\partial x_i}\big(\gamma(\alpha)\big)\, d\alpha \\ &= \sum_{i=1}^{n} \text{IntegratedGrads}_i(\mathbf{x}, \mathbf{x}'). \end{aligned}$$

□

This completeness property is particularly valuable because it provides a form of "conservation" (we can account for exactly where the model's prediction comes from).

Remark 3.6. [Regularity Conditions] For the Completeness theorem to hold, it is *sufficient* that F be differentiable almost everywhere and that the partial derivatives be Lebesgue-integrable along the straight segment joining $\mathbf{x}'$ and $\mathbf{x}$. Specifically:

- **Differentiable almost everywhere**: F must have well-defined partial derivatives at all points except possibly on a set of measure zero (e.g., points where ReLU units switch on/off)
- **Lebesgue-integrable partial derivatives**: The gradients $\frac{\partial F}{\partial x_i}$ must be integrable along the path (ensuring the integral in the IG definition exists)
- **Global C^1 smoothness is *not* required**: The function doesn't need to be continuously differentiable everywhere

Neural networks with ReLU, sigmoid, or similar activations satisfy these minimal conditions.

Uniqueness and Symmetry Among all possible path methods, the selection of the straight-line path requires theoretical justification. This choice is not arbitrary but is uniquely determined by a fundamental desirable property known as symmetry preservation.

Definition 3.7 (Symmetry [13])**.** *Two variables i and j are symmetric with respect to function F if swapping them doesn't change the function:*

$$\forall x, F(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = F(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n) \tag{3.19}$$

Definition 3.8. *An attribution method is symmetry-preserving if symmetric variables with identical values in both input and baseline receive identical attributions.*

Theorem 3.9 (Uniqueness [13])**.** *Integrated Gradients is the unique path method that preserves symmetry, that is, if two features play identical roles in the model and have identical values, they receive identical attributions.*

Proof. We prove by contradiction that only the straight-line path preserves symmetry.

Step 1: Setup for contradiction. Assume there exists a non-straight path $\gamma : [0,1] \to \mathbb{R}^n$ with $\gamma(0) = \mathbf{x}'$ and $\gamma(1) = \mathbf{x}$ that is symmetry-preserving.

Since γ is non-straight, there exist symmetric coordinates $i \neq j$ and some parameter $t_0 \in (0,1)$ such that $\gamma_i(t_0) \neq \gamma_j(t_0)$. Without loss of generality, assume $\gamma_i(t_0) > \gamma_j(t_0)$.

Let (t_1, t_2) be the maximal open interval containing t_0 where $\gamma_i(t) > \gamma_j(t)$ holds. Define:

$$a = \gamma_i(t_1) - \gamma_j(t_1), \quad b = \gamma_i(t_2) - \gamma_j(t_2)$$

where $a, b > 0$ by construction.

Step 2: Construct symmetric test function. Define the symmetric function:

$$f(\mathbf{z}) = \begin{cases} 0, & \text{if } \min(z_i, z_j) \leq a \\ (b-a)^2, & \text{if } \max(z_i, z_j) \geq b \\ (z_i - a)(z_j - a), & \text{otherwise} \end{cases}$$

This function treats coordinates i and j symmetrically and ignores all other coordinates.

Step 3: Choose symmetric baseline and input. Set $\mathbf{x}' = \mathbf{0}$ and $\mathbf{x} = \mathbf{1} = (1, \ldots, 1)$. Since $x_i = x_j = 1$ and $x_i' = x_j' = 0$, any symmetry-preserving attribution method must assign equal attributions to coordinates i and j.

Step 4: Analyze gradients along the path. For $t \notin [t_1, t_2]$: $\nabla f(\gamma(t)) = \mathbf{0}$.

For $t \in (t_1, t_2)$:

$$\frac{\partial f}{\partial z_i} = z_j - a, \quad \frac{\partial f}{\partial z_j} = z_i - a$$

Since $\gamma_i(t) > \gamma_j(t)$ throughout (t_1, t_2), we have:

$$\left.\frac{\partial f}{\partial z_j}\right|_{\gamma(t)} > \left.\frac{\partial f}{\partial z_i}\right|_{\gamma(t)}$$

Step 5: Derive the contradiction. The path attributions are:

$$\text{PathGrads}_k^{\gamma}(\mathbf{1}, \mathbf{0}) = \int_0^1 \frac{\partial f}{\partial z_k}(\gamma(t)) \cdot \dot{\gamma}_k(t)\, dt$$

Since the integrand vanishes outside (t_1, t_2) and is strictly larger for j than for i inside this interval:

$$\text{PathGrads}_j^{\gamma}(\mathbf{1}, \mathbf{0}) > \text{PathGrads}_i^{\gamma}(\mathbf{1}, \mathbf{0})$$

This contradicts the assumption that the path method is symmetry-preserving.

Step 6: Conclude uniqueness. Therefore, no non-straight path can be symmetry-preserving. Any admissible path must move all coordinates proportionally: $\gamma(t) = \mathbf{x}' + \alpha(t)(\mathbf{x} - \mathbf{x}')$.

Re-parametrizing with $\alpha(t) = t$ yields the straight line, completing the proof. □

Additional Desirable Properties The Linearity axiom ensures that attribution methods behave predictably when dealing with combinations of functions. For instance, if we have two models F_1 and F_2 and combine them as $F = aF_1 + bF_2$ (a weighted sum), then the attributions for the combined model should simply be the weighted sum of the individual attributions. This property is important because it preserves the natural mathematical structure and makes the attribution method consistent with how we expect linear combinations to work.

Definition 3.10 (Linearity axiom)**.** *An attribution method A satisfies* Linearity *if, for any two models $F_1, F_2 : \mathbb{R}^n \to \mathbb{R}$ and constants $a, b \in \mathbb{R}$,*

$$A_{aF_1+bF_2}(\mathbf{x}, \mathbf{x}') \;=\; a\, A_{F_1}(\mathbf{x}, \mathbf{x}') \;+\; b\, A_{F_2}(\mathbf{x}, \mathbf{x}').$$

Proposition 3.11 (IG satisfies Linearity [13])**.** *For every $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$ and all $a, b \in \mathbb{R}$,*

$$\mathit{IntegratedGrads}_i^{aF_1+bF_2}(\mathbf{x}, \mathbf{x}') \;=\; a\, \mathit{IntegratedGrads}_i^{F_1}(\mathbf{x}, \mathbf{x}') \;+\; b\, \mathit{IntegratedGrads}_i^{F_2}(\mathbf{x}, \mathbf{x}').$$

Proof. The proof follows from the linearity of derivatives and integrals. Set $F = aF_1 + bF_2$ and use the straight-line path $\gamma(\alpha) = \mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}')$:

$$\begin{aligned}
\text{IntegratedGrads}_i^F(\mathbf{x}, \mathbf{x}') &= (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial x_i}\, d\alpha \\
&= (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial}{\partial x_i}\big(aF_1 + bF_2\big)(\gamma(\alpha))\, d\alpha \\
&= (x_i - x_i') \times \int_{\alpha=0}^{1} \Big(a\, \frac{\partial F_1}{\partial x_i} + b\, \frac{\partial F_2}{\partial x_i}\Big)(\gamma(\alpha))\, d\alpha \\
&= a\,(x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F_1(\gamma(\alpha))}{\partial x_i}\, d\alpha \;+\; b\,(x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F_2(\gamma(\alpha))}{\partial x_i}\, d\alpha \\
&= a\, \text{IntegratedGrads}_i^{F_1}(\mathbf{x}, \mathbf{x}') + b\, \text{IntegratedGrads}_i^{F_2}(\mathbf{x}, \mathbf{x}'),
\end{aligned}$$

where the third line uses linearity of the derivative and the fourth line uses linearity of the integral. Since the argument holds coordinate-wise, IG satisfies the Linearity axiom. □

Numerical Implementation

Practical Computation Since we cannot compute the integral analytically in most cases, we approximate it using a Riemann sum [13]:

$$\text{IntegratedGrads}_i^{\text{approx}}(x, x') := (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m} \tag{3.20}$$

This is straightforward to implement: we simply evaluate the gradient at m equally spaced points along the path from baseline to input, then take their average.

Gauss-Legendre Quadrature In our implementation, we use Gauss-Legendre quadrature as provided by the Captum library [33]. This method approximates the integral using optimally chosen evaluation points and weights:

$$\text{IntegratedGrads}_i^{\text{GL}}(\mathbf{x}, \mathbf{x}') = (x_i - x_i') \sum_{k=1}^{m} w_k \frac{\partial F(\mathbf{x}' + \alpha_k(\mathbf{x} - \mathbf{x}'))}{\partial x_i} \tag{3.21}$$

where $\{\alpha_k, w_k\}_{k=1}^m$ are the Gauss-Legendre nodes and weights. The standard Gauss-Legendre quadrature is defined on the interval $[-1, 1]$, so the nodes ξ_k and weights $\tilde{w}_k$ must be transformed to our integration domain $[0, 1]$:

$$\alpha_k = \frac{1 + \xi_k}{2} \quad \text{(scaling nodes from } [-1, 1] \text{ to } [0, 1]) \tag{3.22}$$

$$w_k = \frac{\tilde{w}_k}{2} \quad \text{(scaling weights accordingly)} \tag{3.23}$$

The nodes ξ_k are the roots of the m-th Legendre polynomial, and the weights $\tilde{w}_k$ are computed as:

$$\tilde{w}_k = \frac{2}{(1 - \xi_k^2)[P_m'(\xi_k)]^2} \tag{3.24}$$

where P_m is the m-th Legendre polynomial and P_m' is its derivative.

Error Analysis The approximation error can be bounded based on the smoothness properties of the gradient function along the integration path.

Theorem 3.12 (Riemann Sum Error Bound for Integrated Gradients)**.** *Consider the straight-line path $\gamma(\alpha) = \mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}')$ and define the partial gradient $g_i(\alpha) := \frac{\partial F}{\partial x_i}(\gamma(\alpha))$.*

If $g_i(\alpha)$ is L-Lipschitz continuous on $[0, 1]$, i.e.,

$$|g_i(\alpha) - g_i(\beta)| \leq L|\alpha - \beta| \quad \textit{for all } \alpha, \beta \in [0, 1],$$

then the m-step Riemann approximation error satisfies:

$$\boxed{\left| \textit{IntegratedGrads}_i(\mathbf{x}, \mathbf{x}') - \textit{IntegratedGrads}_i^{\textit{approx}}(\mathbf{x}, \mathbf{x}') \right| \leq \frac{L|x_i - x_i'| \|\mathbf{x} - \mathbf{x}'\|}{2m}}$$

Proof. The derivative of g_i along the path is:

$$g_i'(\alpha) = \nabla \left(\frac{\partial F}{\partial x_i} \right) (\gamma(\alpha)) \cdot (\mathbf{x} - \mathbf{x}')$$

By the Lipschitz assumption: $|g_i'(\alpha)| \leq L\|\mathbf{x} - \mathbf{x}'\|$.

The standard rectangle-rule error for integrating g_i over $[0, 1]$ with m equal subintervals is at most $\frac{\max_\alpha |g_i'(\alpha)|}{2m}$. Multiplying by the constant factor $(x_i - x_i')$ yields the stated bound. □

Practical Convergence Check In practice, a simple check for sufficient approximation accuracy is to verify that:

$$\left| F(\mathbf{x}) - F(\mathbf{x}') - \sum_{i=1}^{n} \text{IntegratedGrads}_i^{\text{approx}}(\mathbf{x}, \mathbf{x}') \right| \leq \epsilon \tag{3.25}$$

for a small tolerance ϵ (e.g., 5% of $|F(\mathbf{x}) - F(\mathbf{x}')|$).

3.1.6 Gradient SHAP

Gradient SHAP is an efficient approximation of SHAP values specifically designed for differentiable models, combining the theoretical foundation of Shapley values with the computational efficiency of gradient-based methods [12].

Theoretical Foundation in Cooperative Game Theory

The mathematical foundation of Gradient SHAP lies in the Shapley values from cooperative game theory. The basic idea of SHAP values is to compare the model output when a feature is present versus when it is absent in all possible coalitions and aggregate these marginal contributions.

Definition 3.13 (Shapley Value for Feature Attribution)**.** *For a model F, input features $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and a baseline (reference) $\mathbf{x}'$, the Shapley value for feature i is defined as:*

$$\phi_i(F, \mathbf{x}, \mathbf{x}') = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[F_{S \cup \{i\}}(\mathbf{x}) - F_S(\mathbf{x}) \right] \tag{3.26}$$

where N is the set of all features, S is a subset of features excluding i, and $F_S(\mathbf{x})$ represents the model's output when only features in subset S are "present" while all others are set to their baseline values.

Example 3.14. [Shapley Value Calculation for Sequential Recommendation] Consider a user with interaction sequence $\mathbf{x} = [\text{Phone}, \text{Headphones}, \text{Case}]$. We wish to explain the contribution of *Phone* to the next-item prediction.

Let $N = \{\text{Phone}, \text{Headphones}, \text{Case}\}$ and enumerate all subsets $S \subseteq N \setminus \{\text{Phone}\}$: $S_1 = \varnothing$, $S_2 = \{\text{Headphones}\}$, $S_3 = \{\text{Case}\}$, $S_4 = \{\text{Headphones}, \text{Case}\}$.

1. S_1 with weight $\frac{0!(3-0-1)!}{3!} = \frac{1}{3}$: $F(\{\text{Phone}\}) - F(\varnothing)$
2. S_2 with weight $\frac{1}{6}$: $F(\{\text{Phone}, \text{Headphones}\}) - F(\{\text{Headphones}\})$
3. S_3 with weight $\frac{1}{6}$: $F(\{\text{Phone}, \text{Case}\}) - F(\{\text{Case}\})$
4. S_4 with weight $\frac{1}{3}$: $F(\{\text{Phone}, \text{Headphones}, \text{Case}\}) - F(\{\text{Headphones}, \text{Case}\})$

With example outputs $F(\varnothing) = 0.1$, $F(\{\text{Phone}\}) = 0.4$, $F(\{\text{Headphones}\}) = 0.2$, $F(\{\text{Case}\}) = 0.15$, $F(\{\text{Phone}, \text{Headphones}\}) = 0.6$, $F(\{\text{Phone}, \text{Case}\}) = 0.5$, $F(\{\text{Headphones}, \text{Case}\}) = 0.3$, $F(\{\text{Phone}, \text{Headphones}, \text{Case}\}) = 0.8$,

$$\phi_{\text{Phone}} = \frac{1}{3}(0.4 - 0.1) + \frac{1}{6}(0.6 - 0.2) + \frac{1}{6}(0.5 - 0.15) + \frac{1}{3}(0.8 - 0.3) = 0.392.$$

This indicates that Phone contributes approximately 0.39 to the probability of the next item prediction on average across all possible coalitions of other items in the sequence.

This formulation requires 2^n evaluations of the model, which becomes computationally intractable for high-dimensional inputs. Moreover, for most models, it's unclear how to handle missing features directly. [12] addresses this by reformulating the problem using conditional expectation:

$$F_S(\mathbf{x}) = E[F(\mathbf{z})|\mathbf{z}_S = \mathbf{x}_S] \tag{3.27}$$

where $\mathbf{z}$ is a random variable representing the input features, and the expectation is taken over $\mathbf{z}_{N\setminus S}$, the features not in S.

In Captum's [33] implementation, GradientShap approximates SHAP values by computing the expectations of gradients by randomly sampling from the distribution of baselines/references. The method adds white noise (Gaussian noise with zero mean and standard deviation) to each input sample n times, selects a random baseline from the baselines' distribution and a random point along the path between the baseline and the input, and computes the gradient of outputs with respect to those selected random points. The final SHAP values represent the expected values of gradients * (inputs - baselines).

In some sense, Gradient SHAP can be viewed as an approximation of integrated gradients by computing the expectations of gradients for different baselines.

Chapter 4

Implementation

This chapter describes how we implemented and tested our approach. We start by explaining the software setup and computing resources we used, then describe the datasets for our experiments. We use two recommendation models, SASRec [3] and BERT4Rec [5], from the RecBole library [34]. To explain their predictions, we implement several techniques: Integrated Gradients, Gradient SHAP, and LIME from the Captum library [33], Jaccard similarity, and our own implementation of the LXR framework. We explain how we built these explanation methods using the metrics from [23].

4.1 Resources and Development Environment

We developed the implementation using Python 3.9 in a Conda environment to ensure reproducibility. The project was structured as a package called `explainrec` with the following dependencies:

Listing 4.1: Key package dependencies

```
install_requires=[
    "torch>=1.10.0",
    "numpy>=1.17.2,<2.0.0",
    "recbole>=1.0.0",
    "captum>=0.4.0",
    "matplotlib>=3.4.0",
    "scipy>=1.6.0",
    "pandas>=1.3.0",
    "tqdm>=4.62.0",
]
```

For development, we used Visual Studio Code as the primary IDE and Jupyter Notebook for data visualization and results analysis. Initially, we used Google Colab for preliminary experiments, but migrated to the University of Barcelona's CLiC [1] clusters when

[1] `https://clic.ub.edu/`

computational complexity required more stable resources. These clusters are equipped with NVIDIA RTX 4090 GPUs (24GB VRAM) and 64GB RAM, providing the necessary computational power for training sequential recommendation models and computing attribution explanations across multiple datasets. Personal laptops with 16GB RAM were also used throughout the development process for coding and lighter computational tasks.

AI assistants (Claude, ChatGPT, and DeepSeek) were used for grammar and spelling correction during thesis writing and occasional code debugging support.

4.2 Datasets

We have considered three datasets for our evaluation: MovieLens-1M [2] (movies), Yelp 2018 [3] (local businesses), and Amazon Movies and TV 2018 [4] (e-commerce).

For compatibility with the RecBole framework, we used versions of the datasets that were already preprocessed and formatted in RecBole's atomic file structure. These processed datasets were obtained from a shared Google Drive folder[5], which provides ready-to-use files derived from the original sources.

Given that our approach focuses on sequential recommendation systems (SASRec [3] and BERT4Rec [5]), we use only the core sequential interaction features from each dataset: *user ID*, *item ID*, *rating*, and *timestamp*. These models are designed to capture temporal patterns and sequential dependencies in user behavior without relying on additional context-aware features. So while datasets contain rich additional information, we focus on learning from interaction sequences alone. Item titles are kept for visualization purposes.

Table 4.1 shows detailed statistics. ML-1M is used without preprocessing for comparison with existing literature. Yelp and Amazon datasets undergo filtering to keep only users and items with at least 10 interactions, ensuring data quality while maintaining realistic sparsity challenges.

Table 4.1: Dataset Statistics: Original and Filtered Characteristics

Metric	**Datasets**		
	ML-1M	**Yelp 2018**	**Amazon Movies & TV**
Original Dataset Statistics			
Users	6,041	1,326,101	3,826,085
Items	3,884	174,567	182,032
Interactions	1,000,209	5,261,669	8,765,568
Filtered Dataset Statistics			
Users	6,041	77,278	84,116
Items	3,884	45,639	30,882
Interactions	1,000,209	2,103,896	1,890,004
Sparsity (%)	95.74	99.94	99.93
Additional Information			
Preprocessing	None	Min. 10 interactions	Min. 10 interactions

[2] https://grouplens.org/datasets/movielens/1m/

[3] https://github.com/RUCAIBox/RecSysDatasets/tree/master/dataset_info/Yelp

[4] https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews

[5] https://drive.google.com/drive/folders/1so0lckI6N6_niVEYaBu-LIcpOdZf99kj?usp=sharing

MovieLens-1M: A standard dataset from GroupLens Research with high user engagement (165.60 ratings per user) and moderate sparsity (95.74%). Used without changes for comparison with existing literature.

Yelp 2018: Shows local business recommendations from the Yelp Dataset Challenge. Filtering kept 77,278 active users (40.0% of original interactions) focusing on engaged users and established businesses.

Amazon Movies & TV: E-commerce recommendation scenario with 84,116 filtered users keeping 21.6% of original interactions while preserving realistic sparsity challenges.

This selection provides different domains (entertainment, local services, e-commerce), sparsity range (95.74% to >99.9%), different sizes for computational assessment, and comparison capabilities with ML-1M serving as a standard reference.

4.3 Base Recommender Models

For the selected recommendation models SASRec [3] and BERT4Rec [5], we leverage the implementations provided by the RecBole library [34]. RecBole is a unified, comprehensive, and efficient framework developed in PyTorch that supports 94 recommendation algorithms across four major categories.

Since our primary focus is on post-hoc explainability rather than achieving optimal model performance, we utilize the default model configurations without extensive hyperparameter tuning.

4.3.1 Framework Architecture

Our implementation builds upon RecBole's modular architecture through a base class `BaseSequentialRecommender` that encapsulates the core functionality for both SASRec and BERT4Rec models. This design provides several advantages:

- **Unified Interface**: Both models inherit from the same base class, ensuring consistent data processing, training procedures, and evaluation protocols across different architectures.
- **RecBole Integration**: We utilize RecBole's `run_recbole()` function, which includes the complete process of training and testing a model on a specified dataset, providing automated dataset creation, model training, validation, and checkpoint saving.
- **Model Loading and Persistence**: The framework supports model persistence through RecBole's `load_data_and_model()` function, which can reload filtered datasets, split dataloaders, and trained models from saved files, reducing preprocessing time for subsequent experiments.

4.3.2 Model Configuration

Both SASRec and BERT4Rec are configured with the parameters shown in Table 4.2.

Table 4.2: Model Configuration Parameters

Parameter Category	Configuration
Architecture Parameters	
Sequence Length	200 (ML-1M), 50 (Amazon Movies & TV), 30 (Yelp 2018)
Hidden Dimensions	64-dimensional representations
Feed-Forward Dimension	256 (4× hidden size)
Transformer Layers	2 layers
Attention Heads	2 heads per layer
Normalization	Layer normalization (= 1e-12)
Dropout	0.5 (hidden states and attention)
Training Configuration	
Training Epochs	10 epochs
Early Stopping	Based on validation NDCG@10
Training Batch Size	128
Evaluation Batch Size	256
Loss Function	Cross-entropy loss
Optimizer	Adam with learning rate scheduling
Evaluation Protocol	
Ranking Strategy	Full-item ranking
Metrics	NDCG@{5,10,20}, Recall@{5,10,20}, Precision@{5,10,20}, MRR, Hit Rate
Validation Metric	NDCG@10
Data Splitting	Temporal split (80-10-10)

4.3.3 SASRec Implementation

We implement SASRec using RecBole's `SASRec` [6] class with custom wrapper `SASRecRecommender` that extends `BaseSequentialRecommender`. Key aspects include:

- **Configuration Management**: Dataset-specific parameters are handled through `get_model_specific_config()`, setting dropout rates (0.5) and sequence lengths based on dataset characteristics.
- **Data Processing**: Leverages RecBole's built-in data loading and preprocessing pipeline, ensuring consistent temporal ordering and sequence truncation.
- **Training Procedure**: Standard next-item prediction objective with causal attention masking, using Adam optimizer with early stopping based on validation NDCG@10.

[6] `https://recbole.io/docs/recbole/recbole.model.sequential_recommender.sasrec.html`

4.3.4 BERT4Rec Implementation

Similarly, BERT4Rec has been implemented based on RecBole's `BERT4Rec` [7] class with a custom `BERT4RecRecommender` wrapper that extends `BaseSequentialRecommender`. Key aspects include:

- **Masking Strategy**: Dynamic mask ratios are applied per dataset (0.2 for MovieLens, 0.6 for others) following the original paper's recommendations, implemented in the constructor.
- **Sequence Handling**: Extended sequence lengths for MovieLens datasets (200 items) are automatically configured through `build_model_from_dataset_movies()`.
- **Training Configuration**: Larger batch sizes (256) for training stability, with specialized constants for masked item handling (`MASK_ITEM_SEQ`, `POS_ITEMS`, etc.).

4.3.5 Data Processing

User interaction histories are converted into fixed-length sequences with temporal ordering preserved. Sequences shorter than the maximum length are zero-padded, while longer sequences are truncated to keep the most recent interactions. The preprocessing pipeline filters users and items with fewer than 10 interactions to ensure adequate signal for both model training and explanation generation.

4.3.6 Technical Implementation Details

Configuration Management: The base class handles comprehensive parameter configuration through nested dictionaries passed to RecBole's configuration system:

```
config_dict = {
    'model': self.model_name,
    'dataset': dataset_name,
    'n_layers': self.n_layers,
    'n_heads': self.n_heads,
    'hidden_size': self.hidden_size,
    'inner_size': self.hidden_size * 4,
    'max_seq_length': self.max_sequence_len,
    'loss_type': 'CE',
    'train_batch_size': self.batch_size,
    'eval_batch_size': 256,
    'topk': [5, 10, 20],
    'metrics': ['Recall', 'NDCG', 'Precision', 'MRR', 'Hit'],
    'valid_metric': 'NDCG@10'
}
```

Field Mapping and Data Schema: The implementation handles RecBole's field mapping requirements:

[7] `https://recbole.io/docs/recbole/recbole.model.sequential_recommender.bert4rec.html`

- `USER_ID_FIELD`: Maps to *user_id* for user identification
- `ITEM_ID_FIELD`: Maps to *item_id* for item identification
- `TIME_FIELD`: Maps to *timestamp* for temporal ordering
- `load_col`: Specifies which columns to load from dataset files

Training Metrics Capture: A custom callback system captures detailed training metrics through monkey-patching RecBole's trainer:

```
def custom_evaluate(eval_data):
    result = original_evaluate(eval_data)
    epoch = trainer.cur_epoch

    # Store epoch metrics
    self.training_metrics['epochs'].append(epoch)
    for metric_name, value in result.items():
        self.training_metrics['valid_metrics'][metric_name].append(float(value))

    return result
```

Reproducibility Management: Ensures consistent results through multiple seed settings:

- Global seed configuration: `seed=42`
- RecBole reproducibility flag: `reproducibility=True`
- Local rank seed offset for distributed training compatibility

4.3.7 Data Processing Pipeline

Sequence Construction: User interaction histories are converted into fixed-length sequences:

- **Padding Strategy**: Zero-padding for sequences shorter than max_seq_length
- **Truncation Strategy**: Keep the most recent max_seq_length items for longer sequences
- **Temporal Ordering**: Interactions sorted by timestamp to preserve sequential nature

Item and User Filtering:

- **Minimum Interaction Threshold**: Users and items with fewer than specified interactions are filtered
- **Cold Start Handling**: New users/items in test set are handled through zero-shot inference

Field Processing:

```
load_col': {
    'inter': ['user_id', 'item_id', 'rating', 'timestamp'],
    # For MovieLens
    'item': ['item_id', 'movie_title', 'release_year', 'genre'],
    # Optional
    'user': ['user_id', 'age', 'gender', 'occupation', 'zip_code']
}
```

ID Mapping and Conversion: RecBole automatically handles the conversion between external IDs (original dataset identifiers) and internal IDs (consecutive integers starting from 0) used during model training. Special attention must be paid to this mapping when implementing explanation methods, as explanations need to be generated using internal IDs but interpreted and presented using external IDs. The framework provides token-to-ID and ID-to-token conversion utilities to handle this mapping consistently across all explanation techniques.

4.4 Explanation Methods

To interpret and explain the predictions generated by our sequential recommendation models, we employ a comprehensive set of explanation techniques spanning both gradient-based and perturbation-based approaches. These methods enable us to understand which items in a user's interaction history are most influential for specific recommendations, providing insights into the decision-making process of SASRec and BERT4Rec models.

4.4.1 Attribution Methods via Captum

For gradient-based and perturbation-based attribution methods, we leverage the Captum library [33], a unified and generic model interpretability library for PyTorch. Captum provides state-of-the-art algorithms for understanding which features contribute to a model's predictions and offers implementations that can be applied to any differentiable PyTorch model with minimal modification.

LIME Implementation

We implement a custom explainer using Captum's `LimeBase` [8] class, specifically adapted for sequential recommendation that addresses the unique challenges of variable-length user sequences and item embeddings.

Algorithmic Overview Our `SimpleLimeExplainer` extends Captum's framework with specialized component functions for sequential data. The complete attribution process follows Algorithm 1:

[8] `https://captum.ai/api/lime.html`

Algorithm 1 LIME Attribution Process for Sequential Recommendation

Require: User sequence $\mathbf{S} = [s_1, s_2, \ldots, s_n]$, target item t, number of samples N
Ensure: Attribution scores $\mathbf{A} = [a_1, a_2, \ldots, a_n]$ for each sequence position

Step 1: Generate Perturbations
for $k = 1$ to N **do**
  $\mathbf{M}_k \leftarrow$ GenerateBinaryMask($\mathbf{S}$) {Binary mask indicating which items to keep}
  $\mathbf{S}'_k \leftarrow$ ApplyMask($\mathbf{S}$, $\mathbf{M}_k$) {Transform to perturbed sequence}
end for
Step 2: Obtain Model Predictions
for $k = 1$ to N **do**
  $p_k \leftarrow$ Model.Predict($\mathbf{S}'_k$, t) {Target item probability}
  $w_k \leftarrow$ Similarity($\mathbf{S}$, $\mathbf{S}'_k$) {Similarity weight}
end for
Step 3: Fit Interpretable Model
$f(\mathbf{M}) \leftarrow$ FitLinearModel($\{\mathbf{M}_k\}_{k=1}^N$, $\{p_k\}_{k=1}^N$, $\{w_k\}_{k=1}^N$)
Step 4: Extract Attributions
$\mathbf{A} \leftarrow$ GetCoefficients(f) {Linear model coefficients as attributions}
return $\mathbf{A}$

This process generates N perturbed versions of the original sequence, evaluates the model's predictions on each perturbation, and fits a local linear model to approximate the model's behavior in the neighborhood of the original sequence. The coefficients of this linear model serve as attribution scores for each position in the sequence.

Key Technical Adaptations Our implementation addresses three critical challenges for applying LIME to sequential recommendation:

- **Sequential Perturbation Strategy**: Unlike standard LIME implementations that work with fixed-size inputs, we generate binary masks for variable-length sequences while ensuring valid perturbations. Our approach uses Bernoulli sampling with $p = 0.5$ but includes boundary condition handling to prevent degenerate cases where all items are masked or none are masked from active positions.

- **Compact Sequence Transformation**: We transform binary interpretable representations back to compact input sequences by maintaining temporal order of selected items while removing gaps created by masking. This prevents CUDA memory issues and ensures the perturbed sequences are valid inputs for the recommendation models.

- **Similarity-Based Weighting**: We implement locality weighting using a Hamming distance-based approach adapted for sequence data. Rather than comparing exact item values, we compare presence/absence patterns with an exponential kernel:

$$w = \exp\left(-\frac{(1 - \text{similarity})^2}{2\sigma^2}\right)$$

where similarity $= \frac{\sum_i \mathbb{I}[(S_{\text{orig}}[i] \neq 0) = (S_{\text{pert}}[i] \neq 0)]}{|S|}$

- **Interpretable Model Selection**: We support LASSO regression as interpretable model, providing feature selection through L1 regularization:

$$\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} w_i (y_i - x_i^T \beta)^2 + \alpha \|\beta\|_1$$

where w_i represents the similarity weights, y_i are the model predictions, and x_i are the binary feature vectors indicating item presence. The LASSO parameter α controls regularization strength and explanation sparsity.

Model Agnosticism As a post-hoc explanation method, our LIME implementation operates solely on model prediction scores without requiring knowledge of internal architecture details. This ensures seamless compatibility with both SASRec and BERT4Rec:

- **Black Box Treatment**: LIME treats models as black boxes, accessing only their prediction interfaces through the `full_sort_predict` method
- **Architecture Abstraction**: Whether the model uses unidirectional (SASRec) or bidirectional (BERT4Rec) attention, LIME only observes the resulting prediction scores
- **Uniform Interface**: Both models provide identical prediction interfaces, allowing consistent explanation generation regardless of underlying architecture

The complete algorithmic details for each component function are provided in Appendix D.

Gradient-Based Attribution Methods

For gradient-based explanations and other perturbation methods, we implement a unified framework using Captum that supports multiple attribution methods: Integrated Gradients [9], GradientSHAP [10], and Occlusion [11]. Our implementation provides model-agnostic explanations that work with both SASRec and BERT4Rec architectures through specialized forward functions.

Unified Attribution Framework We design a `CaptumExplainer` class that abstracts different attribution methods through a common interface while handling model-specific requirements. Algorithm 4.4.1 shows the configuration process for the three primary attribution methods:

- **Integrated Gradients**: Path integral approach with Gauss-Legendre integration method.
- **GradientSHAP**: Combines gradients with Shapley value sampling using multiple random baselines

[9] `https://captum.ai/api/integrated_gradients.html`
[10] `https://captum.ai/api/gradient_shap.html`
[11] `https://captum.ai/api/occlusion.html`

- **Occlusion**: Systematic perturbation-based attribution through sliding window masking

Algorithm 2 Attribution Method Configuration

Input: Attribution method name, model type
Output: Configured attribution method
$method\_configs \leftarrow \{$
  $'integratedgradients' : \{class : IntegratedGradients, params : \{n\_steps : 25\}\},$
  $'gradientshap' : \{class : GradientShap, params : \{n\_samples : 50\}\},$
  $'occlusion' : \{class : Occlusion, params : \{window\_shape : (1, 1)\}\}$
$\}$
$method\_class \leftarrow method\_configs[method\_name]['class']$
$default\_params \leftarrow method\_configs[method\_name]['params']$
$attribution\_method \leftarrow method\_class(forward\_function)$
return $attribution\_method, default\_params$

Parameter Selection:

- **Integrated Gradients (25 steps):** We chose 25 integration steps based on recommendations from the original paper, which suggests 20-50 steps work well for most applications. This number balances accuracy with computation time. Our monitoring shows good convergence ($\delta < 0.05$) in over 95% of cases, confirming this choice works well.

- **GradientSHAP (50 samples):** We used 50 samples following standard recommendations from SHAP literature, which suggests 50-100 samples provide stable results. This gives us reliable Shapley value estimates while keeping computation manageable.

- **Occlusion (window size 1×1):** We mask one item at a time to understand each item's individual contribution to recommendations. This approach fits our goal of explaining item-level influences in user sequences.

- **Baseline Selection:** We use zero embeddings as baselines for Integrated Gradients and Occlusion, representing "no item" in the embedding space. For GradientSHAP, we use up to 10 random baselines to make results more stable and reduce sensitivity to baseline choice.

- **Convergence Threshold ($\delta < 0.05$):** This 5% threshold follows common practice in attribution methods and proved effective in our experiments for identifying good quality explanations.

- **Time Constraints:** While more extensive parameter testing would be valuable, our choices are well-supported by literature and show consistent performance across different datasets and models.

Model-Specific Forward Functions: To handle the architectural differences between SASRec and BERT4Rec, we implement specialized forward functions that properly propagate gradients through embeddings. Figure 4.1 illustrates the forward function flow comparison, highlighting the critical differences in attention mechanisms and output processing.

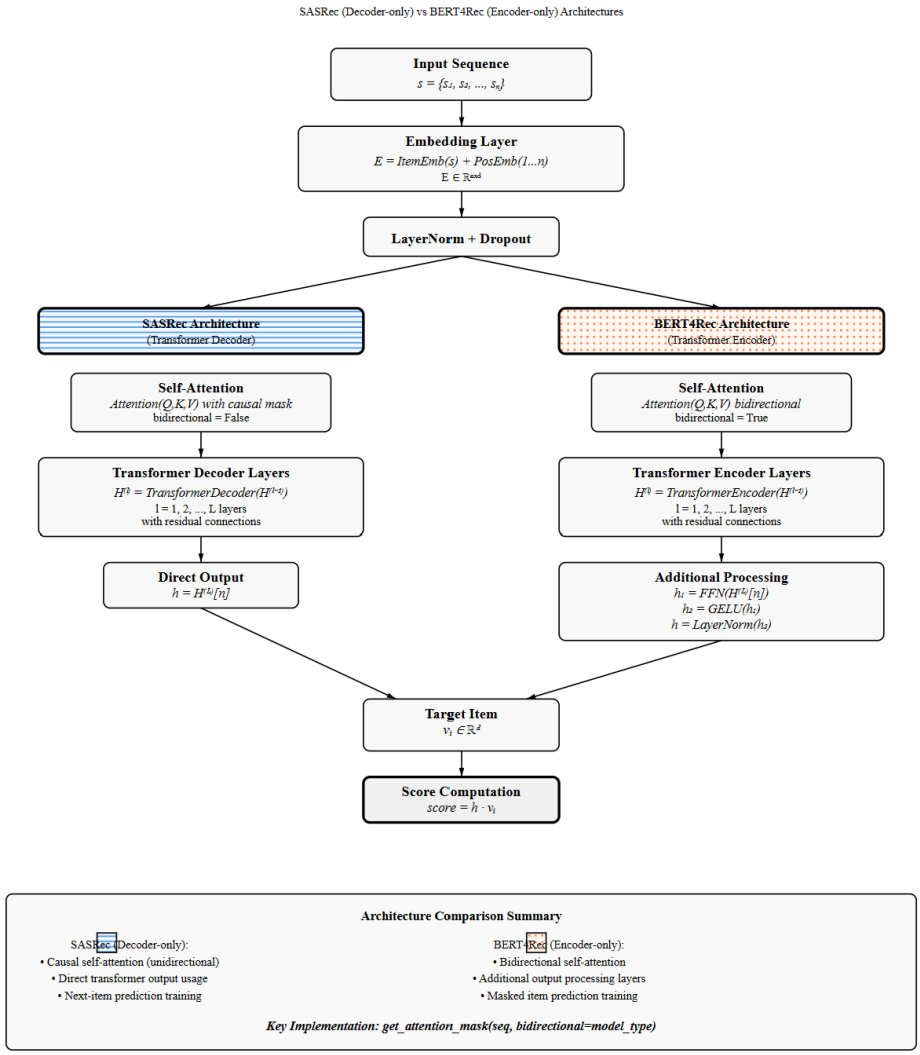

Figure 4.1: Forward function flow comparison between SASRec (decoder-only) and BERT4Rec (encoder-only) architectures. The diagram shows the divergence points where model-specific processing occurs, particularly in attention masking and output layer handling.

The key architectural differences require distinct forward function implementations: SASRec uses causal attention masking (bidirectional=False) and direct transformer output, while BERT4Rec employs bidirectional attention and additional output processing

layers (FFN + GELU + LayerNorm). These forward functions ensure proper gradient propagation through item embeddings for Captum attribution methods. The detailed pseudocode for both architectures is provided in Algorithms 9 and 10 in Appendix E.

Implementation Features

- **Convergence Monitoring**: For Integrated Gradients, We monitor convergence using the completeness axiom where the sum of attributions should equal the difference between the model output and baseline output:

$$\sum_i A_i = f(E) - f(E_0)$$

  The convergence delta $\delta = |f(E) - f(E_0) - \sum_i A_i|$ indicates attribution quality, with $\delta < 0.05$ considered acceptable convergence.

- **Occlusion Configuration**: For occlusion analysis, we systematically mask individual items and measure the impact on predictions. Our implementation uses a sliding window approach where each sequence position is individually evaluated:

$$\mathcal{W}_{i,w} = \{j : i \leq j < i + w\}$$

  where $\mathcal{W}_{i,w}$ defines a window of size w starting at position i. We configure single-token occlusion with `window_shape = (1, embedding_dim)` to isolate individual item contributions.

- **Baseline Selection Strategies**: Our implementation uses adaptive baseline selection based on the attribution method. For Integrated Gradients and Occlusion, we use zero embeddings as baselines representing the absence of items. For GradientSHAP, we sample up to 10 random baselines from the complete item embedding matrix and expand them to match sequence dimensions. This multi-baseline approach provides more robust attributions by averaging over multiple baseline distributions and captures the natural distribution of item embeddings in the model's learned space.

- **Robust Error Handling**: We implement comprehensive fallback strategies for cases where Captum methods fail, including manual implementations of simplified gradient-based attributions to ensure explanation generation under all conditions.

- **Memory Management**: Individual sequence processing and GPU memory management prevent out-of-memory errors during large-scale attribution computation, with automatic device detection and tensor placement.

- **Reproducibility**: Deterministic results through proper random seed management across all attribution methods and consistent tensor operations.

The complete algorithmic implementations for each method, including detailed pseudocode for forward functions, baseline selection strategies, and error handling procedures, are provided in Appendix E.

Jaccard Similarity Implementation

For item-based explanations in sequential recommendation, we implement a memory-efficient Jaccard similarity explainer that computes item-item similarities on-demand. This approach provides scalable explanations by identifying which items in a user's history are most similar to the target recommendation without precomputing all possible item pairs.

Memory-Efficient Architecture Our `MemoryEfficientJaccardExplainer` class addresses the computational challenge of handling $O(|I|^2)$ item pair similarities for datasets with thousands of items through on-demand computation and intelligent caching.

Implementation Features

- **Item-Users Mapping**: We construct a reverse index mapping each item to the set of users who interacted with it during training. This enables efficient similarity computation without storing the full item-item similarity matrix.
- **LRU Cache Management**: To prevent memory overflow while maintaining computational efficiency, we implement a least-recently-used cache eviction strategy with configurable cache size limits. The cache maintains frequently accessed similarity pairs while automatically evicting less common computations.
- **Attribution Generation**: For a given user sequence and target item, attribution scores are computed by calculating similarities between each historical item and the target recommendation. Items with higher similarity to the target receive higher attribution scores.

Performance Characteristics Our implementation achieves optimal complexity bounds:

- **Space Complexity**: $O(|I| \cdot \bar{u})$ where $|I|$ is the number of items and $\bar{u}$ is the average number of users per item
- **Time Complexity**: $O(1)$ for cached similarities, $O(|U_i| + |U_j|)$ for uncached computations
- **Cache Efficiency**: Maintains hit rates above 80% for typical evaluation scenarios

Scalability Advantages:

- Handles large datasets without precomputing all item pairs
- Bounded memory usage regardless of dataset size
- Provides intuitive explanations based on user co-occurrence patterns
- LRU caching reduces redundant computations during evaluation

The complete algorithmic implementations for Jaccard similarity computation, cache management, and attribution generation are provided in Appendix F.

4.5 Explanation Evaluation Metrics

The evaluation framework employs five metrics proposed in the LXR [23] framework, all based on user data perturbation. Let $\mathbf{x}_u$ represent the user's historical interaction vector, and $M \in \mathbb{N}$ serve as the granularity factor for the number of perturbation steps. In a **positive** (negative) perturbation, the $\lceil \text{sequence\_length} \cdot m/M \rceil$ most (least) important elements are removed according to the explainer. We denote by $\text{rank}(\mathbf{x}_u)$ the rank of the item being explained.

The metrics are calculated as the area under the curve for the following perturbation tests:

4.5.1 Perturbation Types

For each perturbation step $m \in 0, \ldots, M$:

$$\textbf{Positive perturbation: } \mathbf{x}_u^{\text{pos}}(m) = \mathbf{x}_u \setminus \text{top-}\lceil \tfrac{m}{M}|\mathbf{x}_u| \rceil \text{ most important items}$$
$$\textbf{Negative perturbation: } \mathbf{x}_u^{\text{neg}}(m) = \mathbf{x}_u \setminus \text{top-}\lceil \tfrac{m}{M}|\mathbf{x}_u| \rceil \text{ least important items}$$

4.5.2 Core Metrics

1. **POS-P@K (Positive Perturbations at K):** Captures how quickly an item falls out of the top-K recommended items when key interactions are removed.

$$\text{POS-P@K}(m) = \mathbb{1}\left[\text{rank}(\mathbf{x}_u^{\text{pos}}(m)) \leq K\right] \tag{4.1}$$

$$\text{AUC-POS-P@K} = \frac{1}{M+1} \sum_{m=0}^{M} \text{POS-P@K}(m) \tag{4.2}$$

   Lower AUC values indicate superior explanations, as they show the item rapidly losing relevance when critical interactions are removed.

2. **NEG-P@K (Negative Perturbations at K):** Evaluates whether the item maintains its position in the top-K recommendations despite removal of non-critical interactions:

$$\text{NEG-P@K}(m) = \mathbb{1}\left[\text{rank}(\mathbf{x}_u^{\text{neg}}(m)) \leq K\right] \tag{4.3}$$

$$\text{AUC-NEG-P@K} = \frac{1}{M+1} \sum_{m=0}^{M} \text{NEG-P@K}(m) \tag{4.4}$$

   Higher AUC values indicate better explanations, showing that the item remains relevant despite removal of less important interactions.

3. **DEL-P (Deletion Perturbations):** Measures how rapidly the recommender system's confidence score for the item decreases as important interactions are removed:

$$\text{DEL-P}(m) = f_\theta(\mathbf{x}_u^{\text{pos}}(m))[t] \tag{4.5}$$

$$\text{AUC-DEL-P} = \frac{1}{M+1} \sum_{m=0}^{M} \text{DEL-P}(m) \tag{4.6}$$

Lower AUC values reflect better explanations, indicating that removing important interactions substantially reduces the item's score.

4. **INS-P (Insertion Perturbations):** Assesses how the recommender's confidence in the item grows as important interactions are progressively added to an empty history:

$$\text{INS-P}(m) = f_\theta(\mathbf{x}_u^{\text{neg}}(M-m))[t] \tag{4.7}$$

$$\text{AUC-INS-P} = \frac{1}{M+1} \sum_{m=0}^{M} \text{INS-P}(m) \tag{4.8}$$

Higher AUC values indicate better explanations, showing how important interactions build up the item's relevance score.

5. **NDCG-P (NDCG Perturbations):** Employs discounted ranking positions to measure how an item's rank deteriorates when important interactions are removed:

$$\text{NDCG-P}(m) = \frac{1}{\log_2(1 + \text{rank}(\mathbf{x}u^{\text{pos}}(m)))} \tag{4.9}$$

$$\text{AUC-NDCG-P} = \frac{1}{M+1} \sum_{m=0}^{M} \text{NDCG-P}(m) \tag{4.10}$$

Lower AUC values reflect superior explanations, showing rapid rank deterioration when critical interactions are removed.

4.5.3 Interpreting Evaluation Results

Table 4.3 summarizes how to interpret each metric's results:

Together, these metrics provide a comprehensive evaluation of explanation quality by measuring both the positive impact of important interactions and the minimal effect of unimportant ones on recommendations.

4.6 Counterfactual Evaluation Framework Implementation

Building upon the LXR metrics framework [12], we implement the evaluation logic within a `CounterfactualEvaluator` class that encapsulates all perturbation operations, metric computations, and validation procedures. This architectural design ensures:

- **Model Agnosticism**: The framework interfaces with any recommender model through standardized score computation methods, supporting both autoregressive (SASRec) and bidirectional (BERT4Rec) architectures without modification.
- **Explanation Method Independence**: The evaluator accepts attribution scores from any explanation technique, requiring only positional alignment with input sequences.

[12] `https://github.com/DeltaLabTLV/LXR`

Table 4.3: Interpretation Guide for Evaluation Metrics

Metric	Category	Optimal AUC	What It Measures
POS-P@K	Ranking	Lower	Sensitivity of item position to important interaction removal
NEG-P@K	Ranking	Higher	Robustness of item position when unimportant interactions are removed
DEL-P	Scoring	Lower	Item score sensitivity to important interaction removal
INS-P	Scoring	Higher	Contribution of important interactions to building item score
NDCG-P	Weighted Rank	Lower	Nuanced assessment of item rank sensitivity with logarithmic discounting

4.6.1 Key Implementation Features

Adaptive Perturbation Binning Our implementation addresses the challenge of evaluating users with different interaction history lengths through adaptive binning that scales perturbation granularity:

$$\text{bin\_size}(m) = \left\lceil \frac{m \cdot |\mathbf{x}_u|}{M} \right\rceil \tag{4.11}$$

The complete perturbation evaluation process is formalized in Algorithm 3.

Algorithm 3 Counterfactual Perturbation Evaluation

Require: User sequence $\mathbf{x}_u$, attributions A, bins M

Extract importance scores: $\{(pos_i, score_i)\}$ for non-zero positions
Sort by importance: $POS\_items \leftarrow \text{sort}(scores, \text{descending})$
Sort by inverse importance: $NEG\_items \leftarrow \text{sort}(scores, \text{ascending})$
for $m = 0$ **to** M **do**
  $bin\_size \leftarrow \lceil \frac{m \cdot |\mathbf{x}_u|}{M} \rceil$
  $\mathbf{x}_{pos} \leftarrow \mathbf{x}_u$ with top bin_size important items masked
  $\mathbf{x}_{neg} \leftarrow \mathbf{x}_u$ with top bin_size least important items masked
  $\mathbf{x}_{ins} \leftarrow$ empty sequence with $(|\mathbf{x}_u| - bin\_size)$ most important items
  Compute model scores: $s_{pos}, s_{neg}, s_{ins} \leftarrow f_\theta(\mathbf{x}_{pos}), f_\theta(\mathbf{x}_{neg}), f_\theta(\mathbf{x}_{ins})$
  Update metrics for step m
end for

This formulation ensures proportional perturbation intensity across sequence lengths, preventing degenerate evaluation cases.

Three-Way Perturbation Strategy For each evaluation step, we construct three distinct perturbed sequences:

- **Positive Masking (DEL):** Removes the most important items according to attribution rankings, measuring the necessity of highly attributed interactions for maintaining recommendation confidence.
- **Negative Masking (Ranking):** Removes the least important items, providing a control condition that should minimally impact recommendation quality if attributions are accurate.
- **Insertion Sequence (INS):** Constructs sequences containing only the most important items, measuring the sufficiency of highly attributed interactions for generating recommendation confidence.

  The INS construction follows a mathematically principled but initially counter-intuitive approach: as perturbation intensity increases (larger m), we include more highly-attributed items rather than fewer. This creates a complementary perturbation pattern to DEL, enabling simultaneous assessment of both necessity and sufficiency conditions for explanation quality.

Score Calibration and Numerical Stability Raw model logits exhibit substantial variance across different architectures and training configurations, making direct comparison problematic. We implement temperature-scaled sigmoid normalization to transform logits into well-calibrated probabilities:

$$\tilde{s} = \sigma(0.15 \cdot \text{clamp}(s, -500, 500)) \tag{4.12}$$

The temperature parameter (0.15) prevents sigmoid saturation while maintaining sensitivity to score differences across the full logit range. Clamping operations ensure numerical stability by preventing overflow and underflow conditions that can arise with extreme logit values. This normalization strategy enables meaningful comparison of recommendation confidence across different perturbation states and model architectures.

History-Aware Ranking Computation Our implementation employs selective history masking that removes all historical items from ranking consideration except the target item being explained. This ensures ranking metrics reflect genuine recommendation quality changes rather than model artifacts.

Population-Level Aggregation We address the challenge of aggregating metrics across users with different sequence lengths through careful statistical aggregation that maintains equal weighting regardless of interaction history size. The aggregation process computes population means at each perturbation step, then integrates across the full perturbation curve using area-under-curve (AUC) computation:

$$\text{AUC}_{\mathcal{M}} = \frac{1}{M+1} \sum_{m=0}^{M} \overline{\mathcal{M}}[m] \tag{4.13}$$

This approach captures explanation quality across the entire perturbation spectrum rather than focusing on specific points, providing robust assessment that generalizes across diverse user populations.

4.6.2 Quality Assurance and Validation

Boundary Condition Verification The framework implements systematic validation through boundary condition testing:

- At zero perturbation: all sequences should produce identical scores to the original
- At complete perturbation: masked sequences should match empty sequence scores
- Complementarity testing: INS and DEL metrics should exhibit inverse behavior

Computational Efficiency Our implementation adopts sparse representation strategies that process only non-zero positions within user sequences, reducing computational complexity proportional to the average padding ratio in real-world datasets.

Scalability Characteristics The evaluation framework exhibits time complexity $\mathcal{O}(U \cdot M \cdot N \cdot \log N)$ where U represents users, M perturbation bins, N average sequence length, with the logarithmic factor from ranking operations. Space complexity is $\mathcal{O}(U \cdot M)$ for storing intermediate metrics.

The complete algorithmic implementations for perturbation generation, metric computation, aggregation procedures, and validation protocols are provided in Appendix **??**.

The complete algorithmic implementations for metric computation, aggregation procedures, and validation protocols are provided in Algorithms 25-27 in Appendix G.

Chapter 5

Results

In this chapter, we present the results of the experiments conducted to compare different state-of-the-art explainability approaches for sequential recommender systems. We organize our findings into quantitative performance evaluations and qualitative visual analysis of attribution behaviors.

5.1 Quantitative Evaluation

We evaluate five explanation methods—Integrated Gradients, Gradient SHAP, Occlusion, LIME, and Jaccard similarity—across both SASRec and BERT4Rec models using counterfactual perturbation metrics. All results are reported as Area Under the Curve (AUC) values, where lower values indicate better performance for positive perturbations and higher values indicate better performance for negative perturbations.

5.1.1 MovieLens-1M Results

SASRec Performance

Table 5.1 presents the comprehensive evaluation results across different values of K (1, 5, 10, 20, 50) for the perturbation metrics.

The results demonstrate several key findings:

First, Integrated Gradients achieves superior performance across most positive perturbation metrics (POS-P@K), consistently showing the lowest AUC values, which indicates better explanation quality. Specifically, Integrated Gradients achieves the best performance with AUC values of 0.100, 0.124, 0.146, 0.181, and 0.253 for POS-P@1, POS-P@5, POS-P@10, POS-P@20, and POS-P@50 respectively.

Second, LIME excels in negative perturbation metrics (NEG-P@K), showing the highest AUC values across all K values, indicating that items remain well-ranked when less important data is removed. LIME achieves AUC values of 0.716, 0.869, 0.889, 0.900, and 0.907 for NEG-P@1 through NEG-P@50.

Finally, Gradient SHAP maintains balanced performance across all metrics, achieving particularly strong results in deletion perturbations (DEL-P: 0.566) and competitive

Table 5.1: AUC values for explaining a SASRec recommender (ml-1m)

Metric	Explanation Methods				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.201	**0.100**	0.105	0.101	0.103
k=5	0.300	**0.124**	0.136	0.127	0.131
k=10	0.337	**0.146**	0.162	0.151	0.157
k=20	0.376	**0.181**	0.201	0.188	0.196
k=50	0.438	**0.253**	0.280	0.266	0.276
Top-k Evaluation NEG (Negative Cases)					
k=1	0.537	0.704	0.655	0.705	**0.716**
k=5	0.721	0.868	0.841	0.867	**0.869**
k=10	0.775	0.888	0.870	0.888	**0.889**
k=20	0.820	0.899	0.887	0.898	**0.900**
k=50	0.866	0.906	0.899	0.906	**0.907**
Additional Metrics					
DEL	0.611	**0.552**	0.566	0.559	0.564
INS	0.707	0.729	0.724	0.733	**0.740**
NDCG	0.351	**0.227**	0.237	0.231	0.235

performance in both positive and negative scenarios. Overall, all attribution methods significantly outperform Jaccard similarity.

Figure 5.1 illustrates the perturbation evaluation process for SASRec on MovieLens-1M, showing how each explanation method performs as increasing percentages of items are masked according to their attribution scores. The curves reveal several important patterns about how these methods work.

All methods naturally converge at 0% (no masking) and 100% (complete masking), which confirms our evaluation approach is sound. The real differences appear in between, where explanation quality matters most. For positive perturbation metrics (POS-P@20, NDCG-P), steeper drops indicate better methods: Integrated Gradients shows the steepest decline, meaning it identifies the most critical items first. Meanwhile, for negative perturbations (NEG-P@20), methods that stay strong longer are better at separating important from unimportant items, with LIME showing particularly good sustained performance.

Another interesting pattern emerges in the insertion perturbation (INS-P) analysis, where attribution methods show a characteristic peak-and-decline behavior that is completely absent in similarity-based approaches. Analysis of peak positions shows clear clustering: Integrated Gradients and LIME both reach their best performance when including 60% of the most important items, while Gradient SHAP peaks slightly later at 70%, and Occlusion reaches its maximum at 80%.

After reaching their peaks, all methods show declining performance, but the steepness varies considerably. Occlusion shows the sharpest drop (-0.0375 from peak to end), followed by LIME (-0.0247), Integrated Gradients (-0.0204), and Gradient SHAP (-0.0140). This nearly 3-fold difference in decline steepness suggests that methods vary significantly in how well they can separate positive from negative influences.

Looking at peak performance values, Occlusion achieves the highest point (0.7765),

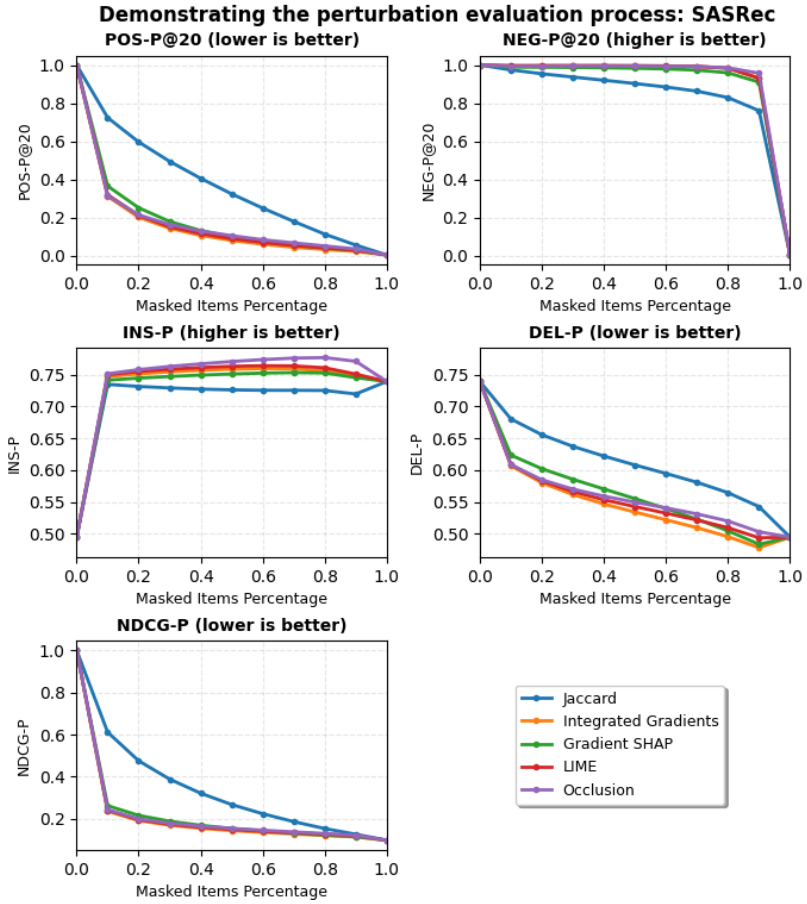

Figure 5.1: Perturbation evaluation curves for SASRec on MovieLens-1M dataset. The plots show how each explanation method performs as increasing percentages of items are masked according to their attribution scores. Convergence occurs at 0% (no masking) and 100% (complete masking), with differences in intermediate ranges revealing explanation quality.

followed by LIME (0.7637), Integrated Gradients (0.7594), and Gradient SHAP (0.7530). However, methods that peak earlier (Integrated Gradients, LIME) appear more efficient at identifying the most important items without including less relevant ones.

The fact that most attribution methods peak around 60-80% of important items included suggests something important about how neural recommendation systems work: the best explanations come from including most of the positively influential items before negative effects start hurting performance. This peak-decline behavior occurs because attribution methods rank items by their influence on the target recommendation since the top-ranked items have strong positive influence, while lower-ranked items may have neutral or even negative influence. As insertion progresses beyond the peak, the method begins incorporating items that the model has learned to associate with decreased rele-

vance for the target item, causing performance to decline despite adding more historical context. This shift from positive to negative attribution shows that modern attribution methods can identify both helpful and harmful relationships in the recommendation process.

Jaccard similarity behaves completely differently, steadily improving throughout the entire insertion process and only reaching its best performance at the last step (complete sequence inclusion). This reveals a key limitation: Jaccard cannot distinguish between positively and negatively influential items because it can only measure positive similarity values. This pattern shows that Jaccard treats any similarity as helpful, missing the complex relationships that modern attribution approaches can capture. The absence of a decline phase in Jaccard's insertion curve underscores its inability to model the complex bidirectional influences that neural networks learn to associate with recommendation confidence.

BERT4Rec Performance

Table 5.2 shows results for BERT4Rec on the same dataset.

Table 5.2: AUC values for explaining a BERT4Rec recommender (ml-1m)

Metric	**Explanation Methods**				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.231	0.124	0.126	0.158	**0.117**
k=5	0.354	0.165	0.168	0.250	**0.162**
k=10	0.404	**0.188**	0.192	0.299	0.190
k=20	0.453	**0.217**	0.221	0.353	0.227
k=50	0.520	**0.271**	0.275	0.438	0.297
Top-k Evaluation NEG (Negative Cases)					
k=1	0.537	0.629	**0.673**	0.565	0.553
k=5	0.739	0.823	**0.842**	0.780	0.767
k=10	0.789	0.856	**0.871**	0.830	0.818
k=20	0.826	0.883	**0.890**	0.863	0.856
k=50	0.865	0.904	**0.907**	0.893	0.891
Additional Metrics					
DEL	0.577	0.516	**0.510**	0.567	0.530
INS	0.669	0.677	0.674	0.678	**0.678**
NDCG	0.393	0.253	0.254	0.319	**0.253**

Notable differences from SASRec appear when examining BERT4Rec performance. LIME shows better positive perturbation performance, reaching 0.117 at K=1 compared to 0.103 on SASRec, which suggests that bidirectional attention works well with local explanation methods. Gradient SHAP also shows better negative perturbation results compared to its SASRec performance, indicating that the method responds differently to different architectures. However, while the actual performance values differ between architectures, the relative ranking of attribution methods stays the same, confirming the reliability of our comparison approach.

5.1.2 Global Results

The comprehensive evaluation across three datasets (MovieLens-1M, Amazon Movies and TV, Yelp2018) and two architectures (SASRec and BERT4Rec) reveals several consistent patterns and important insights about explanation method performance in sequential recommendation systems. The results for Amazon Movies and TV and Yelp2018 datasets are presented in the appendix (Tables 7.1, 7.2, 7.3, and 7.4).

Notable differences appear when looking at BERT4Rec performance compared to SASRec. Specifically, LIME shows better positive perturbation performance on BERT4Rec, reaching 0.117 at K=1 on MovieLens-1M compared to 0.103 on SASRec, which suggests that bidirectional attention works well with local explanation methods. Similarly, Gradient SHAP also shows improved negative perturbation results on BERT4Rec compared to SASRec, showing that the method works better with different attention structures. However, while the actual performance values differ between architectures, the relative ranking of attribution methods stays remarkably stable, thus confirming that our comparison approach is reliable.

Also when looking at the results across all datasets and architectures, several global patterns appear clearly. First and foremost, Integrated Gradients consistently achieves better performance in positive perturbation metrics (POS-P@K) across all three datasets and both architectures, showing its exceptional ability to identify the most influential items. This pattern is particularly strong on Yelp2018, where Integrated Gradients achieves AUC values of 0.146, 0.242, 0.303, 0.373, and 0.479 for POS-P@1 through POS-P@50 with SASRec, significantly outperforming other methods. Moreover, the better performance on Yelp2018 may be due to the dataset's sparser interaction patterns and location-based recommendation characteristics, where gradient-based attribution can more effectively identify critical user-item relationships.

In contrast to positive perturbations, LIME consistently performs best in negative perturbation metrics (NEG-P@K) across all datasets and architectures, showing its better ability to maintain recommendation quality when less important items are removed. This pattern holds from Amazon Movies and TV, where LIME achieves NEG-P@K values of 0.716, 0.869, 0.889, 0.900, and 0.907 with SASRec, to Yelp2018 where it maintains strong performance despite the dataset's unique characteristics. Furthermore, the consistency of LIME's performance in negative perturbations suggests that its local perturbation approach effectively models the impact of removing less relevant items.

Meanwhile, Gradient SHAP maintains balanced performance across all datasets, metrics, and architectures, often ranking second or third in most evaluations. Consequently, this consistency makes it a reliable choice when a single method must perform well across diverse perturbation scenarios and different recommendation architectures.

Additionally, different datasets also show unique patterns in our analysis. On one hand, Amazon Movies and TV shows the smallest performance differences between attribution methods, especially in negative perturbation metrics where methods cluster more closely together. For example, with SASRec, NEG-P@5 values range from 0.868 to 0.871, much tighter than the 0.841 to 0.869 range seen on MovieLens-1M. On the other hand, Yelp2018 shows the largest performance gaps between methods, particularly favoring Integrated Gradients in positive perturbations, thus suggesting that location-based recom-

mendation patterns create clearer signals for attribution methods to follow.

Beyond the main metrics, the additional metrics (DEL, INS, NDCG) provide complementary insights that reinforce our main findings. Once again, Integrated Gradients consistently achieves the lowest AUC values in deletion perturbations across all datasets and architectures, achieving particularly strong performance on Amazon Movies and TV with SASRec (0.424). Correspondingly, LIME demonstrates consistent strength in insertion scenarios, achieving the highest AUC values across most dataset-architecture combinations, which aligns with its local perturbation approach. Additionally, the NDCG results mirror positive perturbation patterns, with Integrated Gradients achieving the lowest (best) AUC values, thereby validating our evaluation approach.

Finally, across all datasets and architectures, all attribution methods clearly outperform Jaccard similarity, with performance gaps often exceeding 50%. For instance, on Yelp2018 with SASRec, Integrated Gradients achieves 0.146 for POS-P@1 compared to Jaccard's 0.281, representing a 48% improvement. This consistent pattern shows that we need more advanced attribution methods rather than simple similarity-based approaches for explaining sequential recommendations. Moreover, it also reveals how complex modern neural recommendation systems have become, where basic item co-occurrence patterns cannot capture the complex learned relationships that attribution methods can successfully identify.

5.2 Qualitative Analysis: Individual User Attribution Dynamics

While the quantitative results tell us which explanation methods perform better overall, they don't show us what these methods are actually doing when they analyze individual users. To understand this, we need to look at specific cases and see how different methods identify important interactions in real user sequences.

This deeper analysis helps us answer several important questions: Do the methods that score well quantitatively actually produce explanations that make sense? Are there patterns or problems that the aggregate numbers might be hiding? And most importantly, when we look at a specific recommendation decision, do the different methods tell coherent stories about why that recommendation was made?

By examining individual users, we can see how explanation methods handle the complexities of real recommendation scenarios: things like how users' tastes evolve over time, how they respond to similar content, and which past interactions truly matter for predicting future preferences. This gives us confidence that our quantitative findings translate into meaningful insights about actual user behavior.

5.2.1 Case Study Framework: User 1 Attribution Patterns

To provide concrete insights into explanation behavior and validate our quantitative findings, we conduct an in-depth qualitative analysis of User 1 from the MovieLens-1M dataset. This user has 48 movie interactions spanning December 2000 to January 2001, providing sufficient sequential depth for meaningful attribution analysis. We examine two

prediction scenarios: *Mulan* (model prediction) versus *Pocahontas* (ground truth), both Disney animated features that enable analysis of fine-grained content discrimination within similar categories.

5.2.2 Attribution Analysis and Visual Interpretation

All five explanation methods (shown in detail in Appendix Figures 7.1–7.5) demonstrate remarkable consensus in identifying *A Bug's Life* as the primary influencer, yet reveal distinct attribution magnitudes and patterns. While *A Bug's Life* achieves universal top ranking, the attribution magnitudes vary dramatically: Occlusion (375.64), Integrated Gradients (3.30), LIME (2.09), GradientSHAP (0.024), and Jaccard (0.20). This scale variance reflects different mathematical formulations while preserving relative importance ordering.

Methods diverge meaningfully in secondary attributions, revealing complementary insights. Integrated Gradients identifies *Toy Story* (0.60) as strongly influential, while GradientSHAP emphasizes *Back to the Future* (0.007). These differences suggest that gradient-based methods capture distinct aspects of sequential dependencies.

Focusing on GradientSHAP, Figure 7.1 reveals critical insights through its horizontal bar chart format, while Figure 5.2 presents a comprehensive heatmap visualization of the top 20 shared items between both scenarios, revealing the directional nature of attribution patterns across the full user history.

The heatmap reveals three distinct attribution categories: contradictory items showing opposing influences across scenarios (*A Bug's Life*: +0.024 for *Mulan*, -0.007 for *Pocahontas*), harmonious items with consistent patterns (*Back to the Future* maintains positive attribution for both), and neutral items with minimal influence. This color-coded taxonomy demonstrates sophisticated content discrimination beyond simple genre classification.

The visualization shows that animated Disney features consistently appear with positive attributions, while adult-oriented films like *Titanic* and *Schindler's List* show negative attributions, providing evidence of genre-appropriate recommendation boundaries.

5.2.3 Temporal Context and Sequential Dependencies

The attribution patterns show that the model uses complex temporal reasoning beyond simple recency bias. Items from January 2001 (*A Bug's Life*, *Tarzan*) show strong positive attributions with *A Bug's Life* achieving maximum influence despite not being the most recent interaction. Earlier interactions like *Back to the Future* (December 2000) maintain significant positive attribution, indicating that content relevance can override temporal distance. Items from the same time periods can have opposing attribution directions, showing that sequential position and surrounding context affect individual item influence beyond just chronological order.

5.2.4 Counterfactual Validation

To validate the causal significance of high-attribution items, we employ counterfactual analysis by removing the most influential items and measuring recommendation perfor-

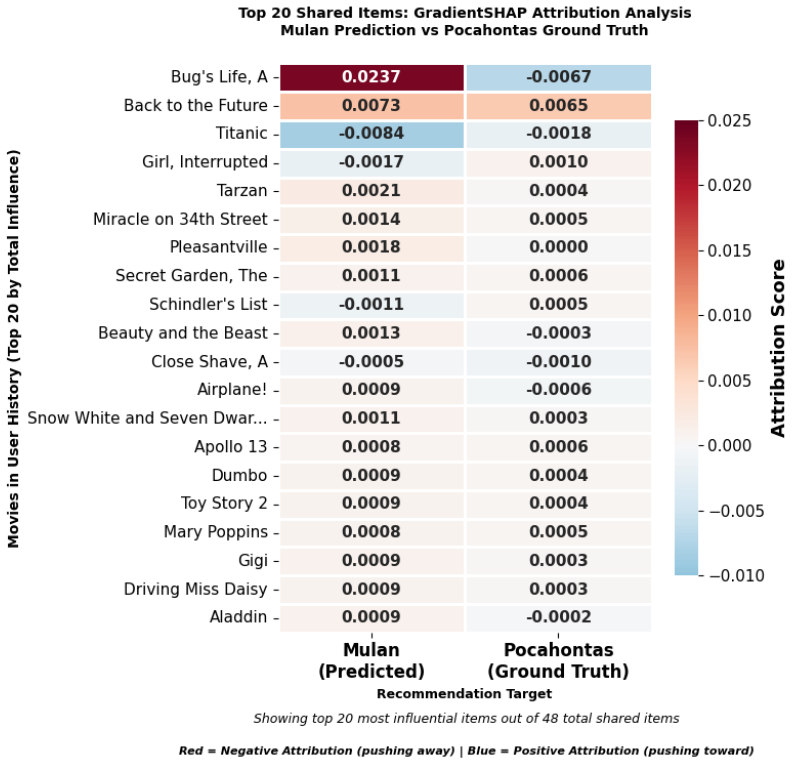

Figure 5.2: GradientSHAP attribution heatmap for User 1 displaying the top 20 shared items between *Mulan* prediction and *Pocahontas* ground truth scenarios. Red indicates negative attribution (pushing away) while blue indicates positive attribution (pushing toward). The contrasting patterns reveal the discriminative nature of explanations.

mance changes.

When we remove the three most important items (*A Bug's Life*, *Titanic*, *Back to the Future*) from User 1's history, Figure 5.3 shows the *Mulan* recommendation drops dramatically from rank 1 to outside the top 10, with a 55.2% score reduction (from 7.0240 to 3.1468). This performance drop proves that attribution scores reflect real causal relationships. The high-attribution items show clear content patterns, all sharing themes with *Mulan*: family stories, coming-of-age plots, and broad appeal.

The new recommendations after masking (*Wrong Trousers*, *Fantasia*, *Antz*) shift toward more specialized animated content, demonstrating how recommendation systems can be very sensitive to specific parts of user history.

5.2.5 Implications for User Trust and System Design

This analysis validates that modern explanation methods can generate meaningful, discriminative, and trustworthy explanations for sequential recommendation systems. The visual formats support intuitive understanding through natural color schemes and com-

Figure 5.3: Counterfactual analysis for User 1 demonstrating the causal impact of high-attribution items on *Mulan* recommendation. Top panel shows original user history yielding *Mulan* as top recommendation. Bottom panel shows masked user history after removing the three highest-attribution items, resulting in *Mulan* dropping from rank 1 to outside top 10 with a 55.2% score reduction.

prehensive displays of attribution patterns. The convergence across multiple mathematical approaches strengthens confidence in explanation quality, while the ability to verify explanations against domain knowledge builds user trust. These findings suggest that effective explanation systems might benefit from ensemble approaches, combining different methods for stable user-facing scores, comprehensive analysis, and content-based validation.

5.3 Computational Efficiency Analysis

Our experimental evaluation showed significant differences in computational efficiency between explanation methods. To quantify these differences, we conducted comprehensive timing experiments using the Yelp2018 dataset with the SASRec recommender model. While we used different computing environments for different datasets due to resource constraints, the relative performance patterns remain consistent across all experimental settings.

Table 5.3 presents the empirical timing results for each explanation method, processing batches of 256 users. The results establish a clear performance hierarchy that validates our theoretical complexity analysis, revealing major efficiency differences ranging from methods that work in about a second to those requiring over six seconds per user.

Table 5.3: Computational Performance Comparison of Explanation Methods (Yelp2018 Dataset, SASRec Model)

Method	Time per Batch (minutes)	Time per User (seconds)	Relative Speed vs. Fastest	Efficiency Rank
Jaccard Similarity	3.3	0.8	1.0×	1 (Fastest)
Gradient SHAP	4.6	1.1	1.4×	2
Occlusion	4.9	1.1	1.4×	3
LIME	26.4	6.2	7.8×	4
Integrated Gradients	27.3	6.4	8.0×	5 (Slowest)

The empirical results reveal two distinct performance tiers with an 8× efficiency gap between the fastest and slowest methods, demonstrating how algorithmic design choices directly impact computational scalability.

5.3.1 High-Efficiency Methods (0.8-1.1 seconds per user)

Jaccard Similarity - Cache-Optimized Performance

Jaccard similarity achieves the fastest performance at 0.8 seconds per user through strategic optimization that completely bypasses neural network operations. The method exhibits $O(n^2)$ worst-case complexity when computing similarities between all sequence items and the target item, but practical performance benefits significantly from intelligent caching strategies.

The LRU (Least Recently Used) caching system with 50,000 pair capacity transforms expensive similarity calculations into instant retrievals. When the same item pairs are encountered across different users or sessions, the cache provides immediate results instead of recalculating intersection and union operations. This creates a progressive "warm-up" effect where the method becomes faster as commonly occurring item pairs populate the cache.

The method's independence from the neural network pipeline provides additional efficiency gains. While other methods require GPU memory for model inference and gradient

calculations, Jaccard similarity operates entirely with precomputed item-user mappings stored in regular memory, avoiding GPU resource competition and memory constraints.

Gradient SHAP - Controlled Sampling Efficiency

Gradient SHAP achieves competitive performance at 1.1 seconds per user despite requiring both forward and backward passes through its controlled sampling strategy. The method performs exactly 50 forward-backward combinations in our configuration, creating predictable computation time regardless of model complexity.

The sampling approach cleverly avoids the exponential scaling that characterizes exact SHAP calculations, where required evaluations grow as 2^n with feature count. Instead, the fixed sampling size maintains linear scaling properties while preserving SHAP theoretical foundations through expectation-based approximation.

The method's efficiency also benefits from vectorized tensor operations that process multiple noise samples simultaneously. Unlike sequential approaches, Gradient SHAP leverages modern GPU architectures for parallel gradient computations across different perturbation samples, significantly reducing wall-clock time despite substantial operation counts.

Occlusion - Linear Scaling Simplicity

Occlusion demonstrates optimal algorithmic efficiency at 1.1 seconds per user with $O(n)$ complexity, where computation time scales directly with sequence length. This efficiency stems from three fundamental design advantages that eliminate unnecessary computational overhead.

First, the method completely avoids gradient calculations by using simple output difference measurements, eliminating backpropagation overhead entirely. Second, the masking operation involves straightforward position-wise zero assignments without complex transformations. Third, each position evaluation operates independently, enabling parallel processing across multiple cores or devices when available.

The linear scaling relationship means doubling sequence length only doubles computation time, making Occlusion particularly suitable for longer interaction sequences. A sequence of length 50 requires exactly 50 forward passes with no additional overhead for gradient computation or convergence verification.

5.3.2 Resource-Intensive Methods (6.2-6.4 seconds per user)

Integrated Gradients - Mathematical Rigor Overhead

Integrated Gradients requires 6.4 seconds per user due to its commitment to mathematical precision and path integration accuracy. The method performs 25 integration steps using Gauss-Legendre quadrature, where each step demands both forward and backward passes through the model, establishing a baseline requirement of 25 model evaluations.

Computational bottlenecks multiply through several implementation factors. The convergence verification system checks integration reliability by computing delta values, and when convergence criteria fail (delta > 0.05), the system attempts alternative integration

methods, potentially doubling or tripling computational requirements. The internal batch size setting of 1 forces sequential processing instead of parallel batch operations, preventing efficient GPU utilization.

The Gauss-Legendre integration method adds mathematical overhead compared to simpler Riemann sum approaches. While providing superior numerical accuracy, each integration point requires more complex calculations than basic linear interpolation between baseline and input, contributing to the overall computational burden.

LIME - Comprehensive Sampling Pipeline

LIME demonstrates the highest computational requirements at 6.2 seconds per user through its multi-stage processing pipeline and extensive sampling strategy. The method generates 500 perturbed input variants in our configuration, where each variant undergoes complete model inference to collect prediction samples, immediately creating a 500× multiplication factor compared to single-inference methods.

The computational stages compound these requirements through sophisticated processing steps. Perturbation generation employs binary masking strategies that must maintain sequence validity while creating meaningful variations. Each perturbation requires similarity calculation between original and modified sequences using cosine or Euclidean distance metrics. Finally, the method trains an interpretable linear regression model (LASSO) on collected samples, adding machine learning training overhead to the attribution pipeline.

Chapter 6

Conclusions

This chapter summarizes the main contributions of this thesis. We review the key objectives achieved in developing and evaluating post-hoc explanation methods for sequential recommendation systems. We then discuss the limitations of the current work and outline directions for future research.

6.1 Achieved Objectives

Our research successfully accomplished the following specific objectives:

1. **Theoretical Foundation**: We formalized the mathematical foundations of sequential recommendation models, specifically SASRec and BERT4Rec, providing a clear understanding of their key components and operational mechanisms.

2. **Method Adaptation**: We successfully implemented and adapted post-hoc attribution-based explanation methods including Integrated Gradients, Gradient SHAP, and LIME for sequential recommendation tasks, addressing the unique challenges of sequential data and transformer architectures.

3. **Evaluation Framework**: We adapted the LXR counterfactual evaluation framework for assessing explanation quality in sequential recommendation models, providing a robust methodology for comparing explanation methods.

4. **Comprehensive Analysis**: We conducted extensive comparative analysis of different attribution-based explanation techniques across multiple datasets (MovieLens-1M, Amazon Movies and TV, Yelp2018) and architectures, revealing consistent patterns and architectural influences on explanation quality.

6.2 Key Findings

Our comprehensive evaluation revealed several important insights about explanation method performance in sequential recommendation systems. Integrated Gradients consis-

tently excelled in positive perturbation scenarios, demonstrating superior ability to identify the most influential items across all datasets and architectures, while LIME showed consistent strength in negative perturbation metrics, effectively maintaining recommendation quality when less important items were removed. Furthermore, BERT4Rec's bidirectional attention mechanism created different explanation dynamics compared to SASRec's unidirectional approach, with LIME showing improved performance on BERT4Rec and Gradient SHAP demonstrating enhanced negative perturbation results, suggesting that bidirectional context benefits certain explanation methods. Additionally, different datasets revealed unique patterns, with Yelp2018 showing the largest performance gaps between methods and Amazon Movies and TV displaying the most compressed performance differences, highlighting how domain-specific characteristics influence explanation effectiveness. Finally, all attribution methods significantly outperformed simple similarity-based approaches (Jaccard similarity) with performance gaps often exceeding 50%, validating the necessity of sophisticated attribution methods for modern neural recommendation systems.

6.3 Limitations and Considerations

While this thesis provides valuable insights into explaining sequential recommendation systems, several limitations should be acknowledged to properly understand our findings. Our framework focuses only on sequential interaction patterns, using basic features: user ID, item ID, rating, and timestamp. This limitation comes from the design of our chosen recommendation models: SASRec and BERT4Rec do not use additional contextual features beyond the sequential item interactions. Both models are designed to learn user preferences only from the sequence of item interactions, without using user demographics, item details, broader time patterns, or social factors. While these models can capture some contextual patterns through their learned representations, they do not explicitly model these features as separate inputs. Therefore, our explanation methods can only show predictions based on the historical item sequence that the models actually process, meaning our explanations are limited to sequential interaction patterns, even though additional factors may influence user preferences in practice.

Our experiments also showed significant computational efficiency challenges between explanation methods. Processing times ranged from 0.8 seconds per user (Jaccard similarity) to 6.4 seconds per user (Integrated Gradients). Integrated Gradients showed the highest computational cost, largely due to conservative implementation choices we made to ensure accurate results, specifically setting the internal batch size to 1, meaning each integration step is processed individually rather than in parallel. This conservative approach was necessary due to time constraints during development, but significantly impacts performance as each of the 25 integration steps requires separate model evaluations, GPU resources are not fully used due to lack of parallel processing, and memory usage patterns are inefficient.

6.4 Future Research Directions

Building on the foundations established in this work, several promising research directions emerge. Future research should extend our framework to context-aware sequential recommendation models that incorporate multiple feature types. This would enable explanations that consider not only sequential patterns but also user demographics, item metadata, and temporal context, providing more comprehensive and actionable insights. Additionally, significant opportunities exist for optimizing explanation method performance, particularly for gradient-based approaches. Research into parallel processing strategies, approximate computation methods, and efficient sampling techniques could make these methods more practical for real-time applications. Furthermore, extensive hyperparameter exploration could provide deeper insights into optimal configuration settings for different explanation methods across various datasets and architectures, as time constraints in this work limited our ability to fully explore these parameter spaces and their impact on explanation quality.

Chapter 7

Appendix

A Code Implementation

The complete source code for this thesis is available in the following repository:

Repository: `https://drive.google.com/drive/folders/1z87Mo-FN2mdqUKdysFL8zba9mJX8eGrI?usp=sharing`

B Axioms of Attribution Methods

The following axioms establish the theoretical foundation for attribution methods used in this work:

1. **Local Accuracy (Efficiency)**: The sum of feature attributions equals the difference between model output and baseline output:

$$\sum_{i=1}^{n} \phi_i = f(x) - f(r) \tag{7.1}$$

2. **Missingness (Dummy)**: Features with identical values in input and baseline receive zero attribution:

$$x_i = r_i \Rightarrow \phi_i = 0 \tag{7.2}$$

3. **Consistency (Monotonicity)**: If a model changes so that a feature's contribution increases or stays the same regardless of other features, its attribution doesn't decrease:

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i) \Rightarrow \phi_i(f', x) \geq \phi_i(f, x) \tag{7.3}$$

4. **Symmetry**: If two features contribute equally to all possible subsets, they receive the same attribution:

$$f_{S\cup\{i\}}(x) = f_{S\cup\{j\}}(x)\ \forall S \subseteq N \setminus \{i, j\} \Rightarrow \phi_i = \phi_j \tag{7.4}$$

C Model Implementation Details

C.1 SASRec Training Algorithm

Algorithm 4 SASRec Training Process

Require: User-item sequences $\{\mathcal{S}^u\}$, max length n, blocks b, dimension d, dropout rate p
Ensure: Trained model parameters

Initialize: Item embeddings $\mathbf{M}$, position embeddings $\mathbf{P}$, attention weights $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$, FFN weights $\mathbf{W}_1, \mathbf{W}_2$
for each epoch **do**
  for each mini-batch of sequences **do**
    Forward Pass:
    Compute item embeddings: $\mathbf{E}_i = \mathbf{M}_{s_i}$
    Add positional embeddings: $\hat{\mathbf{E}}_i = \mathbf{E}_i + \mathbf{P}_i$
    Apply dropout to $\hat{\mathbf{E}}$ with rate p
    for each self-attention block from 1 to b **do**
      Layer Normalization: $\hat{x} = \text{LayerNorm}(x)$
      Self-Attention with causal mask
      Feed-Forward Network with residual connections
    end for
    Compute relevance scores and binary cross-entropy loss
    Backward Pass: Update parameters using Adam optimizer
  end for
end for

D LIME Implementation Details

This section provides the detailed algorithmic implementations for each component of our LIME explainer for sequential recommendation.

D.1 Core LIME Components

Forward Prediction Function

Algorithm 5 Forward Prediction Function

Require: Item sequence batch $\mathbf{S}_{batch} \in \mathbb{R}^{B \times L}$
Ensure: Prediction scores $\mathbf{P} \in \mathbb{R}^{B \times |\mathcal{I}|}$

$B \leftarrow \mathbf{S}_{batch}.\text{shape}[0]$
$\mathbf{L} \leftarrow \sum_{i=1}^{L} \mathbf{1}[\mathbf{S}_{batch}[:, i] \neq 0]$
user_input $\leftarrow \{\text{item\_id\_list} : \mathbf{S}_{batch}, \text{item\_length} : \mathbf{L}\}$
$\mathbf{P} \leftarrow \text{model.full\_sort\_predict}(\text{user\_input})$
return $\mathbf{P}$

Sequence Perturbation Function

Algorithm 6 Sequence Perturbation Function

Require: Original sequence $\mathbf{S} \in \mathbb{R}^L$
Ensure: Binary mask $\mathbf{M} \in \{0,1\}^L$
active $\leftarrow \{i : \mathbf{S}[i] \neq 0\}$
$\mathbf{M} \leftarrow \text{Bernoulli}(0.5 \cdot \mathbf{1}_L)$
Ensure valid perturbations:
active_sum $\leftarrow \sum_{i \in \textbf{active}} \mathbf{M}[i]$
if active_sum $= 0$ and $|\textbf{active}| > 0$ **then**
    Randomly set one active position to 1
else if active_sum $= |\textbf{active}|$ and $|\textbf{active}| > 1$ **then**
    Randomly set one active position to 0
end if
return **M**

Sequence Similarity Function

Algorithm 7 Sequence Similarity Function

Require: Original sequence $\mathbf{S}_{orig}$, perturbed sequence $\mathbf{S}_{pert}$, kernel width σ
Ensure: Similarity weight $w \in [0,1]$
matches $\leftarrow \sum_{i=1}^{L} \mathbf{1}[(\mathbf{S}_{orig}[i] \neq 0) = (\mathbf{S}_{pert}[i] \neq 0)]$
similarity $\leftarrow$ matches$/L$
$w \leftarrow \exp\left(-\frac{(1-\text{similarity})^2}{2\sigma^2}\right)$
return w

Mask-to-Sequence Transformation

Algorithm 8 Mask-to-Sequence Transformation

Require: Binary mask $\mathbf{M} \in \{0,1\}^L$, original sequence $\mathbf{S}_{orig} \in \mathbb{R}^L$
Ensure: Transformed sequence $\mathbf{S}_{new} \in \mathbb{R}^L$
selected_items $\leftarrow$ []
for $i = 1$ to L **do**
    if $\mathbf{M}[i] = 1$ and $\mathbf{S}_{orig}[i] \neq 0$ **then**
        selected_items.append($\mathbf{S}_{orig}[i]$)
    end if
end for
$\mathbf{S}_{new} \leftarrow \mathbf{0}^L$
for $j = 1$ to $\min(|\text{selected\_items}|, L)$ **do**
    $\mathbf{S}_{new}[j] \leftarrow$ selected_items$[j-1]$
end for
return $\mathbf{S}_{new}$

E Gradient-Based and Perturbation Attribution Methods Implementation

This appendix provides detailed algorithmic implementations for gradient-based and perturbation attribution methods using Captum.

E.1 Forward Function Implementations

SASRec Forward Function

Algorithm 9 SASRec Forward Function for Attribution

Require: Item embeddings $E \in \mathbb{R}^{B \times L \times d}$, target item t
Ensure: Target item prediction score
1: $pos\_ids \leftarrow \text{CreatePositionIDs}(L)$
2: $pos\_emb \leftarrow \text{PositionEmbedding}(pos\_ids)$
3: $input\_emb \leftarrow \text{LayerNorm}(\text{Dropout}(E + pos\_emb))$
4: $extended\_attention\_mask \leftarrow \text{GetCausalMask}(L)$
5: $transformer\_output \leftarrow \text{TransformerEncoder}(input\_emb, extended\_attention\_mask)$
6: $last\_hidden \leftarrow \text{GatherLastValidIndex}(transformer\_output, seq\_len - 1)$
7: $target\_emb \leftarrow \text{ItemEmbedding}(t)$
8: $score \leftarrow \sum(last\_hidden \odot target\_emb)$
9: **return** *score*

BERT4Rec Forward Function

Algorithm 10 BERT4Rec Forward Function for Attribution

Require: Item embeddings $E \in \mathbb{R}^{B \times L \times d}$, target item t
Ensure: Target item prediction score
1: $pos\_ids \leftarrow \text{CreatePositionIDs}(L)$
2: $pos\_emb \leftarrow \text{PositionEmbedding}(pos\_ids)$
3: $input\_emb \leftarrow \text{LayerNorm}(\text{Dropout}(E + pos\_emb))$
4: $extended\_attention\_mask \leftarrow \text{GetBidirectionalMask}(L)$
5: $transformer\_output \leftarrow \text{TransformerEncoder}(input\_emb, extended\_attention\_mask)$
6: $ffn\_output \leftarrow \text{OutputFFN}(transformer\_output)$
7: $gelu\_output \leftarrow \text{GELU}(ffn\_output)$
8: $output \leftarrow \text{OutputLayerNorm}(gelu\_output)$
9: $last\_hidden \leftarrow \text{GatherLastValidIndex}(output, seq\_len - 1)$
10: $target\_emb \leftarrow \text{ItemEmbedding}(t)$
11: $score \leftarrow \sum(last\_hidden \odot target\_emb) + bias[t]$
12: **return** *score*

E.2 Integrated Gradients Implementation

Algorithm 11 Integrated Gradients Attribution

Require: Item embeddings E, baseline E_0, steps n, target t
Ensure: Attribution scores A

$methods \leftarrow [\text{'}gausslegendre\text{'}, \text{'}riemann\_middle\text{'}, \text{'}riemann\_left\text{'}]$
$best\_attrs \leftarrow None$, $min\_error \leftarrow \infty$
for each *method* in *methods* **do**
  $attrs, \delta \leftarrow$ IntegratedGradients.attribute(E, E_0, n, *method*)
  $error \leftarrow |\delta|$
  if $error < min\_error$ **then**
    $min\_error \leftarrow error$
    $best\_attrs \leftarrow attrs$
  end if
  if $error < 0.05$ **then**
    break
  end if
end for
$A \leftarrow \sum_d best\_attrs[:, :, d]$
return A

E.3 GradientSHAP Implementation

Algorithm 12 GradientSHAP Attribution

Require: Item embeddings E, number of samples n, noise level σ
Ensure: Attribution scores A

$all\_embeddings \leftarrow$ GetAllItemEmbeddings()
$n\_baselines \leftarrow \min(10, |all\_embeddings|)$
$baseline\_indices \leftarrow$ RandomSample($|all\_embeddings|$, $n\_baselines$)
$baselines \leftarrow all\_embeddings[baseline\_indices]$
$baselines \leftarrow$ ExpandDimensions($baselines$, $E.shape[1]$)
$attrs \leftarrow$ GradientShap.attribute(E, $baselines$, $n\_samples = n$, $stdevs = \sigma$)
$A \leftarrow \sum_d attrs[:, :, d]$
return A

E.4 Occlusion Attribution

Algorithm 13 Occlusion Attribution

Require: Item embeddings E, baseline E_0
Ensure: Attribution scores A

$emb\_dim \leftarrow E.shape[2]$
$window\_shape \leftarrow (1, emb\_dim)$
$strides \leftarrow (1, emb\_dim)$
$occlusion\_params \leftarrow \{$
  $sliding\_window\_shapes : window\_shape,$
  $strides : strides,$
  $baselines : E_0,$
  $perturbations\_per\_eval : 1$
$\}$
$attrs \leftarrow$ Occlusion.attribute(E, $occlusion\_params$)
$A \leftarrow \sum_d attrs[:, :, d]$
return A

E.5 Robust Attribution with Fallback

Algorithm 14 Robust Attribution with Fallback Implementation

Require: Method name, embeddings E, target t
Ensure: Attribution scores A

try:
  $A \leftarrow$ CaptumMethod.attribute(E, parameters)
catch Exception e:
  Print("Captum method failed: " + e)
  if method == "integratedgradients":
    $A \leftarrow$ ManualIntegratedGradients(E, $steps = 10$)
  elif method == "occlusion":
    $A \leftarrow$ ManualOcclusion(E)
  else:
    $A \leftarrow$ SimpleGradientInput(E)
return A

E.6 Device and Memory Management

Algorithm 15 Device-Aware Attribution Computation

Require: Sequences, target items, device
Ensure: Device-consistent attributions
device ← DetermineOptimalDevice()
Move all tensors to *device*
Clear GPU memory cache
for each sequence s in batch **do**
    Process s individually to minimize memory footprint
    attrs ← ComputeAttribution(s, *device*)
    Store *attrs* and clear intermediate tensors
end for
Concatenate all attribution results
return device-consistent attributions

E.7 Attribution Quality Validation

Algorithm 16 Attribution Quality Assessment

Require: Attributions A, original score s_{orig}, baseline score s_{base}
Ensure: Quality metrics and validation status
$completeness\_error \leftarrow |s_{orig} - s_{base} - \sum A|$
$sensitivity\_check \leftarrow$ VerifyNonZeroAttribution(A)
$implementation\_invariance \leftarrow$ CompareMultipleRuns(A)
if $completeness\_error > 0.1$ **then**
    Print("Warning: Poor completeness - consider more integration steps")
end if
if not *sensitivity_check* **then**
    Print("Warning: Zero attributions for non-zero input")
end if
quality_score ← CombineValidationMetrics()
return quality metrics

E.8 Baseline Selection Strategy

Algorithm 17 Adaptive Baseline Selection

Require: Attribution method, embedding matrix E_{all}, sequence S
Ensure: Appropriate baseline B
if method == "integratedgradients" **then**
  $B \leftarrow$ ZeroTensor($S.shape$)
else if method == "gradientshap" **then**
  $n_{baselines} \leftarrow \min(10, |E_{all}|)$
  $random\_indices \leftarrow$ RandomSample($|E_{all}|$, $n_{baselines}$)
  $B \leftarrow E_{all}[random\_indices]$
  Expand B to match sequence dimensions
else if method == "occlusion" **then**
  $B \leftarrow$ MeanEmbedding(E_{all}) **or** ZeroTensor($S.shape$)
end if
Validate baseline produces reasonable scores
return B

F Jaccard Similarity Implementation Details

This appendix provides detailed algorithmic implementations for the memory-efficient Jaccard similarity explainer.

F.1 Memory-Efficient Explainer Initialization

Algorithm 18 Memory-Efficient Jaccard Explainer Initialization

Require: Recommender model, cache size limit C
Ensure: Initialized explainer with item-users mapping
$item\_users \leftarrow \{\}$ {Mapping from items to user sets}
$similarity\_cache \leftarrow \{\}$ {LRU cache for computed similarities}
$cache\_access\_count \leftarrow \{\}$ {Access frequency tracking}
for each interaction (u, i) in training data **do**
  $item\_users[i]$.add(u)
end for
return initialized explainer

F.2 Item-Users Mapping Construction

Algorithm 19 Item-Users Mapping Construction

Require: Training interactions $\mathcal{D} = \{(u, i)\}$
Ensure: Item-users mapping M
Initialize $M \leftarrow \{\}$
for each interaction $(user, item)$ in $\mathcal{D}$ **do**
  $M[item] \leftarrow M[item] \cup \{user\}$
end for
return M

F.3 On-Demand Jaccard Similarity Computation

Algorithm 20 On-Demand Jaccard Similarity Computation

Require: Item IDs i_1, i_2, item-users mapping M, cache C
Ensure: Jaccard similarity score $s \in [0, 1]$
$cache\_key \leftarrow (\min(i_1, i_2), \max(i_1, i_2))$ {Order-independent key}
if $cache\_key \in C$ **then**
  Update access count for $cache\_key$
  return $C[cache\_key]$
end if
$users_1 \leftarrow M[i_1]$, $users_2 \leftarrow M[i_2]$
if $users_1 = \emptyset$ or $users_2 = \emptyset$ **then**
  $similarity \leftarrow 0.0$
else
  $intersection \leftarrow |users_1 \cap users_2|$
  $union \leftarrow |users_1 \cup users_2|$
  $similarity \leftarrow \frac{intersection}{union}$ if $union > 0$ else 0.0
end if
Apply LRU eviction if cache is full
$C[cache\_key] \leftarrow similarity$
return $similarity$

F.4 LRU Cache Management

Algorithm 21 LRU Cache Management

Require: Current cache C, access counts A, max size max_size
Ensure: Updated cache with space for new entry
if $|C| \geq max\_size$ **then**
  $lru\_key \leftarrow \arg\min_{k \in A} A[k]$ {Find least recently used}
  Remove lru_key from C and A
end if
return updated cache

F.5 Jaccard Attribution Generation

Algorithm 22 Jaccard Attribution Generation

Require: User sequence S, target item t
Ensure: Attribution dictionary A
$sequence\_items \leftarrow \{i : S[i] \neq 0\}$ {Non-zero items}
Initialize $A \leftarrow \{\}$
for each $item\_id$ in $sequence\_items$ **do**
  if $item\_id \neq t$ **then**
    $similarity \leftarrow \text{ComputeJaccardSimilarity}(item\_id, t)$
    $A[item\_id] \leftarrow similarity$
  else
    $A[item\_id] \leftarrow 1.0$ {Perfect self-similarity}
  end if
end for
return A

F.6 Memory-Efficient Evaluation Pipeline

Algorithm 23 Memory-Efficient Jaccard Evaluation

Require: Jaccard explainer, user sequence, target item, number of bins
Ensure: Evaluation metrics
Compute Jaccard similarities for sequence items on-demand
Sort items by similarity scores (most to least similar)
Initialize evaluation metrics for each bin
Apply cache statistics tracking
for each bin size **do**
    Create masked sequences based on similarity rankings
    Compute model predictions for masked sequences
    Calculate DEL, INS, P@K, and NDCG metrics
    Update cache access patterns
end for
Aggregate metrics across bins
return evaluation results with cache statistics

F.7 Cache Performance Optimization

Algorithm 24 Cache Performance Statistics

Require: Similarity cache C, access counts A
Ensure: Cache performance metrics
$cache\_size \leftarrow |C|$
$total\_accesses \leftarrow \sum_{k \in A} A[k]$
$cache\_hits \leftarrow |\{k : A[k] > 1\}|$
$hit\_rate \leftarrow \frac{cache\_hits}{\max(1, |A|)}$
return $\{size : cache\_size, hit\_rate : hit\_rate, total\_accesses : total\_accesses\}$

G Evaluation Framework Details

G.1 Counterfactual Evaluation Pipeline

Algorithm 25 Metric Computation Pipeline

Require: Perturbed sequences $\mathbf{x}_{pos}, \mathbf{x}_{neg}, \mathbf{x}_{ins}$, target item t
Ensure: Evaluation metrics
$scores_{pos}, scores_{neg}, scores_{ins} \leftarrow f_\theta(\mathbf{x}_{pos}), f_\theta(\mathbf{x}_{neg}), f_\theta(\mathbf{x}_{ins})$
Normalize: $\tilde{s}_{pos}, \tilde{s}_{neg}, \tilde{s}_{ins} \leftarrow \sigma(0.15 \cdot \text{clamp}(scores, -500, 500))$
Score metrics:
$DEL(m) \leftarrow \tilde{s}_{pos}[t]$, $INS(m) \leftarrow \tilde{s}_{ins}[t]$
Ranking metrics:
Mask history items from scores
$rank_{pos}, rank_{neg} \leftarrow \text{get\_rank}(scores'_{pos}[t]), \text{get\_rank}(scores'_{neg}[t])$
$POS\_P@K(m) \leftarrow \mathbb{1}[rank_{pos} \leq K]$
$NEG\_P@K(m) \leftarrow \mathbb{1}[rank_{neg} \leq K]$
$NDCG(m) \leftarrow \frac{1}{\log_2(1+rank_{pos})}$

G.2 Multi-User Aggregation

Algorithm 26 Cross-User Metric Aggregation

Require: User metrics $\{\mathcal{M}_u\}_{u=1}^{N}$, perturbation steps M
Ensure: Aggregated metrics
for all metric type $\mathcal{M} \in \{DEL, INS, POS\_P@K, NEG\_P@K, NDCG\}$ **do**
  $\overline{\mathcal{M}} \leftarrow \frac{1}{N}\sum_{u=1}^{N} \mathcal{M}_u$ {Compute mean across users}
  $AUC_{\mathcal{M}} \leftarrow \frac{1}{M+1}\sum_{m=0}^{M} \overline{\mathcal{M}}[m]$ {Area under curve}
end for

G.3 Validation Protocol

Algorithm 27 Implementation Validation

Require: Original sequence $\mathbf{x}_u$, empty sequence $\mathbf{x}_{empty}$
Ensure: Validation passed
Boundary validation:
Assert $\|f_\theta(\mathbf{x}_{pos}(0)) - f_\theta(\mathbf{x}_u)\|_2 < \epsilon$
Assert $\|f_\theta(\mathbf{x}_{neg}(0)) - f_\theta(\mathbf{x}_u)\|_2 < \epsilon$
Completeness validation:
Assert $\|f_\theta(\mathbf{x}_{pos}(M)) - f_\theta(\mathbf{x}_{empty})\|_2 < \epsilon$
Assert $\|f_\theta(\mathbf{x}_{neg}(M)) - f_\theta(\mathbf{x}_{empty})\|_2 < \epsilon$
Target masking validation:
Ensure target item $t \notin$ history_mask during ranking

H Additional Experimental Results

This appendix contains the complete experimental results for Amazon Movies and TV and Yelp2018 datasets across both SASRec and BERT4Rec architectures.

H.1 SASRec Results on Additional Datasets

Table 7.1: AUC values for explaining a SASRec recommender (Amazon_Movies_and_TV)

Metric	Explanation Methods				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.211	0.112	0.116	0.116	**0.103**
k=5	0.307	0.151	0.159	0.161	**0.131**
k=10	0.344	0.177	0.187	0.192	**0.157**
k=20	0.381	0.211	0.223	0.232	**0.196**
k=50	0.439	**0.269**	0.283	0.299	0.276
Top-k Evaluation NEG (Negative Cases)					
k=1	0.644	0.729	0.734	**0.739**	0.716
k=5	0.799	0.868	**0.869**	0.871	0.869
k=10	0.828	0.885	0.886	0.886	**0.889**
k=20	0.850	0.894	0.894	0.894	**0.900**
k=50	0.872	0.901	0.901	0.901	**0.907**
Additional Metrics					
DEL	0.496	**0.424**	0.429	0.434	0.564
INS	0.625	0.672	0.668	0.676	**0.740**
NDCG	0.354	**0.234**	0.241	0.245	0.235

Table 7.2: AUC values for explaining a SASRec recommender (yelp2018)

Metric	Explanation Methods				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.281	**0.146**	0.176	0.150	0.183
k=5	0.446	**0.242**	0.296	0.256	0.301
k=10	0.510	**0.303**	0.365	0.323	0.364
k=20	0.571	**0.373**	0.440	0.401	0.433
k=50	0.654	**0.479**	0.519	0.526	0.537
Top-k Evaluation NEG (Negative Cases)					
k=1	0.579	**0.674**	0.664	0.667	0.593
k=5	0.782	**0.860**	0.840	0.849	0.794
k=10	0.831	**0.886**	0.869	0.878	0.840
k=20	0.864	**0.900**	0.887	0.894	0.870
k=50	0.896	**0.912**	0.908	0.908	0.898
Additional Metrics					
DEL	0.544	**0.438**	0.469	0.457	0.485
INS	0.597	0.662	0.646	**0.671**	0.640
NDCG	0.461	**0.313**	0.353	0.326	0.357

H.2 BERT4Rec Results on Additional Datasets

Table 7.3: AUC values for explaining a BERT4Rec recommender (Amazon_Movies_and_TV)

Metric	Explanation Methods				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.244	0.157	0.159	0.156	**0.139**
k=5	0.397	0.239	0.243	0.249	**0.216**
k=10	0.462	0.281	0.286	0.301	**0.260**
k=20	0.524	0.324	0.331	0.357	**0.310**
k=50	0.601	0.386	0.394	0.437	**0.383**
Top-k Evaluation NEG (Negative Cases)					
k=1	0.536	0.630	**0.657**	0.655	0.637
k=5	0.751	0.817	**0.832**	0.831	0.826
k=10	0.802	0.851	**0.860**	0.858	0.858
k=20	0.836	0.872	**0.877**	0.875	0.877
k=50	0.865	0.887	**0.891**	0.888	0.891
Additional Metrics					
DEL	0.587	**0.529**	0.532	0.550	0.538
INS	0.640	0.652	0.651	0.658	**0.661**
NDCG	0.419	0.298	0.301	0.310	**0.284**

Table 7.4: AUC values for explaining a BERT4Rec recommender (yelp2018)

Metric	Explanation Methods				
	Jaccard	Integrated Gradient	Gradient SHAP	Occlusion	LIME
Top-k Evaluation POS (Positive Cases)					
k=1	0.322	0.216	0.227	0.201	**0.170**
k=5	0.526	0.345	0.358	0.346	**0.292**
k=10	0.595	0.397	0.411	0.412	**0.352**
k=20	0.654	0.445	0.459	0.478	**0.412**
k=50	0.718	0.508	0.520	0.564	**0.494**
Top-k Evaluation NEG (Negative Cases)					
k=1	0.579	0.618	0.596	**0.634**	0.632
k=5	0.807	0.824	0.812	**0.818**	0.824
k=10	0.852	**0.862**	0.854	0.853	0.859
k=20	0.879	**0.883**	0.879	0.874	0.881
k=50	0.898	**0.899**	0.897	0.892	0.897
Additional Metrics					
DEL	0.610	**0.564**	0.567	0.585	0.572
INS	0.643	0.647	0.644	**0.650**	0.652
NDCG	0.516	0.376	0.387	0.381	**0.340**

I Detailed Attribution Method Comparisons

This appendix provides comprehensive visual comparisons of all five explanation methods for User 1's attribution analysis, supporting the cross-method consensus find-

ings discussed in the main text.

I.1 Individual Method Attribution Visualizations

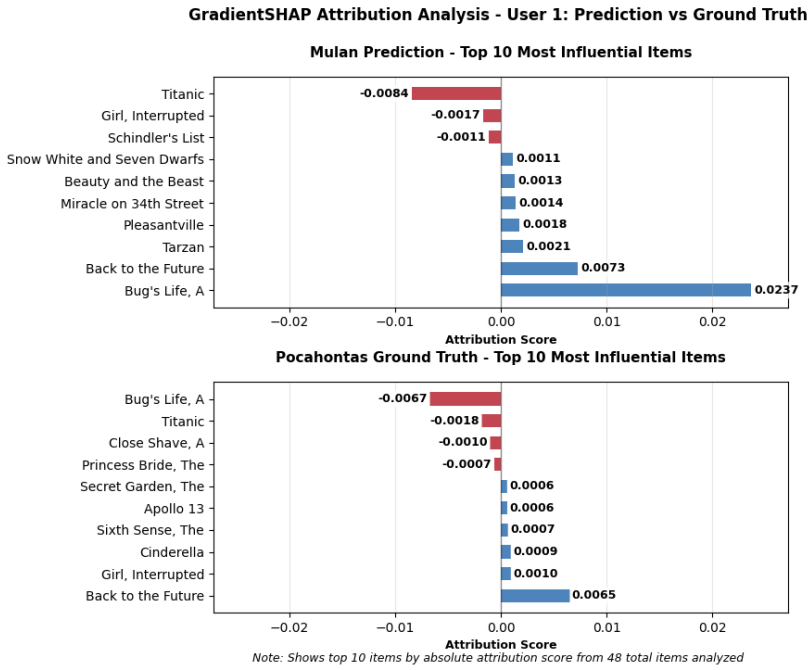

Figure 7.1: GradientSHAP attribution comparison for User 1 showing top 10 most influential items for Mulan prediction versus Pocahontas ground truth. Positive attributions (blue) support the recommendation while negative attributions (red) oppose it. Magnitude differences reveal prediction-specific reasoning patterns.

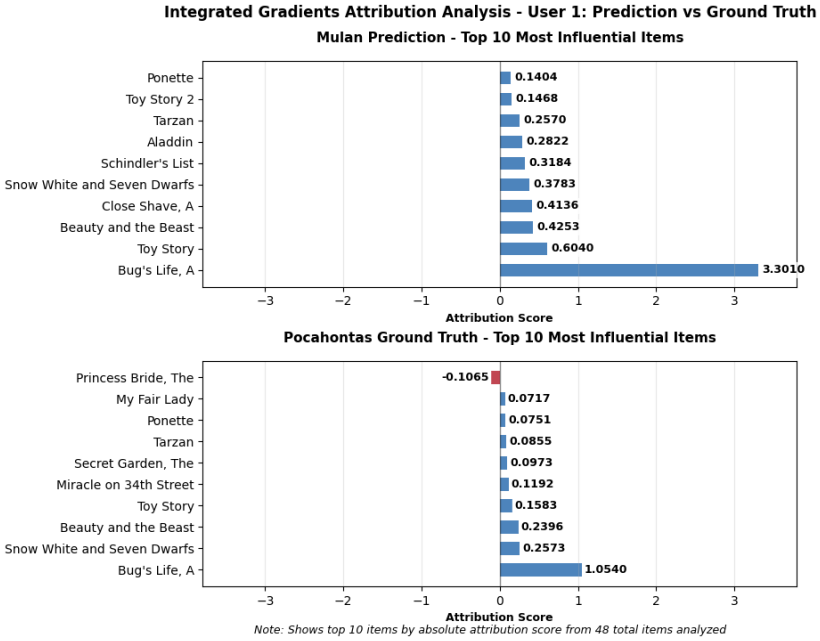

Figure 7.2: Integrated Gradients attribution comparison for User 1 showing top 10 most influential items for Mulan prediction versus Pocahontas ground truth. Positive attributions (blue) support the recommendation while negative attributions (red) oppose it. Magnitude differences reveal prediction-specific reasoning patterns.

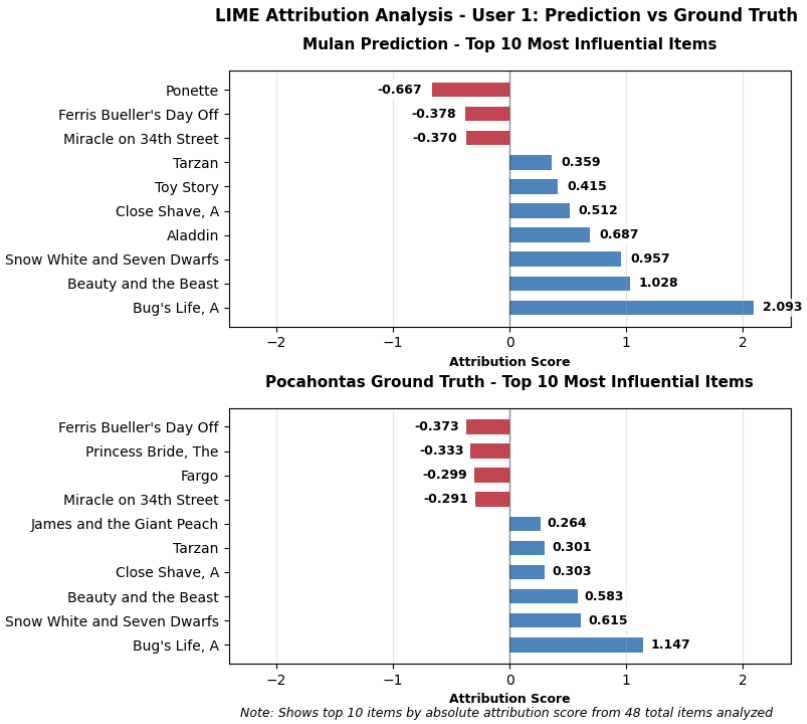

Figure 7.3: LIME attribution comparison for User 1 showing top 10 most influential items for Mulan prediction versus Pocahontas ground truth. Positive attributions (blue) support the recommendation while negative attributions (red) oppose it. Magnitude differences reveal prediction-specific reasoning patterns.

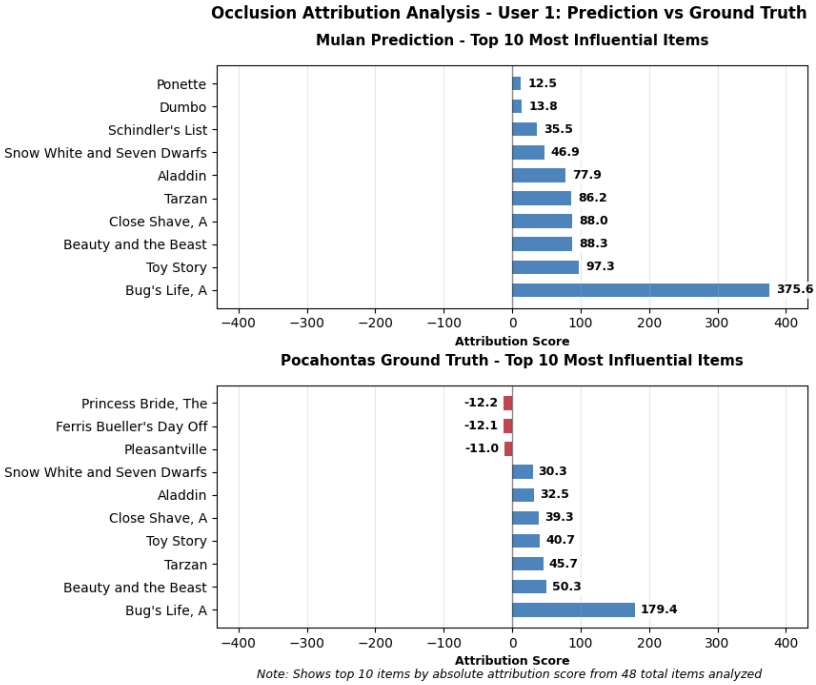

Figure 7.4: Occlusion attribution comparison for User 1 showing top 10 most influential items for Mulan prediction versus Pocahontas ground truth. Positive attributions (blue) support the recommendation while negative attributions (red) oppose it. Magnitude differences reveal prediction-specific reasoning patterns.

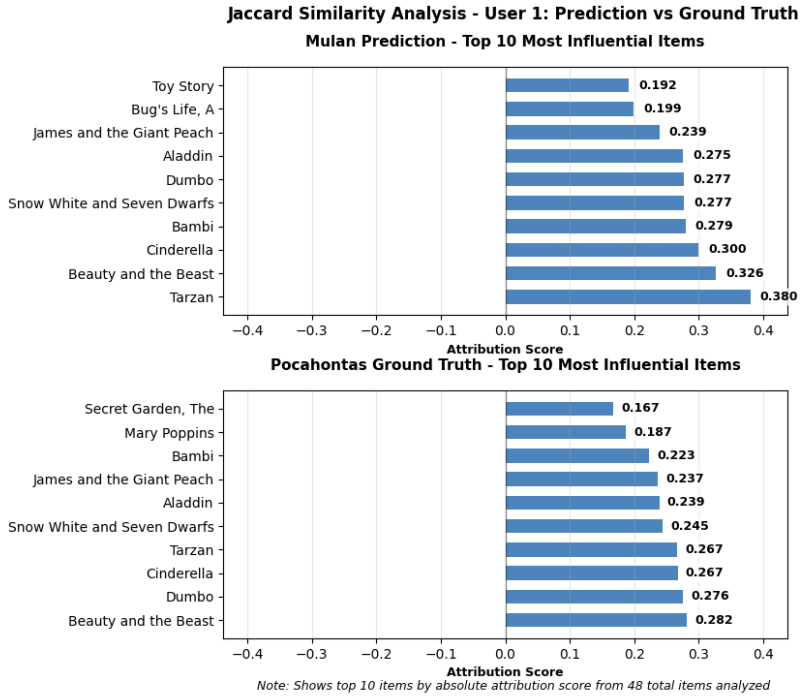

Figure 7.5: Jaccard similarity attribution comparison for User 1 showing top 10 most influential items for Mulan prediction versus Pocahontas ground truth. Positive attributions (blue) support the recommendation while negative attributions (red) oppose it. Magnitude differences reveal prediction-specific reasoning patterns.

Bibliography

[1] Tesfaye Boka, Zhendong Niu, and Rama Neupane. "A survey of sequential recommendation systems: Techniques, evaluation, and future directions". In: *Information Systems* 125 (July 2024), p. 102427. DOI: 10.1016/j.is.2024.102427.

[2] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Recommender Systems Handbook". In: vol. 1-35. Oct. 2010, pp. 1–35. ISBN: 978-0-387-85819-7. DOI: 10.1007/978-0-387-85820-3_1.

[3] Wang-Cheng Kang and Julian McAuley. *Self-Attentive Sequential Recommendation*. 2018. arXiv: 1808.09781 [cs.IR]. URL: https://arxiv.org/abs/1808.09781.

[4] Balázs Hidasi et al. *Session-based Recommendations with Recurrent Neural Networks*. 2016. arXiv: 1511.06939 [cs.LG]. URL: https://arxiv.org/abs/1511.06939.

[5] Fei Sun et al. "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, 1441–1450. ISBN: 9781450369763. DOI: 10.1145/3357384.3357895. URL: https://doi.org/10.1145/3357384.3357895.

[6] David Zibriczky. "Recommender Systems meet Finance: A literature review". In: June 2016. DOI: 10.13140/RG.2.1.1249.2405.

[7] Maryam Etemadi et al. "A systematic review of healthcare recommender systems: Open issues, challenges, and techniques". In: *Expert Systems with Applications* 213 (2023), p. 118823. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.118823. URL: https://www.sciencedirect.com/science/article/pii/S0957417422018413.

[8] Bryce Goodman and Seth Flaxman. "EU regulations on algorithmic decision-making and a "right to explanation"". In: *AI Magazine* 38 (June 2016). DOI: 10.1609/aimag.v38i3.2741.

[9] Finale Doshi-Velez and Been Kim. "Towards A Rigorous Science of Interpretable Machine Learning". In: *arXiv: Machine Learning* (2017). URL: https://api.semanticscholar.org/CorpusID:11319376.

[10] Andrea Ferrario and Michele Loi. "How Explainability Contributes to Trust in AI". In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '22. Seoul, Republic of Korea: Association for Computing Machinery, 2022, 1457–1466. ISBN: 9781450393522. DOI: 10.1145/3531146.3533202. URL: https://doi.org/10.1145/3531146.3533202.

[11] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. arXiv: 1602.04938 [cs.LG]. URL: https://arxiv.org/abs/1602.04938.

[12] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI]. URL: https://arxiv.org/abs/1705.07874.

[13] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG]. URL: https://arxiv.org/abs/1703.01365.

[14] Andreas Madsen, Siva Reddy, and Sarath Chandar. "Post-hoc Interpretability for Neural NLP: A Survey". In: *ACM Comput. Surv.* 55.8 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3546577. URL: https://doi.org/10.1145/3546577.

[15] Sandra Wachter, Brent Mittelstadt, and Chris Russell. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI]. URL: https://arxiv.org/abs/1711.00399.

[16] Yehuda Koren, Steffen Rendle, and Robert Bell. "Advances in Collaborative Filtering". In: Nov. 2021, pp. 91–142. ISBN: 978-1-0716-2196-7. DOI: 10.1007/978-1-0716-2197-4_3.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems. IEEE, Computer Journal, 42(8), 30-37". In: *Computer* 42 (Sept. 2009), pp. 30 –37. DOI: 10.1109/MC.2009.263.

[18] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[19] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), 1929–1958. ISSN: 1532-4435.

[20] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable AI: A Review of Machine Learning Interpretability Methods". In: *Entropy* 23.1 (2021). ISSN: 1099-4300. DOI: 10.3390/e23010018. URL: https://www.mdpi.com/1099-4300/23/1/18.

[21] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV]. URL: https://arxiv.org/abs/1311.2901.

[22] Khanh Hiep Tran, Azin Ghazimatin, and Rishiraj Saha Roy. "Counterfactual Explanations for Neural Recommenders". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. ACM, July 2021, 1627–1631. DOI: 10.1145/3404835.3463005. URL: http://dx.doi.org/10.1145/3404835.3463005.

[23] Oren Barkan et al. "A Counterfactual Framework for Learning and Evaluating Explanations for Recommender Systems". In: *Proceedings of the ACM Web Conference 2024*. WWW '24. Singapore, Singapore: Association for Computing Machinery, 2024, 3723–3733. ISBN: 9798400701719. DOI: 10.1145/3589334.3645560. URL: https://doi.org/10.1145/3589334.3645560.

[24] Sujoy Bag, Sri Kumar, and Manoj Tiwari. "An efficient recommendation generation using relevant Jaccard similarity". In: *Information Sciences* 483 (May 2019), pp. 53–64. DOI: 10.1016/j.ins.2019.01.023.

[25] Sarthak Jain and Byron C. Wallace. *Attention is not Explanation*. 2019. arXiv: 1902.10186 [cs.CL]. URL: https://arxiv.org/abs/1902.10186.

[26] Sarah Wiegreffe and Yuval Pinter. "Attention is not not Explanation". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 11–20. DOI: 10.18653/v1/D19-1002. URL: https://aclanthology.org/D19-1002/.

[27] Alessandro B. Melchiorre et al. "ProtoMF: Prototype-based Matrix Factorization for Effective and Explainable Recommendations". In: *Proceedings of the 16th ACM Conference on Recommender Systems*. RecSys '22. Seattle, WA, USA: Association for Computing Machinery, 2022, 246–256. ISBN: 9781450392785. DOI: 10.1145/3523227.3546756. URL: https://doi.org/10.1145/3523227.3546756.

[28] Lloyd S Shapley. "A Value for n-Person Games". In: *Contributions to the Theory of Games II*. Ed. by Harold W. Kuhn and Albert W. Tucker. Princeton: Princeton University Press, 1953, pp. 307–317.

[29] Jinfeng Zhong and Elsa Negre. "Shap-enhanced counterfactual explanations for recommendations". In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. SAC '22. Virtual Event: Association for Computing Machinery, 2022, 1365–1372. ISBN: 9781450387132. DOI: 10.1145/3477314.3507029. URL: https://doi.org/10.1145/3477314.3507029.

[30] Caio Nóbrega and Leandro Marinho. "Towards explaining recommendations through local surrogate models". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, 1671–1678. ISBN: 9781450359337. DOI: 10.1145/3297280.3297443. URL: https://doi.org/10.1145/3297280.3297443.

[31] Avanti Shrikumar et al. *Not Just a Black Box: Learning Important Features Through Propagating Activation Differences*. 2017. arXiv: 1605.01713 [cs.LG]. URL: https://arxiv.org/abs/1605.01713.

[32] Alexander Binder et al. *Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers*. 2016. arXiv: 1604.00825 [cs.CV]. URL: https://arxiv.org/abs/1604.00825.

[33] Narine Kokhlikyan et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG]. URL: https://arxiv.org/abs/2009.07896.

[34] Wayne Xin Zhao et al. "RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, 4653–4664. ISBN: 9781450384469. DOI: 10.1145/3459637.3482016. URL: https://doi.org/10.1145/3459637.3482016.