SCoRD: Semantic-Assisted Continual Retriever-Reranker Distillation for LLM-Based Recommendation

Seunghyun Baek*
Korea University
Seoul, Republic of Korea
seunghb320@korea.ac.kr

Gyuseok Lee*
University of Illinois at Urbana-Champaign
Champaign, IL, USA
gyuseok2@illinois.edu

Seunghan Lee
Korea University
Seoul, Republic of Korea
seunghanlee@korea.ac.kr

Wonbin Kweon
Sungkyunkwan University
Suwon, Republic of Korea
wonbinkweon@skku.edu

Dong Wang
University of Illinois at Urbana-Champaign
Champaign, IL, USA
dwang24@illinois.edu

SeongKu Kang†
Korea University
Seoul, Republic of Korea
seongkukang@korea.ac.kr

arXiv:2608.19998v1 [cs.IR] 20 Aug 2026

Abstract

Recommendation systems increasingly adopt a two-stage pipeline, where an ID-based retriever retrieves candidates and an LLM-based reranker refines their rankings. To improve retrieval quality, reranker-to-retriever distillation is commonly used to transfer the reranker's knowledge to the retriever. For practical deployment, however, this pipeline must continually adapt to evolving interests and incoming interactions. A naive solution is to repeatedly update the LLM reranker and distill its latest knowledge, but this incurs prohibitive costs. Updating the retriever alone is cheaper, but its limited capacity makes adaptation from sparse data difficult. We propose SCoRD, a continual knowledge distillation framework for LLM-based reranking pipelines under a non-stationary data stream. SCoRD introduces a *semantic reasoning assistant* that distills the LLM's ability to infer underlying user intents into reusable intent-level guidance. It selectively distills reranker knowledge to the retriever on low-confidence sequences, guides retriever-only updates without repeated LLM inference, and feeds retriever-derived representations and intent-drift signals back to the reranker. Experiments on real-world datasets show that SCoRD enables effective and efficient retriever-reranker co-adaptation.

CCS Concepts

• **Information systems** → **Recommender systems**.

Keywords

Knowledge Distillation, Retrieve-Rerank pipeline, Recommendation

ACM Reference Format:

Seunghyun Baek, Gyuseok Lee, Seunghan Lee, Wonbin Kweon, Dong Wang, and SeongKu Kang. 2026. SCoRD: Semantic-Assisted Continual Retriever-Reranker Distillation for LLM-Based Recommendation. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM '26), November 07–11, 2026, Rome, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3799682.3840906

*Both authors contributed equally to this research.
†Corresponding author.

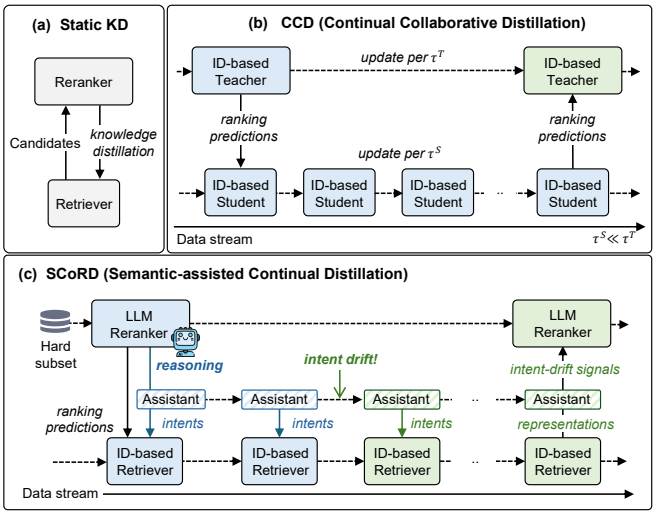

Figure 1: A conceptual comparison of (a) static distillation, (b) CCD, and (c) our approach. Best viewed in color.

1 Introduction

Recently, large language models (LLMs) have advanced recommendation by using world knowledge to reason over interaction sequences and capture semantic user preferences [17, 30, 59]. However, directly applying LLMs on large numbers of users and items incurs prohibitive inference costs [10, 61, 72]. Recent systems therefore adopt a retrieve-then-rerank pipeline: (1) an ID-based retriever (e.g., SASRec [24]) filters the vast item space to a small candidate set, and (2) an LLM-based reranker then reorders them [9, 13, 41, 54, 70]. This balances scalability with ranking precision, enabling the practical deployment of LLMs in recommendation [65].

As rerankers possess greater model capacity and richer ranking signals than lightweight retrievers, knowledge distillation (KD) has been widely employed to transfer these capabilities [5, 18, 37, 60]. Specifically, the reranker (teacher) transfers its ranking knowledge (e.g., ranked lists or ranking scores) into the retriever (student),

This work is licensed under a Creative Commons Attribution 4.0 International License.

CIKM '26, Rome, Italy

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2539-5/2026/11
https://doi.org/10.1145/3799682.3840906

enabling the retriever to surface high-quality candidate items (Figure 1a). Subsequently, these refined candidates directly improve the reranker's final performance, forming a mutually reinforcing relationship. Consequently, compared with optimizing the two modules independently, KD improves not only retriever but also the effectiveness of the retrieve-then-rerank pipeline as a whole [48].

However, applying such KD under a non-stationary data stream is more challenging than one-time distillation in static settings [29]. In real-world applications, new users, items, and interactions continuously arrive, while user preferences and item popularity evolve over time [33]. This necessitates both the retriever and reranker to adapt accordingly. A naive extension of KD would repeatedly update the reranker on each incoming data and distill its latest knowledge to the retriever. However, this strategy is impractical, as it requires repeated LLM reranker updates and inference, incurring prohibitive computational overhead [14, 45].[1]

In practice, the LLM reranker cannot be updated as frequently as the lightweight retriever. This creates a dilemma. If KD is performed without updating the reranker, its supervision becomes stale because the reranker remains tied to an older data distribution. Consequently, the retriever receives outdated guidance and may fail to reflect recent trends or incorporate newly arrived users and items. Alternatively, one may update the retriever alone between reranker update cycles. Although viable, it leaves the capacity-constrained retriever to learn only from limited interaction data alone, often leading to suboptimal performance. Thus, continual KD requires a mechanism that preserves up-to-date guidance for the retriever without repeatedly updating or invoking the LLM reranker.

One recent work [29] addresses a related problem through continual collaborative distillation (CCD), which decouples teacher and student update cycles (Figure 1b). In CCD, the teacher periodically distills its ranking predictions to the student at longer intervals (e.g., weekly), while the student adapts to incoming data at shorter intervals (e.g., daily). To mitigate student-side performance degradation, CCD uses the student's high-ranked predictions to augment sparse interactions. The updated student predictions are then used to augment interaction data for the next teacher update. Thus, CCD establishes a continual KD framework over data streams, where the teacher and student exchange ranking predictions while allowing the student to adapt between teacher update cycles.

Although CCD offers a practical update scheme for continual KD, it is designed for ID-based teacher-student models. In such settings, the teacher and student have a relatively moderate capacity gap and learn from similar collaborative signals, making direct prediction distillation effective. However, LLM-based reranking pipelines depart from this setting, as the reranker and retriever differ fundamentally in both model capacity and knowledge source. This raises unresolved design questions beyond the scope of CCD.

First, how can limited LLM supervision be used effectively? Unlike ID-based teachers, the LLM reranker incurs substantially higher computational costs, making uniform distillation over all incoming sequences infeasible. Thus, LLM-based distillation should focus on sequences where the retriever most needs guidance, while providing sufficiently informative signals to enable the retriever to learn effectively from only a small subset.

Second, how can LLM semantic reasoning be reused during retriever-only updates? The LLM reranker's predictive strength stems not merely from producing ranking scores, but from reasoning over interaction sequences to infer underlying user intents. However, this reasoning is accessible only when the LLM is invoked for distillation, leaving the continually updated retriever without semantic guidance until the next distillation. Thus, this reasoning capability should be distilled into a reusable form that continuously assists the retriever without repeated LLM inference.

Third, how can retriever's knowledge support LLM reranker updates? As the retriever is updated more frequently, it captures recent collaborative patterns and evolving user interests earlier than the LLM reranker. During reranker updates, relying solely on the LLM's own reasoning over new interactions is inefficient and may overlook these up-to-date signals already captured by the retriever. Thus, the reranker should use retriever-side knowledge to keep its semantic reasoning aligned with recent behavioral patterns.

As a solution, we propose **SCoRD**, a Semantic-assisted Continual Retriever-reranker Distillation framework (Figure 1c). SCoRD redesigns the reranker-retriever knowledge interface with a *Semantic Reasoning Assistant*, a lightweight module that distills the LLM's reasoning ability to infer underlying user intents. Rather than using the LLM merely as a source of ranking predictions, the assistant converts its semantic reasoning into reusable intent-level guidance.

SCoRD addresses the three design questions through three stages.

First, during LLM-to-retriever KD, SCoRD selectively applies distillation to sequences where the retriever exhibits low prediction confidence. For these hard sequences, the reranker provides ranking predictions to the retriever, while also distilling LLM-inferred intents into the assistant, such as "*budget-friendly electronics*" or "*upgrading a gaming setup*". The assistant then converts these intents into semantic guidance for the retriever. Together, these prediction and intent signals provide richer supervision, allowing the retriever to learn effectively from a small LLM-supervised subset.

Second, during retriever-only updates, the assistant infers underlying user intents that best explain newly arriving sequences without additional LLM inference. To improve their reliability, SCoRD reinforces intent signals strongly shared across behaviorally similar sequences. The resulting intents enhance retriever prediction and provide additional learning signals for capturing evolving user interests, helping the retriever adapt to new data.

Third, during reranker updates, SCoRD transfers the retriever's up-to-date behavioral knowledge back to the LLM reranker. This knowledge is conveyed through retriever-derived representations and intent-drift signals, helping the reranker align its semantic reasoning with recent collaborative patterns.

Our contributions are summarized as follows:

- We highlight the underexplored challenges of updating LLM-based reranking pipelines over evolving data streams. To the best of our knowledge, we are the first to formulate this problem for ID-based retriever and LLM reranker co-adaptation.
- We propose SCoRD framework that facilitates continual retriever-reranker knowledge transfer through a semantic reasoning assistant, enabling efficient and effective co-adaptation.

[1] In our experiments, the LLM reranker requires 3.5–9.4× longer training time and 72.4–144.4× longer inference time than the ID-based retriever across datasets.

- We conduct extensive experiments on three real-world datasets, showing that SCoRD improves both retrieval and reranking performance over state-of-the-art baselines.

2 Related Work

Retriever-reranker based recommendation. Two-stage retrieve-then-rerank pipelines have become a practical solution for scalable recommendation [4, 15]. An ID-based retriever efficiently narrows the item space to a small candidate set, which a more expressive reranker then refines with precise scoring. Early rerankers rely on ID-based models, scoring candidates by their feature-level relevance to user history [62], or jointly encoding the candidate list to capture inter-item dependencies [2, 46]. More recently, LLMs have emerged as powerful rerankers by leveraging semantic reasoning over user interactions [9, 70], substantially improving ranking quality. However, existing pipelines assume static settings, leaving their co-adaptation to an evolving data stream underexplored.

Knowledge distillation in recommendation. KD has been widely adopted to transfer knowledge from a large teacher to a compact student [19, 20, 22, 23, 53]. Early work focuses on ranking-based distillation, training the student to mimic the teacher's ranked outputs [28, 34]. Subsequent work explores representation-level distillation to provide richer structural knowledge [20, 21]. With the rise of LLMs, KD has been extended to leverage LLMs as teachers for lightweight models [5, 8], and to bridge the gap between semantic and collaborative signals [25, 52]. In retriever-reranker pipelines, KD further plays a crucial role by distilling knowledge from the reranker into the retriever, improving both retrieval quality and overall pipeline performance [15, 48]. Yet, these methods assume one-time distillation; their extension to a non-stationary data stream remains underexplored. CCD [29] pioneers continual KD via ranking prediction, but focuses on ID-based teacher-student models. It does not address LLM-based reranking pipelines, where the two modules exhibit substantial asymmetry in model capacity and knowledge source.

Continual learning for recommendation. Continual learning (CL) updates models incrementally as new data arrives [27, 36]. The goal of CL is to balance knowledge acquisition (plasticity) and retention (stability) over time [12, 68]. Existing approaches fall into two categories. Regularization-based methods constrain model updates in the parameter space, discouraging the parameters from deviating drastically from previously learned values [56, 57, 64]. Replay-based methods store representative historical interactions and reuse them during updates [47, 71, 73]. More recently, CL has been extended to LLM-based recommendation through parameter-efficient adaptation [51, 69], such as low-rank adaptation [14] with regularization toward its past state [69]. However, existing CL studies mainly focus on single-model updates; how to integrate CL and KD for retriever-reranker co-adaptation remains an open question.

In sum, prior work has studied the above three directions largely in isolation, leaving their intersection underexplored. This gap is practically important because the retriever and reranker must co-adapt over an evolving data stream, while repeated LLM updates and distillation incur prohibitive costs. This work addresses this gap.

3 Problem Formulation

Definition 1 (Continual Sequential Recommendation). Following [29, 32, 57, 64], the data stream $\mathcal{D}$ is viewed as consecutive data blocks $[D_0, D_1, \ldots, D_k, \ldots]$. Each block D_k contains interaction sequences observed during a specific time period τ. Here, D_0 serves as the base block used to build the model before continual updates on subsequent blocks. Let $\mathcal{U}_k$ and $\mathcal{I}_k$ be the sets of users and items appearing in D_k, respectively. For each user $u \in \mathcal{U}_k$, let $S_u^{(k)} = \{i_1, \ldots, i_{|S_u^{(k)}|}\}$ be the interaction sequence within D_k. At k-th block, the system updates its model using D_k, without directly accessing the previous blocks $D_{<k}$. The goal is to predict the next item over the accumulated item set given the current-block sequence, i.e., $\arg\max_{i \in \mathcal{I}_{\leq k}} P(i \mid S_u^{(k)})$.

Definition 2 (Retrieval-Reranking Pipeline). We adopt a retrieval-reranking recommendation pipeline [9, 13, 70]. Let M_S be a lightweight ID-based retriever and M_T be a LLM-based reranker. For each user, M_S first identifies a set of top-N candidate items, which are then reordered by M_T to produce the final recommendation list.

Definition 3 (Asynchronous Update Cycle of Retriever and Reranker). The retriever and reranker differ substantially in model size, training cost, and inference time. Thus, the lightweight retriever can be updated frequently to reflect new interactions, whereas frequently updating the reranker is impractical. Following the update setting of [29], we consider an asynchronous update cycle: the retriever M_S is updated at a short interval τ_S (e.g., daily), while the reranker M_T is updated at a longer interval τ_T (e.g., weekly). Under this cycle, the two modules must co-adapt over time despite their different update frequencies.

We define each data block based on the teacher update cycle; thus, the block interval is set to $\tau = \tau_T$.

Problem definition. We aim to enable co-adaptation of the ID-based retriever M_S and the LLM reranker M_T over a non-stationary data stream, while reducing prohibitive LLM update and distillation costs. Under the asynchronous update cycle, this co-adaptation repeatedly performs three stages: (1) LLM-to-retriever distillation, (2) retriever update at τ_S, and (3) reranker update at τ_T.

4 SCoRD Framework

We first construct the semantic reasoning assistant, the core of SCoRD, using the base block D_0 (§4.1). SCoRD then uses the assistant for continual co-adaptation through three stages: reranker-to-retriever KD (§4.2), retriever update (§4.3), and reranker update (§4.4). Finally, we discuss the efficiency of SCoRD (§4.5).

4.1 Semantic Assistant Construction on D_0

Before continual update begins, SCoRD constructs the assistant $\mathcal{A}$ on the base block D_0. The purpose of $\mathcal{A}$ is to retain semantic knowledge from LLMs and provide continuous guidance to the retriever during subsequent updates. A common approach to provide LLM guidance to an ID-based model is to use LLM-derived outputs, such as ranking results [5, 60] or textual embeddings [5, 39], as additional input features or supervision. Although effective in static settings, this approach is impractical for continual updates over data streams, as it requires repeated LLM calls to encode newly arriving data.

Given that the LLM's strength lies in inferring underlying intents from interactions, we distill this capability into a compact module. One may train a smaller language model to generate intents using LLM-inferred intents as supervision. However, this leads to a model-size dilemma: small models may lack the generation ability to produce reliable intents, whereas larger ones undermine efficiency.

To address this, we redesign the form of LLM reasoning knowledge from *free-form generation* to *memory-based selection*. Specifically, we organize LLM-inferred intents as a dynamic intent memory, a discrete set of intent units that expands along the data stream. Given each sequence, the assistant selects the most relevant intents from this memory. For example, the memory may include intent units such as "budget-friendly electronics" or "upgrading a gaming setup" from previous interactions, which can later be used to comprehend newly arriving sequences. This design provides continuous semantic guidance without frequent LLM calls, while reducing the hallucination risks from free-form generation.

Below, we describe memory construction and assistant training on the base block D_0. The overview is presented in Figure 2.

4.1.1 ***Dynamic intent memory construction.*** Let $\mathcal{G}$ be a dynamic intent memory that incrementally expands along the data stream, initialized as $\mathcal{G}^{(0)} = \emptyset$. For each user sequence $S_u^{(0)}$ in the base block D_0, we construct the memory through three steps: (1) intent inference, (2) intent verification, and (3) memory update.

Intent inference. We instruct the LLM to infer intents by *selecting* suitable ones from the current memory or *proposing* new intents when no existing intent sufficiently explains the sequence:

$$\hat{\mathcal{G}}_u = \text{LLM}\big((S_u^{(0)}, \mathcal{G}^{(0)}); \mathcal{P}_{\text{intent}}\big), \tag{1}$$

where $\hat{\mathcal{G}}_u$ denotes the candidate intents for user u, and $\mathcal{P}_{\text{intent}}$ is the corresponding prompt.[2]

Intent verification. To obtain reliable intents, we evaluate whether the inferred intents can explain the user's observed behavior. Following the verification task of [31, 38], we hold out the last item of S_u and ask the LLM to identify it among $N+1$ candidates (one held-out item and N random negatives), given the remaining sequence and $\hat{\mathcal{G}}_u$. We set $N=5$.

If the LLM selects the correct item, the inferred intents are accepted. Otherwise, we provide the failed prediction as feedback and ask the LLM to revise the intents. This repeats until the verification succeeds, up to three trials. If it still fails, we do not record the intent for user u; the user may be reconsidered in a future block (§4.2.1), after more interactions have accumulated.

Memory update. Finally, the verified intents are incorporated into the dynamic memory:

$$\mathcal{G}^{(0)} \leftarrow \mathcal{G}^{(0)} \cup \mathcal{G}_u, \tag{2}$$

where $\mathcal{G}_u$ denotes the verified intents for user u. Existing intents are reused, while newly identified intents are added to expand the memory. Thus, $\mathcal{G}^{(0)}$ grows only when the current memory lacks suitable intents for explaining a user sequence, preventing unnecessary expansion while covering diverse interactions.

[2]The key instruction is: "Given $S_u^{(0)}[:-1]$ and $\mathcal{G}^{(0)}$, select the intents that best explain the user's underlying preferences. If no suitable intent exists, propose new intents".

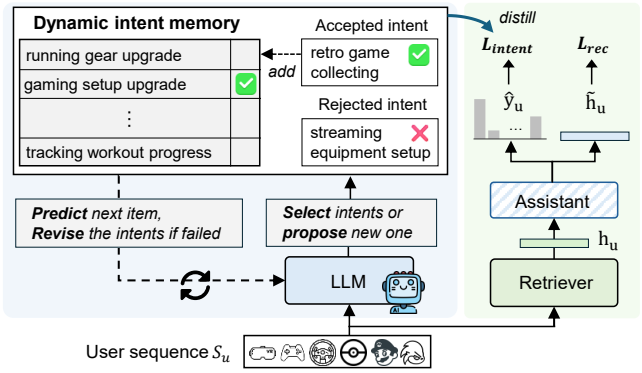

Figure 2: Semantic reasoning assistant construction process.

As a result, we obtain the initial dynamic intent memory $\mathcal{G}^{(0)}$ and the verified intent assignments $\mathcal{G}_u$ for users in the base block.

4.1.2 ***Joint training of assistant and retriever.*** Given the initial memory $\mathcal{G}^{(0)}$ and the intent assignments $\{\mathcal{G}_u\}_{u\in\mathcal{U}_0}$, we train the assistant $\mathcal{A}$ together with the retriever M_S. The goal is to distill the LLM's intent-inference capability into $\mathcal{A}$, so that it can predict the underlying intents and provide semantic guidance to the retriever. As a sequence may have multiple intents, we formulate this intent prediction as a multi-label classification over $\mathcal{G}^{(0)}$.

Memory-based intent prediction. For each user sequence S_u, the retriever produces a sequence representation $\mathbf{h}_u = M_S(S_u) \in \mathbb{R}^d$.[3] To predict relevant intents from the memory, $\mathcal{A}$ adopts a query-key-value design [55], where $\mathbf{h}_u$ serves as the query and each intent $c \in \mathcal{G}^{(0)}$ is represented by a learnable embedding $\mathbf{p}_c \in \mathbb{R}^d$:

$$\mathbf{q}_u = \mathbf{W}_q \mathbf{h}_u, \quad \mathbf{k}_c = \mathbf{W}_k \mathbf{p}_c, \quad \mathbf{v}_c = \mathbf{W}_v \mathbf{p}_c, \tag{3}$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ are learnable projection matrices. The user-intent relevance is computed as: $\hat{y}_{u,c} = \sigma(\mathbf{q}_u^\top \mathbf{k}_c)$, where σ is the sigmoid function.

Semantic-guided representation. While the ID-based retriever captures collaborative patterns, it lacks explicit semantic knowledge of user intents. Using the intent relevance scores, $\mathcal{A}$ converts the intents into semantic guidance for the retriever. Specifically, it aggregates the values of relevant intents and injects them into the original sequence representation:

$$\tilde{\mathbf{h}}_u = \text{LayerNorm}\left(\mathbf{h}_u + \text{Dropout}\left(\sum_{c\in\mathcal{G}^{(0)}} \phi(\hat{y}_{u,c})\mathbf{v}_c\right)\right), \tag{4}$$

where $\phi(n) = n \cdot \mathbb{I}[n > x]$ retains scores above threshold x. We set $x = 0.5$ in this work. This produces a semantic-guided representation $\tilde{\mathbf{h}}_u$ by focusing on the intents most relevant to the current sequence.

Optimization. We jointly optimize the assistant $\mathcal{A}$, parameterized by $\{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \{\mathbf{p}_c\}_{c\in\mathcal{G}}\}$, and the retriever M_S with two objectives: reasoning distillation and semantic-guided recommendation.

(1) **Reasoning distillation**: This aligns the predicted user-intent relevance scores with the LLM-inferred intents. Given $\mathcal{G}_u$, we define the binary label $y_{u,c} = \mathbb{I}[c \in \mathcal{G}_u]$ and optimize:

$$\mathcal{L}_{\text{intent}} = -\frac{1}{|\mathcal{G}^{(0)}|} \sum_{c\in\mathcal{G}^{(0)}} \Big[y_{u,c} \log \hat{y}_{u,c} + (1 - y_{u,c}) \log(1 - \hat{y}_{u,c}) \Big]. \tag{5}$$

[3]For sequential retrievers (e.g., SASRec [24]), $\mathbf{h}_u$ is the hidden state at the last position.

(2) **Semantic-guided recommendation**: This aligns the semantic-guided representation $\tilde{\mathbf{h}}_u$ with the next-item prediction task:

$$\mathcal{L}_{\text{rec}} = -\sum_{S_u \in D_0} \left[\log \sigma(\mathbf{e}_{i_{\text{pos}}}^\top \tilde{\mathbf{h}}_u) + \sum_{i_{\text{neg}} \in \mathcal{N}_u^-} \log(1 - \sigma(\mathbf{e}_{i_{\text{neg}}}^\top \tilde{\mathbf{h}}_u)) \right], \quad (6)$$

where $\mathbf{e}_i$ is the retriever item embedding, i_{pos} is the ground-truth next item, and $\mathcal{N}_u^-$ is the negative item set.

The joint training objective is:

$$\mathcal{L}_{\mathcal{A},M_S} = \mathcal{L}_{\text{rec}} + \lambda_{\text{intent}} \mathcal{L}_{\text{intent}}, \quad (7)$$

where λ_{intent} is the loss-balancing hyperparameter. This jointly trains $\mathcal{A}$ to predict LLM-derived intents and M_S to use the resulting semantic guidance for next-item prediction. The reranker M_T is also warmed up on D_0 using the standard cross-entropy loss.

The following sections describe how $\mathcal{A}$ supports continual distillation. Assume that the retriever and reranker have been trained up to block D_{k-1}. Each new cycle starts with distilling the recent reranker knowledge to the retriever. The distilled retriever then serves as the starting point for subsequent updates on D_k.

4.2 Stage 1: Reranker-to-Retriever KD

In this stage, we use the reranker trained up to D_{k-1} to update the retriever and assistant before processing D_k, so that they start the new block with recent reranker knowledge. We first identify hard sequences based on the retriever's confidence, so that LLM guidance is concentrated where it is most needed. For these sequences, we distill two forms of supervision from the LLM reranker: fine-grained ranking signals to the retriever and inferred intents to the assistant.

4.2.1 ***Confidence-based hard sequence selection.*** Instead of distilling over all sequences, we prioritize sequences where the retriever has low prediction confidence. We estimate this confidence from sequence-item alignment: a well-encoded sequence representation should be closely aligned to the items in the sequence. For each sequence $S_u \in D_{k-1}$, we compute the confidence score:

$$\rho_u = \frac{1}{|S_u|} \sum_{i \in S_u} \sigma(\mathbf{e}_i^\top \tilde{\mathbf{h}}_u). \quad (8)$$

A lower ρ_u indicates that the retriever cannot confidently account for the in-sequence items. We select the bottom $r\%$ of sequences based on ρ_u to form D_{k-1}^{hard}, thereby reducing the cost of distillation. This avoids redundant LLM distillation on sequences that the retriever already predicts well. Semantic knowledge from these hard sequences is later propagated to broader sequences in Stage 2 (§4.3.1).

4.2.2 ***Semantic-assisted ranking distillation.*** Given the selected hard sequences D_{k-1}^{hard}, we first update the intent memory and then perform ranking distillation with semantic guidance from $\mathcal{A}$.

Intent memory update. For each hard sequence, we update the previous memory $\mathcal{G}^{(k-1)}$ using the same inference-verification-update procedure in §4.1.1. This yields the updated memory $\mathcal{G}^{(k)}$ and verified intent assignments $\mathcal{G}_u$ for users in the hard sequence set, with new intents added when needed.

Ranking distillation with semantic guidance. We adopt the listwise ranking distillation loss used in CCD [29], which trains the student to imitate the teacher's ranking order. Our distinction is that the distillation loss is computed with the semantic-guided representation $\tilde{\mathbf{h}}_u$. This allows $\mathcal{A}$ to be jointly optimized with the retriever while guiding ranking transfer with intent semantics.

Let π_u denote the item ranking list produced by the reranker for user u, and let $\pi_u(j)$ be the j-th item in the list. We define the top-N permutation probability [63] using the retriever's prediction scores and minimize its negative log-likelihood:

$$\mathcal{L}_{\text{KD}} = -\sum_{S_u \in D_{k-1}^{\text{hard}}} \log \prod_{j=1}^{N} \frac{\exp(\tilde{s}_{u,\pi_u(j)})}{\sum_{i=j}^{|\pi_u|} \exp(\tilde{s}_{u,\pi_u(i)})}, \quad (9)$$

where $\tilde{s}_{u,i} = \tilde{\mathbf{h}}_u^\top \mathbf{e}_i$ is the semantic-guided item prediction score of the retriever.

4.2.3 ***Optimization of stage 1.*** The overall objective is defined as:

$$\min_{\theta_S, \theta_{\mathcal{A}}} \mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{KD}} + \lambda_{\text{intent}} \mathcal{L}_{\text{intent}}, \quad (10)$$

where θ_S and $\theta_{\mathcal{A}}$ are the parameters of the retriever and assistant, respectively. $\mathcal{L}_{\text{intent}}$ is computed as in Eq. (5) using $\mathcal{G}^{(k)}$ and the verified intents for D_{k-1}^{hard}.

4.3 Stage 2: Continual Update of Retriever

In this stage, the retriever $M_S^{(k)}$ is continuously updated on the incoming block D_k every τ_S. The key challenge is enabling the capacity-constrained retriever to effectively incorporate new sequences and adapt to evolving interests. The assistant $\mathcal{A}$ is designed to provide continuous semantic guidance: given a sequence, it infers the underlying intents and supports semantic-guided prediction. However, $\mathcal{A}$ must also adapt, since new sequences lack verified intent labels and user interests may drift over time. We propose two new techniques to provide up-to-date semantic guidance.

4.3.1 ***Collaborative semantic assistant adaptation.*** $\mathcal{A}$ should accurately predict intents for sequences that have not been directly supervised with intent labels. To this end, we draw inspiration from self-training [11, 44], a well-established semi-supervised learning technique that improves generalization by using pseudo-labels on unlabeled data. However, directly using $\mathcal{A}$'s intent predictions as pseudo-intent labels can be unstable and prone to noise. Instead, we propose to incorporate intents from behaviorally similar sequences.

Specifically, for each user u, we retrieve the top-K behaviorally similar user sequences and construct the collaborative intent label $\tilde{y}_{u,c}$ by aggregating their predicted intent scores:

$$\tilde{y}_{u,c} = \frac{1}{2}\left(\hat{y}_{u,c} + \sum_{u' \in \mathcal{N}(u)} w_{u,u'} \cdot \hat{y}_{u',c} \right), \quad (11)$$

where $\mathcal{N}(u) = \text{TopK}(\{\cos(\mathbf{h}_u, \mathbf{h}_{u'})\}_{u' \in \mathcal{B}})$ denotes the top-K similar users in mini-batch $\mathcal{B}$, and $w_{u,u'}$ is the softmax-normalized cosine similarity over $\mathcal{N}(u)$. For newly arriving sequences, we compute the collaborative intent learning loss $\mathcal{L}_{\text{co-intent}}$ by replacing $y_{u,c}$ with $\tilde{y}_{u,c}$ in Eq. (5). This reinforces intent signals that are strongly and consistently shared across behaviorally similar sequences, providing more reliable semantic guidance.

4.3.2 ***Intent drift-aware negative sampling.*** As user interests evolve over time, some intents fade while new ones emerge. To capture such drift, negative sampling is critical: negatives drawn from items aligned with *faded intents* provide more effective training signals for adapting to evolving interests than random negatives.

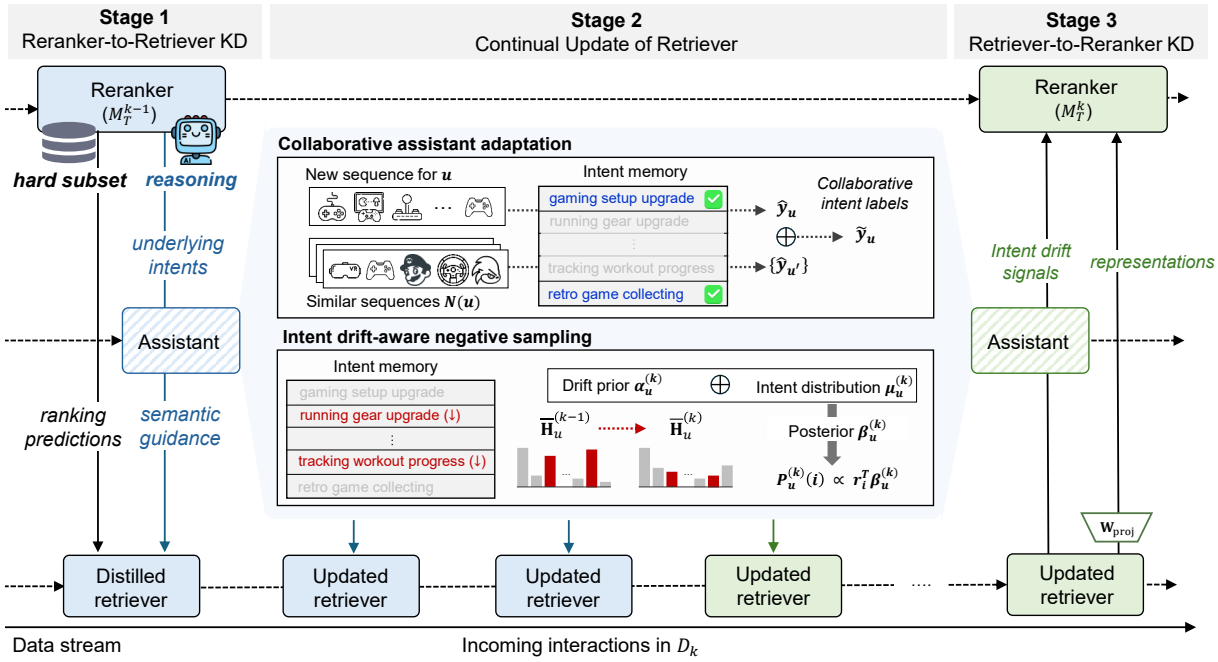

Figure 3: Overview of SCoRD framework. Best viewed in color.

To this end, we quantify how the user's intent distribution shifts across blocks and use this shift to guide negative sampling.

Intent histogram construction. We use the intent distribution as a semantic proxy for the user's interests. To estimate the current intent distribution, we construct a user intent histogram $\mathbf{H}_u^{(k)}$ by aggregating item-level intent relevance over the sequence $S_u^{(k)}$:

$$\mathbf{H}_u^{(k)} = \sum_{i \in S_u^{(k)}} \mathbf{r}_i, \quad \mathbf{r}_i = [\sigma((\mathbf{W}_q \mathbf{e}_i)^\top (\mathbf{W}_k \mathbf{p}_c))]_{c \in \mathcal{G}^{(k)}}, \tag{12}$$

where $\mathbf{r}_i \in \mathbb{R}^{|\mathcal{G}^{(k)}|}$ denotes the relevance vector between item i and intents in $\mathcal{G}^{(k)}$. We normalize it as $\overline{\mathbf{H}}_u^{(k)} = \mathbf{H}_u^{(k)} / \|\mathbf{H}_u^{(k)}\|_1$ and compare it across consecutive blocks to identify faded intents.

Faded-intent negative sampling. To identify faded intents, we first compute a faded-intent score $\boldsymbol{\alpha}_u^{(k)} \in \mathbb{R}^{|\mathcal{G}^{(k)}|}$ by comparing consecutive intent histograms. Intents whose normalized frequency decreases from block $k-1$ to k receive higher weights[4]:

$$\boldsymbol{\alpha}_u^{(k)} = \text{softmax}\left(\overline{\mathbf{H}}_u^{(k-1)} - \overline{\mathbf{H}}_u^{(k)}\right). \tag{13}$$

While this score captures intent drift, using it directly for negative sampling can be unstable because it is estimated only from sparse observed interactions. A decrease in an intent may reflect either a true fading interest or insufficient observations in the current block.

To obtain a more robust distribution, we treat $\boldsymbol{\alpha}_u^{(k)}$ as a drift prior and refine it with evidence from plausible unobserved items. Specifically, we consider the top-Q unobserved items $\mathcal{Q}_u$[5] from the retriever and compute their mean intent relevance: $\boldsymbol{\mu}_u^{(k)} = \frac{1}{Q} \sum_{i \in \mathcal{Q}_u} \mathbf{r}_i$. This reflects the intent distribution of retriever-plausible items. Inspired by Dirichlet-Multinomial conjugate updates [3], we combine the prior with this evidence to obtain a smoothed posterior:

$$\boldsymbol{\beta}_u^{(k)} = (\boldsymbol{\alpha}_u^{(k)} + \boldsymbol{\mu}_u^{(k)}) / \|\boldsymbol{\alpha}_u^{(k)} + \boldsymbol{\mu}_u^{(k)}\|_1. \tag{14}$$

We then sample negatives from top-Q unobserved items with probability proportional to their alignment with the posterior: $P_u^{(k)}(i) \propto \mathbf{r}_i^\top \boldsymbol{\beta}_u^{(k)}$, yielding intent-drift $\mathcal{N}_u^-$ for $\mathcal{L}_{\text{rec}}$ in Eq. (6). This probability becomes high when item i is strongly associated with the faded intents emphasized by $\boldsymbol{\beta}_u^{(k)}$, providing semantic drift-aware negatives that help the retriever adapt to evolving user preferences.

4.3.3 *Optimization of stage 2.*

The overall objective is:

$$\min_{\theta_S, \theta_{\mathcal{A}}} \mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{intent}} \mathcal{L}_{\text{co-intent}} + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}}, \tag{15}$$

where $\mathcal{L}_{\text{rec}}$ uses the intent drift-aware negatives, and $\mathcal{L}_{\text{co-intent}}$ uses collaborative pseudo-intent labels. We additionally adopt a regularization term widely used in continual learning [56, 57, 64, 67], which penalizes excessive shifts in representations to prevent catastrophic forgetting: $\mathcal{L}_{\text{reg}} = \sum_{u \in \mathcal{U}_{k-1} \cap \mathcal{U}_k} \|\mathbf{h}_u^{(k-1)} - \mathbf{h}_u^{(k)}\|_2^2$.

4.4 Stage 3: Retriever-to-Reranker KD

Once the cycle τ_T is due, the reranker is updated on D_k. For effective adaptation, we leverage up-to-date retriever knowledge to reduce the knowledge gap caused by the reranker's infrequent updates.

Retriever-side knowledge injection. Following [35], we incorporate item embeddings from the ID-based retriever into prompt construction. Specifically, each item embedding $\mathbf{e}_i$ from the retriever is projected into the LLM input embedding space as $\mathbf{x}_i = \mathbf{W}_{\text{proj}} \mathbf{e}_i$. The interaction history is then represented by the projected embeddings in the encoding prompt for recommendation.[6]

[4] $\overline{\mathbf{H}}_u^{(k-1)}$ is zero-padded over newly introduced intents in $\mathcal{G}^{(k)}$.
[5] We set Q=100 in our experiments.

[6] The encoding prompt is: "Given the interaction history $[\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_n}]$, extract representations for next-item recommendation."

The LLM reranker then produces a sequence representation from this prompt. We attach an item prediction layer [35], implemented as a linear layer with softmax output. Let $\mathbf{p}_u^{(k)}$ denote the item prediction distribution given $S_u^{(k)}$, where $\mathbf{p}_u^{(k)}[i]$ is the predicted probability of item i. We train the reranker with the standard cross-entropy loss for next-item prediction:

$$\mathcal{L}_{\text{CE}} = -\sum_{S_u^{(k)} \in D_k} \log \mathbf{p}_u^{(k)}[i_{\text{pos}}], \tag{16}$$

where i_{pos} is the ground-truth next item.

Drift-aware negative signals. To explicitly reflect intent drift captured by $\mathcal{A}$, we introduce an auxiliary cross-entropy loss $\mathcal{L}_{\text{drift}}$ that compares the ground-truth next item i_{pos} against intent-drift negatives $\mathcal{N}_u^-$. This is implemented by normalizing the softmax only over i_{pos} and $\mathcal{N}_u^-$, providing focused supervision against faded-intent items and complementing the standard full-item prediction loss.

4.4.1 ***Optimization of stage 3.*** The overall objective is:

$$\min_{\theta_T, \mathbf{W}_{\text{proj}}} \mathcal{L}_{\text{stage3}} = \mathcal{L}_{\text{CE}} + \lambda_{\text{drift}} \mathcal{L}_{\text{drift}}, \tag{17}$$

where θ_T denotes the trainable parameters of the reranker, comprising the LoRA adapters [14] and item prediction layer. λ_{drift} controls the strength of the drift-aware auxiliary loss.

4.5 Training and Inference Efficiency

Training. SCoRD addresses the computational bottleneck in continual KD: repeated LLM inference, distillation, and fine-tuning.

- **Stage 1.** LLM distillation is performed only on the top-r% hardest sequences in each block; we set $r = 20$ in our experiments.
- **Stage 2.** Frequent retriever updates are assisted by semantic guidance from $\mathcal{A}$, requiring no additional LLM calls.
- **Stage 3.** Reranker fine-tuning is performed only every τ_T, avoiding updates at every retriever cycle.

Together, these designs reduce LLM inference and update costs while maintaining continuous semantic guidance, making retriever–reranker co-adaptation over a non-stationary data stream practical. Figure 3 and Algorithm 1 present the overall procedure.

Inference. Given S_u, the retriever ranks all items using the semantic-guided representation $\tilde{\mathbf{h}}_u$, where relevant intents are selected by $\mathcal{A}$. Items are scored by $\tilde{\mathbf{h}}_u^\top \mathbf{e}_i$, yielding top-$N$ candidates for reranking.[7] The reranker then scores each candidate using the predicted distribution $\mathbf{p}_u$. Intent prediction adds negligible overhead to the total latency; we provide a detailed analysis in Section 5.2.3.

5 Experiments

5.1 Experimental Setup

Datasets. We use three real-world datasets: Books (Amazon) [42], Yelp [66], and Movies & TV (Amazon) [42], using the most recent 3, 5, and 2 years of interactions, respectively. Following existing CL setups [32, 57, 64], we simulate a non-stationary data stream by chronologically splitting each dataset into five interaction blocks: the base block D_0 contains the earliest 60% of interactions for initial training, while the remaining 40% is equally divided into four incremental blocks D_1–D_4 (10% each) for continual updates. For each block, we apply k-core filtering [50] with a user threshold of 5 for all datasets and item thresholds of 3, 3, and 2 for Books, Yelp, and Movies & TV, respectively. Each block consists of interaction sequences, where the last item serves as the test label, the second-to-last as the validation label, and the remaining items for training. Table 1 presents detailed statistics per block for each dataset, including the number of accumulated intents (i.e., $|\mathcal{G}^{(k)}|$).

[7] We set N=20 in our experiments.

Algorithm 1 SCoRD algorithm over data blocks

Input: $M_S^{(k-1)}$, $M_T^{(k-1)}$, $\mathcal{A}$ with intent memory $\mathcal{G}^{(k-1)}$
Output: Updated $M_S^{(k)}$, $M_T^{(k)}$, $\mathcal{A}$, $\mathcal{G}^{(k)}$
/ Base Block */*
if $k = 0$ **then**
  Train $M_T^{(0)}$ on D_0
  for each sequence $S_u^{(0)} \in D_0$ **do**
    Infer & verify intent → update memory $\mathcal{G}^{(0)}$ ▷ Eqs. (1–2)
  end for
  Jointly train $\mathcal{A}$ and $M_S^{(0)}$ on D_0 ▷ Eq. (7)
  return
end if
/ Incremental Block */*
Stage 1: Reranker-to-retriever KD
Construct D^{hard} and update $\mathcal{G}^{(k-1)} \to \mathcal{G}^{(k)}$
Update M_S and $\mathcal{A}$ ▷ Eq. (10)
Stage 2: Continual retriever update
for every τ_S **do**
  Enhance intent labels and sample intent-drift $\mathcal{N}_u^-$ ▷ Eqs. (11–14)
  Update M_S and $\mathcal{A}$ ▷ Eq. (15)
end for
Stage 3: Retriever-to-reranker KD
Project item embeddings and sample intent-drift $\mathcal{N}_u^-$
Update M_T ▷ Eq. (17)

Table 1: Data block statistics after preprocessing.

	Data Blocks	D_0 (60%)	D_1 (10%)	D_2 (10%)	D_3 (10%)	D_4 (10%)
Book	# users (new)	30,031(30,031)	1,529(218)	1,735(376)	2,021(510)	1,822(524)
	# items (new)	38,981(38,981)	2,051(1,599)	2,241(1,854)	2,649(2,116)	2,664(2,278)
	# interactions	358,600	12,693	14,957	17,673	16,644
	# intents	140	154	167	180	192
	Avg. Seq Length	11.94	8.30	8.62	8.74	9.14
	Sparsity	0.9997	0.9960	0.9962	0.9967	0.9966
Yelp	# users (new)	61,734(61,734)	1,270(463)	1,998(815)	3,005(1,420)	2,527(1,030)
	# items (new)	47,616(47,616)	1,982(144)	3,061(424)	4,407(608)	3,820(408)
	# interactions	744,167	10,184	16,169	25,688	21,421
	# intents	72	78	80	83	88
	Avg. Seq Length	12.05	8.02	8.09	8.55	8.48
	Sparsity	0.9997	0.9960	0.9974	0.9981	0.9978
Movies	# users (new)	20,380(20,380)	700(520)	955(751)	598(449)	627(459)
	# items (new)	37,591(37,591)	1,851(274)	2,425(390)	1,481(299)	1,494(297)
	# interactions	197,918	5,564	7,775	4,705	4,714
	# intents	61	63	64	65	66
	Avg. Seq Length	9.71	7.95	8.14	7.87	7.52
	Sparsity	0.9997	0.9957	0.9966	0.9947	0.9950

Evaluation protocol. Following prior CL studies [43, 57], models are trained on D_0 and continually updated on each incremental block without access to previous data. For each block, we evaluate both the reranker and the retriever using Hit Rate (H) [6] and NDCG (N) [16] at cutoffs {5, 10, 20}, following [29, 39, 60, 67]. Following [32], we report results on D_2–D_4, as preference drift is less pronounced in the early stages. While some CL studies [7, 29] adopt retained average ($\frac{1}{t}\sum_{i=1}^{t} a_{t,i}$, where $a_{t,i}$ is the performance on D_i after training on D_t) as a metric, recent work emphasizes plasticity in adapting to newly arriving data [67, 68]. Following this trend, we

Table 2: The overall performance comparison. * denotes $p < 0.05$ for the paired t-test on SCoRD with the best baseline.

	Book	After D_2						After D_3						After D_4					
		N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20
Reranker	Full-Batch	0.4579	0.5177	0.4716	0.5603	0.4910	0.6383	0.3612	0.4436	0.3824	0.5086	0.3992	0.5754	0.3293	0.4122	0.3567	0.4967	0.3774	0.5792
	Fine-Tune	0.4185	0.4638	0.4311	0.5026	0.4490	0.5746	0.4392	0.4933	0.4523	0.5343	0.4682	0.5983	0.4972	0.5416	0.5110	0.5850	0.5301	0.6623
	PISA	0.4465	0.4956	0.4632	0.5473	0.4816	**0.6208**	0.4502	0.5085	0.4658	0.5570	0.4853	**0.6351**	0.4981	0.5505	0.5138	0.5992	0.5318	0.6715
	Reloop2	0.4261	0.4756	0.4426	0.5270	0.4588	0.5916	0.4249	0.4852	0.4363	0.5202	0.4502	0.5761	0.4913	0.5517	0.5058	0.5968	0.5212	0.6594
	CCD	0.4477	0.5000	0.4623	0.5458	0.4807	0.6197	0.4324	0.4987	0.4475	0.5456	0.4657	0.6195	0.5072	0.5567	0.5218	0.6024	0.5403	0.6765
	LLMD4Rec	0.4515	0.4904	0.4677	0.5410	0.4846	0.6094	0.4456	0.4783	0.4598	0.5223	0.4751	0.5845	0.5249	0.5514	0.5393	0.5956	0.5531	0.6517
	CoT-Rec	0.4359	0.4752	0.4526	0.5273	0.4711	0.6020	0.4535	0.5004	0.4698	0.5510	0.4879	0.6243	0.5153	0.5640	0.5283	0.6048	0.5444	0.6700
	SCoRD	**0.4925***	**0.5606***	**0.5055***	**0.6009***	**0.5099***	0.6179	**0.4953***	**0.5761***	**0.5078***	**0.6141***	**0.5115***	0.6282	**0.5386***	**0.6172***	**0.5538***	**0.6635***	**0.5579***	**0.6795**
Retriever	Full-Batch	0.2424	0.3666	0.2893	0.5109	0.3218	0.6383	0.1942	0.3035	0.2391	0.4431	0.2726	0.5754	0.1853	0.2835	0.2328	0.4299	0.2706	0.5792
	Fine-Tune	0.1927	0.2967	0.2404	0.4449	0.2734	0.5746	0.2292	0.3539	0.2745	0.4927	0.3015	0.5983	0.3128	0.4463	0.3551	0.5764	0.3771	0.6623
	PISA	0.3244	0.4531	0.3550	0.5477	0.3737	**0.6208**	0.3003	0.4310	0.3394	0.5504	0.3608	**0.6351**	0.3487	0.4817	0.3851	0.5945	0.4047	0.6715
	Reloop2	0.1565	0.2627	0.2144	0.4424	0.2526	0.5916	0.1775	0.2893	0.2302	0.4538	0.2614	0.5761	0.3051	0.4537	0.3472	0.5823	0.3669	0.6594
	CCD	0.3158	0.4372	0.3492	0.5388	0.3698	0.6197	0.2944	0.4215	0.3306	0.5328	0.3527	0.6195	0.3787	0.5171	0.4111	0.6163	0.4264	0.6765
	LLMD4Rec	0.1914	0.3171	0.2433	0.4775	0.2770	0.6094	0.2115	0.3276	0.2570	0.4681	0.2866	0.5845	0.3201	0.4637	0.3559	0.5732	0.3759	0.6517
	CoT-Rec	0.2223	0.3407	0.2673	0.4797	0.2985	0.6020	0.2427	0.3703	0.2876	0.5085	0.3171	0.6243	0.2880	0.4327	0.3339	0.5729	0.3587	0.6700
	SCoRD	**0.4787***	**0.5606***	**0.4895***	**0.5942***	**0.4956***	0.6179	**0.4866***	**0.5740***	**0.4986***	**0.6108***	**0.5031***	0.6282	**0.5327***	**0.6210***	**0.5442***	**0.6561***	**0.5501***	**0.6795**

	Yelp	After D_2						After D_3						After D_4					
		N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20
Reranker	Full-Batch	0.3953	0.4119	0.4028	0.4350	0.4098	0.4631	0.3633	0.4178	0.3744	0.4518	0.3823	0.4827	0.2461	0.3025	0.2656	0.3630	0.2816	0.4264
	Fine-Tune	0.3381	0.3505	0.3436	0.3677	0.3527	0.4038	0.4083	0.4297	0.4138	0.4472	0.4199	0.4716	0.3758	0.3935	0.3817	0.4118	0.3906	0.4473
	PISA	0.3261	0.3402	0.3325	0.3599	0.3421	0.3983	0.3949	0.4158	0.3999	0.4311	0.4077	0.4624	0.3692	0.3844	0.3761	0.4061	0.3864	0.4476
	Reloop2	0.3342	0.3512	0.3416	0.3741	0.3490	0.4035	0.3885	0.4210	0.3942	0.4387	0.4013	0.4669	0.3672	0.4000	0.3733	0.4190	0.3818	0.4534
	CCD	0.3235	0.3425	0.3296	0.3615	0.3394	0.4006	0.3988	0.4185	0.4041	0.4351	0.4118	0.4660	0.3731	0.3865	0.3794	0.4064	0.3884	0.4424
	LLMD4Rec	0.3409	0.3522	0.3479	0.3741	0.3539	0.3980	0.4145	0.4273	0.4196	0.4432	**0.4268**	0.4720	0.3859	0.3979	0.3916	0.4157	0.3982	0.4419
	CoT-Rec	0.3413	0.3567	0.3478	0.3770	0.3574	0.4154	0.3869	0.4230	0.3944	0.4460	0.4028	**0.4798**	0.3664	0.4005	0.3734	0.4223	0.3820	**0.4572**
	SCoRD	**0.3505***	**0.3748***	**0.3568***	**0.3941***	**0.3642***	**0.4235***	**0.4150**	**0.4456***	**0.4202**	**0.4617***	0.4237	0.4756	**0.3942**	**0.4204***	**0.3982**	**0.4328***	**0.4008**	0.4431
Retriever	Full-Batch	0.2233	0.3358	0.2509	0.4197	0.2619	0.4631	0.1676	0.2744	0.2063	0.3934	0.2289	0.4827	0.1203	0.1964	0.1575	0.3119	0.1866	0.4264
	Fine-Tune	0.1881	0.2786	0.2123	0.3528	0.2253	0.4038	0.2427	0.3494	0.2671	0.4241	0.2792	0.4716	0.2169	0.3160	0.2418	0.3921	0.2560	0.4473
	PISA	0.2016	0.2821	0.2222	0.3454	0.2356	0.3983	0.2680	0.3586	0.2859	0.4140	0.2982	0.4624	0.2585	0.3352	0.2763	0.3895	0.2910	0.4476
	Reloop2	0.2017	0.2963	0.2245	0.3664	0.2340	0.4035	0.2573	0.3496	0.2797	0.4185	0.2921	0.4669	0.2733	0.3584	0.2909	0.4125	0.3013	0.4534
	CCD	0.2024	0.2853	0.2222	0.3464	0.2360	0.4006	0.2733	0.3633	0.2906	0.4167	0.3030	0.4660	0.2524	0.3345	0.2707	0.3909	0.2837	0.4424
	LLMD4Rec	0.2145	0.3183	0.2316	0.3702	0.2386	0.3980	0.2875	0.3751	0.3050	0.4290	0.3159	0.4720	0.2902	0.3684	0.3049	0.4132	0.3123	0.4419
	CoT-Rec	0.1950	0.2954	0.2186	0.3677	0.2307	0.4154	0.2002	0.3069	0.2350	0.4138	0.2518	**0.4798**	0.1996	0.3064	0.2294	0.3979	0.2447	**0.4572**
	SCoRD	**0.3389***	**0.3731***	**0.3464***	**0.3961***	**0.3532***	**0.4235***	**0.4068***	**0.4460***	**0.4105***	**0.4575***	**0.4150***	0.4756	**0.3885***	**0.4195***	**0.3927***	**0.4321***	**0.3955***	0.4431

	Movies	After D_2						After D_3						After D_4					
		N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20	N@5	H@5	N@10	H@10	N@20	H@20
Reranker	Full-Batch	0.5625	0.5871	0.5700	0.6100	0.5852	0.6717	0.4905	0.5236	0.5033	0.5639	0.5197	0.6309	0.4451	0.4808	0.4575	0.5192	0.4706	0.5725
	Fine-Tune	0.5647	0.5805	0.5715	0.6018	0.5883	0.6705	0.5218	0.5392	0.5338	0.5766	0.5508	0.6464	0.4972	0.5200	0.5087	0.5553	0.5248	0.6212
	PISA	0.5638	0.5781	0.5708	0.6002	0.5881	0.6713	0.5315	0.5476	0.5440	0.5865	0.5637	0.6662	0.5050	0.5239	0.5173	0.5616	0.5383	0.6471
	Reloop2	0.5613	0.5875	0.5698	0.6141	0.5837	0.6713	0.5336	0.5603	0.5423	0.5872	0.5589	0.6549	0.5036	0.5349	0.5108	0.5576	0.5257	0.6180
	CCD	0.5621	0.5756	0.5713	0.6043	0.5892	0.6774	0.5377	0.5568	0.5485	0.5907	0.5677	**0.6690**	0.4997	0.5200	0.5115	0.5561	0.5336	0.6463
	LLMD4Rec	0.5657	0.5814	0.5732	0.6047	0.5859	0.6570	0.5320	0.5533	0.5394	0.5766	0.5554	0.6415	0.5068	0.5310	0.5178	0.5655	0.5333	0.6290
	CoT-Rec	0.5658	0.5846	0.5737	0.6096	0.5903	**0.6783**	0.5354	0.5540	0.5437	0.5801	0.5643	0.6648	0.4990	0.5176	0.5097	0.5514	0.5347	**0.6541**
	SCoRD	**0.6072***	**0.6492***	**0.6141***	**0.6701***	**0.6149***	0.6733	**0.5831***	**0.6295***	**0.5916***	**0.6556***	**0.5927***	0.6598	**0.5658***	**0.6188***	**0.5720***	**0.6376***	**0.5733***	0.6424
Retriever	Full-Batch	0.3386	0.5041	0.3759	0.6177	0.3899	0.6721	0.2041	0.3451	0.2604	0.5194	0.2886	0.6302	0.1590	0.2839	0.2086	0.4369	0.2433	0.5733
	Fine-Tune	0.3529	0.5135	0.3850	0.6112	0.4002	0.6705	0.2663	0.3980	0.3130	0.5413	0.3398	0.6464	0.2098	0.3310	0.2624	0.4941	0.2948	0.6212
	PISA	0.3509	0.4971	0.3850	0.6018	0.4027	0.6713	0.3309	0.4686	0.3692	0.5872	0.3895	0.6662	0.3315	0.4792	0.3670	0.5875	0.3821	0.6471
	Reloop2	0.4168	0.5597	0.4411	0.6349	0.4505	0.6713	0.3212	0.4686	0.3579	0.5822	0.3767	0.6549	0.2056	0.3224	0.2550	0.4753	0.2914	0.6180
	CCD	0.3584	0.5135	0.3914	0.6149	0.4075	0.6774	0.3773	0.5138	0.4076	0.6062	0.4237	**0.6690**	0.2784	0.4235	0.3194	0.5498	0.3441	0.6463
	LLMD4Rec	0.3804	0.5180	0.4076	0.6018	0.4217	0.6570	0.3386	0.4601	0.3769	0.5773	0.3934	0.6415	0.2854	0.4039	0.3239	0.5239	0.3507	0.6290
	CoT-Rec	0.4146	0.5462	0.4407	0.6267	0.4538	**0.6778**	0.3188	0.4651	0.3554	0.5773	0.3778	0.6648	0.3142	0.4353	0.3527	0.5545	0.3781	**0.6541**
	SCoRD	**0.6003***	**0.6533***	**0.6051***	**0.6680***	**0.6064***	0.6733	**0.5716***	**0.6316***	**0.5782***	**0.6514***	**0.5804***	0.6598	**0.5572***	**0.6149***	**0.5634***	**0.6337***	**0.5656***	0.6424

report the performance of the reranker and retriever at each block, averaged over five independent runs with different random seeds.

Retriever-Reranker setup. We simulate a retriever-reranker pipeline. For a fair comparison, all methods use the same backbone configuration: E4SRec [35] (reranker), which injects ID-based item embeddings into an LLM for reranking, and SASRec [24] (retriever), which captures sequential preferences via self-attention [55]. Note that the reranker and retriever are paired for each method.

Baselines. We group baselines into four categories and compare SCoRD with state-of-the art methods in a reranker–retriever pipeline.

(1) **Reference methods**: serve as upper and lower bounds.
   - **Full-Batch** retrains from scratch on all accumulated data.
   - **Fine-Tune** updates models only on the current data block.
(2) **CL-based methods**: employ regularization or replay for CL.
   - **PISA** [67] is a regularization-based method that balances stability and plasticity per user based on preference shifts.
   - **Reloop2** [73] is a replay-based method that uses error memory to self-correct future recommendations.
(3) **Continual KD**: applies KD across incremental data blocks.
   - **CCD** [29] pioneers continual KD via ranking predictions between teacher-student models with asynchronous updates.
(4) **LLM-based KD methods**: transfer knowledge between LLM-based rerankers and ID-based retrievers.
   - **LLMD4Rec** [60] jointly refines reranker and retriever via bidirectional exchange of semantic and collaborative signals.
   - **CoT-Rec** [39] uses Chain-of-Thought (CoT) [58] reasoning to generate text descriptions, encodes them as semantic embeddings, and initializes the embeddings of the retriever.

As LLMD4Rec and CoT-Rec assume a static environment, we extend them to the CL setting: both methods update the retriever every τ_s; at every τ_T, LLMD4Rec performs bidirectional KD while CoT-Rec provides the reranker with the updated retriever's item embeddings.

Implementation details. Hyperparameters are tuned via grid search on the validation set. Baseline hyperparameters follow the search ranges in the original papers. We describe details in four aspects.

- **Common.** All models are implemented in PyTorch and trained on NVIDIA A100 GPUs using Adam [26]. Following CCD [29], we set the retriever update interval τ_S to one-tenth of the reranker update interval τ_T. For intent generation, each item is represented by its title, category, and five sampled reviews, following [40, 49].
- **Reranker.** We fine-tune Llama-3.2-3B-Instruct [1] with LoRA [14] (rank r=16, α=16, dropout=0.1) for 4 epochs. The learning rate is set to 3e-4 and L_2 regularization to 5e-2.
- **Retriever.** We use SASRec [24] (d=128, 2 layers, 2 heads) trained for 100 epochs. We search the learning rate over {1e-3, 1e-2} and L_2 regularization over {5e-5, 1e-4}. We use two negatives.

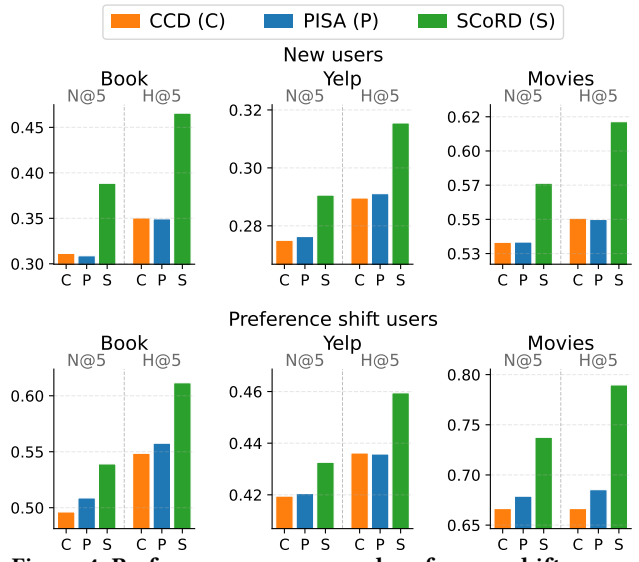

Figure 4: Performance on new and preference shift users.

- **SCoRD hyperparameters.** We search $\lambda_{\text{intent}} \in \{0.5, 1.0, 1.5\}$, $\lambda_{\text{reg}} \in \{0.3, 0.5, 0.7\}$, and $\lambda_{\text{drift}} \in \{0.1, 0.2, 0.5\}$. The number of negatives for $\mathcal{L}_{\text{drift}}$ is searched over $\{5, 10\}$.

5.2 Performance Comparison

5.2.1 ***Main results.*** Table 2 presents the performance comparison. Overall, SCoRD outperforms all baselines across all datasets and metrics. We analyze these gains from three perspectives.

Overall. We attribute the overall gains of SCoRD to two key advantages. First, SCoRD surpasses CL-based methods (PISA, Reloop2) by enabling mutual knowledge transfer over time, whereas CL methods update each model independently. Second, SCoRD gains over CCD by leveraging auxiliary semantic guidance from the assistant beyond surface-level ranking predictions.

Reranker side. SCoRD beats LLM-based KD methods (LLMD4Rec, CoT-Rec). This is attributed to two external guidance sources: (1) collaborative signals from the retriever and (2) intent drift-aware negatives from the assistant, which other methods do not exploit.

Retriever side. SCoRD surpasses LLM-based KD methods as the assistant provides continuous guidance during retriever-only updates via semantic-guided representation and negatives. However, other methods offer no guidance until the LLM is updated again.

Interestingly, SCoRD yields stronger gains on fine-grained metrics (N@5 and H@5) than on H@20. We conjecture that continuous semantic assistance mainly enhances fine-grained recommendation, while adaptive control of assistant guidance could further improve broader candidate coverage. We leave this direction for future work.

5.2.2 ***Analysis under CL setup.*** We evaluate reranker performance of SCoRD against CCD and PISA under the CL setup, focusing on challenging user groups and stability-plasticity balance.

Analysis on user groups. Figure 4 reports performance (N@5, H@5) on new users and preference shift users[8] across three datasets. SCoRD consistently outperforms CCD and PISA across all setups.

[8]Preference shift users are the top 25% by cosine distance $1 - \cos(\bar{\mathbf{e}}_u^{(k-1)}, \bar{\mathbf{e}}_u^{(k)})$, where $\bar{\mathbf{e}}_u^{(k)}$ is the mean item embedding given $S_u^{(k)}$ from the fine-tuned SASRec backbone.

Table 3: Stability (RA) and plasticity (LA) comparison.

Dataset	Model	RA	LA	H-mean
Book	CCD	0.1159	0.4293	0.1825
	PISA	0.1037	0.4322	0.1673
	SCoRD	**0.1276**	**0.4762**	**0.2013**
Yelp	CCD	0.0289	0.3376	0.0532
	PISA	0.0274	0.3376	0.0507
	SCoRD	**0.0478**	**0.3642**	**0.0845**
Movies	CCD	0.0857	0.5554	0.1485
	PISA	0.0839	0.5544	0.1457
	SCoRD	**0.1189**	**0.6034**	**0.1987**

Table 4: Accuracy-efficiency trade-off comparison.

Dataset	Model	Gen time	Tr. time	Inf. time	N@5	H@5
Book	LLMD4Rec	N/A	157m	75s	0.4740	0.5067
	CoT-Rec	15.4h	32.4m	75s	0.4682	0.5132
	SCoRD	1.4h	106.5m	75s	**0.5088**	**0.5846**
Yelp	LLMD4Rec	N/A	156m	89s	0.3804	0.3925
	CoT-Rec	17.6h	37m	88s	0.3649	0.3934
	SCoRD	1.7h	125.3m	88s	**0.3866**	**0.4136**
Movies	LLMD4Rec	N/A	78m	43s	0.5348	0.5552
	CoT-Rec	5.4h	19.5m	43s	0.5334	0.5521
	SCoRD	0.5h	52m	44s	**0.5719**	**0.6266**

The analysis for each group is as follows. For new users, CCD and PISA perform at a similar level, suggesting that CL methods struggle to handle users with no historical interactions. SCoRD addresses this through intent-level guidance from $\mathcal{A}$, which provides semantic context even in the absence of prior user history. For preference shift users, PISA slightly outperforms CCD, as PISA is explicitly designed to handle preference shifts. However, SCoRD achieves substantially larger gains over both, showing that intent-drift aware learning effectively captures evolving user preferences.

Together, these results indicate that SCoRD is effective not only in overall recommendation quality but also in handling the two most challenging user groups under a non-stationary data stream.

Stability-plasticity analysis. Following [7, 29, 32], we adopt two CL-specific metrics[9]: Retained average (RA) for stability and Learning average (LA) for plasticity, both computed on N@5. We also report their harmonic mean (H-mean) to summarize their balance.

Table 3 shows that SCoRD improves over both baselines on RA, LA, and H-mean simultaneously, confirming that stability and plasticity are achieved without trading one off against the other. We attribute the stability gain to item embedding regularization in the retriever, whose stable representations are projected into the reranker, enabling more stable user representation construction. The plasticity gain is driven by intent-drift aware guidance from the assistant and collaborative signals from the retriever, which facilitate adaptation to newly arriving data.

5.2.3 ***Accuracy-efficiency trade-off analysis.*** Table 4 presents accuracy (N@5, H@5) and time for semantic intent generation (Gen), model training (Tr.), and inference (Inf.) over D_2–D_4. SCoRD achieves the best performance while maintaining competitive efficiency. Specifically, SCoRD reduces generation time[10] by ~10× compared to CoT-Rec (e.g., 1.4h vs. 15.4h on Book). For training,

[9]RA = $\frac{1}{t}\sum_{i=1}^{t} a_{t,i}$ (for stability); LA = $\frac{1}{t}\sum_{i=1}^{t} a_{i,i}$ (for plasticity), where a_{ij} is performance on block j after training on block i. We report results on D_2 through D_4.
[10]LLMD4Rec reports no generation time (N/A) as it relies solely on prediction logits without generating any semantic content.

Table 5: Ablation study across three stages.

Model Variant	Reranker N@5	Reranker H@5	Retriever N@5	Retriever H@5
Stage 1				
w/o $\mathcal{A}$ (removes both $\mathcal{L}_{\text{intent}}$ and $\tilde{\mathbf{h}}_u$)	0.3748	0.4945	0.2675	0.3972
w/o $\mathcal{L}_{\text{KD}}$	0.4959	0.5748	0.4935	0.5798
w/o $\mathcal{A}$ & $\mathcal{L}_{\text{KD}}$	0.3598	0.4809	0.2467	0.3768
Stage 2				
w/o $\mathcal{L}_{\text{co-intent}}$	0.5012	0.5782	0.4960	0.5780
w/o $\tilde{h}_u$ (uses h_u instead)	0.4934	0.5659	0.4790	0.5726
w/o Intent-drift $\mathcal{N}_u^-$ (uses random $\mathcal{N}_u^-$)	0.3549	0.4797	0.2455	0.3738
w/o $\mathcal{L}_{\text{reg}}$	0.5061	0.5729	**0.5096**	0.5803
Stage 3				
w/o $\mathcal{L}_{\text{drift}}$	0.4957	0.5746	0.4947	0.5795
Full method	**0.5088**	**0.5846**	0.4993	**0.5852**

LLMD4Rec takes the longest due to full-sequence distillation, CoT-Rec the shortest as it fine-tunes each model independently without cross-model distillation, and SCoRD falls in between. Inference time is comparable across all methods. As a result, SCoRD achieves a better balance between recommendation quality and efficiency.

5.3 Study of SCoRD

5.3.1 ***Ablation study.*** Table 5 reports reranker and retriever performance averaged over D_2–D_4 to examine the contribution of each component on the Book dataset. Overall, the full method with all components achieves the best performance, confirming the contribution of each. We analyze the results stage by stage as follows.

Stage 1. Ablating $\mathcal{A}$ removes both $\mathcal{L}_{\text{intent}}$ supervision and intent-guided user representation $\tilde{\mathbf{h}}_u$, causing the largest drop. Removing $\mathcal{L}_{\text{KD}}$ alone leads to a smaller drop, while removing both $\mathcal{A}$ & $\mathcal{L}_{\text{KD}}$ yields the worst performance, showing their complementary roles.

Stage 2. Replacing intent-drift negatives with random negatives causes the largest drop, highlighting their importance for learning evolving user interests. Replacing the semantic-guided representation $\tilde{\mathbf{h}}_u$ with the original retriever representation $\mathbf{h}_u$ leads to the second largest drop, showing the importance of semantic-guidance during retriever updates. Removing $\mathcal{L}_{\text{co-intent}}$ and $\mathcal{L}_{\text{reg}}$ leads to relatively small drops compared to other components.

Stage 3. Removing $\mathcal{L}_{\text{drift}}$ degrades reranker performance, indicating the importance of drift-aware training signals. Note that retriever-side knowledge injection cannot be ablated, as item embeddings from the retriever are integral to the LLM prompt construction.

5.3.2 ***Hyperparameter analysis.*** We analyze key hyperparameters for each stage (N@5, H@5), reporting retriever performance for stages 1–2 and reranker for stage 3.

Stage 1. Figure 5 shows the effect of the hard sequences selection ratio r. Performance improves sharply from r=0 to r=5 and saturates thereafter for most metrics. This implies that targeting a small proportion of hard users is sufficient for meaningful gains without requiring LLM supervision over the full user set.

Stage 2. Figure 6 examines how the loss weights λ_{intent} and λ_{reg} jointly affect retriever updates. Overall, λ_{intent}=1.0 yields the best results, with performance remaining stable across $\lambda_{\text{reg}} \in \{0.3, 0.5\}$, suggesting that SCoRD is not overly sensitive to loss weight selection within the search range.

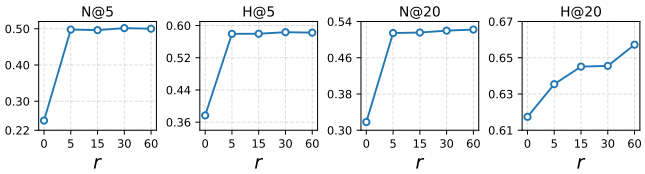

Figure 5: Effect of hard sequences selection ratio r.

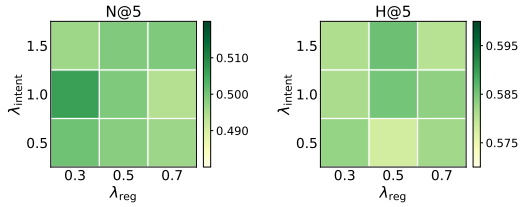

Figure 6: Effect of λ_{intent} and λ_{reg} on retriever performance.

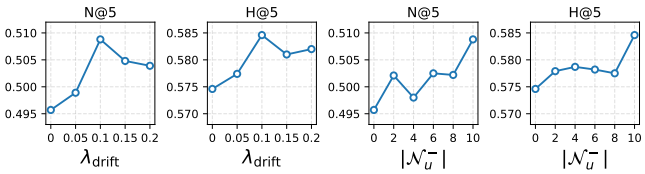

Figure 7: Effect of λ_{drift} and $|\mathcal{N}_u^-|$ on reranker performance.

Stage 3. Figure 7 presents the effect of λ_{drift} and the number of intent-drift negatives $|\mathcal{N}_u^-|$ on reranker training. For λ_{drift}, performance peaks at 0.1 and slightly degrades beyond that, suggesting that too strong a drift signal can hurt recommendation accuracy. For $|\mathcal{N}_u^-|$, performance consistently improves as more intent-drift negatives are added, indicating that a larger set of faded-intent negatives provides richer contrastive supervision. In both cases, removing these components (i.e., setting them to zero) leads to notable performance drops, validating the effectiveness of explicit intent drift-aware supervision.

6 Conclusion

We propose SCoRD, a semantic-assisted continual KD framework that enables co-adaptation of an ID-based retriever and an LLM reranker under a non-stationary data stream. With the semantic reasoning assistant, SCoRD converts the LLM's intent inference ability into reusable guidance for retriever updates. It supports selective reranker-to-retriever distillation, intent-guided retriever updates, and retriever-informed reranker updates. Extensive experiments show that SCoRD improves both the retriever and the reranker, highlighting the effectiveness of semantic-assisted co-adaptation. We expect this work to contribute to the practical deployment of LLM-based reranking pipelines in dynamic recommendation environments.

Acknowledgments

This work was supported by a Korea University Grant, ICT Creative Consilience Program through the IITP grant funded by the MSIT (IITP-2026-RS-2020-II201819), Basic Science Research Program through the NRF funded by the Ministry of Education (NRF-2021R1A6A1A03045425), the NRF grant funded by the MSIT (RS-2026-25486220), and the IITP grant funded by the MSIT (IITP-2026-RS-2026-25616664, AI Star Fellowship Support Program).

GenAI Usage Disclosure

Generative AI tool was used as part of the research methodology. In this work, LLM (LLaMA3.2-3B-Instruct) was used to infer users' underlying intents and serve as a reranking model within the proposed framework. Beyond the role in the research methodology, generative AI tools were limitedly used for writing and grammar refinement. All outputs were carefully reviewed and revised by the authors. Experimental results, datasets, and scientific contributions are entirely the authors' own work.

References

[1] Meta AI. 2024. Llama 3.2. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *The 41st international ACM SIGIR conference on research & development in information retrieval*. 135–144.

[3] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[5] Yu Cui, Feng Liu, Pengbo Wang, Bohao Wang, Heng Tang, Yi Wan, Jun Wang, and Jiawei Chen. 2024. Distillation matters: empowering sequential recommenders to match the performance of large language models. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 507–517.

[6] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 143–177.

[7] Jaime Hieu Do and Hady W Lauw. 2023. Continual collaborative filtering through gradient alignment. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 1133–1138.

[8] Yingpeng Du, Zhu Sun, Ziyan Wang, Haoyan Chua, Jie Zhang, and Yew-Soon Ong. 2025. Active large language model-based knowledge distillation for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 11607–11615.

[9] Jingtong Gao, Bo Chen, Xiangyu Zhao, Weiwen Liu, Xiangyang Li, Yichao Wang, Wanyu Wang, Huifeng Guo, and Ruiming Tang. 2025. Llm4rerank: Llm-based auto-reranking framework for recommendations. In *Proceedings of the ACM on Web Conference 2025*. 228–239.

[10] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chat-rec: Towards interactive and explainable llms-augmented recommender system. *arXiv preprint arXiv:2303.14524* (2023).

[11] Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised learning by entropy minimization. *Advances in neural information processing systems* 17 (2004).

[12] Bowei He, Xu He, Yingxue Zhang, Ruiming Tang, and Chen Ma. 2023. Dynamically Expandable Graph Convolution for Streaming Recommendation. In *WWW*.

[13] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large language models are zero-shot rankers for recommender systems. In *European conference on information retrieval*. Springer, 364–381.

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *Iclr* 1, 2 (2022), 3.

[15] Xu Huang, Defu Lian, Jin Chen, Liu Zheng, Xing Xie, and Enhong Chen. 2023. Cooperative retriever and ranker in deep recommenders. In *Proceedings of the ACM Web Conference 2023*. 1150–1161.

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[17] Jian Jia, Yipei Wang, Yan Li, Honggang Chen, Xuehan Bai, Zhaocheng Liu, Jian Liang, Quan Chen, Han Li, Peng Jiang, et al. 2025. LEARN: Knowledge Adaptation from Large Language Model to Recommendation for Practical Industrial Application. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 11861–11869.

[18] Fan Jiang, Qiongkai Xu, Tom Drummond, and Trevor Cohn. 2023. Boot and switch: Alternating distillation for zero-shot dense retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 912–931.

[19] SeongKu Kang, Junyoung Hwang, Wonbin Kweon, and Hwanjo Yu. 2020. DE-RRD: A knowledge distillation framework for recommender system. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 605–614.

[20] SeongKu Kang, Junyoung Hwang, Wonbin Kweon, and Hwanjo Yu. 2021. Topology distillation for recommender system. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 829–839.

[21] SeongKu Kang, Wonbin Kweon, Dongha Lee, Jianxun Lian, Xing Xie, and Hwanjo Yu. 2023. Distillation from heterogeneous models for top-k recommendation. In *Proceedings of the ACM Web Conference 2023*. 801–811.

[22] SeongKu Kang, Dongha Lee, Wonbin Kweon, Junyoung Hwang, and Hwanjo Yu. 2022. Consensus learning from heterogeneous objectives for one-class collaborative filtering. In *Proceedings of the ACM Web Conference 2022*. 1965–1976.

[23] SeongKu Kang, Jianxun Lian, Dongha Lee, Wonbin Kweon, Sanghwan Jang, Jaehyun Lee, Jindong Wang, Xing Xie, and Hwanjo Yu. 2025. BPL: Bias-Adaptive Preference Distillation Learning For recommender System. *IEEE Transactions on Knowledge and Data Engineering* (2025).

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

[25] Sein Kim, Hongseok Kang, Kibum Kim, Jiwan Kim, Donghyun Kim, Minchul Yang, Kwangjin Oh, Julian McAuley, and Chanyoung Park. 2025. Lost in Sequence: Do Large Language Models Understand Sequential Recommendation?. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1160–1171.

[26] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[27] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[28] Wonbin Kweon, SeongKu Kang, and Hwanjo Yu. 2021. Bidirectional distillation for top-K recommender system. In *Proceedings of the Web Conference 2021*. 3861–3871.

[29] Gyuseok Lee, SeongKu Kang, Wonbin Kweon, and Hwanjo Yu. 2024. Continual collaborative distillation for recommender system. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1495–1505.

[30] Gyuseok Lee, Wonbin Kweon, Zhenrui Yue, SeongKu Kang, Jiawei Han, and Dong Wang. 2026. Uncertainty-Aware Variational Reward Factorization via Probabilistic Preference Bases for LLM Personalization. *arXiv preprint arXiv:2604.00997* (2026).

[31] Gyuseok Lee, Wonbin Kweon, Zhenrui Yue, Yaokun Liu, Yifan Liu, Susik Yoon, Dong Wang, and SeongKu Kang. 2026. SPRINT: Scalable and Predictive Intent Refinement for LLM-Enhanced Session-based Recommendation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 902–912.

[32] Gyuseok Lee, Hyunsik Yoo, Junyoung Hwang, SeongKu Kang, and Hwanjo Yu. 2026. Capturing User Interests from Data Streams for Continual Sequential Recommendation. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining*. 313–323.

[33] Gyuseok Lee, Yaochen Zhu, Hwanjo Yu, Yao Zhou, and Jundong Li. 2025. Collaborative diffusion model for recommender System. In *Companion Proceedings of the ACM on Web Conference 2025*. 1091–1095.

[34] Jae-woong Lee, Minjin Choi, Jongwuk Lee, and Hyunjung Shim. 2019. Collaborative distillation for top-N recommendation. In *2019 IEEE international conference on data mining (ICDM)*. IEEE, 369–378.

[35] Xinhang Li, Chong Chen, Xiangyu Zhao, Yong Zhang, and Chunxiao Xing. 2023. E4srec: An elegant effective efficient extensible solution of large language models for sequential recommendation. *arXiv preprint arXiv:2312.02443* (2023).

[36] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[37] Zizhong Li, Haopeng Zhang, and Jiawei Zhang. 2024. Intermediate distillation: Data-efficient distillation from black-box llms for information retrieval. *arXiv preprint arXiv:2406.12169* (2024).

[38] Hongyang Liu, Zhu Sun, Tianjun Wei, Yan Wang, Jiajie Zhu, and Xinghua Qu. 2026. Diagnostic-Guided Dynamic Profile Optimization for LLM-based User Simulators in Sequential Recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 15306–15314.

[39] Jiahao Liu, Xueshuo Yan, Dongsheng Li, Guangping Zhang, Hansu Gu, Peng Zhang, Tun Lu, Li Shang, and Ning Gu. 2025. Improving llm-powered recommendations with personalized information. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2560–2565.

[40] Yibin Liu, Jianyu Zhang, and Shijian Li. 2025. Enhancing Recommendation with Reliable Multi-profile Alignment and Collaborative-aware Contrastive Learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 1936–1946.

[41] Sichun Luo, Bowei He, Haohan Zhao, Wei Shao, Yanlin Qi, Yinya Huang, Aojun Zhou, Yuxuan Yao, Zongpeng Li, Yuanzhang Xiao, et al. 2025. Recranker: Instruction tuning large language model as ranker for top-k recommendation. *ACM Transactions on Information Systems* 43, 5 (2025), 1–31.

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

[43] Fei Mi, Xiaoyu Lin, and Boi Faltings. 2020. Ader: Adaptively distilled exemplar replay towards continual learning for session-based recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 408–413.

[44] Kamal Nigam and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*. 86–93.

[45] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[46] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM conference on recommender systems*. 3–11.

[47] Jiarui Qin, Weiwen Liu, Weinan Zhang, and Yong Yu. 2025. D2K: Turning historical data into retrievable knowledge for recommender systems. In *Proceedings of the ACM on Web Conference 2025*. 472–482.

[48] Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qiaoqiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2825–2835.

[49] Xubin Ren, Wei Wei, Lianghao Xia, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. 2024. Representation learning with large language models for recommendation. In *Proceedings of the ACM web conference 2024*. 3464–3475.

[50] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[51] Tianhao Shi, Yang Zhang, Zhijian Xu, Chong Chen, Fuli Feng, Xiangnan He, and Qi Tian. 2024. Preliminary study on incremental learning for large language model-based recommender systems. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4051–4055.

[52] Zhongxiang Sun, Zihua Si, Xiaoxue Zang, Kai Zheng, Yang Song, Xiao Zhang, and Jun Xu. 2024. Large language models enhanced collaborative filtering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2178–2188.

[53] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2289–2298.

[54] Runchu Tian, Xueqiang Xu, Bowen Jin, SeongKu Kang, and Jiawei Han. 2025. CoRank: LLM-based compact reranking with document features for scientific retrieval. *arXiv preprint arXiv:2505.13757* (2025).

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[56] Yuening Wang, Yingxue Zhang, and Mark Coates. 2021. Graph structure aware contrastive knowledge distillation for incremental learning in recommender systems. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3518–3522.

[57] Yuening Wang, Yingxue Zhang, Antonios Valkanas, Ruiming Tang, Chen Ma, Jianye Hao, and Mark Coates. 2023. Structure aware incremental learning with personalized imitation weights for recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4711–4719.

[58] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[59] Wei Wei, Xubin Ren, Jiabin Tang, Qinyong Wang, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. 2024. Llmrec: Large language models with graph augmentation for recommendation. In *Proceedings of the 17th ACM international conference on web search and data mining*. 806–815.

[60] Jiongran Wu, Jiahao Liu, Dongsheng Li, Guangping Zhang, Mingzhe Han, Hansu Gu, Peng Zhang, Li Shang, Tun Lu, and Ning Gu. 2025. Bidirectional Knowledge Distillation for Enhancing Sequential Recommendation with Large Language Models. *arXiv preprint arXiv:2505.18120* (2025).

[61] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2024. A survey on large language models for recommendation. *World Wide Web* 27, 5 (2024), 60.

[62] Yunjia Xi, Weiwen Liu, Jieming Zhu, Xilong Zhao, Xinyi Dai, Ruiming Tang, Weinan Zhang, Rui Zhang, and Yong Yu. 2022. Multi-level interaction reranking with user behavior history. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1336–1346.

[63] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*. 1192–1199.

[64] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. 2020. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2861–2868.

[65] Fan Yang, Zheng Chen, Ziyan Jiang, Eunah Cho, Xiaojiang Huang, and Yanbin Lu. 2023. Palr: Personalization aware llms for recommendation. *arXiv preprint arXiv:2305.07622* (2023).

[66] Yelp. [n. d.]. Yelp Open Dataset. https://business.yelp.com/data/resources/open-dataset/. Accessed: 2026-01-20.

[67] Hyunsik Yoo, SeongKu Kang, Ruizhong Qiu, Charlie Xu, Fei Wang, and Hanghang Tong. 2025. Embracing plasticity: Balancing stability and plasticity in continual recommender systems. In *Proceedings of the 48th International ACM SIGIR conference on research and development in Information Retrieval*. 2092–2101.

[68] Hyunsik Yoo, SeongKu Kang, and Hanghang Tong. 2025. Continual Recommender Systems. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 6857–6860.

[69] Hyunsik Yoo, Ting-Wei Li, SeongKu Kang, Zhining Liu, Caizhi Charlie Xu, Qilin Qi, and Hanghang Tong. 2026. Continual low-rank adapters for llm-based generative recommender systems. In *International Conference on Learning Representations*, Vol. 2026. 6618–6642.

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

[71] Xinni Zhang, Yankai Chen, Chenhao Ma, Yixiang Fang, and Irwin King. 2024. Influential exemplar replay for incremental learning in recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 9368–9376.

[72] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, et al. 2024. Recommender systems in the era of large language models (llms). *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 6889–6907.

[73] Jieming Zhu, Guohao Cai, Junjie Huang, Zhenhua Dong, Ruiming Tang, and Weinan Zhang. 2023. Reloop2: Building self-adaptive recommendation models via responsive error compensation loop. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5728–5738.