FAST-WEIGHT MEMORY FOR ADAPTIVE RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern sequential recommenders repeatedly process interaction histories to capture evolving user preferences, coupling preference adaptation with sequential history modeling. We propose FWPRec, a recommendation model that separates user representation into a stable preference anchor and a fast associative memory for recent behavioral changes. The anchor captures persistent preference information, while fast-weight matrices incorporate new positive and negative feedback through learned gradient-free updates at inference time without modifying global model parameters. This separation allows item representations and long-term user states to be reused across requests, while preference changes are directly incorporated into a fixed-size user-specific state. We evaluate FWPRec on six real-world interaction datasets against static, lightweight adaptation, recurrent, attention-based, and state-space recommenders under controlled ranking settings. FWPRec achieves competitive ranking performance while providing effective adaptation to preference changes, with particularly strong gains when informative positive and negative feedback is available. Further analysis shows that learned memory construction and the separation of stable preferences from fast behavioral corrections are important for effective adaptation. The results demonstrate that fast-weight memory provides an alternative and complementary mechanism to sequential history modeling for capturing dynamic user preferences.

1 INTRODUCTION

User preference is neither completely static nor fully determined by the most recent interaction. A user may have persistent interests represented by a profile, search query, or long-term interaction history, while recent behavior reflects temporary changes within this broader preference. Modern sequential recommenders capture such changes by modeling ordered interaction histories, and recent work has substantially improved this paradigm through generative modeling, large-scale sequence architectures, and more efficient recurrent or state-space updates (Rajput et al., 2023; Zhai et al., 2024; Liu et al., 2024; Yue et al., 2024).

Despite these advances, adaptation is still typically coupled with sequence inference. The current user state is updated through repeated transformations of the interaction sequence or its maintained hidden state. Recent methods improve this process through linear attention, recurrent units, semantic item tokenization, or explicit modeling of different temporal preference components (Liu et al., 2023; Wang et al., 2024; Qu et al., 2025; Jiang et al., 2023; Shin et al., 2024). These approaches make sequential modeling more expressive or efficient, but largely retain the same underlying formulation in which preference evolution is represented through an evolving sequence model.

In this work, we study a different formulation. Rather than using a single evolving representation to jointly capture persistent and transient preference, we separate the user state into a stable preference anchor and a fast associative memory. The anchor represents persistent user intent, while the memory stores interaction-specific corrections induced by recent feedback. We implement this memory using fast weights (Schlag et al., 2021), where user-specific parameters are updated during inference through globally learned update functions. The global model parameters remain fixed, while each interaction modifies only the fast user state without gradient-based optimization.

The separation of stable preference anchor and fast associative memory is particularly useful when item representations and persistent user information can be reused across requests. Item embeddings

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

may be precomputed and shared across users, while a query, profile, or long-term representation provides the initial anchor. Indeed, precomputed item representations and cached user representations are common infrastructure choices in large-scale industrial recommendation systems (Yi et al., 2019; Pancha et al., 2022; Satuluri et al., 2020). New feedback then updates only a bounded user-specific memory, making the adaptive state independent of interaction-history length. The formulation therefore provides a different quality–adaptation–state trade-off from conventional sequential recommendation.

Based on this formulation, we propose FWPRec, a fast-weight recommender for dynamic preference modeling. FWPRec maintains a stable user anchor together with associative memories for positive and negative behavioral corrections. Each interaction produces a learned memory update, and the resulting fast state is combined with the anchor to rank reusable candidate representations. We evaluate FWPRec on six real-world recommendation datasets under chronological splits and shared candidate sets, comparing against static, lightweight adaptation, recurrent, attention-based, and state-space recommenders while controlling item representations and candidate construction. FWPRec achieves competitive ranking performance across different recommendation settings and is particularly effective when informative positive and negative feedback is available. Further analysis shows that learned value construction and the separation of stable preferences from fast behavioral corrections are important for adaptation, while the associative memory allows individual interactions contributing to the current preference state to be traced.

2 Preliminaries and Related Work

2.1 Sequential Preference Modeling

Sequential recommenders infer user preference from an ordered interaction history. Recent work improves this formulation through efficient attention, recurrent units, state-space models, and generative sequence modeling (Liu et al., 2023; Yue et al., 2024; Liu et al., 2024; Rajput et al., 2023; Zhai et al., 2024). Other methods model long- and short-term preference or different temporal components of user behavior (Jiang et al., 2023; Du et al., 2023; Shin et al., 2024). These approaches effectively model behavioral dynamics and have shown strong performance at industrial scale (Zhai et al., 2024). However, processing interaction histories introduces additional computation and memory costs under the strict latency and throughput constraints of large-scale recommendation systems (Gupta et al., 2020; Zhai et al., 2024). Recent industrial systems therefore reduce long-history computation through efficient sequence operators, history compression, and cached representations (Zhai et al., 2024; Chen et al., 2026; Li et al., 2026). This becomes particularly important as new interactions need to be incorporated without repeatedly processing a growing history (Liu et al., 2022).

2.2 Lightweight Adaptation and Representation Reuse

Large-scale recommendation systems often separate expensive representation construction from online serving. Two-tower and embedding-based systems precompute or cache item representations so that recommendation can be performed through lightweight similarity computation (Covington et al., 2016; Huang et al., 2020). Industrial systems further apply this principle to user representations. PinnerFormer, for example, avoids fully real-time sequential inference by generating user embeddings in batch, citing the cost of repeated history inference and the complexity of maintaining mutable hidden states (Pancha et al., 2022). PinnerSage similarly combines fixed item embeddings with a long-term user profile and an online component that incorporates recent activity (Pal et al., 2020). A simple alternative to sequence inference is to update the user representation directly. Exponential moving averages provide one such mechanism, $\mathbf{u}_t = (1-\alpha)\mathbf{u}_{t-1} + \alpha\mathbf{e}_t$, where $\mathbf{e}_t$ represents the latest interaction. Moving-average and recency-weighted mechanisms have been used to emphasize recent behavior without replaying the full history (Wu et al., 2023; Chen et al., 2024). Such updates are inexpensive and independent of history length, but the adaptation rule is predefined and does not account for how a particular interaction should modify the existing preference.

These limitations motivate a learnable update mechanism that retains the constant-state adaptation of lightweight methods without prescribing the update rule.

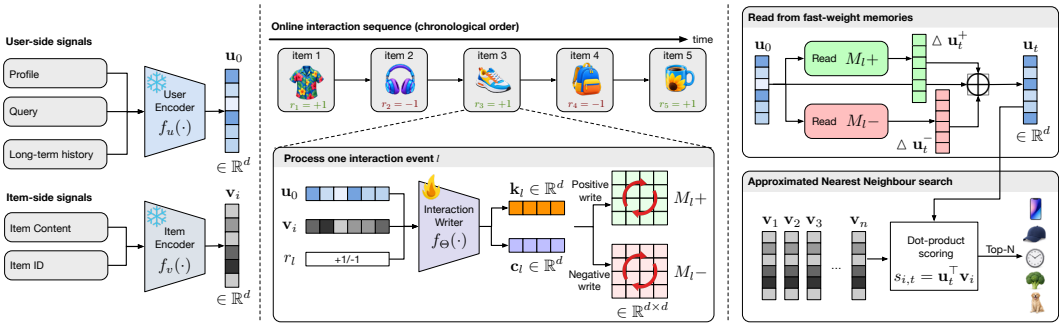


Figure 1: Overview of the proposed fast-weight preference modeling framework. Each interaction sequentially updates the positive or negative fast-weight memory through an interaction writer. The memories are then queried to recover preference changes and construct the updated user representation, which is used for retrieval and ranking. Since the user and item encoders, as well as the approximate nearest-neighbor retrieval stage, remain unchanged, the proposed framework can be easily integrated into existing large-scale recommendation pipelines.

3 Fast-Weight Preference Adaptation

3.1 Preference Adaptation

We consider one user and omit the user index for clarity. Let $\mathbf{u}_0 \in \mathbb{R}^d$ denote the user's stable preference representation before observing the interactions used for adaptation. Depending on the application, $\mathbf{u}_0$ can be obtained from an explicit user profile, a search query, or a cached representation constructed from long-term interaction history. When such information is unavailable, it can be estimated from earlier positive interactions.

After $\mathbf{u}_0$ is constructed, suppose the user generates a sequence of interactions $\mathcal{H}_t = \{(i_1, r_1), \ldots, (i_t, r_t)\}$, where i_ℓ denotes the item involved in the ℓ-th interaction and $r_\ell \in \mathbb{R}$ denotes its feedback. We use $r_\ell > 0$ for positive feedback and $r_\ell < 0$ for negative feedback.

Our goal is to learn an adaptation function

$$\mathbf{u}_t = F_\Theta\left(\mathbf{u}_0, \mathcal{H}_t\right), \tag{1}$$

where $\mathbf{u}_t \in \mathbb{R}^d$ is the adapted user representation and Θ denotes the global model parameters. For an item i that is represented by a fixed vector $\mathbf{v}_i \in \mathbb{R}^d$, we score the user's preference $\mathbf{u}_t$ to the candidate item i by simple dot product such that $s_{i,t} = \mathbf{u}_t^\top \mathbf{v}_i$.

The main question is therefore how to construct F_Θ. Instead of repeatedly encoding $\mathcal{H}_t$ using a sequential model, FWPRec keeps $\mathbf{u}_0$ fixed and represents the preference changes introduced by $\mathcal{H}_t$ using fast-weight memory.

3.2 Fast-Weight Memory

FWPRec maintains two fast-weight matrices for positive and negative preference updates,

$$\mathbf{M}_t^+, \mathbf{M}_t^- \in \mathbb{R}^{d\times d}, \tag{2}$$

where $\mathbf{M}_t^+$ summarizes positive preference changes and $\mathbf{M}_t^-$ summarizes negative preference changes after processing the first t interactions. Both matrices are initialized as zero matrices, $\mathbf{M}_0^+ = \mathbf{M}_0^- = \mathbf{0}$, before processing any interactions.

Following the paradigm of Fast Weight Programmers (Schlag et al., 2021), these memory matrices themselves are not globally learned model parameters but the outcome of memory updating functions. In particular, a group of global parameters Θ determine how an interaction modifies the memories, while the memories themselves are updated directly during adaptation.

3.2.1 MEMORY UPDATE WHEN NEW INTERACTION OBSERVED

Given a new observed interaction (i_ℓ, r_ℓ), we first construct an interaction representation such that $\mathbf{x}_\ell = [\mathbf{u}_0||\mathbf{v}_{i_\ell}||r_\ell]$, where $[\cdot||\cdot]$ denotes vector concatenation. The interaction representation is then provided to learned writer functions that generate a key vector $\mathbf{k}_\ell \in \mathbb{R}^d$ and a correction vector $\mathbf{c}_\ell \in \mathbb{R}^d$:

$$\mathbf{k}_\ell = \mathrm{Norm}\left(\mathbf{u}_0 + f_k(\mathbf{x}_\ell)\right), \qquad \mathbf{c}_\ell = \mathrm{Norm}\left(\mathbf{v}_{i_\ell} + f_c(\mathbf{x}_\ell)\right). \tag{3}$$

Here, f_k and f_c are globally learned functions, and Norm stands for $\mathbf{x}/||x||_2$. The key $\mathbf{k}_\ell$ determines where the interaction is written in memory, while $\mathbf{c}_\ell$ specifies the preference correction associated with the interaction. We use the term "correction" because $\mathbf{c}_\ell$ captures how the interaction shifts the original user preference $\mathbf{u}_0$.

In the following description, we will omit the $+$ or $-$ superscript to make our description more clearly. Given the generated key $\mathbf{k}_\ell$ of the current interaction, we query the existing memory through

$$\widehat{\mathbf{c}}_\ell = \mathbf{M}_{\ell-1}\mathbf{k}_\ell. \tag{4}$$

Here, $\widehat{\mathbf{c}}_\ell$ represents the correction that the current memory already associates with $\mathbf{k}_\ell$. To see this more directly, consider an idealized memory constructed from previous key-correction pairs

$$\mathbf{M}_{\ell-1} = \sum_{j<\ell} \mathbf{c}_j \mathbf{k}_j^\top. \tag{5}$$

Querying this memory with key $\mathbf{k}_\ell$ gives

$$\mathbf{M}_{\ell-1}\mathbf{k}_\ell = \sum_{j<\ell} \mathbf{c}_j \left(\mathbf{k}_j^\top \mathbf{k}_\ell\right). \tag{6}$$

Intuitively, a fast-weight matrix acts as an associative memory that maps a key vector to its stored preference correction. If the historical keys are orthogonal, the query returns the correction previously stored for the same key and returns a zero vector when no matching key has been observed. With learned non-orthogonal keys, it instead retrieves a similarity-weighted combination of related historical corrections.

The fast-weight memory allows the effect of each interaction to depend on its learned key. An interaction written with $\mathbf{k}_\ell$ has a larger influence when the memory is queried by a similar key, while its effect becomes smaller for unrelated preference directions. Therefore, the memory does not treat every interaction as a global update to the user representation. Instead, it can store different preference corrections for different regions of the representation space. It allows FWPRec to adapt user preference locally, rather than updating a single user vector that affects all subsequent predictions.

The current interaction produces the correction $\mathbf{c}_\ell$ (from Equation 3). We therefore update the memory based on the difference between this correction and what is already stored:

$$\mathbf{M}_\ell = \rho\mathbf{M}_{\ell-1} + \beta_\ell \left(\mathbf{c}_\ell - \widehat{\mathbf{c}}_\ell\right)\mathbf{k}_\ell^\top, \tag{7}$$

where $\rho \in [0, 1]$ controls the retention of previous memory and $\beta_\ell \geq 0$ controls the strength of the current write. This delta-rule update writes only the residual correction that is not already represented in memory. If the existing memory already returns $\mathbf{c}_\ell$ for $\mathbf{k}_\ell$, the update becomes zero apart from memory decay.

For signed feedback, we use separate write strengths for the positive and negative memories:

$$\beta_\ell^+ = \sigma\left(f_+(\mathbf{x}_\ell)\right)[r_\ell]_+, \qquad \beta_\ell^- = \sigma\left(f_-(\mathbf{x}_\ell)\right)[-r_\ell]_+, \tag{8}$$

where $[x]_+ = \max(x, 0)$, and f_+ and f_- are learned scalar functions. Positive feedback therefore updates only $\mathbf{M}_\ell^+$, while negative feedback updates only $\mathbf{M}_\ell^-$.

3.2.2 ADAPTING USER REPRESENTATION THROUGH FAST WEIGHTED MEMORY

After processing the t interactions in $\mathcal{H}_t$, we obtain the adapted user representation by reading both memories using the stable preference $\mathbf{u}_0$:

$$\mathbf{u}_t = \mathrm{Norm}\left(\mathbf{u}_0 + \mathbf{M}_t^+\mathbf{u}_0 - \alpha\mathbf{M}_t^-\mathbf{u}_0\right), \tag{9}$$

where $\alpha \geq 0$ controls the contribution of negative feedback.

Equation 9 gives the adaptation function introduced in Eq. 1. The stable representation $\mathbf{u}_0$ is not modified. Instead, $\mathbf{M}_t^+\mathbf{u}_0$ and $\mathbf{M}_t^-\mathbf{u}_0$ provide the positive and negative corrections supported by the subsequent interactions. Since these corrections are retrieved from memories constructed through the learned interaction keys, the resulting update depends on both the stable preference represented by $\mathbf{u}_0$ and the preference changes stored during adaptation. The model can therefore preserve the original user preference while selectively adjusting the representation according to the observed positive and negative feedback.

3.3 Simple Learning Objective and Parameters to Learn

We train the model using an InfoNCE objective. At prediction time t, let i^+ denote the positive target item and let $\mathcal{N}_t$ denote the set of negative items used for contrastive learning. Given the matching score $s_{i,t}$ between the current user representation and item i, the training objective is

$$\mathcal{L}_{\mathrm{NCE}} = -\log \frac{\exp\left(s_{i^+,t}/\tau\right)}{\exp\left(s_{i^+,t}/\tau\right) + \sum_{i^- \in \mathcal{N}_t} \exp\left(s_{i^-,t}/\tau\right)}, \tag{10}$$

where τ is the temperature parameter. This objective encourages the updated user representation to assign a higher similarity to the observed next item than to the negative items.

The learnable parameters are those associated with the functions f_k, f_c, f_+, and f_-, which are collectively denoted by Θ. The memory retention coefficient ρ, negative-feedback weight α, and InfoNCE temperature τ are treated as hyperparameters and are not included in Θ. During training, gradients propagate through the memory updates in Eq. 7 to optimize these functions. In contrast, the fast-weight matrices $\mathbf{M}_t^+$ and $\mathbf{M}_t^-$ are dynamically updated from the interaction sequence and are not trainable model parameters.

At inference time, the learned functions in Θ, $\mathbf{u}_0$, and the item representations remain fixed. Each new interaction only updates the corresponding fast-weight memory, which produces the updated user representation $\mathbf{u}_t$ for subsequent retrieval. Importantly, the item encoder and the resulting item representations are unaffected by this adaptation. Existing item indexing and approximate nearest-neighbor retrieval can therefore be retained without modification. FWPRec changes only the user-side representation used to query the index, allowing preference adaptation to be introduced without repeatedly encoding the interaction history or rebuilding the retrieval infrastructure.

4 Experiments and Evaluations

Datasets. We evaluate on six real-world recommendation datasets, including MIND and EB-NeRD for news recommendation, Amazon Reviews 2023 Video Games for product recommendation, MovieLens-1M for movie recommendation, and KuaiRand-Pure and KuaiRec for short-video recommendation. These datasets cover different forms of user feedback, including clicks, non-clicks, explicit ratings, and watch behavior. All datasets are split chronologically, and only interactions observed before the prediction time are used to construct user histories. Dataset statistics and preprocessing details are provided in Appendix A.1.

Baselines. We compare FWPRec with static user profiles, lightweight preference-update methods, and sequential recommenders. The preference-update baselines include PositiveEMA, SignedEMA (Ding & Li, 2005), LastFeedback (Huang & Stakhiyevich, 2021), CumulativeMean, SlidingWindowMean (Vinagre & Jorge, 2012), OnlineSGD (Vinagre et al., 2014), and Diagonal-Kalman. These methods maintain an explicit user state and provide direct controls for whether fast adaptation can be achieved with simpler update rules. Sequential baselines include GRU4Rec (Hidasi et al., 2016), SASRec (Kang & McAuley, 2018), DuoRec (Qiu et al., 2022), LRURec (Yue et al., 2024), HSTU (Zhai et al., 2024), and Mamba4Rec (Liu et al., 2024) where applicable.

Evaluation Protocol. For the main comparison, all methods use the same item representations. We encode item content using Qwen3-Embedding-0.6B and obtain fixed 64-dimensional ℓ_2-normalized item vectors. The item representations are frozen during recommender training. Within each dataset, all methods use the same chronological split, user histories, candidate items, and labels.

We report AUC, MRR, nDCG@5, and nDCG@10 for overall ranking performance.

Table 1: Comparison with preference adaptation methods without sequential models. LF, CM, SWM, and D-KF denote LastFeedback, CumulativeMean, SlidingWindowMean, and DiagonalKalman, respectively. Results are averaged over three training seeds. For each setting, boldface denotes the highest result and underline denotes the second-best result.

Method	MIND				EB-NeRD				Amazon Video Games			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
SignedEMA	0.6034	0.3287	0.3074	0.3752	0.5356	0.3415	0.3764	0.4562	<u>0.7918</u>	<u>0.2217</u>	<u>0.2153</u>	<u>0.2588</u>
LF	0.5806	0.3103	0.2867	0.3559	0.5367	0.3417	0.3770	0.4564	0.6888	0.1892	0.1802	0.2114
CM	<u>0.6036</u>	<u>0.3292</u>	<u>0.3082</u>	<u>0.3761</u>	<u>0.5415</u>	0.3455	<u>0.3813</u>	0.4600	0.7619	0.1998	0.1912	0.2323
SWM	0.5948	0.3218	0.2998	0.3682	0.5388	0.3427	0.3784	0.4577	0.7576	0.2042	0.1970	0.2367
OnlineSGD	0.5870	0.3144	0.2903	0.3607	0.5412	<u>0.3455</u>	0.3812	<u>0.4601</u>	0.5627	0.0834	0.0636	0.0866
D-KF	0.5886	0.3186	0.2962	0.3619	0.5325	0.3326	0.3690	0.4493	0.7810	**0.2314**	**0.2259**	**0.2680**
FWPRec (Ours)	**0.6259**	**0.3591**	**0.3387**	**0.4043**	**0.5752**	**0.3693**	**0.4092**	**0.4821**	**0.8105**	0.2086	0.1999	0.2496

Method	MovieLens-1M				KuaiRand				KuaiRec			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
SignedEMA	<u>0.7104</u>	0.1542	0.1400	0.1779	<u>0.6945</u>	<u>0.3699</u>	<u>0.3846</u>	<u>0.4420</u>	0.4620	0.0527	0.0321	0.0448
LF	0.6587	0.1210	0.1026	0.1352	0.6701	0.3429	0.3571	0.4177	0.4504	0.0486	0.0281	0.0399
CM	0.6708	0.1290	0.1123	0.1464	0.6934	0.3643	0.3786	0.4357	0.4602	0.0513	0.0307	0.0430
SWM	0.6797	0.1420	0.1267	0.1617	0.6751	0.3442	0.3577	0.4177	0.4589	0.0512	0.0307	0.0431
OnlineSGD	0.6452	0.1074	0.0874	0.1182	0.6723	0.3458	0.3606	0.4205	0.4483	0.0472	0.0264	0.0375
D-KF	0.6758	<u>0.1565</u>	<u>0.1445</u>	<u>0.1782</u>	0.6593	0.3251	0.3349	0.3961	<u>0.4967</u>	<u>0.0653</u>	<u>0.0448</u>	<u>0.0616</u>
FWPRec (Ours)	**0.7694**	**0.2047**	**0.1972**	**0.2421**	**0.7452**	**0.4190**	**0.4348**	**0.4858**	**0.5856**	**0.0775**	**0.0554**	**0.0814**

4.1 Preference Adaptation without Sequential Models

We examine whether FWPRec improves preference adaptation without relying on sequential models. We compare FWPRec with six lightweight adaptation methods under the same mean-content user anchor. SignedEMA maintains an exponentially weighted preference state using signed feedback, while LastFeedback uses only the most recent feedback. CumulativeMean and SlidingWindowMean aggregate all historical feedback or a recent window of feedback, respectively. OnlineSGD updates the preference state through online gradient steps, and DiagonalKalman recursively estimates the preference state with a diagonal covariance approximation.

Table 1 reports the held-out test results across six datasets. FWPRec achieves the best result on all four metrics for MIND, EB-NeRD, MovieLens-1M, KuaiRand, and KuaiRec. On MIND and EB-NeRD, CumulativeMean and SignedEMA provide relatively strong baselines, but FWPRec consistently improves both AUC and top-ranked metrics. The improvement is also clear on MovieLens-1M and KuaiRand, where FWPRec outperforms the strongest baseline by a noticeable margin across all metrics. Amazon Video Games shows a different pattern. FWPRec achieves the highest AUC while DiagonalKalman performs better on MRR, nDCG@5, and nDCG@10. SignedEMA is also competitive on this dataset and gives the second-best AUC and top-ranked metrics among the baselines.

4.2 Comparison to Static Baselines and Advanced Sequential Models

We next compare FWPRec with static and sequential recommendation models. The static models provide a reference for whether preference adaptation is needed, while the sequential models test whether explicit sequence modeling is necessary to achieve strong recommendation performance. Our goal is therefore to examine whether a stronger preference adaptation mechanism can recover much of the benefit normally obtained from sequential user modeling.

Table 2 reports the results across six datasets. FWPRec achieves the strongest performance on MIND and EB-NeRD across all four metrics, showing clear gains over both static and sequential baselines. It also remains competitive on KuaiRec. On MovieLens-1M and KuaiRand, simple static models are particularly strong, with ItemKNN performing best on MovieLens-1M and BPRMF performing best on KuaiRand. MovieLens-1M provides limited item metadata, while KuaiRand provides no item text in our setup, making collaborative signals relatively more important. This is consistent with previous findings that carefully tuned classical recommendation methods can remain competitive with more complex neural models (Ferrari Dacrema et al., 2019). FWPRec nevertheless ranks

Table 2: Comparison with static and sequential recommendation models using shared frozen content representations. For each setting, the boldface refers to the highest result, and the underline indicates the second best result. KuaiRand provides no item text, so its column uses learned ID item embeddings in each model's native configuration rather than the shared frozen content representation.

Method	MIND				EB-NeRD				Amazon Video Games			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
Static Models												
Popularity	0.4941	0.2527	0.2253	0.2910	0.4436	0.2635	0.2893	0.3854	0.3126	0.0707	0.0598	0.0745
ItemKNN	0.5034	0.2643	0.2366	0.3028	0.4894	0.3014	0.3330	0.4197	0.5320	0.0965	0.0789	0.0953
BPRMF	0.5771	0.3223	0.2974	0.3635	0.5623	0.3612	0.3997	0.4749	0.5572	0.0761	0.0553	0.0769
Sequential Models												
GRU4Rec	0.6173	0.3492	0.3276	0.3958	0.5633	0.3586	0.3976	0.4729	0.7957	0.1910	0.1790	0.2292
SASRec	0.5755	0.3153	0.2889	0.3610	0.5328	0.3388	0.3747	0.4544	0.7718	0.1823	0.1713	0.2153
DuoRec	0.6147	0.3546	0.3320	0.3977	0.5424	0.3422	0.3792	0.4577	0.7793	0.1621	0.1456	0.1935
LRURec	0.6197	0.3538	0.3321	0.3998	0.5635	0.3586	0.3976	0.4731	0.8114	0.2145	0.2075	0.2566
Mamba4Rec	0.6211	0.3552	0.3336	0.4011	0.5640	0.3594	0.3984	0.4736	0.8055	0.2046	0.1951	0.2442
HSTU	0.6215	0.3542	0.3324	0.4005	0.5620	0.3572	0.3963	0.4717	**0.8146**	**0.2247**	**0.2184**	**0.2674**
FWPRec (Ours)	**0.6259**	**0.3591**	**0.3387**	**0.4043**	**0.5752**	**0.3693**	**0.4092**	**0.4821**	0.8105	0.2086	0.1999	0.2496

Method	MovieLens-1M				KuaiRand				KuaiRec			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
Static Models												
Popularity	0.8286	0.2063	0.1948	0.2478	0.5224	0.2275	0.2211	0.2876	0.1419	0.0195	0.0059	0.0101
ItemKNN	**0.8557**	**0.2946**	**0.3035**	**0.3640**	0.4638	0.2082	0.1969	0.2609	0.3615	0.0189	0.0006	0.0015
BPRMF	0.6631	0.1043	0.0826	0.1150	**0.7770**	**0.4371**	**0.4714**	**0.5198**	0.5567	0.0670	0.0444	0.0645
Sequential Models												
GRU4Rec	0.7997	0.2372	0.2356	0.2833	0.6709	0.3532	0.3675	0.4223	0.5864	0.0784	0.0567	0.0801
SASRec	0.7909	0.2408	0.2392	0.2848	0.6805	0.3436	0.3582	0.4195	0.5757	**0.0935**	**0.0723**	**0.0955**
DuoRec	0.7988	0.2560	0.2565	0.3015	0.6488	0.3074	0.3183	0.3827	0.5294	0.0628	0.0407	0.0583
LRURec	0.8112	0.2816	0.2842	0.3278	0.6740	0.3334	0.3469	0.4088	**0.5926**	0.0821	0.0611	0.0860
Mamba4Rec	0.8102	0.2803	0.2828	0.3264	0.6742	0.3360	0.3497	0.4108	0.5824	0.0765	0.0550	0.0796
HSTU	0.8159	0.2914	0.2949	0.3378	0.6748	0.3408	0.3551	0.4152	0.5811	0.0785	0.0571	0.0801
FWPRec (Ours)	0.7694	0.2047	0.1972	0.2421	0.7452	0.4190	0.4348	0.4858	0.5856	0.0775	0.0554	0.0814

second across all metrics on KuaiRand, while the strong ItemKNN results on MovieLens-1M leave less room for preference adaptation or sequential modeling to provide additional gains. On Amazon Video Games, sequential models provide a clearer advantage, with HSTU achieving the strongest overall performance. The results in Table 2 use the minimal FWPRec input construction throughout. As shown in Appendix C.3, incorporating additional feedback-type and temporal features further improves FWPRec on four of the six datasets and can outperform the strongest baselines reported here in several settings. Overall, these results show that preference adaptation can substantially improve over static user modeling and, on several datasets, achieve performance comparable to explicit sequential models. The relative benefit depends on the strength of the static, collaborative, and sequential signals available in each dataset.

4.3 Ablation Study

We examine the contribution of the main design choices in our model across the six datasets. Figure 2 reports the change in nDCG@10 relative to the full model. Figure 2a changes one component at a time, while Figure 2b considers stronger simplifications involving multiple components.

The leave-one-out results show that removing the negative memory, removing the user anchor from memory construction, sharing the write network, or reducing the memory rank generally lowers performance. The low-rank memory and ablation variants are described in Appendix B.3. The effect is particularly strong on KuaiRand, where using only the positive memory or a single signed memory causes a large drop. In contrast, replacing the fixed write state with an evolving state gives mixed results, improving ML-1M and KuaiRec but degrading the remaining datasets. This suggests that recursively updating the write state is not consistently beneficial.

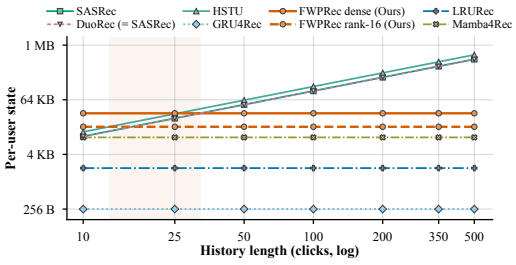
	MIND-L	EB-NeRD	Amazon	ML-1M	KuaiRand	KuaiRec
Positive only	**−0.0061**	**−0.0099**	−0.0008	−0.0020	**−0.0823**	**−0.0083**
Single memory	−0.0004	−0.0001	+0.0010	−0.0002	−0.0817	−0.0048
Evolving write	−0.0013	−0.0002	**−0.0044**	**+0.0056**	−0.0045	+0.0080
Raw k/v	−0.0038	−0.0062	−0.0004	−0.0018	−0.0135	−0.0009
Shared writer	−0.0010	−0.0006	+0.0019	−0.0020	−0.0016	−0.0012
Rank-16	−0.0037	−0.0028	−0.0001	−0.0000	−0.0098	−0.0001

(a) Leave-one-out ablations

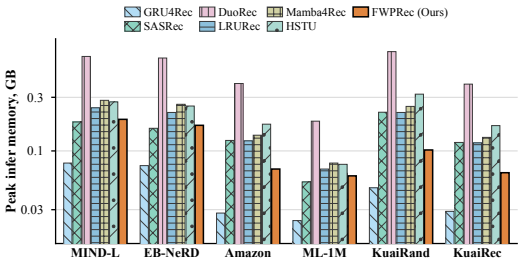
	MIND-L	EB-NeRD	Amazon	ML-1M	KuaiRand	KuaiRec
Anchor only	−0.0277	**−0.0267**	−0.0017	**−0.1321**	**−0.0885**	**−0.0449**
Single + Hebbian	−0.0004	+0.0041	−0.0009	−0.0085	−0.0081	+0.0060
Pos-only + raw k/v	−0.0063	−0.0098	−0.0039	−0.0021	**−0.0843**	−0.0076
Dual, fixed writes	**−0.0343**	−0.0240	**+0.0142**	−0.0870	−0.0527	−0.0292
R16 + shared + raw	−0.0027	−0.0069	−0.0001	−0.0016	−0.0395	−0.0024
Evolving + single	−0.0021	−0.0005	−0.0026	+0.0070	−0.0654	+0.0044

(b) Combined ablations

Figure 2: Ablation results across the six datasets. Each cell reports the change in nDCG@10 relative to the full model. Negative values indicate lower performance, while positive values indicate improvement over the full model. (a) isolates individual design choices, while (b) evaluates more substantial simplifications of the proposed architecture.



(a) Persistent per user state as interaction grows



(b) Peak inference memory across the six datasets

Figure 3: Computational efficiency of FWPRec and the baselines. (a) reports the persistent state maintained for each user as a function of history length. FWPRec keeps a fixed-size memory while the storage required by several sequence-based models increases with history length. (b) reports the peak memory used during inference across the six datasets. The two figures measure different aspects of memory efficiency.

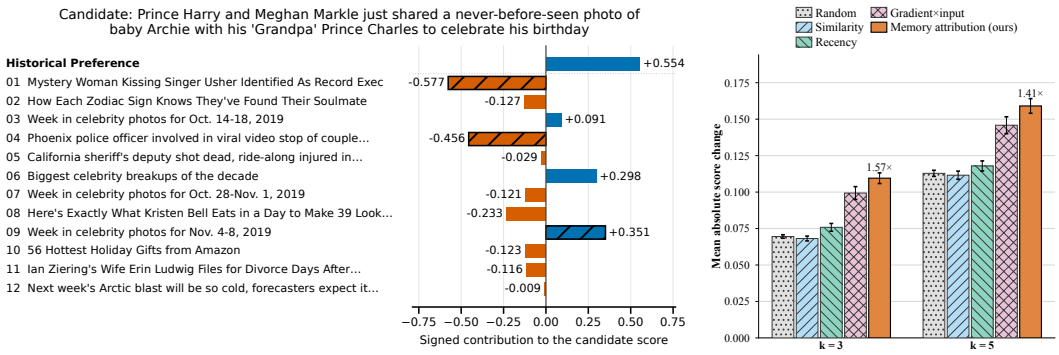
The combined ablations show a clearer trend. Removing memory entirely causes substantial degradation, while using unlearned writes also performs notably worse on most datasets. Simpler variants can remain competitive on individual datasets, but their performance is less consistent across datasets. Overall, the results support the combination of learned associative memory, separate positive and negative memories, and the stable user anchor used in the full model.

4.4 Computational Efficiency

We evaluate efficiency from two aspects, including the persistent user state and peak inference memory. Figure 3a shows how the per-user state changes with history length. Sequence models that retain interaction histories require increasing per-user storage as the history grows. In contrast, FWPRec maintains a fixed-size fast-weight memory, giving $O(1)$ per-user storage with respect to history length. Recurrent and state-space models such as GRU4Rec, LRURec, and Mamba4Rec can also maintain compact persistent states, but differ in their inference-time memory requirements, as shown in Figure 3b.

The full FWPRec model uses approximately 32 KB per user, while the rank-16 variant reduces this to approximately 16 KB (see Appendix B.3 for detail). This reduction introduces only a small performance change on most datasets. As shown in Figure 2a, reducing the memory to rank 16 changes nDCG@10 mainly at the third or fourth decimal place on five of the six datasets, with a larger drop on KuaiRand. This shows that the memory rank provides a simple tradeoff between recommendation performance and per user storage. In particular, rank 16 reduces the user state to about half of the full model while preserving similar performance on most datasets.

Figure 3b reports peak inference memory across the six datasets. Unlike persistent user state, this quantity also includes the memory used by model parameters, item embeddings, batched inputs, in-



(a) Interaction-level contributions for one candidate

(b) Mean absolute score change

Figure 4: Understanding preference updates through interaction-level attribution. **Left**: a case study showing how the static user preference and individual historical interactions contribute to a candidate score. **Right**: removing interactions selected by memory attribution produces a larger score change than random, similarity, recency, and Gradient×Input baselines, indicating that the identified interactions have greater influence on the current prediction.

termediate tensors, and computation workspace. FWPRec remains within a similar memory range to the sequential baselines and below the most memory intensive models in these experiments. Overall, FWPRec avoids memory growth with user history while allowing the memory rank to further reduce the per user storage cost with limited performance change.

4.5 Understanding Preference Updates

Beyond ranking performance, we examine how individual interactions in the fast-weight memory affect a current recommendation. Since the memory update is additive, its contribution to a candidate score can be decomposed across historical interactions. The exact attribution calculation is provided in Appendix C.1.

Figure 4a shows one MIND example. The static user anchor contributes positively to the candidate, while historical interactions introduce both positive and negative preference updates. Several interactions provide strong negative corrections that offset the static preference, and their contributions are not limited to the most recent observations. This illustrates how the memory selectively modifies the static user representation rather than applying a uniform update after each interaction.

We further test whether highly attributed interactions have greater predictive impact. For each test instance, we remove the top-k interactions selected by each attribution method and recompute the candidate score. We compare memory attribution with random selection, item similarity, recency, and Gradient×Input. As shown in Figure 4b, removing interactions selected by memory attribution produces the largest average score change for both $k = 3$ and $k = 5$. Its effect is $1.57\times$ and $1.41\times$ larger than random removal, respectively, and remains consistently larger than Gradient×Input, the strongest competing attribution method. These results show that the interaction contributions identified directly from the fast-weight memory capture historical updates that have greater influence on the current prediction.

5 Conclusion

We introduce a fast-weight memory for preference adaptation in recommender systems. Rather than replacing the recommendation pipeline with a sequential model, our method augments a static user representation with associative memory that stores and retrieves interaction-dependent preference updates. Across multiple datasets, it provides strong ranking performance relative to static and sequential baselines, while preserving the existing item representation and retrieval infrastructure. Our analysis further shows that the learned updates are localized and can be traced back to the interactions that contribute to a current recommendation. Overall, fast-weight memory provides a lightweight way to introduce adaptive user modeling without requiring substantial changes to large-scale recommendation systems.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing and improve the readability of the manuscript. We also used AI-assisted tools for information retrieval and literature discovery, including identifying potentially relevant prior work, with the final selection and verification of references performed by the authors. The research ideas, methodology, experimental design, original code implementation, and interpretation of the results were developed and conducted by authors. Generative AI tools were additionally used to assist with code review and verification, while the implementation and experimental results were independently checked and validated by the authors. We have reviewed all AI-assisted content and take responsibility for the final content of this work, including text or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the model architecture, learning objective, and experimental setup in the main paper. The appendix further provides implementation details, hyperparameter configurations, dataset processing and evaluation protocols, baseline configurations, and additional experimental results to support reproduction of the reported results. Together, these materials specify the model and experimental procedures used throughout the paper.

REFERENCES

Rui Chen, Zonglin Wang, Cundong Tang, Jianwei Zhang, Pu Li, Xiangjie Kong, and Min Huang. EMARec: A sequential recommendation with exponential moving average. *Neural Computing and Applications*, 36(21):12917–12933, 2024. doi: 10.1007/s00521-024-09718-7.

Zhimin Chen, Chenyu Zhao, Ka Mo, Yunjiang Jiang, Jane Lee, Khushhall Chandra Mahajan, Ning Jiang, Kai Ren, Charlie Li, and Wen-Yun Yang. Massive memorization with hundreds of trillions of parameters for sequential transducer generative recommenders. In *International Conference on Learning Representations*, 2026.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198. ACM, 2016. doi: 10.1145/2959100.2959190.

Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pp. 485–492. ACM, 2005. doi: 10.1145/1099554.1099689.

Xinyu Du, Huanhuan Yuan, Pengpeng Zhao, Jianfeng Qu, Fuzhen Zhuang, Guanfeng Liu, Yanchi Liu, and Victor S. Sheng. Frequency enhanced hybrid attention network for sequential recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 78–88. ACM, 2023. doi: 10.1145/3539618.3591689.

Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 101–109, 2019.

Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 488–501. IEEE, 2020. doi: 10.1109/HPCA47549.2020.00047.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations*, 2016.

Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In

Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2553–2561. ACM, 2020. doi: 10.1145/3394486.3403305.

Zhao Huang and Pavel Stakhiyevich. A time-aware hybrid approach for intelligent recommendation systems for individual and group users. *Complexity*, 2021:8826833, 2021. doi: 10.1155/2021/8826833.

Juyong Jiang, Peiyan Zhang, Yingtao Luo, Chaozhuo Li, Jae Boum Kim, Kai Zhang, Senzhang Wang, Xing Xie, and Sunghun Kim. AdaMCT: Adaptive mixture of CNN-transformer for sequential recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 976–986. ACM, 2023. doi: 10.1145/3583780.3614773.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 197–206. IEEE, 2018. doi: 10.1109/ICDM.2018.00035.

Jingyu Li, Zhaocheng Du, Qianhui Zhu, Kaiyuan Li, Zhicheng Zhang, Song-Li Wu, Chaolang Li, and Pengwen Dai. Collectivekv: Decoupling and sharing collaborative information in sequential recommendation. In *International Conference on Learning Representations*, 2026.

Chengkai Liu, Jianghao Lin, Jianling Wang, Hanzhou Liu, and James Caverlee. Mamba4Rec: Towards efficient sequential recommendation with selective state space models. *arXiv preprint arXiv:2403.03900*, 2024.

Langming Liu, Liu Cai, Chi Zhang, Xiangyu Zhao, Jingtong Gao, Wanyu Wang, Yifu Lv, Wenqi Fan, Yiqi Wang, Ming He, Zitao Liu, and Qing Li. LinRec: Linear attention mechanism for long-term sequential recommender systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 289–299. ACM, 2023. doi: 10.1145/3539618.3591717.

Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. Monolith: Real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663*, 2022.

Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. PinnerSage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2311–2320. ACM, 2020. doi: 10.1145/3394486.3403280.

Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. PinnerFormer: Sequence modeling for user representation at pinterest. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3702–3712. ACM, 2022. doi: 10.1145/3534678.3539156.

Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. Contrastive learning for representation degeneration problem in sequential recommendation. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 813–823, 2022. doi: 10.1145/3488560.3498433.

Haohao Qu, Wenqi Fan, Zihuai Zhao, and Qing Li. TokenRec: Learning to tokenize ID for LLM-based generative recommendations. *IEEE Transactions on Knowledge and Data Engineering*, 37 (10):6216–6231, 2025. doi: 10.1109/TKDE.2025.3599265.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, Maciej Kula, Ed Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0452.

Venu Satuluri, Yao Wu, Xun Zheng, Yilei Qian, Brian Wichers, Qieyun Dai, Gui Ming Tang, Jerry Jiang, and Jimmy Lin. Simclusters: Community-based representations for heterogeneous recommendations at twitter. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3183–3193, 2020.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021.

Yehjin Shin, Jeongwhan Choi, Hyowon Wi, and Noseong Park. An attentive inductive bias for sequential recommendation beyond the self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8984–8992, 2024. doi: 10.1609/aaai.v38i8.28747.

João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18:271–282, 2012.

João Vinagre, Alípio Mário Jorge, and João Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In *User Modeling, Adaptation, and Personalization*, pp. 459–470. Springer, 2014.

Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 2400–2409. ACM, 2024. doi: 10.1145/3627673.3679569.

Ga Wu, Shivam Khare, Li Zhou, Yael Brumer, Jun-Ping Ng, and Ruowei Wang. Large-scale user preference tracking via asynchronous and asymmetric updating at twitter. In *2023 IEEE International Conference on Big Data (BigData)*, pp. 315–324. IEEE, 2023.

Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 269–277, 2019.

Zhenrui Yue, Yueqi Wang, Zhankui He, Huimin Zeng, Julian McAuley, and Dong Wang. Linear recurrent units for sequential recommendation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 930–938. ACM, 2024. doi: 10.1145/3616855.3635760.

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58484–58509. PMLR, 2024.

A Detailed Experimental Setup

A.1 Datasets and Preprocessing

We conduct experiments on six real-world recommendation datasets: MIND, EB-NeRD, KuaiRand-Pure, Amazon Reviews 2023 Video Games, MovieLens-1M, and KuaiRec.

MIND-Large. We chronologically split the training behaviors and use the final 10% for validation. The official labeled development set is used as the held-out test set. User histories contain at most 50 interactions, and each impression contains at most 50 candidates. Clicked items are treated as positive feedback. For methods that explicitly support signed feedback, exposed but non-clicked items are assigned a weak negative feedback value of -0.25. Sequential baselines that operate only on positive interaction sequences receive clicked histories only. The resulting test set contains 376,471 impressions.

EB-NeRD. EB-NeRD Small is used as an independent news-domain replication. We construct chronological training and validation partitions and retain at most 50 historical interactions and 50 candidates per impression.

KuaiRand-Pure. We use the standard-policy interaction log and construct a chronological $80/10/10$ train, validation, and test split. User histories contain at most 100 interactions, and each evaluation impression contains at most 50 candidates. Click and non-click events are assigned feedback values of 0.5 and -0.1, respectively, for methods that consume signed feedback. The resulting test set contains 59,190 ranking impressions.

Amazon Reviews 2023. We use the Video Games category and follow the fixed timestamp boundaries used in our implementation. Users are required to have at least five interactions. Ratings of at least four are treated as positive feedback, while ratings of at most two are treated as negative feedback. We retain at most 100 historical interactions and 100 candidates per evaluation instance. The resulting test set contains 19,773 impressions.

MovieLens-1M and KuaiRec. MovieLens-1M is split chronologically within each user using an $80/10/10$ train, validation, and test partition. Users are required to have at least 20 interactions. Ratings of at least four are treated as positive feedback and ratings of at most two as negative feedback. KuaiRec is used for auxiliary behavioral analyses. Positive and negative events are defined using watch-ratio thresholds of 1.0 and 0.2, respectively.

A.2 Item Representations and Candidate Construction

For the representation-controlled experiments, we use fixed item representations generated by Qwen3-Embedding-0.6B. News items are encoded from article text, while Amazon items are encoded from available product metadata. The resulting embeddings are projected to 64 dimensions and ℓ_2 normalized. The item representation table remains frozen during recommender training, and learned item-specific biases are disabled.

Within each dataset, all representation-controlled methods use identical chronological splits, user histories, candidate IDs, relevance labels, and dataset construction seeds. Candidate truncation and negative sampling are deterministic, ensuring that model comparisons differ primarily in the construction and update of the user representation.

NRMS and LSTUR are evaluated separately because they learn their own news encoders jointly with the user model. They use the same dataset split, logged candidates, and positive user histories as the controlled baselines, but do not share the frozen Qwen representation.

A.3 Baseline Implementations

We consider three categories of baselines: static user representations, explicit online preference-update rules, and sequential recommendation models.

Static and lightweight preference-update baselines. StaticProfile represents the user using a fixed mean-content profile and does not maintain a transient adaptive state. PositiveEMA updates the user state using an exponential moving average over positive interactions, while SignedEMA additionally incorporates negative feedback.

LastFeedback uses the representation of the most recent feedback event as the current adaptive user state. CumulativeMean maintains the running mean of all observed feedback representations. SlidingWindowMean computes the mean over a fixed recent interaction window. OnlineSGD updates the user state using an online gradient step after each interaction. DiagonalKalman maintains a Gaussian belief over the user preference vector and performs coordinate-wise Kalman updates using a diagonal covariance approximation. These methods provide lightweight adaptive controls that update an explicit user state without using recurrent, attention-based, or state-space sequence encoders.

Sequential baselines. We compare against GRU4Rec, SASRec, DuoRec, LRURec, HSTU, and Mamba4Rec where applicable. GRU4Rec represents recurrent sequential recommendation, SASRec uses causal self-attention over the interaction history, and DuoRec augments sequential recommendation with contrastive training. LRURec and Mamba4Rec represent efficient recurrent and state-space approaches, while HSTU provides a recent high-capacity sequential baseline.

For news recommendation, we additionally compare with NRMS and LSTUR using their native item encoders. Warm-item auxiliary experiments additionally include ItemKNN, SeqRules, BPRMF, FPMC, BERT4Rec, TiSASRec, BSARec, and LinRec.

FWP-based models. Standalone FWPRec combines a stable mean-content user representation with separate positive and negative fast-weight memories. SASRecFWP attaches the proposed fast-weight memory to the SASRec user representation, while SASRecSignedEMA uses the same SASRec backbone with a signed EMA adaptation module. These models allow us to examine whether explicit adaptive state remains useful when combined with a sequential user encoder.

The ablation study further considers positive-only memory, a shared single memory, fixed versus evolving write states, alternative key and value construction, shared versus independent write networks, and dense versus low-rank memory parameterizations.

A.4 Training and Hyperparameter Selection

All final neural-model results are reported over training seeds $\{42, 43, 44\}$. Dataset construction and candidate sampling use a fixed seed of 42. Hyperparameter selection is performed using training seed 42 and validation nDCG@10, with test evaluation disabled. The selected configuration is then fixed across the three final training seeds. For each run, we use the checkpoint with the highest validation nDCG@10 for held-out test evaluation.

For FWPRec, we tune the shared memory-decay factor ρ, which controls how much previously stored preference information is retained during a memory update, and the negative-feedback weight α, which controls the contribution of the negative-feedback memory to the adapted user representation. Table 3 summarizes the search space and selected values. We evaluate all combinations independently for each dataset using the corresponding training configuration. Ablation variants reuse the selected dataset-specific values without additional tuning. We set InfoNCE temperature $\tau = 1$ as default without tuning in our experiments.

Baseline learning rates follow bounded validation searches conducted during development. On MIND, GRU4Rec, DuoRec, LRURec, and HSTU are evaluated with learning rates $\{5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}\}$, with 2×10^{-3} selected for all four models. These values are retained for the final MIND experiments without an additional search. On Amazon, the same search is performed for HSTU and LRURec, selecting 10^{-3} and 2×10^{-3}, respectively. Other baseline learning rates follow the dataset-specific defaults.

On MIND, models are trained until convergence with batch size 1,024 and early-stopping patience of three epochs. The default learning rate is 10^{-3}, while GRU4Rec, DuoRec, LRURec, and HSTU use 2×10^{-3}.

On KuaiRand, all neural models are trained until convergence with batch size 512, early-stopping patience of two epochs, and learning rate 2×10^{-3}.

Table 3: Hyperparameter search space and selected values for FWPRec.

	ρ	α
Search space	$\{0.9, 0.99, 1.0\}$	$\{0.3, 0.5, 1.0\}$
MIND	1.0	1.0
EB-NeRD	1.0	1.0
Amazon	0.9	0.5
ML-1M	0.9	0.3
KuaiRand	1.0	1.0
KuaiRec	0.9	1.0

On Amazon, models are trained until convergence with batch size 256 and early-stopping patience of three epochs. The default learning rate is 10^{-3}, while LRURec uses 2×10^{-3}.

All neural experiments are run on CUDA. We record the resolved configuration, random seeds, selected checkpoint, aggregate metrics, and impression-level predictions for each final run.

A.5 Statistical Analysis

For confirmatory comparisons, we compute paired hierarchical bootstrap confidence intervals using 2,000 bootstrap replicates. For overall ranking metrics, users are sampled with replacement so that impressions from the same user remain within the same bootstrap cluster. For adaptation experiments, complete preference-shift events are used as the sampling unit.

Training seeds are resampled at the outer level. Prediction archives are aligned by user, impression, candidate IDs, relevance labels, adaptation-event ID, and adaptation step before paired comparison. Statistical analyses are performed separately for each dataset.

For overall recommendation quality, we use relative nDCG@10 non-inferiority with a margin of -1%. For preference adaptation, the primary comparison uses AdaptNDCG@10 with $H = 3$ on MIND and KuaiRand. On Amazon, where explicit negative ratings provide a stronger negative-feedback signal, the primary adaptive endpoint is negative-rich nDCG@10. Other metrics and cohort analyses are treated as complementary descriptive evidence.

B Model Architecture and Input Construction

B.1 Enhanced Learnable Writer Inputs

The main experiments use the minimal writer input $\mathbf{x}_\ell = [\mathbf{u}_0 \| \mathbf{v}_{i_\ell} \| r_\ell]$. Here, we describe an optional enhanced input construction that additionally provides the writer networks with feedback type and temporal context. Specifically, we augment the base user representation, item representation, and scalar feedback with the feedback type and temporal context. These additional features are used only by the writer networks to determine how an interaction should update the memory and are not themselves stored in the fast-weight memory.

For an interaction ℓ with feedback value r_ℓ, we first map the feedback sign to a discrete action type

$$z_\ell = \begin{cases} 0, & r_\ell < 0, \\ 1, & r_\ell = 0, \\ 2, & r_\ell > 0. \end{cases} \tag{11}$$

A learnable embedding table then maps the action type to

$$\mathbf{a}_\ell = \mathbf{E}_{\text{action}}[z_\ell], \qquad \mathbf{E}_{\text{action}} \in \mathbb{R}^{3\times 8}, \tag{12}$$

where $\mathbf{a}_\ell \in \mathbb{R}^8$. The embedding distinguishes negative, neutral, and positive interactions, while the scalar r_ℓ retains the corresponding feedback magnitude.

We further provide the writer networks with temporal information describing the age of each interaction. Let Δt_ℓ denote the elapsed time between interaction ℓ and the current recommendation

Table 4: Inputs to the four writer networks.

Input	Dim.	Learned	Description
$\mathbf{u}_0$	64	No	Base user representation
$\mathbf{v}_{i_\ell}$	64	No	Interaction-item representation
$\mathbf{a}_\ell$	8	Yes	Feedback-type embedding
r_ℓ	1	No	Scalar feedback
$\boldsymbol{\tau}_\ell$	3	No	Temporal features
$\mathbf{x}_\ell$	140	—	Complete writer input

Table 5: Architecture and role of the four writer networks.

Function	Input	Output	Role
f_k	129	64	Key residual
f_c	129	64	Correction residual
f_+	129	1	Positive gate
f_-	129	1	Negative gate

request. We construct

$$\boldsymbol{\tau}_\ell = \left[\min\left(\frac{\log(1+\Delta t_\ell)}{\log(1+T_{\text{long}})}, 4\right), \exp\left(-\frac{\Delta t_\ell}{T_{\text{short}}}\right), \exp\left(-\frac{\Delta t_\ell}{T_{\text{long}}}\right)\right] \in \mathbb{R}^3. \tag{13}$$

The first component captures logarithmic interaction age, while the second and third capture short- and long-term recency. Unless otherwise specified, we use

$$T_{\text{short}} = 3600, \qquad T_{\text{long}} = 86400, \tag{14}$$

corresponding to one hour and one day, respectively.

The complete input shared by the four writer networks is therefore

$$\mathbf{x}_\ell = [\mathbf{u}_0 \| \mathbf{v}_{i_\ell} \| \mathbf{a}_\ell \| r_\ell \| \boldsymbol{\tau}_\ell] \in \mathbb{R}^{140}, \tag{15}$$

where $\mathbf{u}_0 \in \mathbb{R}^{64}$ and $\mathbf{v}_{i_\ell} \in \mathbb{R}^{64}$ denote the base user and interaction-item representations, respectively.

Table 4 summarizes the writer inputs and whether they introduce trainable parameters.

Among these additional features, only $\mathbf{E}_{\text{action}}$ introduces trainable parameters. The temporal features are deterministic transformations of Δt_ℓ and do not introduce additional model parameters.

Note that all results reported in the main paper use this minimal input construction without the additional interaction-level features described above. We include the enhanced construction here as an optional extension to show how additional feedback and temporal information can be incorporated into the writer networks without changing the fast-weight memory formulation. We empirically compare the two input constructions in Section C.3, showing that the additional features can provide substantial gains on some datasets, while the minimal construction remains competitive or performs better on others.

B.2 Writer Network Architecture

For the main experiments, f_k, f_c, f_+, and f_- are implemented as four independent two-layer MLPs. Given the writer input $\mathbf{x}_\ell \in \mathbb{R}^{129}$ defined in Section B.1, each network has the form

$$f(\mathbf{x}) = \mathbf{W}_2\,\text{GELU}\,(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2, \tag{16}$$

with hidden dimension 128. The four writer networks differ only in their output dimensions:

$$\begin{aligned} f_k &: \mathbb{R}^{129} \to \mathbb{R}^{128} \to \mathbb{R}^{64}, \\ f_c &: \mathbb{R}^{129} \to \mathbb{R}^{128} \to \mathbb{R}^{64}, \\ f_+ &: \mathbb{R}^{129} \to \mathbb{R}^{128} \to \mathbb{R}, \\ f_- &: \mathbb{R}^{129} \to \mathbb{R}^{128} \to \mathbb{R}. \end{aligned} \tag{17}$$

The key and correction writers predict residual adjustments to the base user and interacted-item representations:

$$\mathbf{k}_\ell = \text{Norm}\left(\mathbf{u}_0 + f_k(\mathbf{x}_\ell)\right), \tag{18}$$

and

$$\mathbf{c}_\ell = \text{Norm}\left(\mathbf{v}_{i_\ell} + f_c(\mathbf{x}_\ell)\right). \tag{19}$$

In this form, f_k adjusts the stable user representation to determine where an interaction is written in memory, while f_c adjusts the interacted-item representation to determine the corresponding correction.

The remaining two writers determine the strengths of positive and negative memory updates:

$$\beta_\ell^+ = \sigma\left(f_+(\mathbf{x}_\ell)\right)[r_\ell]_+, \qquad \beta_\ell^- = \sigma\left(f_-(\mathbf{x}_\ell)\right)[-r_\ell]_+, \tag{20}$$

where $[x]_+ = \max(x, 0)$. The sigmoid outputs act as learned write gates, while the magnitude of r_ℓ determines the corresponding update strength.

Table 5 summarizes the four learned functions.

The main reported experiments use these four independent MLPs. Some MIND and ablation configurations instead use a shared writer trunk,

$$\mathbf{h}_\ell = \mathrm{GELU}\left(\mathbf{W}_{\mathrm{trunk}}\mathbf{x}_\ell + \mathbf{b}_{\mathrm{trunk}}\right), \tag{21}$$

followed by separate linear heads for the key, correction, positive gate, and negative gate. We report this parameterization separately when it is used, rather than treating it as the same model architecture.

B.3 Low-Rank Fast-Weight Memory

For the rank-16 variant, we represent each positive and negative fast-weight memory using at most $r = 16$ rank-one components:

$$\mathbf{M}_t^\pm = \sum_{j=1}^{r} \mathbf{c}_{t,j}^\pm (\mathbf{k}_{t,j}^\pm)^\top, \qquad r = 16, \tag{22}$$

where $\mathbf{c}_{t,j}^\pm \in \mathbb{R}^d$ stores the scaled residual correction associated with key $\mathbf{k}_{t,j}^\pm \in \mathbb{R}^d$. The resulting memory therefore has rank at most 16. Each new memory write adds one rank-one component following the update in Eq. 7. Once the memory reaches capacity, the oldest component is replaced, while the retained components continue to follow the memory decay. This reduces the storage and read complexity of each memory from $O(d^2)$ to $O(rd)$, while keeping the memory size independent of the interaction-history length.

B.4 Ablation Variants

We provide additional details for the ablation variants in Figure 2. **Raw k/v** removes the residual connections used in the key and correction construction, giving $\mathbf{k}_\ell = \mathrm{Norm}(f_k(\mathbf{x}_\ell))$ and $\mathbf{c}_\ell = \mathrm{Norm}(f_c(\mathbf{x}_\ell))$. **Dual, fixed writes** retains separate positive and negative memories but removes the learned writers, using $\mathbf{k}_\ell = \mathrm{Norm}(\mathbf{u}_0)$, $\mathbf{c}_\ell = \mathrm{Norm}(\mathbf{v}_{i_\ell})$, and $\beta_\ell^\pm = [\pm r_\ell]_+$. **Single + Hebbian** uses one shared memory and replaces the delta-rule update with the direct outer-product write $\mathbf{c}_\ell \mathbf{k}_\ell^\top$, without subtracting the correction already stored in memory. **R16 + shared + raw** combines the rank-16 memory, shared writer trunk, and raw key/correction construction in a single variant. Finally, **Evolving + single** uses an evolving write state together with one shared fast-weight memory. For the evolving-write variant, the writer uses the current adapted user representation $\mathbf{u}_{\ell-1}$ in place of the fixed anchor $\mathbf{u}_0$ when constructing $\mathbf{x}_\ell$.

C Detailed Description of Interaction Attribution

C.1 Interaction Attribution

We compute the importance of each historical interaction by tracing its memory write to the current recommendation score. Since the fast-weight memory is updated additively over interactions, the resulting score can be decomposed into the contribution of the stable user anchor and the contribution of each memory write.

Let $\mathbf{u}_0$ denote the stable user anchor and $\mathbf{M}_T^+$ and $\mathbf{M}_T^-$ denote the positive and negative memories after processing the interaction history. Before normalization, the adapted user representation is

$$\widetilde{\mathbf{u}}_T = \mathbf{u}_0 + \mathbf{M}_T^+ \mathbf{u}_0 - \alpha \mathbf{M}_T^- \mathbf{u}_0, \tag{23}$$

and

$$\mathbf{u}_T = \frac{\widetilde{\mathbf{u}}_T}{\|\widetilde{\mathbf{u}}_T\|}. \tag{24}$$

For candidate item i, the recommendation score is

$$s_{i,T} = \frac{\mathbf{u}_T^\top \mathbf{v}_i}{\tau}. \tag{25}$$

For an interaction ℓ, we isolate the write introduced by that interaction from the corresponding memory update. For the positive memory, we define

$$\mathbf{D}_\ell^+ = \mathbf{M}_\ell^+ - \rho \mathbf{M}_{\ell-1}^+, \tag{26}$$

and similarly for the negative memory,

$$\mathbf{D}_\ell^- = \mathbf{M}_\ell^- - \rho \mathbf{M}_{\ell-1}^-. \tag{27}$$

These terms correspond to the new memory corrections written by interaction ℓ. Their effect on the user representation is obtained by applying the write to the stable anchor,

$$\mathbf{D}_\ell^+ \mathbf{u}_0 \qquad \text{and} \qquad \mathbf{D}_\ell^- \mathbf{u}_0. \tag{28}$$

If m_ℓ subsequent memory updates occur after interaction ℓ, its remaining contribution at prediction time is

$$\rho^{m_\ell} \mathbf{D}_\ell^+ \mathbf{u}_0 \tag{29}$$

for the positive memory and

$$\rho^{m_\ell} \mathbf{D}_\ell^- \mathbf{u}_0 \tag{30}$$

for the negative memory.

We then define the signed contribution of interaction ℓ to candidate item i as

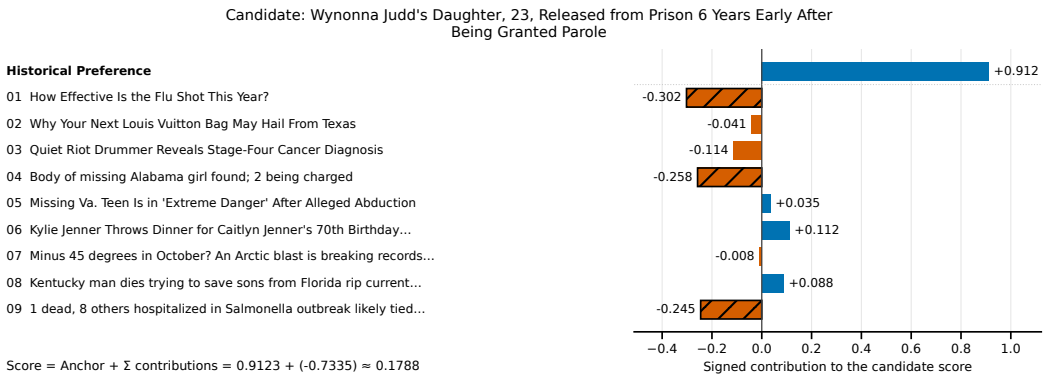
$$\boxed{A_{\ell,i} = \frac{\mathbf{v}_i^\top \left(\rho^{m_\ell} \mathbf{D}_\ell^+ \mathbf{u}_0 - \alpha \rho^{m_\ell} \mathbf{D}_\ell^- \mathbf{u}_0 \right)}{\tau \, \|\widetilde{\mathbf{u}}_T\|}} \tag{31}$$

A positive value increases the score of candidate i, while a negative value decreases it. The sign depends on how the memory write aligns with the candidate representation. Therefore, an interaction written into the positive memory does not necessarily contribute positively to every candidate.

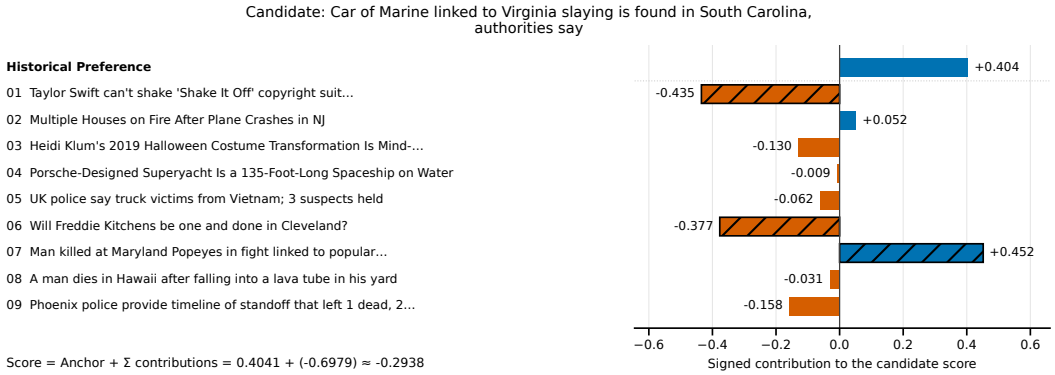
The recommendation score can therefore be written as

$$s_{i,T} = \frac{\mathbf{v}_i^\top \mathbf{u}_0}{\tau \, \|\widetilde{\mathbf{u}}_T\|} + \sum_\ell A_{\ell,i}. \tag{32}$$

We verify this decomposition numerically for each evaluated impression. Therefore, $A_{\ell,i}$ gives the exact contribution of interaction ℓ to the score produced from the observed history.



(a) The static user anchor remains positive after a substantial negative correction from the interaction history.



(b) The interaction history reverses the positive contribution from the static user anchor.

Figure 5: Additional examples of interaction-level preference attribution. The blue bar at the top denotes the contribution from the static user anchor, while the remaining bars show signed contributions from historical interactions.

This attribution should not be interpreted as the counterfactual score change caused by removing interaction ℓ. Removing an earlier interaction can change later memory writes and the final normalization of $\widetilde{\mathbf{u}}_T$. The attribution instead measures how each observed interaction contributes to the prediction produced from the complete history.

C.2 Additional Preference Attribution Case Studies

We provide two additional examples of interaction-level preference attribution on MIND. In each case, the static user anchor contributes to the candidate score, while historical interactions provide signed preference updates. Positive values increase the score and negative values decrease it.

In Figure 5a, the static user anchor contributes $+0.912$ to the candidate score. Historical interactions contribute -0.734 in total, reducing the final score to 0.179. The corrections are distributed across multiple interactions and include both positive and negative effects. For example, the flu-shot article contributes -0.302, the missing-girl article contributes -0.258, and the Salmonella-outbreak article contributes -0.245, while several other interactions contribute positively. The history therefore weakens, but does not reverse, the preference represented by the static user embedding.

In Figure 5b, the static anchor contributes $+0.404$, while the historical interactions contribute -0.698 in total, resulting in a final score of -0.294. The largest negative contributions come from the Taylor Swift copyright article and the Cleveland football article, with contributions of -0.435 and -0.377, respectively. At the same time, the Popeyes-related interaction contributes $+0.452$. The history therefore contains preference updates in both directions, with their combined effect reversing the static preference for this candidate.

Table 6: Held-out test comparison between the paper-literal and enhanced FWPRec parameterizations. Enhanced FWPRec uses $\mathbf{x}_\ell = [\mathbf{u}_0 \| \mathbf{v}_{i_\ell} \| \mathbf{a}_\ell \| r_\ell \| \boldsymbol{\tau}_\ell]$. Results are means over training seeds $\{42, 43, 44\}$; boldface denotes the higher result within each dataset and metric.

Method	MIND				EB-NeRD				Amazon Video Games			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
FWPRec-Literal	**0.6259**	**0.3591**	**0.3387**	**0.4043**	**0.5752**	**0.3693**	**0.4092**	**0.4821**	0.8105	0.2086	0.1999	0.2496
FWPRec-Enhanced	0.6156	0.3498	0.3271	0.3953	0.5699	0.3642	0.4043	0.4777	**0.8233**	**0.2390**	**0.2342**	**0.2825**
Method	**MovieLens-1M**				**KuaiRand**				**KuaiRec**			
	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10	AUC	MRR	N@5	N@10
FWPRec-Literal	0.7694	0.2047	0.1972	0.2421	0.7452	0.4190	0.4348	0.4858	0.5856	0.0775	0.0554	0.0814
FWPRec-Enhanced	**0.7993**	**0.2356**	**0.2327**	**0.2800**	**0.8093**	**0.4952**	**0.5170**	**0.5613**	**0.6088**	**0.0889**	**0.0678**	**0.0933**

These examples further show that fast-weight memory does not apply a single global correction to the user representation. Different historical interactions can contribute in different directions, and their combined effect determines how the static preference is adapted for the current candidate.

C.3 Effect of Enhanced Writer Inputs

We further evaluate the enhanced writer inputs introduced in Section B.1 the simpler formulation used in the main paper. The main FWPRec model constructs each writer input as $\mathbf{x}_\ell = [\mathbf{u}_0 \| \mathbf{v}_{i_\ell} \| r_\ell]$, while the enhanced variant additionally includes the learned action embedding $\mathbf{a}_\ell$ and temporal features $\boldsymbol{\tau}_\ell$. Both variants otherwise use the same fast-weight memory formulation.

Table 6 shows that the additional interaction features can provide substantial gains, but their benefit depends on the dataset. The enhanced variant improves all four metrics on Amazon Video Games, MovieLens-1M, KuaiRand, and KuaiRec. The gains are particularly pronounced on KuaiRand, where AUC increases from 0.7452 to 0.8093 and nDCG@10 from 0.4858 to 0.5613. Similar improvements are observed on MovieLens-1M and Amazon Video Games. In contrast, the simpler formulation performs consistently better on MIND and EB-NeRD, although the differences are relatively small.

These results suggest that explicit feedback-type and temporal information can strengthen preference updates when such interaction context is informative, but is not uniformly beneficial across datasets. Importantly, the simpler writer input used throughout the main paper already provides strong performance without these additional features. We therefore retain it as the core FWPRec formulation and present the enhanced input as an optional extension when richer interaction information is available and useful.