Versioned Late Materialization for Ultra-Long Sequence Training in Recommendation Systems at Scale

Liang Guo, Ge Song, Litao Deng, Jianhui Sun, Chufeng Hu, Lu Zhang, Zhen Ma, Shouwei Chen, Weiran Liu, Sarang Masti Sreeshylan, Xiaoxuan Meng, Yanzun Huang[†]
Meta Platforms, Inc., USA

Abstract

Modern Deep Learning Recommendation Models (DLRMs) follow scaling laws with sequence length, driving the frontier toward ultra-long User Interaction History (UIH). However, the industry-standard "Fat Row" paradigm, which pre-materializes these sequences into every training example, creates a storage and I/O wall where data infrastructure usage exceeds GPU training capacity due to data redundancy that is amplified in multi-tenant environments where models with vastly different sequence length requirements share a union dataset. We present a *versioned late materialization* paradigm that eliminates this redundancy by storing UIH once in a normalized, immutable tier and reconstructing sequences just-in-time during training via lightweight versioned pointers. The system ensures Online-to-Offline (O2O) consistency through a bifurcated protocol that prevents future leakage across both streaming and batch training, while a read-optimized immutable storage layer provides multi-dimensional projection pushdown for heterogeneous model tenants. Disaggregated data preprocessing with pipelined I/O prefetching and data-affinity optimizations masks the latency of training-time sequence reconstruction, keeping training throughput compute-bound by GPUs. Deployed on production DLRMs, the system reduces training data infrastructure resource usage while enabling aggressive sequence length scaling that delivers significant model quality gains, serving as the foundational data infrastructure for modern recommendation model architectures, including HSTU [26] and ULTRA-HSTU [7].

CCS Concepts

• **Information systems → Recommender systems**.

Keywords

Recommendation Systems, Machine Learning, Late Materialization, Training Data Infrastructure, Sequence Modeling

1 Introduction

The advances of DLRMs over the past decade have been fundamentally driven by the length of the User Interaction History (UIH). Early candidate-dependent search approaches (e.g., DIN [28], SIM [19], ETA [5]) select the top-K relevant history items per candidate and scale UIH from 10^1 to 10^4 events. More recently, HSTU [26] reformulates recommendation as a sequential transduction task, applying full causal self-attention over the entire interaction sequence to avoid information loss and pave the way for the Generative Recommender framework. Notably, the work established that recommendation model quality follows a power-law scaling relationship with training compute—providing the first evidence that the scaling laws driving the LLM revolution extend to recommendation systems. This finding has been corroborated across the industry [4, 6, 7, 12, 14, 15, 21, 23], with the consistent result that longer sequences yield monotonically improved recommendation quality—pushing the frontier towards 10^5+ events.

However, these architectural advances have exposed a fundamental infrastructure bottleneck. The industry-standard approach pre-materializes complete UIH sequences into every training example, a "Fat Row," causing K-fold data redundancy as the same user history is duplicated across all ranking requests within the lookback window. The resulting storage and I/O amplification drives the total resource usage of data supporting services to exceed the usage of training GPUs themselves, effectively capping the scaling potential of next-generation architectures.

We observe that this problem is based on a false necessity. The industry adopted physical pre-materialization as a blanket mechanism for Online-to-Offline (O2O) consistency—ensuring training reproduces the exact feature state observed at inference time. While O2O consistency is required, pre-materialization is only sufficient, not necessary, to achieve it. UIH sequence data possesses a structural property—it is an append-only, temporally ordered, immutable sequence—that admits a more efficient alternative: storing the canonical history once and reconstructing the exact inference-time state via a temporal predicate at training time. Although late materialization and multi-version concurrency control (MVCC) are well-established techniques in the database community [1, 3], their application to recommendation training data pipelines—where the interplay of streaming and batch training modes, future-leakage prevention, and multi-tenant sequence projection creates domain-specific challenges absent in traditional query processing—has not been explored. In this paper, we present a *versioned late materialization* paradigm that exploits this insight and breaks through the storage and I/O wall. Our contributions include:

(1) A formalization of the "Fat Row" Problem, with quantitative analysis showing that pre-materialized UIH redundancy drives data infrastructure usage to exceed GPU training deployment in recommendation systems (Section 2).
(2) A *versioned late materialization protocol* that achieves O2O consistency and prevents future leakage through lightweight version metadata, serving as a unified solution for both streaming training and batch training (Section 3).
(3) A *production-grade training-time materialization architecture* combining read-optimized immutable storage with multi-tenant sequence projection pushdown, disaggregated data preprocessing, pipelined I/O prefetching, and data-affinity sharding to mask sequence reconstruction latency and keep training throughput GPU-bound (Section 4).

[†]Work done at Meta Platforms, Inc.

(4) *Deployment at scale*, reducing training data infrastructure resource usage while enabling model quality gains through aggressive sequence length scaling, serving as the foundational infrastructure for HSTU [26] and trillion-parameter sequential transducers (Section 5).

2 The Fat Row Problem

This section examines the fundamental infrastructure bottleneck that emerges when scaling UIH sequences under the industry-standard pre-materialization paradigm. We first describe the Online-to-Offline consistency requirement that motivates the "Fat Row" architecture, then quantify the storage and I/O wall, and finally show how multi-tenant training environments amplify the penalty.

2.1 Online-to-Offline Consistency

A defining challenge of industrial recommendation systems is that model training typically occurs asynchronously: seconds, minutes, or hours after the original ranking request, yet training must reproduce the exact feature state observed during online inference, to avoid a well-documented source of model quality degradation [22] from O2O data skew. The most pernicious form of skew is *future leakage*. In a typical lifecycle, a ranking request is triggered to retrieve videos for a user's next session at T_{request}. The user's engagement (the label) with one of the videos in the next session occurs at T_{label} ($T_{\text{label}} > T_{\text{request}}$). When training on this session asynchronously at T_{train}, if a training example inadvertently includes user events that occurred after T_{request}, the model learns to exploit information unavailable at inference time, producing artificially inflated offline metrics that do not transfer online.

To ensure that a model is trained on the exact state of features seen during online inference, the industry-standard architecture employs a feature snapshotting and pre-materialization mechanism illustrated in Figure 1. During the online inference funnel, the high-dimensional feature vectors used for ranking, including long-sequence UIH features extracted from the frequently updated UIH store, are "frozen" and cached in a high-throughput key-value feature store. As ground-truth labels from user interactions (e.g., likes or video completions) arrive asynchronously, an ingestion service joins them with the cached snapshots to generate a fully materialized training example [9] containing all feature values and associated late-arrival labels—a "Fat Row." This architecture must serve both *streaming* online training (where samples are consumed within seconds) and *batch* offline training (where samples may be replayed days later), making the consistency guarantee non-negotiable. Although this architecture ensures high-fidelity data, it forces the system to store and transport redundant UIH sequence features for every individual training example, creating the infrastructure bottlenecks that our system aims to solve.

2.2 Storage and I/O Wall

As DLRMs evolve towards ultra-long sequences to capture lifelong user interests, the systemic redundancy of the "Fat Row" paradigm has become a bottleneck to model scaling. In this paradigm, training data are generated by physically denormalizing and materializing UIH features into every individual training example. The impact of this approach is best illustrated by the relationship between request

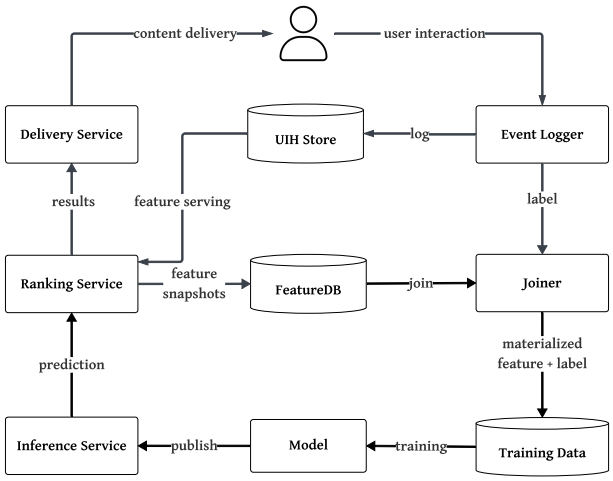

Figure 1: Feature snapshotting and pre-materialization architecture for RecSys training data generation.

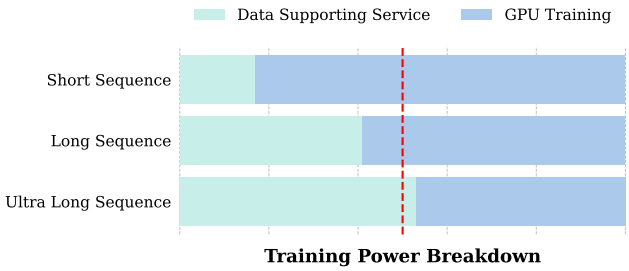

Figure 2: Estimation of data supporting service and GPU training power of a recommendation product to support short, long, and ultra-long sequences.

frequency (K) and historical lookback (N). To provide the model with an N-day UIH window, the user's history for the preceding $N-1$ days—which remains static across K requests in the same day—is physically duplicated into every generated training example. This results in a K-fold amplification in both storage footprint and write volume for the vast majority of the UIH payload.

As shown in Figure 2, our analysis of a recommendation surface reveals the severity of this wall. The total resource consumption of data supporting services—comprising training data storage, online and offline training data ingestion, and preprocessing compute—exceeds that of GPU training when enabling ultra-long UIH sequences. In this imbalance, UIH sequence features account for the majority of the training data volume and read bandwidth. The industry is forced into a "diminishing returns" trap where the infrastructure demands of supporting longer sequences eventually outweigh the marginal gains in model quality, effectively capping the potential of next-generation architectures.

2.3 The Multi-Tenant Penalty

The Fat Row penalty is amplified in multi-tenant environments. In mature recommendation ecosystems, it is standard practice to maintain a union training dataset, a centralized dataset serving multiple model tenants across the recommendation funnel (e.g., Retrieval, Pre-ranking, and Ranking) of the same product. This unified approach offers significant operational benefits by minimizing storage redundancy and pipeline maintenance, and providing a consistent data baseline for cross-model performance ablation.

However, under the "Fat Row" paradigm, this shared infrastructure creates a multi-tenant penalty. Different models have vastly different requirements: a candidate retrieval model may only require the last 100 user events; a late-stage ranking model may require 100× more events to maximize precision. Because the union dataset must accommodate the maximum requirement, it materializes and writes the maximum sequence length into every training example. Without sequence-aware partial projection at the storage layer, simpler models are forced to consume and transport the entire monolithic UIH data, leading to read amplification and making it prohibitive to scale UIH for flagship models without unfairly impacting the rest of the fleet.

3 Methodology

To circumvent this storage and I/O wall, we pivot from the storage-heavy pre-materialization "Fat Row" paradigm to a compute-centric late materialization approach that stores UIH sequence features as lightweight version metadata referencing a normalized, versioned history, decoupling the storage footprint of UIH from the number of training examples. For this approach to be viable, it must ensure O2O consistency—reconstructing the exact inference-time UIH state during training—and preserve data freshness, as any regression in the availability of recent events would impair the model's ability to capture real-time user intent shifts. We first argue why late materialization is safe for UIH sequences (Section 3.1), then present the practical challenge of serving both streaming and batch training (Section 3.2), and finally introduce the *versioned late materialization* protocol (Section 3.3).

3.1 The False Necessity of Pre-Materialization

The universality of the Fat Row architecture rests on an implicit thesis: to ensure O2O consistency and prevent future leakage, the complete UIH sequence must be physically snapshotted into every training example at inference time. This thesis conflates two independent claims: (C1) that O2O consistency is a hard requirement, and (C2) that physical pre-materialization is the *necessary mechanism* to achieve it. Claim C1 is unambiguously true. However, Claim C2 is only sufficient, not necessary, and it is the sole driver of the infrastructure crisis described in Section 2.

Claim C2 does not hold for UIH sequences due to a structural invariant: unlike general features whose values may be overwritten in place, UIH is an append-only, temporally ordered, immutable sequence—a user's history at time t is deterministically defined as the set of events with `timestamp` $\leq t$, and individual events are never retroactively modified or deleted once written. This invariant has three consequences that collectively invalidate C2. First, *reconstructibility*: the exact state of UIH at any historical inference time t can be recovered from a single canonical copy via a temporal range scan, which requires no physical snapshot. Second, *future leakage prevention by predicate*: the temporal predicate `timestamp` $\leq t$ is both necessary and sufficient to exclude post-inference events, providing the same guarantee as physical snapshotting without data duplication. Third, *version metadata sufficiency*: logging a lightweight UIH sequence versioning feature ($O(1)$ per sample) replaces logging the entire sequence ($O(\text{seq\_length})$). In database terms, this admits an MVCC-style protocol [2]—store canonical data once, log version metadata, and reconstruct state at read time—an application of late materialization [1] to the recommendation training domain.

3.2 Batch and Streaming Training

Data freshness is a well-established driver of DLRM quality, enabling models to capture emerging trends and in-session user intent shifts. To satisfy freshness requirements while supporting large-scale experimentation, a unified training data generation pipeline is maintained for each of our products to serve both training paradigms. Online streaming training consumes a distributed real-time messaging stream [11] to power latency-sensitive production models. The same stream is persisted in data warehouse tables [24] hourly to allow batch-based offline training for experimental iterations and new model warm-up.

Eliminating the "Fat Row" redundancy within this unified pipeline is non-trivial. A naive normalization approach, storing UIH features in a side table and joining them during an offline ETL process, fails for two reasons. First, it violates the unified ingestion model: online training workers cannot perform heavy analytical joins on live streams without breaching stringent latency SLAs. Second, it introduces an O2O consistency gap: if the offline join materializes a different UIH snapshot than the one observed during online ranking, the resulting training-serving skew degrades model quality. These constraints motivate the *versioned late materialization* protocol presented in Section 3.3, which provides a unified UIH point-in-time reconstruction mechanism that is transparent to training job types.

3.3 Versioned Late Materialization Protocol

As illustrated in Figure 3, the protocol is built upon the bifurcation of UIH into a Mutable UIH portion, which handles recent events comprising a small percentage of the UIH sequence, and an Immutable UIH portion for a user's long-term history (physical storage optimizations are detailed in Section 4). The protocol consists of lightweight inference-time snapshotting and versioned training-time late materialization.

Inference-Time Snapshotting. During the ranking phase, the system fetches sequences from both Mutable UIH and Immutable UIH to construct a complete UIH sequence for model inference. To minimize feature logging volume, instead of logging the snapshot of complete UIH at inference time, the system only persists the mutable UIH for each training example. For the massive immutable UIH portion, the system logs only the lightweight version metadata required for training-time UIH reconstruction: temporal boundaries (`start_ts`, `end_ts`), sequence length, and an optional checksum to verify the correctness of the reconstructed UIH during training. By logging compact versioned metadata rather than raw sequence

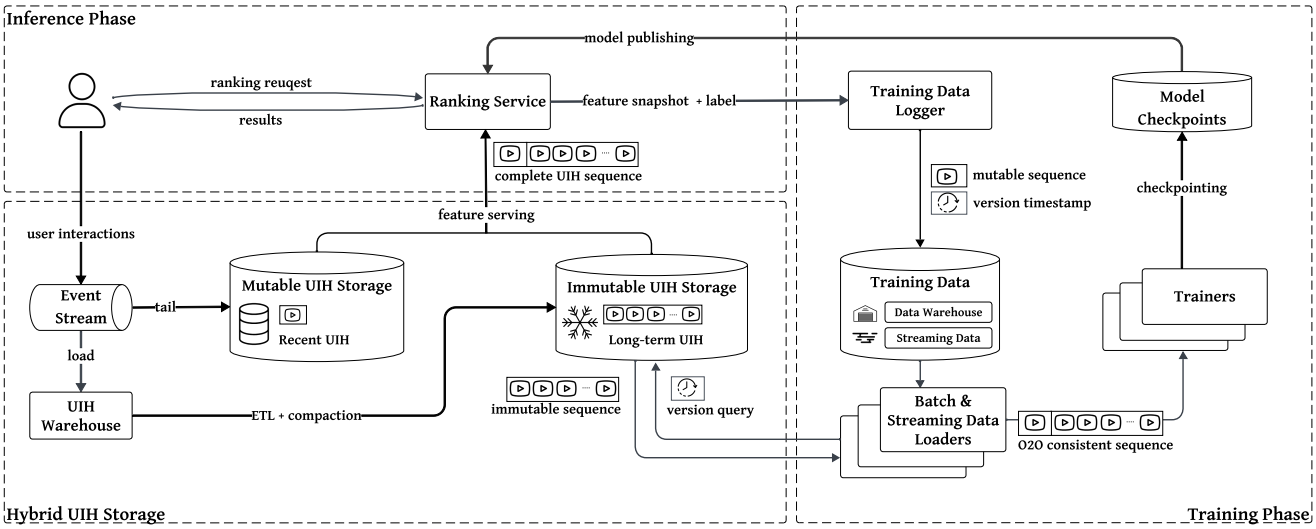

Figure 3: Versioned Late Materialization Protocol

data, we effectively decouple logging write-amplification from the total sequence length.

Immutable Data Organization. To facilitate efficient training-time retrieval, the immutable UIH storage organizes the history into subsequences, each keyed by its initial timestamp (`start_ts`). This layout allows the training-time range scan query to precisely fetch the immutable UIH data within target temporal windows and sequence length, ensuring high-concurrency read performance even when 100+ training jobs access overlapping ranges.

Training-Time Versioned Late Materialization. During the training data preprocessing, the system performs a "Time-Travel" reconstruction to replicate the inference-time state:

(1) Extracts the version metadata and the recent sequence from the mutable UIH logged in the training example.
(2) Simultaneously applies the temporal boundaries from the logged version metadata to issue a bounded range scan against the immutable UIH storage.
(3) Concatenates the two components to reconstruct complete UIH features consistent with the inference-time snapshot.

Correctness Argument. This protocol maintains O2O consistency by construction across both storage systems. The mutable UIH, comprising recent engagements immediately preceding the ranking request, is physically snapshotted and recorded at T_{request}, ensuring that no late-arrival events can contaminate this portion. For immutable UIH, the system issues a bounded range scan at training-time materialization using the temporal boundaries (`start_ts`, `end_ts`) logged at inference time; since the historical data stored in immutable UIH storage are static, the same bounded range scan yields identical results at both inference and training time. Any events loaded into immutable storage after T_{request} (including interactions between T_{request} and T_{train}) are excluded from the versioned window. In production, checksum-based validation between the Fat Row baseline and late-materialized sequences confirms reconstruction fidelity, empirically verifying the consistency guarantee. Importantly, since the reconstruction logic operates entirely within the data preprocessing layer and depends only on the logged version metadata, not on the training paradigm, the protocol can be seamlessly adopted by both streaming and batch training.

4 Productionization

The *versioned late materialization* protocol in Section 3 eliminates the storage redundancy of the Fat Row paradigm and ensures Online-to-Offline consistency for training data correctness. However, productionizing this protocol within the production training ecosystem introduces two key systems challenges. First, hundreds of concurrent training jobs for a single product, spanning production models and experimental iterations across multiple training regions, all issue read queries against a shared UIH storage, creating a degree of read amplification that demands a highly scalable storage architecture (Section 4.1). Second, shifting sequence construction to the training read-path requires each training data preprocessing worker to perform non-contiguous random reads from the immutable UIH storage, risking GPU starvation without aggressive optimization (Section 4.2).

4.1 Scalable UIH Storage

4.1.1 Hybrid Storage Design. The system bifurcates UIH into two storage systems to serve the divergent access patterns of real-time inference and high-concurrency training: a real-time mutable UIH store that captures the most recent engagements to provide second-level feature freshness, and an immutable UIH store that stores the vast majority of the UIH volume as long-term historical sequences populated via daily compaction jobs. The mutable UIH is accessed exclusively on the inference path; according to the versioned protocol, the mutable sequence is captured and logged into the training data at inference time, so training workers never query the mutable UIH. To support high-frequency updates without a Read-Modify-Write penalty, the mutable UIH store employs a fast merge operator

that performs blind-write appends, deferring state resolution to read-time or background compaction, and a write-through cache co-located with the ranking service for minimal access latency. The retention window of the mutable store is coupled to the compaction cadence of the immutable store: events must remain in the mutable tier until the next offloaded compaction cycle (Section 4.1.2) consolidates them into the immutable tier, so a daily compaction schedule requires at least one day of mutable retention. The remainder of this section focuses on the immutable UIH, which dominates both the storage volume and the read traffic from the training fleet and the online ranking service.

4.1.2 Read-Optimized Immutable Storage. The immutable UIH store serves as the normalized repository for long-term user interaction history, eliminating the systemic redundancy of the Fat Row paradigm. Its design is driven by a single objective: maximizing read throughput for range scans over versioned UIH sequences under massive concurrent training load while enabling multi-dimensional projection for heterogeneous model tenants.

Offloaded compaction and optimal data layout. The immutable UIH store has a predictable query pattern. Every query retrieves a contiguous subsequence of a user's history within a temporal range. This constrained access pattern admits a data layout that is provably optimal in I/O efficiency: one disk seek followed by purely sequential I/O per query. The immutable and read-only nature of historical UIH eliminates the need for write-optimized data layout [17], allowing a single-level storage layout that no general-purpose storage engine [8] can match. To maintain this optimal layout, a daily ETL pipeline merges and compacts newly arrived UIH partitions with existing history into complete, chronologically ordered sequences, and generates pre-sorted storage files whose key ordering and sharding match the storage engine's topology. These pre-sorted files are then bulk-loaded into the immutable UIH store as a single-level layout, eliminating the multi-level compaction that LSM-based stores [17] require to reorganize ingested data. This preserves the optimal data layout across daily refreshes, serving massive concurrent read traffic from the training fleet without compaction-induced write amplification or multi-level read amplification. Since each generation cycle rebuilds the full lookback window from source-of-truth data, it coalesces all temporal stripes for a given user into contiguous storage, ensuring that multi-stripe range scans remain purely sequential I/O rather than random reads scattered across multiple files.

Multi-dimensional projection schema. To support the heterogeneous requirements of different model tenants, the offline UIH compaction pipeline organizes each user's history by a multi-dimensional composite key with `user_id`, `feature_group`, and `subsequence_timestamp`. This schema partitions a user's monolithic sequence into fixed-length temporal subsequences—analogous to horizontal stripes of user engagements—each stored as a separate row keyed by its start timestamp. This layout enables two dimensions of server-side projection pushdown that together eliminate the multi-tenant penalty described in Section 2.3. First, *sequence length projection*: the system computes exactly how many stripes to scan for a model's target sequence length, so shorter-sequence tenants never over-fetch the full history. Second, *feature group projection*: different models select only the feature groups relevant to their architecture, so simpler tenants never read the superset of all features. At the storage engine level, the system leverages an optimized multi-range scan with parallel I/O and prefetching to fetch multiple stripes in a single batched request, amortizing per-request latency and maximizing throughput for the co-located sequential reads enabled by the optimal single-layer data layout.

Trait-aware columnar encoding. The system employs a domain-specific columnar encoding format that organizes user history as a column-oriented matrix where rows represent chronologically ordered events and columns denote typed, heterogeneous traits (e.g., post ID, timestamp, event type, video watch time). Notably, these traits exhibit different density and value distributions. For example, post ID and timestamp are dense, while engagement signals like comment and share are highly sparse. Within each horizontal stripe of the user sequence, the system exploits this heterogeneity through a trait-aware columnar encoding [13]—delta-encoding for timestamps, compact presence bitmaps for likes, or dictionary compression for categorical metadata. Besides storage and I/O efficiency benefits, the columnar layout enables selective decoding: at materialization time, it decodes only the subset of traits required by the model's feature specification, skipping irrelevant columns entirely at the byte level. This additional secondary-level projection complements the *sequence-length* and *feature-group* projections, minimizing I/O bandwidth and decoding overhead.

4.2 Training-Time Materialization

4.2.1 Disaggregated Data Preprocessing. Late materialization shifts the burden of ultra-long sequence reconstruction—I/O-intensive joins and decoding against immutable UIH sequences—from pre-materialization pipelines into the training read-path, risking GPU starvation where accelerators idle waiting for training data to be loaded. To address this, we adopt a disaggregated Data PreProcessing (DPP) architecture [27] that decouples training data loading and preprocessing from the training loop by offloading materialization to independently scalable preprocessing clusters. This disaggregated architecture enables elastic scaling: the system monitors the job-level GPU starvation percentage (accelerator idle time) and worker waste percentage (CPU idle time) in real time, and automatically provisions supplemental DPP workers for jobs with complex UIH reconstruction, ensuring that training throughput remains compute-bound by the GPUs rather than bottlenecked by the non-contiguous I/O latency and decoding of sequence materialization. Moreover, modern GPUs require large batch sizes for peak utilization, but materializing ultra-long sequences renders DPP worker threads heavily memory-bound. This disaggregated architecture achieves large training batches within each DPP worker's memory boundary by trainer-side rebatching. DPP workers process smaller base batches that fit within physical memory constraints, which are then asynchronously buffered, merged, and reshuffled by the trainer-side DPP client to satisfy the model's full batch requirements. This reduction in preprocessing granularity enables higher thread concurrency and CPU utilization per worker. In practice, tuning the DPP base batch size to balance memory pressure against thread-level parallelism yields a 15% improvement in per-worker data preprocessing throughput.

4.2.2 Masking Latency. To mask the non-contiguous I/O latency of immutable sequence lookups during training, we integrate an efficient vectorized execution engine [18] as the query engine on DPP workers. The core operator is a specialized index join where the *probe-side* is the primary training table, and the *build-side* is the immutable UIH store. Unlike conventional joins in OLAP workloads, this operator is designed to hide remote storage latency:

Pipelined I/O Prefetching. For each probe-side training batch, the operator extracts join keys and issues a bulk multi-range scan request against the immutable sequence storage. Without waiting for the response, it immediately prefetches the next probe-side batch, overlapping the immutable storage read for batch N with the primary training table read for batch $N+1$. Since these latencies are comparable, concurrent execution effectively masks the sequence lookup latency without stalling the pipeline, achieving an additional 10% improvement in per-worker data preprocessing throughput on models with heavy UIH sequence lookup.

Partial Projection Pushdown. To mitigate I/O amplification across a heterogeneous fleet of model tenants, the system exposes each model's UIH sequence length and feature group requirements to the query engine, which pushes these projections down to the immutable UIH storage, enabling the storage optimizations described in Section 4.1.2 to execute a partial range scan that retrieves only the relevant feature subsets and the specific temporal window requested by the model—avoiding the multi-tenant over-fetch penalty described in Section 2.3.

4.2.3 Data-Affinity I/O Optimization. Streaming training jobs naturally achieve a high block cache hit rate on the immutable UIH store, as concurrent trainers process the same recent temporal partitions. However, offline batch training jobs—including experimental iterations and warm-up runs—access disparate historical partitions with minimal cache locality, requiring explicit I/O optimization.

The system employs two complementary data-affinity strategies for offline batch training. First, during ingestion from real-time streams into the data warehouse for batch training, the system clusters training examples into buckets keyed by users. This groups a user's training examples from the same hourly window into a single batch, allowing DPP workers to perform the UIH reconstruction for all temporally-adjacent examples of the same user with a single immutable sequence lookup, effectively amortizing retrieval workload across session-level interactions. Second, the system enforces symmetric sharding between the primary training data and the immutable UIH storage using an identical hash partitioning scheme with a shared partition key, ensuring that all UIH sequence lookups within a data loading batch map to the same storage shard and eliminating the network fanout that would otherwise bottleneck high-concurrency read-paths. Together, these two optimizations reduce the overall read bandwidth from batch training jobs by 60% and increase the per-worker data processing throughput by 28%.

4.3 Enabling Model Scaling

Removing the data wall via late materialization is necessary but not sufficient—training and serving ultra-long sequences also demands model-side compute efficiency. ULTRA-HSTU [7] and VISTA [6] provide the complementary efficiency stack through a two-stage framework, semi-local attention, load-balanced stochastic sampling, and kernel-level optimizations that together enable higher throughput at doubled sequence length than the original baseline. On the data side, the immutable storage supports a *layered sequence layout* that partitions UIH into density-aware feature groups—e.g., dense view events with short lookback versus sparse explicit actions with long lookback—powered by the multi-dimensional projection pushdown in Section 4.1.2 to serve each group independently for maximizing signal value within the same sequence length. Data infrastructure and model efficiency are complementary investments that amplify each other when combined [7, 12].

Beyond compute efficiency, the offloaded compaction design of the immutable UIH storage (Section 4.1.2) provides important operational benefits that accelerate the model experiment lifecycle. Since each daily compaction regenerates the entire lookback window, it naturally enforces *right-to-delete* compliance: engagements associated with deleted content or accounts are idempotently scrubbed during every bulk load cycle, eliminating the need for expensive retroactive patching of historical sequences. Equally important, when researchers introduce new SideInfo features (e.g., content category, creator attributes) or deprecate unused ones, a single pipeline run regenerates the complete lookback window with the updated schema—enabling rapid offline experimentation without the multi-day backfill latency that incremental append-only systems would require. This agility in feature iteration is essential for improving model quality, as the optimal set of UIH features co-evolves with model architecture and sequence length.

5 Evaluation

5.1 System Efficiency

Table 1 summarizes the system metrics of *versioned late materialization* evaluated on three model tenants that share a union training dataset of a recommendation platform. All metrics are measured under streaming training unless explicitly marked. By storing UIH sequences once in a normalized tier rather than duplicating them into every training example, the system eliminates the K-fold write amplification inherent in denormalized Fat Rows, yielding a 46.2% reduction in primary write bandwidth of the shared training dataset. On the read side, removing the UIH payload from the primary training data reduces per-job read bandwidth by 47–70%, with Models B (mid sequence) and C (short sequence) further benefiting from projection pushdown (Section 4.2): they fetch only the required sequence lengths rather than the full long sequence, eliminating the multi-tenant read amplification penalty (Section 2.3).

Late materialization introduces sequence lookup as new I/O against the immutable UIH store. The streaming training of Model A (long sequence) has the highest sequence lookup bandwidth at 62.7% relative to the baseline primary read, narrowed from the 70.3% primary read reduction by the trait-aware columnar encoding (Section 4.1.2) that achieves a higher encoding density. Although 62.7% appears to nearly offset the 70.3% savings in raw bandwidth, the single-level, compaction-free immutable storage delivers 3.4× higher read throughput per unit of host resource than the append-only primary training data storage, so the effective host resource footprint of the 62.7% lookup bandwidth is substantially lower than the resources freed by the 70.3% primary read reduction. For batch training, the data-affinity optimization (Section 4.2.3) further

Table 1: System Efficiency of Versioned Late Materialization (Baseline = Fat Row Paradigm)

Model Tenant	UIH Seq. Length	Number of Training Jobs	Primary Write Bandwidth[1]	Primary Read Bandwidth[2]	Seq. Lookup Bandwidth[3]		Per-Batch Data Loading Latency
					Streaming	Batch	
Model A	Long	Low	−46.2%	−70.3%	+62.7%	+24.6%	+9.7%
Model B	Mid	High		−50.9%	+16.2%	+6.5%	-26.4%
Model C	Short	High		−47.7%	+8.7%	+3.4%	-36.2%

[1] Write bandwidth reduction is shared across 3 model tenants using the shared training dataset.
[2] Primary training data includes all labels and features, excluding the normalized immutable UIH sequences.
[3] UIH sequence lookup bandwidth from the immutable UIH store, reported as a percentage relative to the baseline primary read bandwidth.

amortizes sequence lookups across temporally adjacent examples of the same user, reducing lookup bandwidth by approximately 60%. Average per-batch data loading latency governs per-worker preprocessing throughput and thus the number of DPP workers each training job provisions to keep GPUs saturated. Model A sees a modest 9.7% increase from training-time sequence reconstruction, which is minimized by pipelined I/O prefetching (Section 4.2.2) and absorbed by elastic DPP scaling (Section 4.2.1) to avoid GPU starvation. Models B and C achieve 26–36% latency improvements through projection pushdown. The system-level efficiency is further amplified by the job distribution: Model A's higher per-job GPU and memory footprint results in longer iteration cycles and lower job concurrency, while Models B and C account for the majority of concurrent training jobs and realize the largest per-job improvements in both read bandwidth and data loading latency.

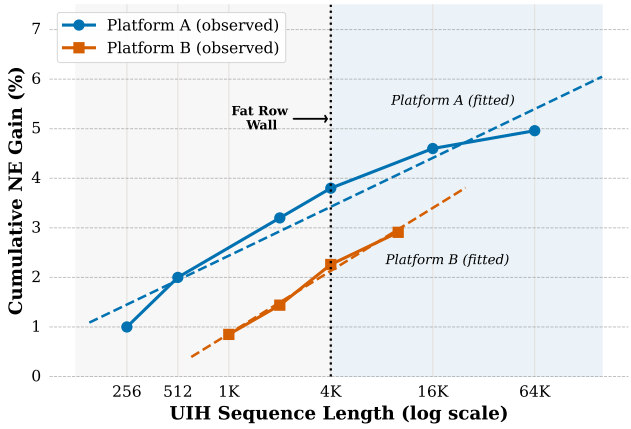

Figure 4: NE improvement of UIH sequence length scaling on two production recommendation surfaces.

5.2 Model Performance

We evaluated the impact of sequence length scaling on the late-stage ranking models of two production recommendation platforms using Normalized Entropy (NE) [10]. As shown in Figure 4, both platforms exhibit a monotonically improving NE as the UIH sequence length increases under the *versioned late materialization* infrastructure. Across the overlapping range where both paradigms are deployable (256–4K), late materialization achieves NE on par with the Fat Row baseline, confirming that training-time sequence reconstruction does not introduce model quality degradation. We define the *Fat Row Wall* as the sequence length at which the ratio of data supporting service resource usage to GPU training power exceeds 0.75. In our production environment, this wall is reached at approximately 4K (Figure 4), beyond which the K-fold amplification (Section 2) makes further scaling under the Fat Row paradigm difficult. The *versioned late materialization* architecture breaks through this wall: Platform A gains an additional 1.2% cumulative NE improvement from 4K to 64K (total >5%), and Platform B gains 0.65% from 4K to 10K. At this production scale, where a 1% NE improvement is considered a success, these gains, enabled by *versioned late materialization*, would be impractical to achieve under the Fat Row paradigm within the same infrastructure envelope.

The impact of sequence length scaling aided by *versioned late materialization* is further validated through rigorous online A/B testing on these two recommendation platforms, as summarized in Table 2. Following prior work [7], we report anonymized metrics across

Table 2: Online A/B testing results of sequence length scaling enabled by versioned late materialization on two production recommendation platforms.

Product	Seq. Length	Metric	Relative Gain
Platform A	4K → 16K	Metric-Topline	+0.22%
		Metrics-C	+4.1%
		Metrics-E-1	+2.3%
		Metrics-E-2	+4.3%
Platform B	4K → 10K	Metric-Topline	+0.14%
		Metrics-C	+0.79%
		Metrics-E-3	+1.4%
		Metrics-E-4	+1.7%

three categories: Metric-Topline (the platform's north-star engagement metric), Metrics-C (consumption-oriented metrics such as time spent), and Metrics-E (explicit engagement signals such as re-shares, comments, and likes). For each platform, the target sequence length—4K → 16K for Platform A and 4K → 10K for Platform B—is selected to balance model quality gains against the marginal GPU and data supporting service usage at each scaling step. Notably, these gains are demonstrated across platforms powered by distinct

model architectures, confirming that the sequence infrastructure's benefits are broadly applicable.

6 Related Work

As discussed in Section 1, a rich body of work has driven UIH scaling to 10^5+ events through increasingly sophisticated model architectures [5, 7, 12, 19, 23, 26, 28]. However, these efforts focus exclusively on model architecture and attention efficiency, largely treating the underlying data availability as a given. Our work addresses the complementary and often bottlenecking challenge: the data infrastructure that determines how far the sequence length can be scaled efficiently for training.

General-purpose ML data pipeline frameworks such as tf.data [16] and disaggregated preprocessing architectures [25, 27] address training data loading and preprocessing bottlenecks, while feature stores [20] provide unified offline/online feature serving to mitigate training-serving skew. These systems target broad ML workloads but do not address the domain-specific challenge of DLRM training, where UIH sequences dominate storage and I/O footprint with K-fold redundancy across multi-tenant fleets. More recently, ROO [9] introduces request-only optimization for session-level training data generation in recommendation systems, reducing redundancy in non-sequential features across impressions within a session. However, UIH sequence redundancy—the dominant system resource driver identified in Section 2—remains untouched, as ROO does not normalize or deduplicate the long-history sequences that are physically duplicated in every training example.

Late materialization—deferring reconstruction until query results are needed—is a well-established technique in columnar database query processing [1] that reduces I/O by avoiding premature assembly of wide rows. Multi-version concurrency control (MVCC) [3] is similarly foundational in transactional databases, enabling concurrent readers to observe consistent snapshots without blocking writers. While both techniques are mature in the database community, none of these systems address the domain-specific challenges of recommendation training: the interplay of streaming and batch training modes with distinct temporal semantics, the need for future-leakage prevention across both modes, and multi-tenant sequence projection over variable-length histories. Our work adapts these design principles to the recommendation systems ML training domain—storing normalized sequences once and reconstructing them just-in-time during training via lightweight version metadata, while introducing a bifurcated consistency protocol that addresses these domain-specific requirements.

7 Conclusion

We presented *versioned late materialization*, a data infrastructure paradigm that eliminates the K-fold storage and I/O redundancy of the Fat Row approach by exploiting a structural invariant of User Interaction History: as an append-only, temporally ordered sequence, its state at any point in time can be reconstructed from a single canonical copy via lightweight version metadata rather than physical duplication. Deployed across multiple production recommendation surfaces, the system not only reduces data infrastructure resource usage but, more importantly, enables UIH sequence scaling to lengths impractical under the Fat Row paradigm, unlocking significant model quality gains. These results demonstrate that data infrastructure is a first-class scaling lever for recommendation quality, complementary to advances in model architecture.

Acknowledgments

This work would not be possible without the work from the following contributors (alphabetical order): Prashasti Baid, Renqin Cai, Xianjie Chen, Huihui Cheng, Patrick Cullen, Chaitanya Datye, Delia David, Fei Ding, Qi Ding, Qin Ding, Omkar Gawde, Sathya Gunasekar, Xinyao Hu, Yanzun Huang, Parichay Kapoor, Manos Karpathiotakis, Hung-Ching Lee, Hongwei Li, Jiahao Liang, Xueyao Liang, Linbin Luo, Yun Mao, Franco Mo, Luning Pan, Pedro Eugenio Rocha Pedreira, Harsha Rastogi, Kai Ren, Aleksandr Shatrovskii, Hongzheng Shi, Yu Shi, Ruohan Sun, Mike Trepanier, Igor Vodov, Shanyue Wan, Hao Wang, Junjie Wang, Zehui Wang, Xiang Xiao, Kostas Xirogiannopoulos, Shuo Xu, Hon Yan, Tao Yang, Stanley Yao, Ximing Yu, Bill Zhang, Xin Zhang, Zhang Zhang, Zhenyuan Zhao, Charles Zheng, Josh Zhou, Yupeng Zou.

References

[1] Daniel J Abadi, Daniel S Myers, David J DeWitt, and Samuel Madden. 2007. Materialization Strategies in a Column-Oriented DBMS. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*.

[2] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 2007. A Critique of ANSI SQL Isolation Levels. arXiv:cs/0701157 [cs.DB] https://arxiv.org/abs/cs/0701157

[3] Philip Bernstein and Nathan Goodman. 1983. Multiversion Concurrency Control - Theory and Algorithms. *ACM Trans. Database Syst.* 8 (12 1983), 465–483. doi:10.1145/319996.319998

[4] Zheng Chai, Qin Ren, Xijun Xiao, Huizhi Yang, Bo Han, Sijun Zhang, Di Chen, Hui Lu, Wenlin Zhao, Lele Yu, et al. 2025. Longer: Scaling up long sequence modeling in industrial recommenders. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 247–256.

[5] Qiwei Chen, Changhua Pei, Shuguang Lv, Cheng Li, Jian Ge, and Wenwu Ou. 2021. End-to-End User Behavior Retrieval in Click-Through Rate Prediction Model. *arXiv preprint arXiv:2108.04468* (2021).

[6] Zhimin Chen, Chenyu Zhao, Ka Chun Mo, Yunjiang Jiang, Jane H. Lee, Khushhall Chandra Mahajan, Ning Jiang, Kai Ren, Jinhui Li, and Wen-Yun Yang. 2026. Massive Memorization with Hundreds of Trillions of Parameters for Sequential Transducer Generative Recommenders. arXiv:2510.22049 [cs.IR] https://arxiv.org/abs/2510.22049

[7] Qin Ding, Kevin Course, Linjian Ma, Jianhui Sun, Ruochen Liu, Zhao Zhu, Chunxing Yin, Wei Li, Dai Li, Yu Shi, Xuan Cao, Ze Yang, Han Li, Xing Liu, Bi Xue, Hongwei Li, Rui Jian, Daisy Shi He, Jing Qian, Matt Ma, Qunshu Zhang, and Rui Li. 2026. Bending the Scaling Law Curve in Large-Scale Recommendation Systems. arXiv:2602.16986 [cs.IR] https://arxiv.org/abs/2602.16986

[8] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. *ACM Trans. Storage* 17, 4, Article 26 (Oct. 2021), 32 pages. doi:10.1145/3483840

[9] Liang Guo, Wei Li, Lucy Liao, Huihui Cheng, Rui Zhang, Yu Shi, Yueming Wang, Yanzun Huang, Keke Zhai, Pengchao Wang, et al. 2025. Request-Only Optimization for Recommendation Systems. *arXiv preprint arXiv:2508.05640* (2025).

[10] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atber, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. 1–9.

[11] Manos Karpathiotakis, Vlassios Rizopoulos, Basri Kahveci, Tiziano Carotti, Artem Gelum, Hazem Nada, and Yuri Dolgov. 2025. Scribe: How Meta Transports Terabytes per Second in Real Time. *Proceedings of the VLDB Endowment* 18 (09 2025), 4817–4830. doi:10.14778/3750601.3750607

[12] Dai Li, Kevin Course, Wei Li, Hongwei Li, Jie Hua, Yiqi Chen, Zhao Zhu, Rui Jian, Xuan Cao, Bi Xue, Yu Shi, Jing Qian, Kai Ren, Matt Ma, Qunshu Zhang, and Rui Li. 2025. Realizing Scaling Laws in Recommender Systems: A Foundation-Expert Paradigm for Hyperscale Model Deployment. arXiv:2508.02929 [cs.IR] https://arxiv.org/abs/2508.02929

[13] Gang Liao, Ye Liu, Jianjun Chen, and Daniel J. Abadi. 2024. Bullion: A Column Store for Machine Learning. arXiv:2404.08901 [cs.DB] https://arxiv.org/abs/2404.08901

[14] Wenhan Lyu, Devashish Tyagi, Yihang Yang, Ziwei Li, Ajay Somani, Karthikeyan Shanmugasundaram, Nikola Andrejevic, Ferdi Adeputra, Curtis Zeng, Arun K Singh, et al. 2025. DV365: Extremely Long User History Modeling at Instagram. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 4717–4727.

[15] Yue Meng, Cheng Guo, Xiaohui Hu, Honghu Deng, Yi Cao, Tong Liu, and Bo Zheng. 2025. User Long-Term Multi-Interest Retrieval Model for Recommendation. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 1112–1116.

[16] Derek G Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk. 2021. tf.data: A Machine Learning Data Processing Framework. *Proceedings of the VLDB Endowment* 14, 12 (2021).

[17] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Inf.* 33, 4 (June 1996), 351–385. doi:10.1007/s002360050048

[18] Pedro Pedreira, Orri Erber, Masha Kandula, Kevin Haas, Yolanda Hao, Anja Gruenheid, Deepak Nair, Hao Liu, Huameng Zhu, Wenlei Fan, et al. 2022. Velox: Meta's Unified Execution Engine. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3372–3384.

[19] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. *arXiv preprint arXiv:2006.05639* (2020).

[20] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data Management Challenges in Production Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*.

[21] Qin Ren, Zheng Chai, Xijun Xiao, Yuchao Zheng, and Di Wu. 2025. LongRetriever: Towards Ultra-Long Sequence based Candidate Retrieval for Recommendation. *arXiv preprint arXiv:2508.15486* (2025).

[22] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in Machine learning systems. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511.

[23] Zihua Si, Lin Guan, ZhongXiang Sun, Xiaoxue Zang, Jing Lu, Yiqun Hui, Xingchao Cao, Zeyu Yang, Yichen Zheng, Dewei Leng, et al. 2024. Twin v2: Scaling ultra-long user behavior sequence modeling for enhanced ctr prediction at kuaishou. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4890–4897.

[24] Ashish Thusoo, Joydeep Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive - A Warehousing Solution Over a Map-Reduce Framework. *PVLDB* 2 (08 2009), 1626–1629. doi:10.14778/1687553.1687609

[25] Taegeon Um et al. 2023. FastFlow: Accelerating Deep Learning Model Training with Smart Offloading of Input Data Pipeline. *Proceedings of the VLDB Endowment* 16, 5 (2023).

[26] 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. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 58484–58509.

[27] Mark Zhao, Niket Tirmazi, Jiyan Erber, Skye Ihm, Aarti Minnich, Shashank Nair, and Dawn Sun. 2022. Understanding Data Storage and Ingestion for Large-Scale Deep Recommendation Model Training. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 1042–1057.

[28] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Changhua Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.