MDPI

Article

Large Language Model Agents for Recommender Systems: Bridging Behavior and Semantics with Long-Short Term Interest Modeling

Yiming Cheng [1], Hao Feng [2], Yitong Ma [3], Qi Yang [1], Zhang Yudong [2] and Yi Yang [2,*]

[1] University of Chicago, Chicago, IL, USA; eaminchan@uchicago.edu (Y.C.); yangqi@uchicago.edu (Q.Y.)
[2] Tsinghua University, Beijing, China; yangyy@tsinghua.edu.cn (Y.Y.); zhangyd16@mails.tsinghua.edu.cn (Z.Y.); fh20@mails.tsinghua.edu.cn (H.F.)
[3] University of California, San Diego, La Jolla, CA, USA; yim030@ucsd.edu (Y.M.)
* Correspondence: yangyy@tsinghua.edu.cn (Y.Y.)

Abstract

A recommender system extracts user preferences from past interactions to suggest items, widely used in platforms like user-generated content, online shopping, and urban services. These systems aim to provide accurate recommendations, reduce user interaction burden, and enhance user experience while improving socio-economic benefits. Current technologies include content-based, behavior-based, and hybrid recommendations. Behavior-based technologies, like collaborative filtering, are mature due to extensive user-item interaction data but focus on behavior over intrinsic features, lacking comprehensive understanding. Advancements in natural language processing (NLP), text vectorization, and large language models (LLMs) have made content-based recommendations based on semantic understanding more prominent. This study presents an exploratory LLM-agent-based hybrid recommender system framework, building upon a multi-armed bandit (MAB) approach, which enables semantic understanding of item content and integrates both short-term and long-term user interaction behaviors for recommendation. Specifically, it optimizes collaborative filtering models to capture long-term and short-term interests, and uses an agentic LLM memory and reasoning component to identify preference shifts and schedule appropriate recommendation modules. It further vectorizes content using NLP and LLMs to generate retrieval terms that enhance candidate recall, and aligns LLM-optimized content recommendations with user behavior to integrate both aspects. This hybrid system achieved a recall rate improvement of up to 26.63% in behavior recommendation and an NDCG improvement of 1.48x in diversity-focused recommendation tasks on the MovieLens dataset. However, this work represents an initial exploration in this emerging research direction, and more comprehensive comparisons with state-of-the-art methods are needed in future work to fully assess the framework's effectiveness.

Keywords: recommendation system; large language model; agent; machine learning; semantic understanding; natural language processing

Academic Editor: Dr. Weibin Wu

Received:
Revised:
Accepted:
Published:

1. Introduction

1.1. Research Background

In the design of modern information retrieval systems, the scale of retrievable items, the number of users, and the volume of interaction demands have grown rapidly, making *accurate identification of user preferences* and *fast, precise delivery of retrieval results* key

optimization objectives. As early as the third century BCE, humans had already begun to design information retrieval systems for the ever-growing body of information. For instance, the first international library—the Library of Alexandria in ancient Greece—which collected the vast majority of books or copies available in the world at that time, had already developed a rudimentary system for information categorization and retrieval [1]. With the sharp increase in interaction scale and update frequency, such passive retrieval systems have gradually become insufficient to meet their original design goals. Therefore, to reduce users' search burden while still achieving efficient and accurate retrieval, R. Armstrong [2] and colleagues first proposed an *active* information retrieval system in 1995, which can be regarded as the early prototype of modern recommender systems. With the development of mobile Internet and the further rise in user–item interaction frequency, industry has an urgent demand for more efficient and more accurate recommendation technologies. For example, Meituan's 2024 financial report [3] states that its food delivery business generated 21.89 billion user–item interactions in the year, representing a year-on-year increase of 23.9%. The growth in information volume and the rapid evolution of interaction categories also pose substantial and novel challenges for recommender system design.

To better cope with the computational load induced by interaction data overload, to more precisely infer user intent, and to generate recommendations that are both efficient and faithful to real user behaviors, researchers have increasingly focused on *recommender systems*. The goal is to mine large-scale behavioral data—such as user ratings and interaction sequences—to learn representations of user preferences and to predict future interactions. For example, on movie platforms, one can predict the ratings that a user might give to unseen movies based on historical ratings, and then generate a ranked recommendation list [4] to help users discover preferred items more efficiently. In the digitization of urban life, interactions between users and POIs (Points of Interest)—e.g., residents' extensive behaviors related to food, clothing, housing, and transportation—can be leveraged to forecast future activities, thereby improving convenience in city living.

These applications heavily rely on the quality and quantity of historical interaction data, aiming to produce recommendation results that are close to users' true future behaviors. The abundance of interaction logs in the mobile Internet era provides a data foundation for behavior-driven predictive techniques. Meanwhile, with the rapid progress of natural language processing, recommendation approaches based on semantic understanding of content have received increasing attention in recent years. Motivated by this trend, this work proposes a hybrid recommender system framework that integrates content and behavior via large language models (LLMs), along with an implementation of the framework.

1.2. Contributions and Paper Organization

This work explores how to integrate LLM-based semantic reasoning with conventional recommendation pipelines that model user interactions and item content. The main contributions are:

- We propose an LLM-agent-based hybrid recommendation framework that combines semantic understanding with interaction modeling, supported by an explicit memory structure to represent and update user intent and context.
- We develop a long–short-term adaptive interaction-based recommender that reweights historical interactions and uses LLM-assisted model scheduling to better handle preference shifts.
- We design a content-based recommendation component with threshold filtering and regular-expression matching to reduce retrieval drift under semantic vectorization.

- We introduce a behavior–content alignment strategy for hybrid-intent recommendation and validate the framework empirically on MovieLens.

The remainder of the manuscript is organized as follows. We review related work and foundational methods in the Related Work section. We then present the LLM-agent-based hybrid system design, followed by experimental evaluation. Finally, we conclude with a discussion of limitations and future directions.

1.3. Research Challenges

As discussed in the above survey, a variety of strategies have been developed for recommender systems, including interaction-based, content-based, and more recent approaches based on LLMs and hybrid combinations. Nevertheless, effectively integrating LLMs into existing recommender system frameworks remains a frontier problem. In practice, each paradigm has its limitations: interaction-based methods depend on abundant prior interactions and suffer from cold start; content-based methods are constrained by feature extraction quality and generalization; LLM-based recommenders are limited by compute requirements and remote inference costs, making it difficult for them to serve as the sole backbone of a recommender system. These observations motivate a balanced architecture that incorporates LLMs, integrates the strengths of existing systems, and enables LLMs to contribute at critical points with reasonable overhead.

To this end, this work designs an LLM-based hybrid recommender framework and optimizes different components using LLMs. The key challenges can be summarized as follows:

First, interaction-based models lack personalized corrections with respect to the recommendation objects. Users exhibit both long-term and short-term interest shifts, while model invocation is typically uniform, which can cause substantial errors for certain users. Since interaction-based models do not explicitly understand content, identifying interest changes incurs additional overhead. Given the massive scale of interaction matrices in modern recommender systems, this can lead to significant computational costs.

Second, content-based recommender systems can better extract user and item features, but without explicit behavioral signals, they may fail to respond to short-lived spikes in large-scale interactions triggered by trending events, leading to recommendation bias. Moreover, if the recall set is too large, per-query computation increases; since fine-grained ranking is ultimately required, substantial unnecessary computation may be wasted, degrading overall system performance.

Third, the role and paradigm of LLMs in recommender workflows remain underspecified. Recommender systems are large engineering systems that implement end-to-end information retrieval functionalities. After incorporating LLMs, one must maximize their benefits in semantic understanding and feature extraction while avoiding significant increases in compute overhead and system complexity, and also align interaction-based and content-based subsystems. This requires a more general and cost-performance balanced framework adaptable across datasets.

1.4. Research Methodology

Building on prior work (reviewed in the Related Work section) and the challenges identified above, this work proposes an LLM-based hybrid recommender system framework that integrates behavior-based and content-based recommender systems, strengthens semantic understanding and feature extraction, and improves overall interaction prediction. The methodology includes the following components:

First, we design a long-/short-term interest change detection and adaptive recommendation module based on user interactions and LLM agents. On top of classical matrix

factorization, we add a linear layer with adjustable weights to manually control the attention range, enabling adaptation to different long-/short-term recommendation scenarios. We further design an agent memory update mechanism to more accurately detect interest changes, quantify their magnitude to some extent, and make model orchestration decisions, thereby addressing the lack of content understanding and the high cost of frequent model updates in traditional behavior-based recommenders.

Second, building on the GPT4Rec framework proposed by Li et al. [5], we correct certain vectorization biases and similarity estimation errors caused by overly long and noisy content. We then propose a regularization-threshold-based Query–Search framework for feature enhancement in content-based recommendation. Leveraging LLMs' semantic understanding and generalization, this module predicts future query terms and generates coarse-ranking results efficiently for downstream refinement, reducing wasted computation in traditional pipelines.

Third, we propose a complete LLM-based hybrid alignment framework. By anchoring the recalled results from the content-based recommender and using LLM-driven semantic understanding together with an interaction-based long-/short-term adaptive network for correction, the final results can jointly reflect behavioral signals and content understanding while reducing LLM resource overhead. This framework provides a practical approach to integrating LLMs into existing recommender systems and offers strong extensibility and generalization potential for future work.

2. Related Work

2.1. Taxonomy and Representative Paradigms

Recommender systems can be broadly categorized into interaction-based recommender systems (Interaction-Based RS), content-based recommender systems (Content-Based RS), and hybrid recommender systems (Hybrid RS) [6]. With the release of large language models such as ChatGPT, their strong natural language understanding and generalization over prior knowledge have further enabled an emerging line of agent-based recommender systems built upon LLM technologies.

Interaction-based methods rely on large-scale user–item interaction records and explicit feedback (e.g., ratings). A representative class is collaborative filtering (CF) [7], which leverages similarity across users and/or items. However, such methods depend critically on the amount and quality of historical interactions, and typically face severe cold-start issues in early deployment and in cross-domain settings. Content-based methods focus on feature extraction and vectorization of users/items as content, deriving similarity from representations. This alleviates heavy dependence on interaction logs and can generalize to new users/items, but performance is sensitive to representation quality. Hybrid systems seek to reconcile these complementary strengths via alignment and fusion strategies (e.g., weighted combination, switching, cascaded or mixed integration) [6].

2.2. Collaborative Filtering (CF)

Resnick [7] and colleagues were among the first to introduce collaborative filtering into recommender systems. CF is motivated by the intuition that similar users tend to prefer similar items, and that similar items tend to attract similar users. Common CF variants include user-based CF [8], item-based CF [9], and model-based CF [10].

User-based CF. Given a user–item rating matrix, user-based CF represents each user's ratings as a vector and computes user–user similarity. A widely used similarity measure is the Pearson correlation coefficient [11]. Specifically, the similarity between users U_a and U_b is:

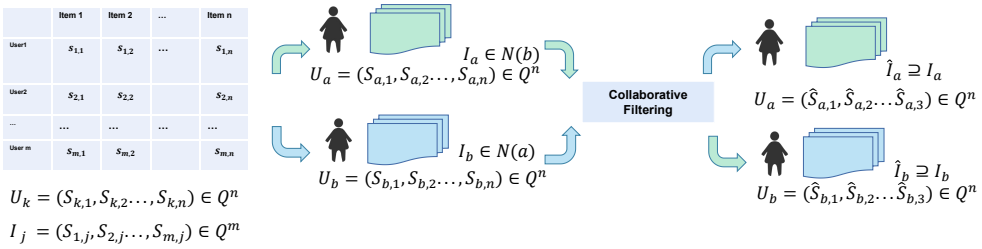

Figure 1. Illustration of collaborative filtering (user-based vs. item-based views).

$$\text{sim}(U_a, U_b) = \frac{\sum_{i \in I_a \cap I_b} (S_{a,i} - \bar{S}_a)(S_{b,i} - \bar{S}_b)}{\sqrt{\sum_{i \in I_a \cap I_b} (S_{a,i} - \bar{S}_a)^2 \sum_{i \in I_a \cap I_b} (S_{b,i} - \bar{S}_b)^2}} \tag{1}$$

where $S_{a,i}$ and $S_{b,i}$ denote ratings of users U_a and U_b on item i, $\bar{S}_a$ and $\bar{S}_b$ are average ratings, and $I_a \cap I_b$ is the set of co-rated items. A predicted rating for user U_a on item i_j can be computed as:

$$\hat{S}_{a,j} = \bar{S}_a + \frac{\sum_{b \in N(a)} \text{sim}(U_a, U_b) \cdot (S_{b,j} - \bar{S}_b)}{\sum_{b \in N(a)} |\text{sim}(U_a, U_b)|} \tag{2}$$

where $N(a)$ is the neighborhood of users most similar to U_a.

Item-based CF. Item-based CF is symmetric in spirit: it represents items as vectors over user interactions and computes item–item similarity. In practice, user cardinality is often much larger than item cardinality; item-based CF reduces the number of similarity computations while increasing vector dimensionality. The similarity between items I_i and I_j is:

$$\text{sim}(I_i, I_j) = \frac{\sum_{u \in U_i \cap U_j} (S_{u,i} - \bar{S}_i)(S_{u,j} - \bar{S}_j)}{\sqrt{\sum_{u \in U_i \cap U_j} (S_{u,i} - \bar{S}_i)^2 \sum_{u \in U_i \cap U_j} (S_{u,j} - \bar{S}_j)^2}} \tag{3}$$

and the predicted rating of user U_u on item I_i is:

$$\hat{S}_{u,i} = \frac{\sum_{j \in N(i)} \text{sim}(I_i, I_j) \cdot S_{u,j}}{\sum_{j \in N(i)} |\text{sim}(I_i, I_j)|} \tag{4}$$

2.3. LLM- and Agent-Based Recommender Systems

LLMs have enabled stronger semantic understanding and long-sequence modeling, and have thus been explored in recommender systems for representation learning, intent inference, candidate reranking, and workflow orchestration [12]. Transformer-based architectures [13] support sequence modeling over interaction histories, while LLMs' semantic capabilities can help mitigate cold-start issues when interaction data is sparse. Representative systems include BERT4Rec [14], Transformers4Rec [15], and GPT4Rec [5].

2.4. Interaction-Based Recommender System Algorithms

Building upon the overview of recommender system techniques introduced in Chapter 1, this chapter selects representative interaction-based and content-based algorithms that align with the characteristics of large language model (LLM) technologies. We briefly review important prior work and the core principles behind these methods. Specifically, we first introduce problem formulations for interaction-based recommendation and foun-

dational, widely used matrix-factorization-based training approaches. We then review common content vectorization techniques and similarity-based retrieval methods.

2.4.1. Modeling Interaction-Based Recommender Systems

For interaction-based recommender systems, the most critical element is to extract statistical regularities from large-scale user interaction behaviors, while paying comparatively less attention to the user and the item content themselves. In modern recommender systems, extracting interaction features via matrix factorization is motivated by several common assumptions and empirical observations: (1) *Large-scale interaction data are available*. Such data are often massive and provide the foundation for designing interaction-based algorithms. Interaction signals are also diverse. The most common signal is explicit ratings, which we primarily adopt in this work. Other signals include clicks and purchases; different interaction types can be assigned different strengths (e.g., clicks as weak signals and purchases as strong signals) and mapped onto a unified rating scale if needed. (2) *The interaction matrix is extremely high-dimensional and rapidly evolving*. The interaction matrix is formed by the Cartesian product of users and items, with dimensionality equal to the product of the number of users and the number of items. In modern systems, both users and items are large-scale and continuously updated, resulting in matrices that are high-dimensional and change quickly. Under this setting, factoring the original high-dimensional matrix into lower-dimensional user and item vectors can dramatically reduce overall storage and update costs while preserving desired modeling fidelity. (3) *The interaction matrix is highly sparse*. In practice, each user interacts with only a small fraction of all items, and most items are interacted with by a small subset of users. Moreover, recommender systems commonly exhibit a pronounced long-tail effect [16,17], where a small number of head items account for the majority of interactions, while the vast majority of items collectively contribute only a small portion. Consequently, although the matrix is large, it is extremely sparse [18], and most entries are missing. Performing feature extraction directly on the sparse matrix is inefficient; representing users and items as low-dimensional vectors improves both storage and computation efficiency. (4) *User preferences exhibit latent stability over time*. Behavior-based methods typically assume that user interactions are driven by latent preferences that remain stable to some extent over time. This latent structure can be inferred from historical interactions and leveraged to predict future behaviors.

Under these assumptions, an interaction-based recommender system can be formulated as follows. Given a user set $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and an item set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, the interaction state space is represented by a matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, where $\mathbf{R}_{ui}$ denotes the rating of user u for item i. If user u has not rated item i, then $\mathbf{R}_{ui}$ is missing. The goal is to predict a user's ratings on a held-out subset of items given observed ratings on a subset $\{I_1, I_2, \ldots, I_k\}$, i.e., to infer $\{I_{k+1}, \ldots, I_n\}$ using the first k observations, and then recommend items the user is likely to prefer. To evaluate recommendation quality, predicted values $\{I'_{k+1}, \ldots, I'_n\}$ can be compared with ground-truth values $\{I_{k+1}, \ldots, I_n\}$; higher similarity indicates better performance.

Although such an evaluation protocol is widely used, it has notable limitations. Because the recommender system intervenes in and alters the interaction sequence, it is difficult to disentangle the causal effect of exposure from intrinsic user preferences; moreover, collecting real-time logged bandit feedback at scale is challenging. These issues are expected to be better addressed in real-world deployment settings and with richer online datasets in future work.

2.4.2. Matrix-Factorization-Based Machine Learning Methods

Among interaction-based recommendation algorithms, a widely used approach is based on matrix singular value decomposition. We first decompose a rating matrix R into three matrices: a user feature matrix U, a singular value matrix Σ, and an item feature matrix V:

$$R \approx U\Sigma V^T \tag{5}$$

In real-world systems where dimensionality is very large, we typically perform dimensionality reduction by keeping the top k singular values and vectors, yielding a truncated interaction matrix $\hat{R}$, which is used for recommendation:

$$R \approx U_k \Sigma_k V_k^T \triangleq \hat{R} \tag{6}$$

To improve prediction accuracy, we commonly minimize the following regularized loss:

$$\mathcal{L} = \sum_{(u,i)\in\mathcal{K}} (R_{ui} - (U_k\Sigma_k V_k^T)_{ui})^2 + \lambda(\|U_k\|^2 + \|V_k\|^2) \tag{7}$$

where $\mathcal{K}$ is the set of observed ratings, λ is a regularization parameter, and $\|\cdot\|^2$ denotes the Frobenius norm for regularization to mitigate overfitting.

During learning, we initialize U_k and V_k randomly and optimize $\mathcal{L}$ via gradient descent [19]. Let η denote the learning rate; gradients are computed and parameters are updated as:

$$U_k' = U_k - \eta\frac{\partial\mathcal{L}}{\partial U_k} = U_k - 2\sum_{(u,i)\in\mathcal{K}} (R_{ui} - (U_k\Sigma_k V_k^T)_{ui})V_k\Sigma_k + 2\lambda U_k \tag{8}$$

$$V_k' = V_k - \eta\frac{\partial\mathcal{L}}{\partial V_k} = V_k - 2\sum_{(u,i)\in\mathcal{K}} (R_{ui} - (U_k\Sigma_k V_k^T)_{ui})\Sigma_k^T U_k^T + 2\lambda V_k \tag{9}$$

We split the observed dataset $\mathcal{K}$ into a training set $\mathcal{K}_{\text{train}}$, a validation set $\mathcal{K}_{\text{val}}$, and a test set $\mathcal{K}_{\text{test}}$:

$$\mathcal{K} = \mathcal{K}_{\text{train}} \cup \mathcal{K}_{\text{val}} \cup \mathcal{K}_{\text{test}} \tag{10}$$

We define training, validation, and test errors [20] using mean squared error (MSE):

$$\text{MSE}_{\text{train}} = \frac{1}{|\mathcal{K}_{\text{train}}|}\sum_{(u,i)\in\mathcal{K}_{\text{train}}} (R_{ui} - \hat{R}_{ui})^2 \tag{11}$$

$$\text{MSE}_{\text{val}} = \frac{1}{|\mathcal{K}_{\text{val}}|}\sum_{(u,i)\in\mathcal{K}_{\text{val}}} (R_{ui} - \hat{R}_{ui})^2 \tag{12}$$

$$\text{MSE}_{\text{test}} = \frac{1}{|\mathcal{K}_{\text{test}}|}\sum_{(u,i)\in\mathcal{K}_{\text{test}}} (R_{ui} - \hat{R}_{ui})^2 \tag{13}$$

The overall procedure can be summarized by the following pseudocode.

2.5. Content-Based Recommender System Algorithms

In contrast to interaction-based methods, content-based recommender systems do not treat interaction logs as the primary behavioral signal for optimizing an objective from a purely statistical perspective. Instead, they vectorize users and items through semantic analysis. Therefore, a reasonable feature extraction strategy and high-quality embeddings are essential, as embedding quality directly determines recommendation performance. This section introduces three vectorization methods used for optimization in subsequent experiments.

Algorithm 1 SVD-Based Recommender System Algorithm (Pseudocode)

Input: user set U, item set I, rating matrix R, rank k, learning rate η, regularization λ, epochs num_epochs
Output: predicted rating matrix $\hat{R}$
Initialize $U_k \in \mathbb{R}^{m \times k}$ and $V_k \in \mathbb{R}^{n \times k}$ randomly
for epoch = 1 to num_epochs **do**
  for each observed (u, i) in R **do**
    compute error $e_{ui} = R_{ui} - (U_k \Sigma_k V_k^T)_{ui}$
    update $U_k[u,:] \leftarrow U_k[u,:] + \eta(e_{ui} V_k[i,:] - \lambda U_k[u,:])$
    update $V_k[i,:] \leftarrow V_k[i,:] + \eta(e_{ui} U_k[u,:] - \lambda V_k[i,:])$
  end for
end for
Compute $\hat{R} = U_k \Sigma_k V_k^T$
return $\hat{R}$

2.5.1. Content Vectorization Methods

Using the same notation as above, we describe three vectorization methods and their applications to content-based recommendation.

TF–IDF

TF–IDF (Term Frequency–Inverse Document Frequency) [21] is widely used in NLP to measure the importance of a token in a document. In recommender systems, it can be adapted to measure the importance of an item in a user's interaction history. TF–IDF consists of two components: term frequency (TF) and inverse document frequency (IDF); in recommendation, these correspond to item frequency and inverse user frequency.

Item frequency (TF) measures how often item i appears in user u's history:

$$\mathrm{TF}(i,u) = \frac{f_{i,u}}{\sum_{i' \in u} f_{i',u}} \tag{14}$$

where $f_{i,u}$ is the frequency of item i in user u's history, and the denominator is the total number of item occurrences for user u.

Inverse user frequency (IDF) measures the global importance of item i:

$$\mathrm{IDF}(i,\mathcal{U}) = \log\left(\frac{|\mathcal{U}|}{|\{u \in \mathcal{U} : i \in u\}|}\right) \tag{15}$$

where $|\mathcal{U}|$ is the total number of users, and the denominator counts the users whose histories contain item i.

TF–IDF combines the above as:

$$\text{TF-IDF}(i,u,\mathcal{U}) = \mathrm{TF}(i,u) \times \mathrm{IDF}(i,\mathcal{U}) \tag{16}$$

A higher TF–IDF score indicates higher importance of the item in the user's interaction record. Each item i can be represented as a TF–IDF vector $\mathbf{v}_i$, whose j-th component is:

$$\mathbf{v}_i[j] = \text{TF-IDF}(i, u_j, \mathcal{U}) \tag{17}$$

where u_j denotes the j-th user. These vectors can then be used for similarity computation and recommendation.

BERT

BERT (Bidirectional Encoder Representations from Transformers) [22] is widely used to obtain contextual text representations in NLP; in content-based recommendation, it can also be used to encode item semantics. As BERT is a large model, training from scratch is expensive; typical usage involves pretraining and fine-tuning. In this work, we primarily use pretrained BERT to validate the correctness of the overall framework. Vectorizing an item i with BERT typically involves: **Preprocessing** tokenizing and normalizing the item profile text Doc_i; and **Encoding** feeding Doc_i into pretrained BERT to obtain a contextual representation $\mathbf{v}_i$:

$$\mathbf{v}_i = \text{BERT}(\text{Doc}_i) \tag{18}$$

User preference modeling represents a user's preference vector $\mathbf{p}_u$ as the average (or weighted average) of item vectors in the user's interaction history. Let $\mathcal{I}_u$ be the set of items interacted by user u; then:

$$\mathbf{p}_u = \frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} \mathbf{v}_i \tag{19}$$

Given $\mathbf{p}_u$ and candidate item vectors $\mathbf{v}_i$, recommendations can be produced via similarity computation.

spaCy

spaCy is an open-source NLP library that provides tools for semantic processing and vectorization. Its typical pipeline and how it can be used for recommendation include: **Preprocessing** tokenizing profile text using a predefined vocabulary; **Part-of-speech tagging** assigning POS tags via statistical models and pretrained components; **Dependency parsing** building a dependency tree to identify syntactic relations and sentence structure; **Named entity recognition (NER)** identifying entities via pretrained models and rules; and **Word embeddings** mapping tokens w to fixed-length vectors $\mathbf{e}_w$ using pretrained embeddings such as GloVe [23] and fastText [24].

Similar to BERT-based representations, spaCy-based vectors can be used to construct user vectors $\mathbf{p}_u$ and candidate item vectors $\mathbf{v}_i$ for subsequent similarity computation and recommendation.

2.5.2. Similarity-Based Retrieval for Recommendation

With vector representations of users and items, recommendation can be performed by computing similarity scores and ranking. The following pseudocode illustrates the procedure:

Algorithm 2 Similarity-Ranking-Based Recommendation (Pseudocode)

Input: user preference vector $\mathbf{p}_u$, item vectors $\{\mathbf{v}_i\}$, top-N
Output: ranked recommendation list
for each candidate item i **do**
  compute similarity $\text{sim}(u,i) = \mathbf{p}_u \cdot \mathbf{v}_i$
end for
Sort candidates by similarity in descending order
return top-N items

2.6. Summary

This chapter reviewed the fundamental ideas and several widely used implementations of interaction-based and content-based recommender systems. These methods also serve as the foundations for the LLM-based hybrid recommender system framework proposed and implemented in Chapter 3. In the following, we will present how LLMs can

be used to optimize these baseline implementations and how to align and fuse interaction-based and content-based recommenders into a unified system.

3. Hybrid Recommender System Design with LLM Agents

3.1. Overview

In recent years, large language model (LLM) technologies [25] have advanced rapidly and have addressed a range of long-standing open problems in natural language processing. In this work, we primarily adopt GPT-3.5-turbo [26] as a pretrained LLM to provide an interface for semantic understanding. Leveraging their extensive pretrained knowledge and strong semantic modeling capabilities, LLMs offer powerful functional primitives for recommender systems, including semantic feature extraction, workflow understanding, and sentiment analysis. These capabilities form the foundation of the hybrid recommender system framework proposed in this work.

The necessity and design advantages of using LLMs in our framework can be summarized as follows:

Interest understanding. In recommender systems, many interaction signals are not stored as quantitative scores in an interaction matrix. Such signals include, but are not limited to, user profiles, item profiles, user textual reviews, and the textual content associated with historical interactions. These heterogeneous data sources lack a unified processing interface under traditional NLP pipelines; LLMs provide a stable and unified semantic interface. In this work, an LLM-based agent [27] analyzes a user's profile, constructs a persona via agent memory, and updates it over time. A user's dynamic interaction history can also be incorporated into memory updates.

Few-shot interaction prediction. Although training LLMs requires large corpora, pretrained LLMs can generalize to new tasks with only a small number of examples, or even in zero-shot settings, by leveraging the knowledge encoded during pretraining. In recommendation, it is often sufficient to provide limited samples to determine whether a user's short-term interests have shifted markedly, enabling the pretrained model to infer and generalize accordingly.

Enhanced interest feature generation. Using an LLM to directly generate recommendations is often inefficient. First, pretrained models do not contain the exact user–item–interaction corpus of the target system. Second, LLM inference is more expensive than conventional models, and thus can be less suitable for real-time recommendation scenarios. To address this, Li et al. proposed GPT4Rec [5], a coarse-retrieval paradigm that uses an LLM to semantically analyze a user's interaction history and infer likely next-step search queries. The retrieved query terms are then used in a retrieval system to obtain coarse candidates, substantially reducing LLM compute while enabling subsequent reranking to improve quality.

3.2. Method

3.2.1. Item Recommendation: An Overview

This section focuses on an agent-based framework design for item recommendation. In urban life, item recommendation spans many aspects of daily activities, including food, clothing, housing, and transportation. Our framework is designed to cover the full set of fields that real-world items may contain. Beyond common interaction features and entity metadata in typical online recommender systems, we explicitly incorporate real-world urban information, such as geo-locations, map data, and integrated city databases. Under this general framework, many conventional online recommender systems with agent memory updates can be viewed as special cases (i.e., item recommendation without

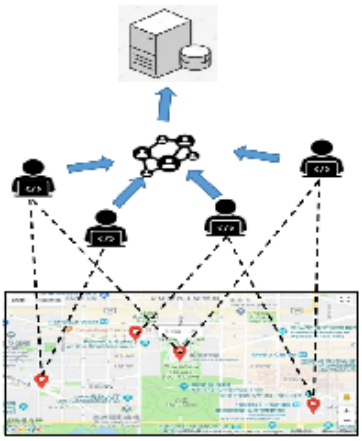

Figure 2. Illustration of item/POI recommendation.

geographic information). Therefore, the recommendation targets can be either online-interactable items or real-world POIs enriched with urban spatial information.

3.2.2. Agent Memory Module: Storage and Updates

Hybrid Static–Dynamic Memory

An external memory module is the core of an agent. For a single user, the agent is initialized with the user's profile and interaction history as *static memory*. As the recommendation process unfolds, *dynamic memory* is formed from real-time context updates and requires a corresponding update mechanism. Therefore, a comprehensive key–value design and update strategy constitutes the core of memory module design. In practical settings, missing values are common; corresponding methods should either implement field-specific handling or adopt a generalizable memory update mechanism.

Different from traditional recommenders that primarily store an interaction matrix and explicit ratings, our memory module stores richer user information and interaction context for each user, retaining more attributes for each single interaction. This enables the recommender to mine more signals during recommendation and produce more accurate results. Moreover, because this memory is stored largely in explicit textual form, semantic reasoning over it was challenging prior to LLMs. With LLMs, the system can infer user profiles at the semantic level in greater detail, and the resulting reasoning is more interpretable to developers.

Below we describe the memory data structures and common operations.

(a) Data structure. Agent memory can be divided into two components: static information and dynamic information. Static information does not change during the recommendation process but is maintained as the system updates over time. Dynamic information is associated with the user's current spatiotemporal context. Both components contribute to modeling the user's preferences but play different roles. Static information primarily captures long-term, relatively stable preference patterns, which supports long-term interest inference and passive recommendation. In real systems, users may issue requests that deviate from long-term preferences, e.g., due to special times (holidays), special locations (unusual places), or short-term exploratory intents. In such cases, short-term recommendation should intervene, and dynamic information can provide necessary corrections to recommendation results.

Static information. Static information is explicitly stored in the database and contains two parts: identification information and interaction information. Identification information includes basic attributes such as name, age, and gender, as well as socio-

economic attributes such as occupation, salary, and deposits, which can help construct user profiles. Interaction information differs from traditional interaction matrices: beyond preference fields, it explicitly stores interaction records, including spatiotemporal context and item review information stored in the database. The data structure is recorded in JSON format [28] and shown in Listing 1.

Dynamic information. Dynamic information is generated at the time of user request. Compared to conventional recommenders, with the semantic understanding capability of LLMs, the system can accept user-provided query strings, as well as spatiotemporal context and additional filtering constraints. The data structure is recorded in JSON format [28] and shown in Listing 2.

Listing 1. User Static Information (JSON-like Representation)

```
{ "Identification Info": {
  "Name": <user name>,
  "UserID": <unique identifier>,
  "Age": <age>,
  "Gender": <gender>,
  "Salary": <salary>,
  "Federation": <organization/group>,
  "Work age": <years of work>,
  "Deposit": <deposits>,
  "Company": <company>,
  "Home Address": <home address>
},
"Interaction Info": {
  "Description of Favor": <preference description>,
  "Interaction History": {
    "Inter1": {
      "Time": <time>,
      "Query": <query>,
      "Filter": <filters>,
      "Final decision": {
        "Item name": <item name>,
        "Item category": <category>,
        "Item comment grade": <rating>,
        "Item price average": <avg price>,
        "Item comment count": <review count>,
        "Item distance": <distance>
      }
    }
  }
}
```

Listing 2. User Dynamic Information (JSON-like Representation)

```
{ "Query": <user query>,
  "Filter": {
    "filter type": <distance/reviews/price/rating/category>,
    "rank type": <1: ascending, 0: descending, none>}
  "Location": <user location>,
  "Time": <request time> }
```

(b) Data operations

Based on the above memory data structures, we design a set of operations and memory update mechanisms. These operations are executed by a collection of specialized agents, each responsible for a specific function, and collectively they cooperate to complete recommendation. The key agent group and their operations are as follows:

Static memory initialization (Passive intent recognition; Pir Agent). During initialization, the agent reads static memory. The Pir Agent consumes basic user information and historical interactions to infer a passive intent profile and writes it into the `Interaction Info.Description of Favor` field.

Dynamic memory update (Active intent recognition; Air Agent). When receiving a user-initiated request, the Air Agent extracts the primary intent from the query, parses spatiotemporal signals, infers mixed intents, and updates agent memory accordingly.

Cross-updating hybrid memory (Supplement and Widen; Sew Agent). Within a recommendation round, Pir and Air not only read/update their own parts but also perform cross-updates. New dynamic interactions are written into static interaction history and static preference descriptions, while semantic features from static memory can serve as defaults when dynamic fields are missing.

Candidate recall (Gathering/Linkage Out-of Versatile Explorer; Glove Agent). After memory update, the system can use the memory to infer a temporary persona and retrieve candidates from the recommender, mimicking human retrieval behavior. Details are introduced later.

Reflection (Reasoning in General and Specific; Rings Agent). To improve explainability and ranking quality, the agent interprets each candidate result, evaluates whether it aligns with user preferences, and adjusts ranking accordingly.

Exploratory recommendation (PLAN-B Agent). In certain contexts, users may prefer non-stationary recommendations, e.g., special holidays, unusual locations, special requests, or when long-term interest drift is small. Exploratory recommendation can also address the long-tail issue by giving additional exposure to high-quality but under-exposed content in UGC platforms.

Feedback reception (Shake Agent). Traditional recommenders typically cannot significantly refine results for every user after returning recommendations. In our framework, the system can accept user feedback requests, interpret them, and iteratively refine recommendations.

3.2.3. LLM-Agent-Based POI Recommendation Workflow

After describing the functionality of each agent component, we outline the overall recommendation workflow. Subsequent optimizations will focus on selected components within this framework. As suggested by the agent names, the recommendation system resembles a process of "sewing a glove", "wearing rings", and finally "shaking hands" with the user. The overall pipeline consists of: (i) memory initialization (Pir/Air and Sew), (ii) candidate recall (Glove, Rings, and Plan-B), and (iii) post-feedback (Shake).

Memory initialization. Pir and Air initialize the memory using static profiles and dynamic requests, respectively. The Sew Agent then performs cross-updates. Static memory is maintained, and the enriched dynamic request is passed to the subsequent recommendation steps.

Candidate recall. After sufficiently understanding the user via semantic reasoning, the system retrieves candidates. Recall includes regular recall for ordinary long-/short-term interests and exploratory recall for special scenarios.

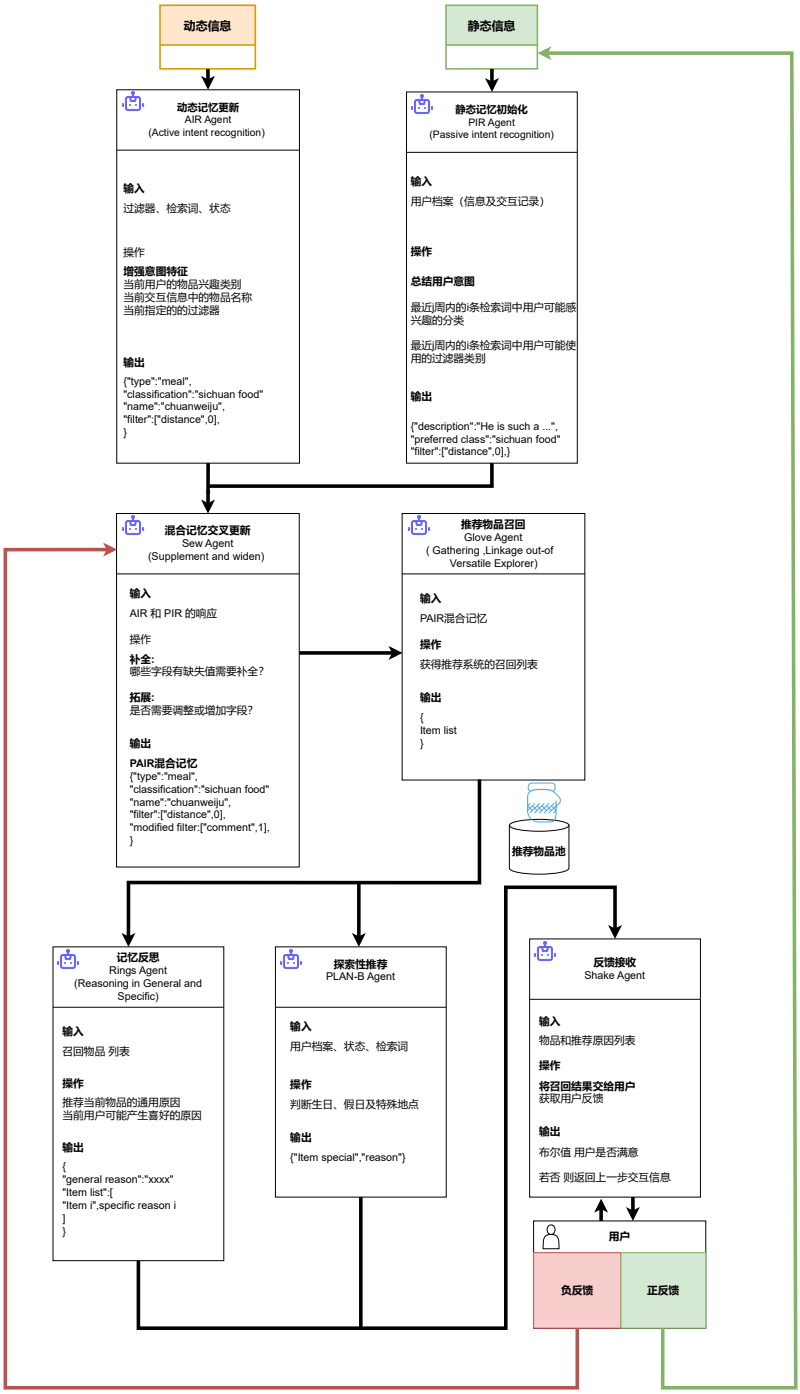

Figure 3. End-to-end POI recommendation workflow driven by LLM agents.

Post-feedback. After returning results, the system interacts with the user once. If the user accepts the results, the process terminates and memory is updated. Otherwise, the reflection module is invoked to adjust the ranking until a satisfactory list is produced.

3.2.4. Long–Short-Term Interest Change Detection and Adaptive Recommendation

Linear-Network-Based Long–Short-Term Recommender

As discussed above, when modeling large-scale historical interactions, model-based interaction recommenders are often used, under the assumption that users exhibit stable, extractable features over long-term interactions. However, traditional matrix-based meth-

ods do not explicitly consider the temporal effectiveness of interactions. In particular, a user may experience a substantial recent interest shift, yet long-ago interactions are weighted equally to recent ones, reducing sensitivity to short-term changes. Therefore, we introduce time-aware weighting to adjust the attention range: recent interactions receive higher attention weights, while earlier interactions are down-weighted. This section focuses on the introduction of a linear weighting layer and its effect on the algorithm.

Consider an SVD-based interaction recommender. To emphasize recent behaviors, we introduce multiple weight modulation layers that reweight interactions by recency. To avoid confusion with the item index i used in R_{ui}, we use $t \in \{1, \ldots, T\}$ to index the position of an interaction in a user's time-ordered history (from oldest to most recent), and let T denote the total length (time horizon) of that history. A simple linear recency modulation can be written as:

$$w_t^{(n)} = \frac{t}{T}. \tag{20}$$

More generally, we use a smooth activation (sigmoid) to control the attention range at different time scales. Let $j_n \in \{1, \ldots, T\}$ denote the high-attention center index at time scale n, and define the normalized center as $c_n = j_n/T \in (0, 1]$. Let $\sigma(x) = \frac{1}{1+\exp(-x)}$. Then:

$$w_t^{(n)} = \sigma\left(\alpha_n\left(\frac{t}{T} - c_n\right)\right) = \frac{1}{1 + \exp\left(-\alpha_n\left(\frac{t}{T} - c_n\right)\right)}, \tag{21}$$

where α_n controls the slope and c_n controls the center of the n-th modulation. In the matrix form used below, we associate each observed interaction (u, i) with its position $t(u, i)$ in user u's time-sorted interaction sequence (e.g., obtained by sorting that user's interactions by timestamp and taking the rank). We then broadcast the recency weights to the interaction matrix as $W_{ui}^n = w_{t(u,i)}^{(n)}$.

Let η denote the learning rate. The gradient updates become:

$$U_k' = U_k - \eta\left(\frac{\partial \mathcal{L}}{\partial U_k}\right) = U_k - \eta\left(-2\sum_{(u,i)\in\mathcal{K}}(R_{ui} - \sum_{n=1}^{N} W_{ui}^n \cdot (U_k\Sigma_k V_k^T)_{ui}) \sum_{n=1}^{N} W_{ui}^n V_k \Sigma_k + 2\lambda U_k\right) \tag{22}$$

$$V_k' = V_k - \eta\left(\frac{\partial \mathcal{L}}{\partial V_k}\right) = V_k - \eta\left(-2\sum_{(u,i)\in\mathcal{K}}(R_{ui} - \sum_{n=1}^{N} W_{ui}^n \cdot (U_k\Sigma_k V_k^T)_{ui}) \sum_{n=1}^{N} W_{ui}^n \Sigma_k^T U_k^T + 2\lambda V_k\right) \tag{23}$$

For conciseness and to leverage CUDA parallelism, the above computation can be reformulated using tensor operations. Let $\circ$ denote element-wise multiplication. The predicted rating tensor becomes:

$$\hat{\mathcal{R}} = \sum_{n=1}^{N} \mathcal{W}_n \circ (\mathcal{U}\Sigma\mathcal{V}^T) \tag{24}$$

The gradient updates follow $\mathcal{U} \leftarrow \mathcal{U} - \eta\frac{\partial \mathcal{L}}{\partial \mathcal{U}}$ and $\mathcal{V} \leftarrow \mathcal{V} - \eta\frac{\partial \mathcal{L}}{\partial \mathcal{V}}$, where:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = -2\left(\mathcal{R} - \sum_{n=1}^{N} \mathcal{W}_n \circ (\mathcal{U}\Sigma\mathcal{V}^T)\right) \sum_{n=1}^{N} \mathcal{W}_n \circ (\mathcal{V}\Sigma) + 2\lambda\mathcal{U} \tag{25}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{V}} = -2\left(\mathcal{R} - \sum_{n=1}^{N} \mathcal{W}_n \circ (\mathcal{U}\Sigma\mathcal{V}^T)\right)^T \sum_{n=1}^{N} \mathcal{W}_n \circ (\Sigma\mathcal{U}^T) + 2\lambda\mathcal{V} \tag{26}$$

The GPU-parallel training procedure of the weighted attention-range recommender can be summarized by the following pseudocode.

Algorithm 3 Attention-Range-Controlled Recommender with Weight Layers (GPU; Pseudocode)

Input: user set U, item set I, rating matrix R, rank k, learning rate η, regularization λ, epochs num_epochs, attention ranges $\{K_n\}$
Output: model set $\{\hat{\mathcal{R}}^n\}$ for different attention ranges
for each K_n in $\{K_n\}$ **do**
  initialize $\mathcal{U}^n \in \mathbb{R}^{m\times k}$ and $\mathcal{V}^n \in \mathbb{R}^{n\times k}$ and move to GPU
  for epoch = 1 to num_epochs **do**
    compute $\hat{\mathcal{R}}^n = \sum_{i=1}^{N} \mathcal{W}_i^n \circ (\mathcal{U}^n \Sigma \mathcal{V}^{nT})$
    compute gradients $\partial\mathcal{L}^n/\partial\mathcal{U}^n$ and $\partial\mathcal{L}^n/\partial\mathcal{V}^n$
    update $\mathcal{U}^n \leftarrow \mathcal{U}^n - \eta\,\partial\mathcal{L}^n/\partial\mathcal{U}^n$
    update $\mathcal{V}^n \leftarrow \mathcal{V}^n - \eta\,\partial\mathcal{L}^n/\partial\mathcal{V}^n$
  end for
  save $\hat{\mathcal{R}}^n$ from GPU to CPU
end for
return $\{\hat{\mathcal{R}}^n\}$

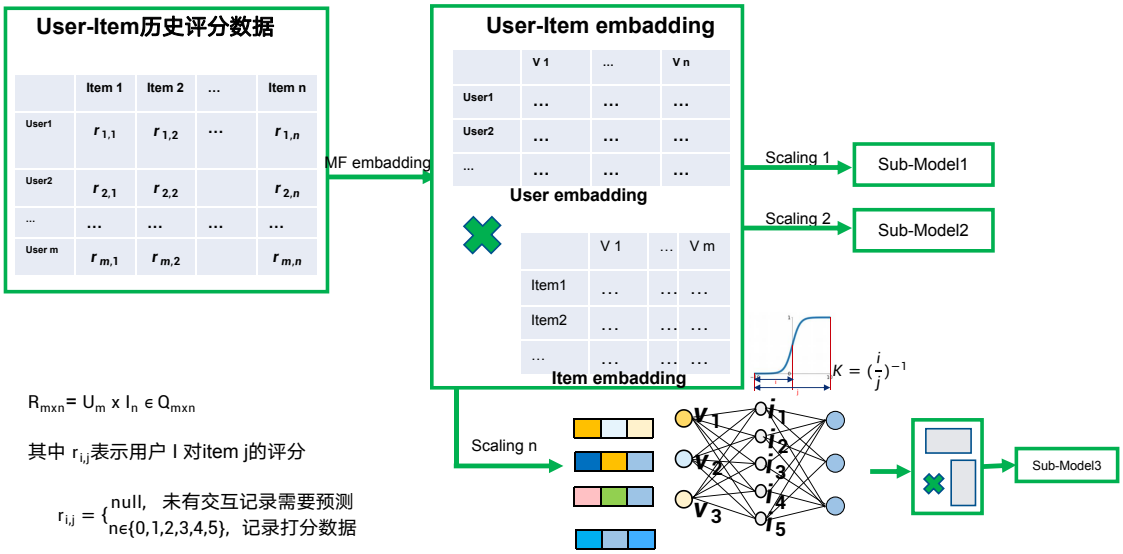

Figure 4. Illustration of the weighted long–short-term recommendation network.

LLM-Based Preference Shift Identification

After obtaining a family of sub-models with different attention ranges, the key challenge is to select which model to schedule for a given user, which depends on detecting preference shifts. We consider two approaches: qualitative shift detection and quantitative shift estimation.

For qualitative detection, the LLM's semantic analysis does not reliably yield an exact high-attention ratio. Instead, with a small number of prompts/examples, the LLM can provide a qualitative judgment of whether a user's short-term interests differ substantially from long-term interests. In this setting, short-term model parameters are shared across users, and the LLM selects between a long-term and a short-term model. Concretely, if the difference between long- and short-term interests is large, the short-term model is prioritized; if long-term preferences remain stable, the system maintains the long-term model with high probability and schedules an exploratory model with some probability [29].

For quantitative estimation, the LLM attempts to infer when the user's interest shift occurred and estimate the high-attention ratio, and then select the sub-model whose attention-range parameter just exceeds the inferred ratio. This approach depends on

accurate shift detection and few-shot capabilities, and in our experimental setting it is less effective than qualitative detection.

LLM-Based Adaptive Long–Short-Term Recommender

Based on Algorithm 3 and the quantitative shift estimation approach above, we propose the following LLM-based adaptive long–short-term recommendation procedure.

Algorithm 4 Interest Shift Detection and Model Scheduling (Pseudocode)

Input: user profile set $\{P_u\}$, model set $\{\hat{\mathcal{R}}^n\}$
Output: scheduled model $\{\hat{\mathcal{R}}_u^*\}$
for each user profile P_u **do**
  Sew Agent analyzes interest change and **computes** ΔI_u
  if ΔI_u exceeds a threshold **then**
    select $\hat{\mathcal{R}}_u^* = \arg\max_n \texttt{Fitness}(K_n, \Delta I_u)$
  else keep the current model
  save $\hat{\mathcal{R}}_u^*$
end for
return $\{\hat{\mathcal{R}}_u^*\}$

3.2.5. Semantic Content Recommendation

Vectorization and Similarity-Based Recall

Chapter 2 introduced content-based recommendation methods. In this work, we adopt three common NLP vectorization models/toolkits: TF–IDF, BERT, and spaCy. We first generate a set of query terms via the LLM-based framework and vectorize these terms. The resulting vectors are used to compute user–item similarities and produce recommendation lists. The advantage of using an LLM is that it provides stronger semantic understanding and generation capabilities, capturing subtle changes in user preferences and enabling higher-level semantic analysis and preference modeling for more accurate and personalized recommendations.

Algorithm 5 Vectorization and Similarity-Based Recall (Pseudocode)

Input: users $\{u\}$, items $\{i\}$, top-N, vectorizer (TF-IDF/BERT/spaCy)
Output: recommendation lists $\{L_u\}$
for each item i **do**
  if TF-IDF **then compute** TF-IDF vector $\mathbf{v}_i$
  else if BERT **then** preprocess Doc_i and **compute** $\mathbf{v}_i = \text{BERT}(\text{Doc}_i)$
  else if spaCy **then** preprocess, parse, NER, and **compute** embedding vector $\mathbf{v}_i$
end for
for each user u **do**
  compute preference vector $\mathbf{p}_u = \frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} \mathbf{v}_i$
  for each candidate item i **do**
    compute $\text{sim}(u,i) = \mathbf{p}_u \cdot \mathbf{v}_i$
  end for
  rank items by similarity to obtain L_u
  save L_u
end for
return $\{L_u\}$

Regular-Expression Matching and Threshold Filtering

In practical vectorization and retrieval, we observed several special cases and applied corresponding corrections [30–33]. In general, similarity-based recall can be implemented in three ways: (i) direct string-level regular-expression matching, (ii) embedding

similarity-based recall, and (iii) collaborative-filtering-based recall from interaction logs. Collaborative-filtering-based recall and its LLM-based optimization were introduced earlier; here we focus on correcting semantic drift in embedding-based recall using regex matching and threshold filtering.

For example, in an existing retrieval system, searching for "Star War" [30] may tokenize and average "Star" and "War", and then retrieve items whose vectors are closest to the query vector. In practice, later Star Wars entries (e.g., *Star Wars: The Force Awakens*) may contain subtitle tokens that deviate from the original query, resulting in low-priority retrieval under pure embedding similarity. Conversely, *Guardians of the Galaxy*, which is thematically different, may be retrieved with high priority due to superficial lexical proximity (e.g., "Guardian" vs. "War", "Star" vs. "Galaxy"). Additionally, *The Mandalorian* as a Star Wars spin-off is highly relevant to Star Wars fans, yet its title shares few tokens with the original query and is difficult to retrieve by naive embedding similarity. The Mandalorian case can be further corrected using interaction-based recommendation and will be revisited in the fusion framework. This section proposes a correction strategy that combines direct regex matching and threshold filtering to mitigate semantic drift in embedding-based retrieval.

We model the correction process as follows. Suppose we have query terms $Q_1, Q_2, \ldots, Q_n$. For each query Q_i, we retrieve candidates $R_{Q_i} = \{R_{Q_i}1, R_{Q_i}2, \ldots, R_{Q_i}m\}$. Let $\mathbf{v}_{Q_i}$ be the vector of Q_i, and let $\mathbf{v}_j$ be the vector of item item_j in the database. Define similarity $\text{sim}(\mathbf{v}_{Q_i}, \mathbf{v}_j)$ and threshold θ. Let $\text{Sorted}_{R_{Q_i}}$ be the similarity-sorted list, Match be regex matches, and $\text{Final}_{R_{Q_i}}$ be the final recall list. The procedure is summarized below:

1) **Vectorization.** Vectorize queries and items:

$$\mathbf{v}_{Q_i} = \text{Vectorize}(Q_i), \quad \mathbf{v}_j = \text{Vectorize}(\text{item}_j)\ \forall j. \tag{27}$$

2) **Similarity and thresholding.** Filter items by:

$$\text{sim}(\mathbf{v}_{Q_i}, \mathbf{v}_j) > \theta. \tag{28}$$

3) **Sorting.** Sort by decreasing similarity:

$$\text{Sorted}_{R_{Q_i}} = \{R_{Q_i}1, R_{Q_i}2, \ldots\} \text{ with decreasing } \text{sim}(\mathbf{v}_{Q_i}, \mathbf{v}_{R_{Q_i}j}). \tag{29}$$

4) **Regex correction.** For insufficient recall, use regex matches:

$$\text{Match} = \{M_1, M_2, \ldots\}. \tag{30}$$

5) **Fusion and insertion.** Produce:

$$\text{Final}_{R_{Q_i}} = \begin{cases} \text{Sorted}_{R_{Q_i}} + \text{Match}, & \text{if } \exists j : \text{sim}(\mathbf{v}_{Q_i}, \mathbf{v}_j) > \theta, \\ \text{Match}, & \text{otherwise.} \end{cases} \tag{31}$$

The corresponding pseudocode is shown below.

Algorithm 6 Query-Term-Based Item Recall and Ranking with Thresholding and Regex (Pseudocode)

Input: query set $\{Q_i\}$, item database, threshold θ
Output: recalled item list Final_R
Initialize Final_R = []
for each query Q_i **do**
  vectorize Q_i to $\mathbf{v}_{Q_i}$; Temp_R = []
  for each item j in database **do**
    vectorize item j to $\mathbf{v}_j$ and **compute** sim
    if sim($\mathbf{v}_{Q_i}, \mathbf{v}_j$) $> \theta$ **then** add j to Temp_R
  end for
  if Temp_R not empty **then** sort Temp_R by sim desc and merge into Final_R
end for
if Final_R empty **then** perform regex matching, sort matches, and insert into Final_R
return Final_R

LLM-Based Query–Search Recommendation Architecture

A pretrained LLM can analyze a user's interaction history to predict a set of keywords that the user may use in the next search. These keywords are then fed into retrieval and ranking to optimize recommendation. Direct LLM-based retrieval is inefficient; generating query terms and using a retrieval engine is significantly more efficient. The procedure is:

1) **Interaction history initialization.** Represent the interaction history as $H = \{h_1, h_2, \ldots, h_k\}$.

2) **History analysis.** Use a pretrained LLM to generate predicted query terms:

$$Q = \mathrm{PTM}(H), \tag{32}$$

where $Q = \{q_1, q_2, \ldots, q_n\}$ is the predicted query set.

3) **Retrieval using Algorithm 6.** For each q_i, retrieve similar items:

$$R_{q_i} = \mathrm{FindSimilarItems}(q_i, \theta, \mathrm{top\_n}). \tag{33}$$

The pseudocode is as follows.

Algorithm 7 Pretrained-LLM-Based Query–Search Recommendation (Pseudocode)

Input: interaction history H, item database, threshold θ
Output: recalled item list Final_R
Initialize Final_R = []
Use pretrained LLM to produce query set $Q = \{q_1, \ldots, q_n\}$
for each query q in Q **do**
  invoke threshold+regex corrected recall (Algorithm 6) to get R_q
  merge R_q into Final_R
end for
return Final_R

Through these steps, the pretrained LLM generates plausible future query terms from historical interactions, and the retrieval system performs thresholded similarity matching with regex correction, enabling content-similarity recommendation with reduced LLM overhead.

3.2.6. LLM-Based Hybrid Intent Recognition and Recommendation Framework

Aligned Recall for Content–Interaction Hybrid Recommendation

This section introduces a behavior-aware correction method that uses the optimized interaction-based model in Section 3.2.3 to correct the content-similarity recall results produced in Section 3.2.4, which lack behavioral signals.

Let a user's interaction history be $H = \{h_1, \ldots, h_k\}$. As above, a pretrained LLM predicts a query sequence $Q = \{q_1, \ldots, q_n\}$ and obtains coarse recall results $R_{q_i} = \text{FindSimilarItems}(q_i, \theta, \text{top\_n})$. We then perform behavior-based aligned correction as follows:

Behavior-based correction recall. For each coarse-recalled item $r \in R_{q_i}$, we use the attention-range-controlled model (Algorithm 3) to perform further recall and then fuse results. Let $R = \bigcup_{i=1}^{n} R_{q_i}$. For each $r \in R$, we obtain:

$$V_r = \hat{\mathcal{R}}_n(r),$$

where $\hat{\mathcal{R}}_n(r)$ denotes further recall using the model from the weighted attention-range recommender.

(1) Recall alignment and fusion. Fuse all V_r to form:

$$F = \bigcup_{r \in R} V_r.$$

(2) Composite scoring. For each item $v \in F$, compute a composite score:

$$S(v) = \sum_{r \in R} w_r \cdot \text{score}(v, r),$$

where w_r is a weight for item r, and $\text{score}(v, r)$ measures similarity between v and r.

(3) Ranking. Sort F by $S(v)$ to obtain the final recommendation list.

The overall procedure is summarized below.

Algorithm 8 Behavior-Aligned Correction for Query–Search Recall (Pseudocode)

```
Input: interaction history H, item database, threshold θ
Output: final ranked list
Initialize Final_R = []
Use pretrained LLM to generate queries Q = {q_1, ..., q_n}
for each query q do
  R_q = FindSimilarItems(q, θ, top_n); Intermediate_R=[]
  for each item r in R_q do
    V_r = R̂_n(r); add V_r to Intermediate_R
  end for; merge Intermediate_R into Final_R
end for
Compute composite scores S(v) for candidates
Sort candidates by S(v) in descending order
return Final_R
```

LLM-Driven Adaptive Hybrid Recommender System

Building upon the above components, we describe the final integrated design. First, using the adaptive matrix factorization network, we obtain a family of sub-models with different attention ranges, enabling item representations at different time scales. The LLM then qualitatively (or quantitatively, in significant cases) infers whether a user's short-

term interest has shifted substantially relative to long-term preferences, and selects an appropriate sub-model for behavior-based recall.

Second, the content-based recommender with regex matching and threshold filtering predicts the next likely search queries from the user's historical textual signals and performs similarity-based coarse recall.

Finally, given both the behavior-based correction model and the content-based recall results, we apply the fusion framework in Algorithm 8 to correct content recall using behavioral signals. The final ranked list thus jointly reflects both item content semantics and user interaction behaviors, and is returned for evaluation.

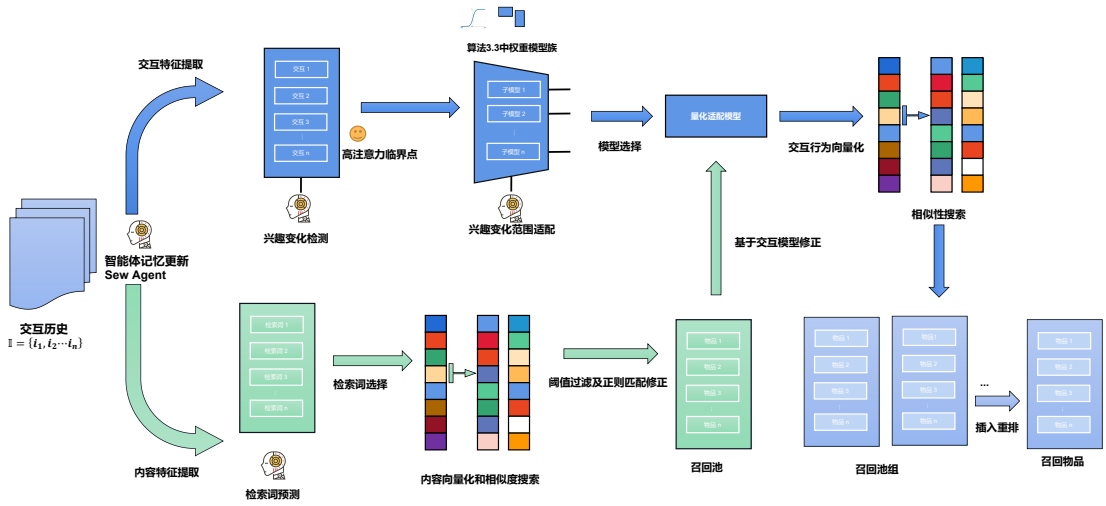

Figure 5. LLM-driven adaptive hybrid recommender system framework.

3.3. Summary

This chapter presented (i) an LLM-enhanced interaction-based long–short-term adaptive recommendation network, (ii) a content-based recommendation network corrected by regex matching and threshold filtering, and (iii) an integrated framework combining Query–Search and behavior–content alignment for hybrid recommendation. We provided mathematical formulations, algorithmic designs, and an overall system architecture. This chapter constitutes the core theoretical foundation of this work and directly supports the subsequent experimental evaluation.

4. Experiments

4.1. Experimental Setup

4.1.1. Datasets

To validate the effectiveness of the proposed recommender system, we require a real-world dataset that contains (i) user information, (ii) item textual information, and (iii) user–item interaction data. Based on these requirements, we adopt the MovieLens dataset [4]. The dataset is collected by the GroupLens organization [34] from an online movie recommendation platform and consists of users' real rating records for movies. Spanning nearly two decades, MovieLens provides multiple versions with different scales, including 100K, 1M, 10M, and 20M [35–38]. Considering computational cost and feature extraction difficulty, we primarily use MovieLens-1M in our experiments. The dataset used in this work is an offline dataset, and the fields contained in MovieLens can be viewed as a subset of the full item schema discussed in the previous chapter; at the current stage, these fields are sufficient to validate the soundness of the proposed framework. In future

iterations, we plan to develop and evaluate an online version using real-time data from production environments and a complete data schema.

MovieLens-1M contains 1,000,209 rating records from 6,040 users on 3,883 movies. Each user has rated at least 20 movies. The dataset covers diverse genres, and the rating timestamps are relatively evenly distributed over time. We use MovieLens-1M to evaluate the performance of the recommender system.

In this work, we split the dataset into training/validation/test sets with a ratio of 8:1:1 for training model-based recommenders. The dataset also includes basic user attributes (e.g., age, gender, and occupation) and basic movie attributes (e.g., title and genre).

In addition, to better process rating data, we preprocess rating timestamps by converting them into year/month/day formats. This facilitates subsequent time-scale modeling. We also implement an interface to convert user information into complete sentences to support later semantic analysis.

Additional statistical characteristics of the dataset are summarized in Table 1.

Table 1. Statistical Summary of the MovieLens-1M Dataset

Statistic	Value
Number of users	6,040
Number of movies	3,883
Number of ratings	1,000,209
Avg. ratings per user	165.5
Avg. ratings per movie	257.6
Avg. rating	3.58

4.1.2. Evaluation Metrics

This section introduces the evaluation metrics used in this work. For the final recommender system, we quantify both accuracy and ranking quality. Specifically, we adopt Recall@k and NDCG@k [39,40] to evaluate the top-k retrieval recall and the ranking quality, respectively. We detail their definitions and computation below.

Following the notation in previous chapters, let the user set be $\mathcal{U} = \{u_1, u_2, \cdots, u_m\}$ and the item set be $\mathcal{I} = \{i_1, i_2, \cdots, i_n\}$. The interaction space is represented by a matrix $R \in \mathbb{R}^{m \times n}$, where R_{ui} denotes the rating of user u on item i. If user u did not rate item i, then R_{ui} is missing. The goal is to predict the ratings of the held-out items $\{I_{n-k+1}, \cdots, I_n\}$ based on the observed ratings $\{I_1, I_2, \cdots, I_{n-k}\}$. The recommender produces predicted values $\{I'_{n-k+1}, \cdots, I'_n\}$, which are compared with the ground truth $\{I_{n-k+1}, \cdots, I_n\}$.

Recall@k. Recall measures the overlap between the recommended list and the true interacted items. It captures accuracy but does not reflect ranking quality. The value ranges from 0 to 1; higher values indicate higher overlap. The parameter k specifies the cutoff.

$$\text{Recall@k} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\{I_{n-k+1}, I_{n-k+2}, \ldots, I_n\} \cap \{I'_1, I'_2, \ldots, I'_k\}|}{|\{I_{n-k+1}, I_{n-k+2}, \ldots, I_n\}|} \tag{34}$$

NDCG (Normalized Discounted Cumulative Gain). NDCG evaluates ranking quality. If items with higher user ratings are ranked earlier, the score is higher. The value ranges from 0 to 1; higher values indicate better ranking quality. The parameter k specifies the cutoff.

1) DCG@k (Discounted Cumulative Gain):

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{2^{R_{u,I'_i}} - 1}{\log_2(i+1)} \tag{35}$$

2) IDCG@k (Ideal DCG):

$$\text{IDCG@k} = \sum_{i=1}^{k} \frac{2^{R_{u,I_i^*}} - 1}{\log_2(i+1)} \tag{36}$$

3) NDCG@k:

$$\text{NDCG@k} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\text{DCG@k}}{\text{IDCG@k}} \tag{37}$$

where $R_{u,I_i'}$ is user u's rating for the recommended item I_i'; I_i' denotes the i-th recommended item; and I_i^* is the i-th item in the ideal ranking sorted by true ratings.

4.2. Experimental Results

4.2.1. Interaction-Based Long–Short-Term Interest Shift Detection and Adaptive Recommendation

Based on the long–short-term adaptive recommendation network with linear weighting layers described in Section 3.2.3 and the training procedure in Algorithm 3.3, we train models with $K \in \{1, 2, 3, 4\}$. The case $K = 1$ corresponds to the original MF model and serves as the baseline, yielding the MF@Kfold_agent_free model after training. We then integrate an LLM memory module to infer model invocation and obtain the final MF@Kfold_agent model. We evaluate on a validation set consisting of the last 600 users in MovieLens-1M. The results are shown in Table 2.

Table 2. K-Fold MF Results

Experiment	Recall@1	Recall@5	Recall@20	Recall@100
MF@1fold (baseline)	0.00522	0.02642	0.09155	0.25371
MF@2fold_agent_free	0.00466	0.02621	0.08443	0.26017
MF@2fold_agent	0.00631 (+20.88%)	0.03345 (+26.63%)	0.10639 (+16.19%)	0.29908 (+17.88%)
MF@3fold_agent_free	0.00510	0.02617	0.09087	0.25734
MF@3fold_agent	0.00489 (-6.31%)	0.02878 (+8.93%)	0.09788 (+6.92%)	0.27161 (+7.05%)
MF@4fold_agent_free	0.00477	0.02596	0.09040	0.26133
MF@4fold_agent	0.00518 (-0.77%)	0.02766 (+4.69%)	0.09702 (+5.98%)	0.27142 (+6.98%)

Key observations from the above results are as follows:

Benefit of attention-range quantization. As shown, when $K = 2, 3, 4$, recall improves notably compared with the baseline. Recall@1 is more stringent and may decrease due to its strictness, whereas recall at larger cutoffs improves consistently. The largest gain is achieved by MF@2fold_agent, which improves Recall@5 by 26.63% over the baseline.

Effect of varying K on long-term prediction. Comparing different base models, performance gains are most pronounced when the high-attention range is approximately half of the overall interaction horizon. As K increases, the high-attention window becomes shorter, which reduces extracted features and shortens attention span, leading to reduced long-term prediction performance.

LLM-enhanced semantic scheduling. Across experimental groups, models with LLM-driven semantic reasoning and scheduling consistently outperform their agent-free counterparts, demonstrating the LLM's ability to understand user preference shifts and to schedule models effectively.

For the best-performing MF@2fold_agent model, the cumulative Recall@5 variation is illustrated in Figure 6.

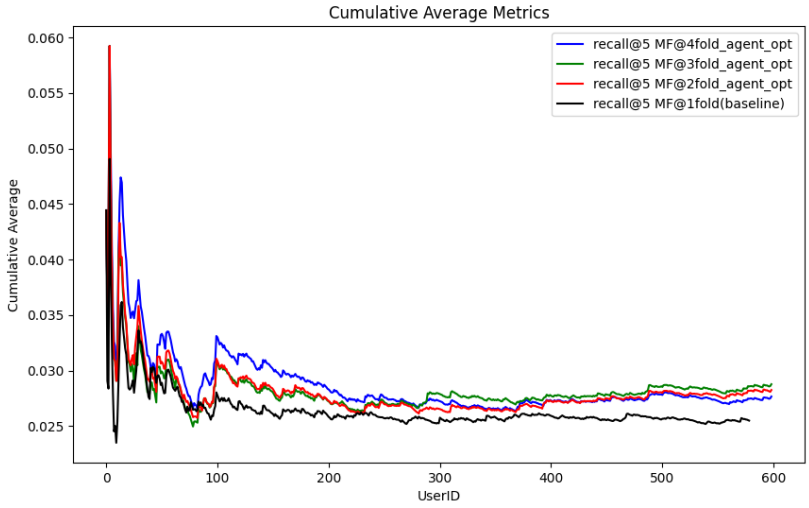

Figure 6. MF@2fold_agent Recall@5 Results.

4.2.2. Semantic Content-Based Recommendation

Based on the system design in Section 3.2.4 and Algorithm 3.5, we implement a retrieval system that performs vectorization-based search. We evaluate three vectorization models/toolkits—BERT, TF–IDF, and spaCy—and use an LLM to predict users' future query terms. We then perform item recall using the predicted queries through the vectorization interface. Since multiple queries are generated and multiple recall rounds are needed, we set the smallest cutoff to $k = 10$. The results are shown in Table 3.

	NDCG@k				Recall@k			
	k=10	k=20	k=50	k=100	k=10	k=20	k=50	k=100
BERT	0.1695	0.2215	0.2789	0.3305	0.0034	0.0073	0.0171	0.0341
TF–IDF	0.2151	0.2489	0.3007	0.3541	0.0046	0.0080	0.0184	0.0416
spaCy	0.1912	0.2324	0.3017	0.3380	0.0044	0.0082	0.0206	0.0361

Table 3. Query–Search Recommendation Performance with Different Vectorization Methods

Based on the results, we select TF–IDF and spaCy for subsequent experiments due to their stronger performance. BERT performs worse in our setting and is also incompatible with other models in later experiments due to excessive GPU memory usage. Therefore, for both compute and performance considerations, we do not use BERT in later experiments; instead, we validate the framework using the remaining vectorization methods.

4.2.3. LLM-Based Hybrid Intent Recognition Recommendation System

In this section, we implement the hybrid intent recognition recommendation framework designed in Section 3.2.5 and Algorithm 3.8. As a baseline, we adopt a *simple* multi-armed bandit (MAB) setup to balance exploitation of observed behavioral regularities and exploration. Concretely, we use an epsilon-greedy strategy to trade off known and novel information [41], and aim to maximize cumulative rewards [42,43]. The exploration probability is set to 0.1.

We emphasize that this MAB baseline is primarily used as a *trivial* exploratory reference rather than a strong state-of-the-art (SOTA) bandit baseline; thus, the comparison should be interpreted as an initial proof-of-concept for the proposed framework. More rigor-

ous comparisons against stronger bandit variants and competitive modern recommenders are left for future work.

Table 4. Performance Comparison of Recommendation Systems

		NDCG@k				Recall@k			
		k=10	k=20	k=50	k=100	k=10	k=20	k=50	k=100
MAB		0.1101	0.1120	0.1245	0.1428	0.0206	0.0373	0.0784	0.1293
spaCy	Base	0.1912 (+73.59%)	0.2324 (+107.14%)	0.3017 (+142.29%)	0.3380 (+136.73%)	0.0044 (-78.64%)	0.0082 (-78.00%)	0.0206 (-73.72%)	0.0361 (-72.08%)
	K1_align	0.2435 (+121.06%)	0.2514 (+124.52%)	0.3019 (+142.49%)	0.3461 (+142.43%)	0.0042 (-79.61%)	0.0061 (-83.65%)	0.0142 (-81.89%)	0.0291 (-77.49%)
	K2_align	0.1937 (+75.88%)	0.2387 (+113.19%)	0.2834 (+127.59%)	0.3304 (+131.38%)	0.0028 (-86.41%)	0.0057 (-84.73%)	0.0138 (-82.40%)	0.0286 (-77.87%)
	K3_align	0.1820 (+65.04%)	0.2060 (+83.93%)	0.2626 (+110.85%)	0.3221 (+125.56%)	0.0033 (-83.98%)	0.0055 (-85.27%)	0.0124 (-84.18%)	0.0272 (-78.95%)
	K4_align	0.1759 (+59.76%)	0.2057 (+83.71%)	0.2609 (+109.54%)	0.3202 (+124.21%)	0.0033 (-83.98%)	0.0058 (-84.47%)	0.0126 (-83.93%)	0.0265 (-79.49%)
TF–IDF	Base	0.2151 (+95.23%)	0.2489 (+122.23%)	0.3007 (+141.48%)	0.3541 (+148.02%)	0.0046 (-77.67%)	0.0080 (-78.56%)	0.0184 (-76.55%)	0.0416 (-67.76%)
	K1_align	0.2190 (+98.19%)	0.2482 (+121.61%)	0.3019 (+142.49%)	0.3343 (+134.14%)	0.0039 (-81.07%)	0.0066 (-82.30%)	0.0150 (-80.87%)	0.0279 (-78.43%)
	K2_align	0.1927 (+75.03%)	0.2265 (+102.24%)	0.2729 (+119.17%)	0.3281 (+129.91%)	0.0031 (-84.95%)	0.0056 (-84.98%)	0.0126 (-83.93%)	0.0274 (-78.80%)
	K3_align	0.1538 (+39.68%)	0.1985 (+77.23%)	0.2520 (+102.39%)	0.3110 (+117.86%)	0.0022 (-89.32%)	0.0047 (-87.40%)	0.0113 (-85.60%)	0.0245 (-81.04%)
	K4_align	0.1530 (+38.87%)	0.2020 (+80.36%)	0.2543 (+104.21%)	0.3134 (+119.45%)	0.0024 (-88.35%)	0.0049 (-86.86%)	0.0114 (-85.46%)	0.0248 (-80.83%)

Overall, compared with this simple MAB baseline, our framework improves ranking-oriented metrics (NDCG) under the diversity- and ranking-focused evaluation setting. For example, at $k = 100$, NDCG increases from 0.1428 (MAB) to 0.3541 (TF–IDF, Base), i.e., an absolute increase of 0.2113; this corresponds to a relative gain of approximately $1.48\times$ over the baseline at that cutoff. At the same time, Recall@k may decrease, reflecting a trade-off where the framework prioritizes ranking quality/diversity of the returned list rather than maximizing pure hit-rate. Given the simplicity of the MAB baseline used here, these results should be viewed as exploratory rather than a definitive SOTA comparison.

4.3. Experimental Conclusions

In this chapter, we experimentally validate the interaction-based long–short-term adaptive recommendation network, the content-based recommendation network corrected by regex matching and threshold thresholding, and the Query–Search plus behavior–content alignment framework proposed in Chapter 3. The adaptive long–short-term module improves recall by up to 26.63%. For the end-to-end hybrid framework, we observe notable improvements in ranking quality (NDCG) under diversity-focused recommendation settings, with representative cutoffs showing gains consistent with the $\sim 1.48\times$ NDCG description in the abstract. However, since the MAB baseline adopted here is intentionally lightweight, these findings are preliminary; future work should compare against stronger bandit baselines and more competitive SOTA recommender systems on broader datasets to more thoroughly assess effectiveness and generalizability.

5. Conclusions and Future Work

5.1. Summary

Motivated by recent advances in large language models (LLMs) and by the complementary strengths of content-based and behavior-based recommender systems, this work proposes a hybrid recommendation framework that integrates semantic understanding with interaction modeling. Specifically, we develop: (i) a long–short-term adaptive recommender that combines interaction reweighting with an LLM-agent-based scheduling mechanism; (ii) a content-based recommender enhanced by threshold filtering and regular-expression matching to mitigate retrieval drift; and (iii) an LLM-driven hybrid-intent recommendation framework that aligns and corrects content recall using interaction signals. To support semantic reasoning and interpretability, we also introduce an agent memory structure to represent and update user intent and context, and provide an end-to-end system design that integrates these components.

We further conduct experiments to evaluate the proposed framework. The results suggest that hybrid-intent recommendation with LLM-assisted alignment can improve ranking-oriented metrics and enhance overall interpretability and semantic reasoning capabilities. As an early exploration of integrating LLMs with traditional recommender algorithms, this work offers a practical design paradigm and empirical evidence that may support subsequent research and real-world applications.

5.2. Future Work

Future work will focus on interactive recommendation in realistic settings and on recommendation techniques that more deeply leverage semantic understanding and reasoning. In particular, most offline metrics are computed on historical logged data, whereas recommendation itself can influence user behavior through exposure effects. Building on LLMs and other semantic technologies, it is promising to incorporate richer user interaction loops and to optimize recommendation models using real-time datasets from production environments. Such settings may also motivate evaluation protocols beyond Recall and NDCG that better reflect causal effects, user satisfaction, and long-term utility.

In addition, LLM-based recommenders are well-positioned to handle more complex and mixed-context scenarios. For example, as discussed in the urban-space setting, POI recommendation includes rich spatiotemporal signals beyond typical online items, and thus requires stronger capabilities in language understanding, semantic grounding, and spatiotemporal reasoning—which remain challenging for existing systems. Further research on LLMs and agentic architectures may enable more effective and valuable solutions for city-scale recommendation problems.

Author Contributions: For research articles with several authors, a short paragraph specifying their individual contributions must be provided. The following statements should be used "Conceptualization, X.X. and Y.Y.; methodology, X.X.; software, X.X.; validation, X.X., Y.Y. and Z.Z.; formal analysis, X.X.; investigation, X.X.; resources, X.X.; data curation, X.X.; writing—original draft preparation, X.X.; writing—review and editing, X.X.; visualization, X.X.; supervision, X.X.; project administration, X.X.; funding acquisition, Y.Y. All authors have read and agreed to the published version of the manuscript.", please turn to the CRediT taxonomy for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported.

Funding: Please add: "This research received no external funding" or "This research was funded by NAME OF FUNDER grant number XXX." and and "The APC was funded by XXX". Check carefully that the details given are accurate and use the standard spelling of funding agency names at https://search.crossref.org/funding, any errors may affect your future funding.

Institutional Review Board Statement: In this section, you should add the Institutional Review Board Statement and approval number, if relevant to your study. You might choose to exclude this statement if the study did not require ethical approval. Please note that the Editorial Office might ask you for further information. Please add "The study was conducted in accordance with the Declaration of Helsinki, and approved by the Institutional Review Board (or Ethics Committee) of NAME OF INSTITUTE (protocol code XXX and date of approval)." for studies involving humans. OR "The animal study protocol was approved by the Institutional Review Board (or Ethics Committee) of NAME OF INSTITUTE (protocol code XXX and date of approval)." for studies involving animals. OR "Ethical review and approval were waived for this study due to REASON (please provide a detailed justification)." OR "Not applicable" for studies not involving humans or animals.

Informed Consent Statement: Any research article describing a study involving humans should contain this statement. Please add "Informed consent was obtained from all subjects involved in the study." OR "Patient consent was waived due to REASON (please provide a detailed justification)." OR "Not applicable" for studies not involving humans. You might also choose to exclude this statement if the study did not involve humans.

Written informed consent for publication must be obtained from participating patients who can be identified (including by the patients themselves). Please state "Written informed consent has been obtained from the patient(s) to publish this paper" if applicable.

Data Availability Statement: We encourage all authors of articles published in MDPI journals to share their research data. In this section, please provide details regarding where data supporting reported results can be found, including links to publicly archived datasets analyzed or generated during the study. Where no new data were created, or where data is unavailable due to privacy or ethical restrictions, a statement is still required. Suggested Data Availability Statements are available in section "MDPI Research Data Policies" at https://www.mdpi.com/ethics.

Acknowledgments: In this section you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments). Where GenAI has been used for purposes such as generating text, data, or graphics, or for study design, data collection, analysis, or interpretation of data, please add "During the preparation of this manuscript/study, the author(s) used [tool name, version information] for the purposes of [description of use]. The authors have reviewed and edited the output and take full responsibility for the content of this publication."

Conflicts of Interest: Declare conflicts of interest or state "The authors declare no conflicts of interest." Authors must identify and declare any personal circumstances or interest that may be perceived as inappropriately influencing the representation or interpretation of reported research results. Any role of the funders in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results must be declared in this section. If there is no role, please state "The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results".

Abbreviations

The following abbreviations are used in this manuscript:

Air	Active intent recognition agent
BERT	Bidirectional Encoder Representations from Transformers
CF	Collaborative Filtering
CUDA	Compute Unified Device Architecture
DCG	Discounted Cumulative Gain
IDCG	Ideal Discounted Cumulative Gain
IDF	Inverse Document Frequency
JSON	JavaScript Object Notation
LLM	Large Language Model
MAB	Multi-Armed Bandit
MF	Matrix Factorization
MSE	Mean Squared Error
NDCG	Normalized Discounted Cumulative Gain
NER	Named Entity Recognition
NLP	Natural Language Processing
Pir	Passive intent recognition agent
POI	Point of Interest
POS	Part-of-Speech
PTM	Pretrained language Model (used for query generation)
RFC	Request for Comments
RS	Recommender System
SOTA	State of the Art
SVD	Singular Value Decomposition
Sew	Supplement and Widen agent
TF	Term Frequency
TF–IDF	Term Frequency–Inverse Document Frequency
UGC	User-Generated Content

References

1. Phillips, H. Great library of Alexandria. *Library philosophy and practice* **2010**.
2. Armstrong, R.; Freitag, D.; Joachims, T.; Mitchell, T.; et al. Webwatcher: A learning apprentice for the world wide web. In Proceedings of the AAAI Spring symposium on Information gathering from Heterogeneous, distributed environments. Stanford, 1995, Vol. 93, p. 107.
3. Meituan. Announcement of the Results for the Year Ended December 31, 2023, 2024. Accessed: 2024-05-19.
4. Harper, F.M.; Konstan, J.A. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* **2015**, *5*, 1–19.
5. Li, J.; Zhang, W.; Wang, T.; Xiong, G.; Lu, A.; Medioni, G. GPT4Rec: A generative framework for personalized recommendation and user interests interpretation. *arXiv preprint arXiv:2304.03879* **2023**.
6. Ko, H.; Lee, S.; Park, Y.; Choi, A. A survey of recommendation systems: recommendation models, techniques, and application fields. *Electronics* **2022**, *11*, 141.
7. Resnick, P.; Iacovou, N.; Suchak, M.; Bergstrom, P.; Riedl, J. Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of the Proceedings of the 1994 ACM conference on Computer supported cooperative work, 1994, pp. 175–186.
8. Goldberg, D.; Nichols, D.; Oki, B.M.; Terry, D. Using collaborative filtering to weave an information tapestry. *Communications of the ACM* **1992**, *35*, 61–70.
9. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the Proceedings of the 10th international conference on World Wide Web, 2001, pp. 285–295.
10. CarlKadie, J.B.D. Empirical analysis of predictive algorithms for collaborative filtering. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA* **1998**, *98052*.
11. Pearson, K. VII. Mathematical contributions to the theory of evolution.—III. Regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* **1896**, pp. 253–318.
12. Xu, L.; Zhang, J.; Li, B.; Wang, J.; Cai, M.; Zhao, W.X.; Wen, J.R. Prompting Large Language Models for Recommender Systems: A Comprehensive Framework and Empirical Analysis. *arXiv preprint arXiv:2401.04997* **2024**.
13. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In Proceedings of the Advances in Neural Information Processing Systems. Curran Associates, Inc., 2017, Vol. 30.

14. Sun, F.; Liu, J.; Wu, J.; Pei, C.; Lin, X.; Ou, W.; Jiang, P. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In Proceedings of the Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 1441–1450.
15. Khorasani, S.M.H.; Huber, S.; Falk, J.; Jannach, D. Transformers4Rec: Bridging the Gap between NLP and Sequential/Session-Based Recommendation. In Proceedings of the Fifteenth ACM Conference on Recommender Systems, 2021, pp. 143–149.
16. Anderson, C. *The Long Tail: Why the Future of Business is Selling Less of More*; Hyperion: New York, 2006.
17. Brynjolfsson, E.; Hu, Y.J.; Smith, M.D. From Niches to Riches: Anatomy of the Long Tail. *MIT Sloan Management Review* **2006**, *47*, 67–71.
18. Steck, H. Embarrassingly shallow autoencoders for sparse data. In Proceedings of the The World Wide Web Conference, 2019, pp. 3251–3257.
19. Cauchy, A.L. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences* **1847**, *25*, 536–538.
20. Robbins, H.; Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* **1951**, *22*, 400–407. https://doi.org/10.1214/aoms/1177729586.
21. Jones, K.S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* **1972**, *28*, 11–21.
22. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* **2019**.
23. Pennington, J.; Socher, R.; Manning, C. Glove: Global vectors for word representation. In Proceedings of the Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.
24. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* **2017**, *5*, 135–146.
25. Wang, H.; Li, J.; Wu, H.; Hovy, E.; Sun, Y. Pre-trained language models and their applications. *Engineering* **2022**.
26. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* **2020**.
27. Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science* **2024**, *18*, 1–26.
28. Crockford, D. The application/json Media Type for JavaScript Object Notation (JSON). Internet Engineering Task Force, 2006. RFC 4627.
29. Robbins, H. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society* **1952**, *58*, 527–535.
30. Star Wars: Episode I - The Phantom Menace. Internet. Accessed: 2023-05-25.
31. Guardians of the Galaxy. Internet. Accessed: 2023-05-25.
32. Star Wars: Episode VII - The Force Awakens. Internet. Accessed: 2023-05-25.
33. The Mandalorian. Internet. Accessed: 2023-05-25.
34. GroupLens Research. GroupLens Research Project. https://grouplens.org/. Accessed: 2024-05-27.
35. GroupLens Research. MovieLens 100K Dataset, 1998. Accessed: 2024-05-27.
36. GroupLens Research. MovieLens 1M Dataset, 2003. Accessed: 2024-05-27.
37. GroupLens Research. MovieLens 10M Dataset, 2009. Accessed: 2024-05-27.
38. GroupLens Research. MovieLens 20M Dataset, 2016. Accessed: 2024-05-27.
39. Herlocker, J.L.; Konstan, J.A.; Terveen, L.G.; Riedl, J.T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* **2004**, *22*, 5–53.
40. Järvelin, K.; Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. In Proceedings of the ACM Transactions on Information Systems (TOIS). ACM New York, NY, USA, 2002, Vol. 20, pp. 422–446.
41. Sutton, R.S.; Barto, A.G. Reinforcement Learning: An Introduction, Cambridge, MA, USA, 1998. Chapter 2.3: Epsilon-Greedy Methods.
42. Lattimore, T.; Szepesvári, C. *Bandit Algorithms*; Cambridge University Press: Cambridge, UK, 2020.
43. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* **2002**, *47*, 235–256. Upper Confidence Bound (UCB) Algorithm.