

An Event is Worth One Token: Event Tokenization for Industrial-scale LLM Recommendation

Fan Xia^{*,†}, Zhaoheng Zheng^{*}, Iman Setayesh^{*}, Ruogu Lin^{*}, Yiqin Pan^{*}, Samarth Mittal^{*}, Wentao Bao, Vinti Pandey, Sachin Patil, Jianpeng Cheng, Jun Xiao, Zhuang Wang, Xiangjun Fan, Sri Reddy, Minghai Chen

^{*}Core contributors, [†]Correspondence, AI at Meta

LLM-based recommendation has scaled along model capacity and sequence length, yet each position encodes only text, semantic IDs, or a few categorical features—discarding rich user, item, context, and outcome signals available at each event. Under autoregressive modeling, this yields weak queries at each position and, since each position becomes context for the next, the degradation compounds across the sequence. We propose an event-centric paradigm that represents each interaction by its full temporal snapshot, and identify a new scaling dimension we term **snapshot resolution**: the amount of information encoded per event. To efficiently scale snapshot resolution, we introduce **AMBER** (Autoregressive Modeling via Bottlenecked Event Representation), which compresses each temporal snapshot into a compact **Event Token**, a new LLM input modality. The representation is learned end-to-end, while Event Tokens are pre-computed and cached for serving, decoupling snapshot resolution from real-time serving compute. On industrial-scale ranking and retrieval benchmarks, AMBER advances the compute-quality Pareto frontier relative to alternative recommendation paradigms. At sufficient capacity, a single unified tokenizer even outperforms dedicated per-entity tokenizers, demonstrating positive transfer across structurally different entity types. AMBER’s Event Tokens also transfer across model architectures: when integrated into a heavily optimized non-LLM ranker as serving-time historical features, they yield statistically significant improvements. Further scaling Event Tokenizer capacity provides additional improvements.

Date: August 24, 2026

Correspondence: Fan Xia at xiafanaifb@meta.com



1 Introduction

Large autoregressive models are increasingly applied to sequential prediction tasks that operate over real-world *events*: recommendation model sequences of user interaction events [5, 24, 26], world models predict future states from observation-action sequences [3, 11, 19], and financial models forecast market movements from financial event sequences [23]. These domains share a common structure: each sequence position encodes a **temporal snapshot**—a structured record of the system’s observations at event time—collectively described by a heterogeneous set of signals. We refer to foundational models that predict future states from structured event sequences as **Large Event Models (LEMs)**. While LLMs operate over a fixed vocabulary, LEMs operate over temporal snapshots of real-world events.

The distinction between LEMs and LLMs lies in what we call **snapshot resolution**. At each moment, the true system state contains far more information than any system can record; snapshot resolution is how much of it a model actually captures—formally, the number of distinct signals encoded per event. For example, recommendation events contain user, item, context, and outcome signals, while robotic-control events contain visual observations, joint states, and active subgoals. Under autoregressive training,

higher snapshot resolution at each position improves both sides of the attention mechanism: the query expresses more fine-grained intent, and the key-value history retains more contextual detail for all downstream positions.

Industrial recommendation—a representative large-scale LEM instance—currently faces a strict trade-off between sequence length and snapshot resolution, driven by serving compute requirements: in large-scale recommendation systems, feature-related operations (storage, deserialization, and network transfer) can rival or even exceed GPU compute. This bottleneck forces architectures into compromises. Traditional pointwise models [6, 28] retain comprehensive heterogeneous features for the current query but represent the user history sparsely. Conversely, recent autoregressive models excel at sequence modeling but are forced to reduce detailed historical events to simple item-semantic representations [4, 5, 7, 24] or manually curated feature subsets [12, 26]. In both cases, this forced information loss weakens individual predictions, and because each position acts as context for the next, the degradation compounds progressively across the sequence.

Prior work demonstrates that LLMs can process compressed latent representations, such as visual tokens, after alignment with their embedding space [21, 22]. Therefore, we propose **AMBER**

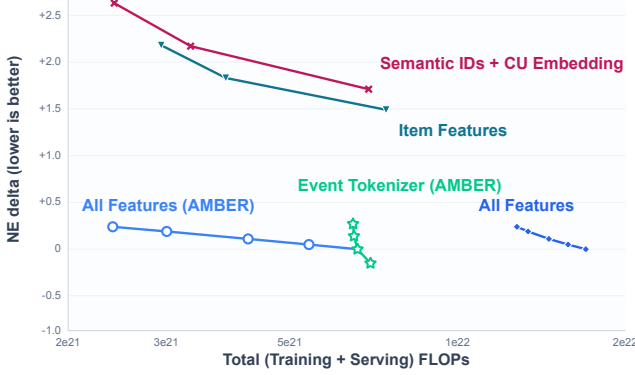


Figure 1: Scaling snapshot resolution under total training and serving FLOPs, including CPU feature materialization. Moving from Semantic IDs and content-understanding embeddings to item features and then full event features progressively advances the Pareto frontier. By asynchronously computing and caching Event Tokens, AMBER makes full-feature scaling practical and gives Event Tokenizer scaling a more favorable compute-quality trend than User LLM scaling (Sec. 5.4).

(Autoregressive Modeling via Bottlenecked Event Representation), an architecture centered around a unified Event Tokenizer. Rather than feeding raw features directly to the sequence model, the tokenizer encapsulates the vast heterogeneous signals of each temporal snapshot into a compact Event Token, which is then consumed autoregressively by the downstream sequence model. The entire pipeline is trained end-to-end, learning an efficiently compressed event token representation that scales effectively with downstream LLM capacity. At serving time, because tokenization is per-event and history-independent, it is triggered asynchronously as events occur, and the resulting Event Tokens are cached into the user’s history sequence (Figure 2). Consequently, caching each Event Token after a single asynchronous computation avoids materializing raw features during real-time inference, decoupling snapshot resolution from serving compute requirements.

We verify that **snapshot resolution** is a viable and effective scaling dimension for industrial recommendation. On top of this finding, we demonstrate that the event token paradigm has the following additional properties that make it well-suited for industrial recommendation at scale:

First, Event Tokenizer capacity can be scaled independently to improve downstream performance with minimal impact on real-time serving compute. Since tokenization is triggered once per impression and cached, scaling Event Tokenizer capacity yields a favorable compute-quality trade-off. It yields substantial downstream quality improvements while adding negligible compute requirements to the primary online serving path.

Second, downstream LLM performance scales predictably with capacity over Event Tokens. By treating these compact tokens as a standardized continuous input modality, the downstream sequential model yields monotonic performance improvements as

its parameter capacity increases. This indicates that rather than experiencing excessive compression, the Event Tokens retain sufficient fine-grained semantic information to successfully unlock the scaling laws of the downstream model.

In addition to these contributions, we share practical lessons from operating AMBER in a large-scale recommendation system, focusing on recurrent representation drift and Event Token storage. Periodic tokenizer retraining required by open vocabularies and evolving user trends shifts the representation space, making newly generated tokens incompatible with cached histories. We mitigate this instability with an adversarial recurrent alignment strategy that constrains drift across encoder updates. We also apply Matryoshka Dropout and quantization-aware training to reduce Event Token storage.

We evaluate AMBER in a large-scale production environment, covering both ranking and retrieval. Under total training and serving compute, AMBER advances the compute-quality Pareto frontier relative to alternative recommendation paradigms. By aligning Event Tokens with the LLM embedding space, AMBER enables pretrained weights to outperform random initialization, supporting Event Tokens as a new LLM input modality. Large-scale evaluation further shows that an existing non-LLM ranker can directly consume Event Tokens as additional historical features, yielding a statistically significant improvement.

In summary, AMBER introduces event-level tokenization to incorporate heterogeneous features into event sequences, enables compute-efficient scaling through asynchronous tokenization, and addresses the representation drift and storage challenges of large-scale use.

2 Related Work

Sequence Modeling in Recommendation. Industrial recommenders primarily rely on two feature sets: per-impression heterogeneous features and historical behavior features. Due to strict serving-compute constraints, the snapshot resolution of a per-impression query is typically tens to hundreds of times larger than that of a historical event. This asymmetry forces a strict modeling dichotomy. **Pointwise models** [6, 28] use the high-resolution per-impression features to attend over the low-resolution history; while this yields strong performance, it severely degrades training efficiency by redundantly re-encoding the history for every impression. Conversely, **autoregressive models** [15, 26] process history sequentially in an efficient single causal pass, but the query snapshot resolution at each position is inherently weak, which ultimately caps overall performance. AMBER bridges this gap, achieving the single-pass training efficiency of autoregressive modeling while unlocking the high per-event resolution characteristic of pointwise models.

LLM Recommendation. Most LLM-based recommenders serialize events into text [9] or Semantic IDs [4, 7, 12, 24] or both. However, bounded context windows under strict serving constraints force a direct trade-off between per-event token budget and history coverage. Consequently, both paradigms compromise: Semantic IDs efficiently encode items but discard detailed event context, while text can describe full events but suffers from low information density. Similar to multimodal LLMs [1, 21, 22], HLLM [5]

trains end-to-end to compress item text into embeddings but discards relevant event-level signals. Concurrent work, LoopFM [14], forms sequences from intermediate foundation model (FM) representations; however, this decoupled distillation blocks downstream gradients, diminishing improvements when the downstream model matches the FM’s capacity. AMBER overcomes both by training an *event-centric* tokenizer end-to-end, compressing the full heterogeneous event into high-density Event Tokens optimized directly for downstream sequential modeling.

Representation Drifting and System Efficiency. Caching learned representations introduces two systemic challenges: representation drift during recurrent training and substantial storage requirements at scale. To address temporal drift, we draw on Domain Adaptation [8, 10, 25], a field that aligns feature spaces across varying distributions. While traditionally applied to cross-domain tasks, we adapt these adversarial principles [8] to stabilize representations across successive temporal checkpoints. To resolve the storage bottleneck, we build upon recent advances in Representation Compression, specifically Matryoshka Representation Learning [17, 18] and quantization-aware training [2], to substantially reduce the memory footprint of cached tokens.

3 Methodology

3.1 AMBER: Model Architecture

Current industrial recommendation systems exhibit a substantial computational asymmetry between training and serving. Because offline training operates without strict latency constraints and processes only actual logged behaviors alongside synthetic negatives, its computational cost is merely a fraction of online serving, which score large candidate pools in real time. Our architecture exploits this structural asymmetry by deliberately using additional offline training compute to reduce online serving compute requirements.

Following this philosophy, we propose **AMBER**, an event-centric paradigm that uses additional offline training to encapsulate each interaction’s heterogeneous features into compact tokens, so that at serving time the downstream model consumes cached Event Tokens instead of materializing detailed per-event signals, achieving comparable quality at far lower serving compute. It has two components:

- **Event Tokenizer** g_θ : compresses the raw per-event signals into d_z -dimensional **Event Tokens** z_i .
- **User LLM** f_ψ : consumes the Event Tokens and makes predictions autoregressively.

In our experiments, f_ψ is initialized from a pre-trained 1B-parameter Llama model. For RoPE, we use event timestamps relative to a fixed reference date rather than sequence indices. During training, the two models (g_θ , f_ψ) are jointly optimized. During serving, they are decoupled: the Event Tokenizer runs asynchronously, and the downstream model makes predictions from the cached Event Token sequence (Figure 2). As the User LLM is a decoder-only Transformer, the rest of this subsection focuses on the Event Tokenizer.

Tokenizer architecture. The Event Tokenizer maps raw signals to an Event Token in three steps (Figure 7(a); detailed in Appendix B):

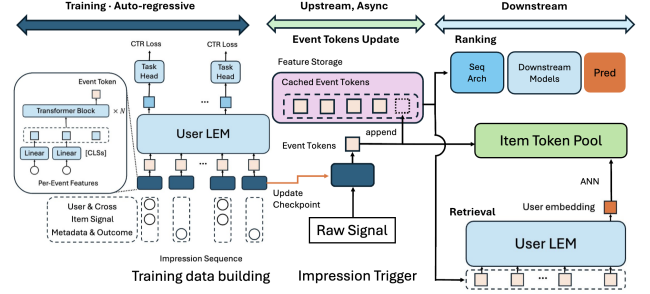


Figure 2: Overview of AMBER. Training (left): the Event Tokenizer is trained end-to-end with the User LLM under an autoregressive objective, learning to compress heterogeneous features into latent embeddings (Event Tokens). Asynchronous upstream (middle): when an event occurs, the Event Tokenizer compresses its full feature set into a compact Event Token and appends it to the user’s existing Event Token sequence; item tokens are also used to update the existing item-token pool. Downstream (right): ranking and retrieval models consume cached Event Token sequences as sequential features. By compressing full event features into Event Tokens asynchronously, AMBER increases the snapshot resolution of user histories without materializing the full feature set at serving time, thereby avoiding the associated serving compute (Sec. 3.1).

- (1) **Feature encoding.** Sparse categorical features (via embedding lookup), embedding features (via linear projection), and dense numerical features (normalized, concatenated, and projected) are each mapped to independent d_{model} -dimensional tokens $\mathbf{h}_1, \dots, \mathbf{h}_m$.
- (2) **Interaction & summarizing.** We append c learnable [CLS] tokens $\mathbf{h}_{\text{CLS}}^{(1)}, \dots, \mathbf{h}_{\text{CLS}}^{(c)}$ to the feature tokens and process the sequence through an L -layer bidirectional Transformer encoder, so features interact and are summarized into the c [CLS] outputs.
- (3) **LLM-space projection.** An MLP maps the contextualized [CLS] outputs to the final d_z -dimensional Event Token.

Concretely, the Event Token is

$$z_i = \text{MLP}\left(\text{BiTransformer}(\mathbf{h}_1, \dots, \mathbf{h}_m, \mathbf{h}_{\text{CLS}}^{(1:c)})[\text{CLS}]\right). \quad (1)$$

We also compare against other architectures in Section 5.

Unified Event Tokenizer. Instead of task-specific encoders, a single shared Event Tokenizer g_θ generates tokens via dynamic feature masking:

$$z_i = g_\theta(\mathbf{f}_i, \mathbf{m}), \quad z_i \in \mathbb{R}^{d_z} \quad (2)$$

where $\mathbf{f}_i$ denotes the heterogeneous features of event e_i , and $\mathbf{m}$ is a role based binary mask dictating feature visibility (Table 1). Beyond simplifying serving, this shared design outperforms dedicated per-entity encoders via positive transfer across entity (§5.3.2).

Table 1: Per-role feature masking. A single shared Event Tokenizer serves all roles; only the input mask differs. ✓: kept, ✗: masked (metadata = position, weight) (Sec. 3.2).

Token role	User	Item	Ctx & Cross	Metadata & Outcome
Ranking context (A)	✓	✓	✓	✗
Ranking label (B)	✗	✗	✗	✓
Retrieval history	✓	✓	✓	✓
Retrieval target	✗	✓	✗	✗

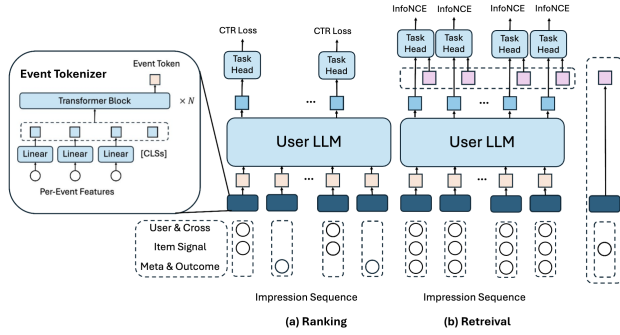


Figure 3: AMBER for two tasks. (a) Ranking uses an interleaved sequence of context and label tokens with a BCE objective. (b) Retrieval uses fused event tokens with an InfoNCE objective; the candidate item (pink) is tokenized separately. All Event Tokenizer instances share weights and differ only in their feature masks for different token types (Sec. 3.2).

3.2 Training and Sequence Design

Three-stage training. Directly training a randomly initialized Event Tokenizer with a pre-trained LLM would corrupt the LLM’s world knowledge. We therefore use two-stage alignment inspired by vision-language model training [21, 22], followed by recurrent training:

Stage 1 (Pre-alignment). We freeze the User LLM and train only the Event Tokenizer to align event features with the LLM’s pre-trained embedding space.

Stage 2 (Joint training). We unfreeze the User LLM and train both components end-to-end to co-adapt their representations.

Stage 3 (Recurrent training). We periodically retrain to track distribution shifts. Section 4 addresses the resulting drift of cached Event Tokens across updates.

Sequence Design. Context length is a bottleneck in sequential recommendation: under a fixed context window, every extra token per event evicts a historical event. Our experiments (Section 5.3.2) show that prioritizing historical coverage over tokens per event yields better performance. We seek the minimum token budget each task requires, leading to one token per event for retrieval and two tokens per event for ranking (Figure 3, Table 1).

Ranking objective (autoregressive BCE). We interleave, for each event, a context token $z_i^{(A)}$ and a label token $z_i^{(B)}$ under complementary masks (Table 1), forming the sequence

$$(z_1^{(A)}, z_1^{(B)}, z_2^{(A)}, z_2^{(B)}, \dots, z_n^{(A)}, z_n^{(B)}). \quad (3)$$

Each context position $z_i^{(A)}$ predicts event i ’s label via an MLP head. Since the label token $z_i^{(B)}$ follows $z_i^{(A)}$ in the sequence, it is never attended to, keeping the prediction leakage-free. Training runs over the whole sequence in a single forward pass:

$$\mathcal{L}_{\text{rank}} = -\frac{1}{n} \sum_{i=1}^n [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)], \quad (4)$$

where $y_i \in \{0, 1\}$ is the behavior label. At inference, the candidate is appended as a context token with no label.

Retrieval objective (InfoNCE). We fuse each event into a single unmasked token, forming the sequence

$$(z_1, z_2, \dots, z_n), \quad (5)$$

which maximizes the history the LLM attends over. At each position the LLM emits a user embedding u_i ; targets are pre-encoded by the same tokenizer from item-only features and cached independently of the user. We train with a noise-contrastive estimation loss:

$$\mathcal{L}_{\text{ret}} = -\frac{1}{n} \sum_{i=1}^n \log \frac{\exp(u_i^T v_i^+ / \tau)}{\exp(u_i^T v_i^+ / \tau) + \sum_{j \in \mathcal{N}_i} \exp(u_i^T v_j^- / \tau)}, \quad (6)$$

where v_i^+ is a positively engaged item in the user’s near future, τ is a temperature, and $\mathcal{N}_i$ combines *hard negatives* (items shown but not engaged) and *easy negatives* (randomly sampled items).

4 Serving at Scale

4.1 Asynchronous Upstream Architecture

AMBER operates as a two-stage system that decouples feature computation from real-time serving:

Offline tokenization and caching. The Event Tokenizer runs asynchronously: triggered when an impression event occurs, it queries the event’s full heterogeneous feature set through the existing training-data pipeline and compresses it into an Event Token. Each token is appended to the user’s Event Token sequence in the feature store.

Downstream modes. Cached Event Tokens support two downstream modes:

- **Ranking.** Serving end-to-end LLMs for real-time ranking introduces substantial infrastructure bottlenecks, particularly regarding KV cache maintenance for long user histories. While resolving these serving challenges is beyond the scope of this work, we demonstrate the universal transferability of our representation: cached Event Tokens are served as plug-and-play sequential features to the existing non-LLM ranking model (validated in §5.6).
- **Retrieval.** The User LLM directly consumes the cached token sequence to produce real-time user embeddings for approximate nearest-neighbor (ANN) search.

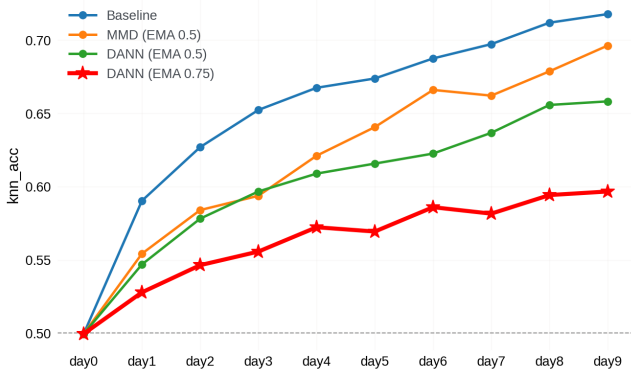


Figure 4: Mitigating representation drift over time, measured by checkpoint separability (k -NN accuracy; higher = more drift, 0.5 = indistinguishable). The unconstrained baseline (no regularization) rises monotonically, indicating severe progressive drift in the learned embedding space. Our stabilization configurations (EMA(0.5)+DANN, EMA(0.5)+MMD, and EMA(0.75)+DANN) suppress this separation, keeping the representation consistent across continuous retraining cycles (Sec. 4.2).

4.2 Mitigating Representation Drift

During recurrent training, updating the encoder from $g_{\theta_{old}}$ to $g_{\theta_{new}}$ alters its output space, causing *representation drift* that accumulates across retraining (Figure 4) and renders tokenized events incompatible with previously cached tokens (t-SNE visualization in Appendix F), destabilizing downstream LLM predictions (Appendix G).

To mitigate representation drift, we combine **adversarial regularization** based on domain-adversarial neural network (DANN) training [8] with **exponential moving average (EMA) updates**.

Adversarial regularization. During recurrent training, a lightweight discriminator D_ϕ is trained to distinguish whether a token was produced by the old or new encoder:

$$\mathcal{L}_{adv} = -\frac{1}{n} \sum_{i=1}^n [s_i \log D_\phi(z_i) + (1 - s_i) \log(1 - D_\phi(z_i))], \quad (7)$$

where $s_i \in \{0, 1\}$ indicates the encoder version. A gradient reversal layer pushes $g_{\theta_{new}}$ to produce representations indistinguishable from $g_{\theta_{old}}$, minimizing the joint objective $\mathcal{L}_{task} + \lambda \mathcal{L}_{adv}$. We use a small λ ($\sim 5 \times 10^{-3}$), as larger values can produce degenerate representations that fool the discriminator without reducing drift.

EMA weight update. We maintain an EMA encoder across daily cycles as the stable reference to which DANN aligns each new encoder, rather than anchoring only to the previous day.

We evaluate DANN against alternative drift-mitigation objectives in Section 5.5.

Sharded recurrent training. Retraining on the full user base every day is wasteful: only a small fraction of each user’s sequence is new, so most data is re-seen. We instead randomly partition users into N shards and retrain on one shard per day, cycling through all users every N days to maximizing the fresh signal per training.

When model converged, we find this suffices to catch up to the latest trends.

4.3 Embedding Compression

Storing hundreds of pre-computed Event Tokens per user requires compression to improve storage efficiency. We apply two complementary techniques.

Matryoshka Dropout. We introduce Matryoshka Dropout to support flexible token dimensions without repeatedly retraining the model. Standard Matryoshka Learning [17] requires multiple downstream forward passes per step, a computationally prohibitive operation for LLMs. We instead achieve continuous ordering in a single pass via structured suffix dropout. With probability ρ , we truncate z to its leading $d_r \sim \text{Uniform}(d_{min}, d_z)$ dimensions and apply d_z/d_r rescaling:

$$\tilde{z}^{(j)} = \begin{cases} \frac{d_z}{d_r} z^{(j)} & \text{if } j \leq d_r, \\ 0 & \text{if } j > d_r, \end{cases} \quad (8)$$

so the model learns to retain the most important information in each prefix.

Quantization-aware training. We expose the encoder to simulated INT8 noise via STE [2] during training and serve at INT4, achieving an 8× storage reduction (vs. FP32) while preserving ~80% of the full-precision embedding’s impact.

5 Experiments

We evaluate AMBER via a **dual-track** validation:

Model Effectiveness and Scaling (§5.2–§5.4). We benchmark AMBER against a highly optimized ranking model (hereafter, *Incumbent*) and other paradigms (§5.2), conduct ablation studies of input features and tokenizer designs (§5.3–§5.3.2), and analyze scaling behavior across tokenizer capacity, feature count, and downstream model size (§5.4).

Cross-Architecture Transfer (§5.6). We demonstrate the model-agnostic nature of Event Tokens by serving the decoupled tokenizer as plug-and-play features for an existing non-LLM ranker.

5.1 Setup

5.1.1 Dataset. Models are trained on petabyte-scale impression logs from a large-scale industrial recommendation platform using chronological splits: training and feature selection use past data, calibration uses only recent data preceding evaluation, and all metrics are computed on unseen future dates. Each event carries the few hundred most important features across user, item, context, and outcome signals; ablations (§5.3–§5.3.2) use a 10% subsample, while main results (§5.2) use the full dataset.

AMBER targets real-world system constraints that arise from feature materialization, recurrent training, and non-stationarity. Evaluating these constraints requires hundreds of heterogeneous features per event and billion-scale temporal data. Petabyte-scale industrial data provides both properties, which common public recommendation benchmarks do not jointly expose.

5.1.2 Metrics. For ranking, our primary evaluation metric is **Normalized Entropy (NE)** [13], a standard offline metric widely used as a proxy for online ranking quality in industrial recommendation.

NE is defined as the average log loss normalized by the entropy of the empirical positive rate p :

$$\text{NE} = \frac{-\frac{1}{N} \sum_{i=1}^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]}{- [p \log p + (1 - p) \log(1 - p)]}, \quad (9)$$

where $y_i \in \{0, 1\}$ is the ground-truth label, and p is the empirical interaction rate observed in the evaluation dataset. All NE improvements are reported as relative deltas (NE Δ). We use the following metrics:

NE. Because NE is computed from log loss, it reflects both ranking quality and probability calibration: over- or under-confident predictions increase NE even when the ranking remains unchanged. We therefore calibrate each model on leakage-free data from the recent past so that the aggregate predicted interaction rate matches the observed interaction rate (i.e., a calibration ratio close to 1), and compute NE on unseen future data.

Ensemble NE. Direct cross-paradigm comparison is insufficient to assess complementary predictive benefit. Therefore, Ensemble NE measures the complementary predictive benefit of AMBER relative to the highly optimized Incumbent. On unseen future data, we compute NE after linearly blending their predictions, with the mixing weight tuned on a preceding held-out slice. To account for feature-materialization latency, we mask preceding events within either a **10-minute** or **1-minute** delay window. Lower Ensemble NE indicates that AMBER captures additional signal over the Incumbent. We leverage this as an offline measure of complementary predictive benefit.

Soft Recall. For retrieval, we report the value-weighted fraction of top-item value retained after the model’s candidates are merged with those retrieved by the Incumbent and re-scored by the downstream ranker, capturing end-to-end utility rather than isolated retrieval precision.

NE differences within 0.02% are averaged over two runs; the observed run-to-run variation is at most 0.02%.

5.1.3 Baselines.

Incumbent models. *Incumbent* denotes the highly optimized ranking model, trained on trillions of examples, against which we report Ensemble NE. *Incumbent (fair)* matches AMBER on features, FLOP budget, and training target, serving as the controlled baseline for NE.

HSTU-style input. This baseline follows HSTU’s input schema (item categorical features per event and user features once at the sequence front) while retaining AMBER’s LLM to isolate the input schema; prior work finds Transformer and HSTU backbones comparable at scale [16] (Appendix C)¹.

Semantic IDs + CU embeddings. Our ablation study (§5.3.2) shows that history coverage contributes more than additional tokens per event under a fixed context length. We therefore fuse multiple Semantic IDs and content-understanding (CU) embeddings into a single token. CU embeddings are included as a lossless upper bound for Semantic ID decompression. To control for tokenizer architecture and capacity, this baseline uses the same Event Tokenizer architecture and scale as AMBER.

¹The downstream sequence-model architecture is not a focus of AMBER. The Event Tokenizer can also be paired with HSTU as the downstream sequence model. For a controlled comparison, we follow the existing HSTU input schema while using the same 1B-parameter LLM as AMBER.

Table 2: Ranking performance. NE Δ is reported relative to Incumbent (fair). Ensemble NE Δ measures the complementary predictive signal provided when linearly blending predictions with the Incumbent, evaluated under simulated 10-minute and 1-minute feature delays (Sec. 5.2).

Method	NE Δ ↓	Ensemble NE Δ ↓ (vs. Incumbent)	
	vs. Incumbent (fair)	10-min delay	1-min delay
SID + CU emb	+1.20%	−0.03%	−0.05%
HSTU-style	+0.60%	−0.03%	−0.06%
Pointwise [†]	−0.10%	—	—
AMBER	−0.40%	−0.10%	−0.16%

[†] See Appendix D for the Pointwise setup.

Pointwise[†]. To separate autoregressive training efficiency from history resolution, this variant reproduces the pointwise input pattern within AMBER’s backbone: the current query uses the full event, while history retains only item features. We include the complete item-feature set as a strong upper bound, although materializing this many historical features online would be computationally costly (Appendix D).

5.2 Main Results

5.2.1 Ranking. As established in §5.1, we evaluate both controlled model quality and complementary predictive signal. Table 2 shows that AMBER improves on both fronts.

Performance against Controlled Baselines. AMBER achieves the lowest NE, outperforming Incumbent (fair), Pointwise, HSTU-style input, and Semantic IDs + CU embeddings by 0.40%, 0.30%, 1.00%, and 1.60%, respectively. The 1.60% improvement over Semantic IDs + CU embeddings highlights the information bottleneck of item-centric representations. AMBER’s 0.30% edge over Pointwise confirms that maintaining high snapshot resolution throughout the history—rather than solely at the current query—is beneficial. Meanwhile, Pointwise’s 0.10% improvement over Incumbent (fair) isolates the efficiency benefit of autoregressive modeling under matched compute. Finally, while HSTU-style input reduces NE by 0.60% over Semantic IDs + CU embeddings through static user-side and item categorical features, its similar Ensemble NE suggests that these signals are largely redundant with the Incumbent.

Complementary Signal to the Incumbent. Blending AMBER with the Incumbent reduces Ensemble NE by 0.10% under a 10-minute feature delay and 0.16% under a 1-minute window. In contrast, Semantic IDs + CU embeddings yield only 0.03% and 0.05% reductions—less than one-third of AMBER’s improvements—and degrade further without CU embeddings (§5.3.1). Despite this gap, the item-semantic baseline still provides non-zero ensemble improvements, which may partly result from the LLM capacity.

5.2.2 Retrieval. We evaluate retrieval via Soft Recall (§5.1). AMBER maintains a 0.31–0.51% improvement across all checkpoint ages, outperforming CU embeddings by 0.28–0.32 percentage points (Table 3). While the latter becomes neutral by Day 8, AMBER retains a 0.31% improvement, indicating that high-resolution event

Table 3: Retrieval Soft Recall Δ over days since the last re-training, relative to the Incumbent (Sec. 5.2).

Soft Recall $\uparrow$	Day 1	Day 2	Day 4	Day 8
CU emb.	+0.23%	+0.09%	+0.20%	−0.01%
AMBER	+0.51%	+0.35%	+0.50%	+0.31%

Table 4: Feature group ablation (pre-alignment). SID denotes Semantic IDs; Ensemble NE is measured against the Incumbent under a 10-minute delay. Leave-one-out results are grouped by feature type, and cumulative addition starts from SID + outcome signals (Sec. 5.3.1).

Configuration	NE $\Delta \downarrow$	Ensemble NE $\Delta \downarrow$
	vs. Full AMBER	10-min delay
<i>Item representation</i>		
– Item features	+6.65%	–
– Item categorical signals	+1.00%	–
– Item emb	+0.06%	–
– SID	neutral	–
– (SID + item emb.)	+0.15%	–
– (SID + Content Hashes)	+0.07%	–
<i>Event context</i>		
– User features	+0.30%	–
– Outcome signals	+0.24%	–
– Metadata (weight, position)	+0.08%	–
– User-item cross features	+0.03%	–
<i>Cumulative addition</i>		
SID + outcome signals	+2.40%	−0.024%
+ CU emb	+1.62%	−0.034%
+ sparse features	+1.52%	−0.036%
+ item emb	+1.50%	−0.036%
+ user signals	+0.40%	−0.060%

information improves retrieval and remains robust to checkpoint staleness.

5.3 Ablation Study

We further ablate feature groups and tokenizer architectures to derive AMBER’s design choices, followed by a scaling analysis that accounts for both training and serving compute (§5.4).

5.3.1 Feature Group Ablation. We conduct two complementary analyses: (1) removing one feature group at a time from the full AMBER, and (2) starting from a minimal Semantic ID representation and progressively adding features. All experiments use the same encoder and pre-alignment setup.

Item information is important, but Semantic IDs are insufficient. Removing all item features degrades NE by 6.65%, confirming that item representation remains the primary signal in sequential recommendation. However, retaining only Semantic IDs and outcome signals incurs a 2.40% gap to full AMBER. Adding CU embeddings and non-CU item categorical features narrows this gap to 1.52%, showing that substantial item information is not captured by Semantic IDs alone.

Learned item representations contain largely overlapping signals. Adding a behavior-based item embedding on top of CU embeddings and item categorical features improves NE by only 0.02% (1.52% $\rightarrow$ 1.50%). Likewise, removing Semantic IDs from full AMBER is neutral, but removing them together with item embeddings incurs a 0.15% loss, compared with 0.06% from removing item embeddings alone. This suggests that Semantic IDs provide little information beyond that already encoded by learned item embeddings.

User features capture signals beyond the behavior sequence. Adding user features improves NE by 1.10%, showing that user-side state contains information absent from item interactions. Ensemble NE echoes this result: its largest improvements come from user signals and CU embeddings, both computationally costly to materialize online, while behavior-based item embeddings provide little additional information when reused in the event history. The 0.07% loss from jointly removing Semantic IDs and content hashes further shows that identifier types provide complementary information.

5.3.2 Tokenizer & Architecture Design. We validate AMBER’s architecture through four data-driven ablations.

Unified tokenization enables positive transfer. A shared tokenizer improves NE by 0.02% under both the four-token (user, item, other, label) and two-token (context, label) schemas, showing positive transfer across entity types.

History coverage determines the token budget. Two tokens per event help with unlimited context (0.10%) but hurt under a limited window (0.16%); AMBER therefore prioritizes history coverage.

The Transformer provides the best practical trade-off. A four-layer Transformer matches DHEN ($\times 12$) at similar FLOPs while offering greater flexibility and FlashAttention efficiency; PMA and Concat+FFN (Appendix B) perform worse.

Pre-trained knowledge transfers to event modeling. Even when frozen, the LLM improves over the Event Tokenizer alone by 0.83%, demonstrating the potential of using Event Tokens directly as LLM inputs. Unfreezing the LLM adds another 1.04%. Although the advantage of pre-trained initialization narrows with more data, it stabilizes at 0.20% over random initialization (Appendix H).

5.4 Efficiency & Scaling Analysis

Scaling protocol. We scale User LLM depth over $\{1, 2, 4, 8, 16\}$, Event Tokenizer depth over $\{1, 2, 4\}$, width over $\{64, 128, 256, 512\}$, and snapshot resolution over $\{100, 200, 400\}$ features. Features are ranked by permutation importance, measured by the NE increase after shuffling each feature across examples. We preserve the feature-type distribution at every resolution and always retain label and metadata features.

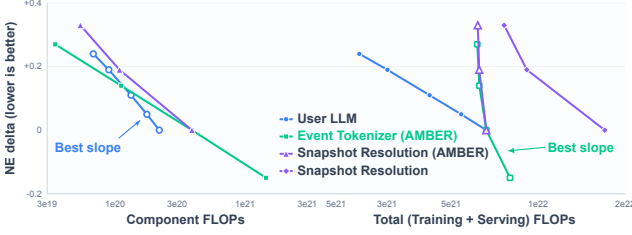
Computational-cost model. We define total computational cost as

$$C_{\text{total}} = m_u C_{\text{user}} + m_e C_{\text{event}} + C_{\text{mat}}, \quad (10)$$

where ranking workloads yield a 30–100 $\times$ serving-to-training ratio; we use the lower bound. Cached tokenization gives $m_e \ll m_u$, and C_{mat} captures sequence materialization. NE is measured, whereas serving compute is estimated (Appendix A).

Table 5: Ablations of AMBER’s design. (a) Contribution of each training stage. (b) Transfer from sharing one tokenizer across entity types. (c) Trade-off between per-event token capacity and history coverage. (d) Comparison of event encoders (Sec. 5.3.2).

(a) Training		(b) Unified vs. Per-Entity		(c) Number of Tokens per Event		(d) Encoder	
Setting	NE Δ ↓	Schema	NE Δ ↓	Context	NE Δ ↓ (2 vs. 1 token/event)	Encoder	NE Δ ↓
Tokenizer only vs. frozen LLM	+0.83%	User / item / other / label		Unlimited context	−0.10%	Bi-Transformer (×4, baseline)	—
Unfrozen vs. frozen LLM	−1.04%	(4 tokens/event)	−0.02%	Limited context	+0.16%	Bi-Transformer (×2)	+0.09%
Pre-trained vs. random init	−0.20%	Context / label				Bi-Transformer (×1)	+0.19%
Pre-aligned vs. direct joint	−0.10%	(2 tokens/event)	−0.02%			DHEN (×12) [27]	0.00%
						PMA (×4) [20]	+0.20%
						Concat+FFN	+0.22%

**Figure 5: Scaling under training-only and total-compute accounting. User LLM scaling is favored when only training FLOPs are counted (left). Once serving FLOPs are included (right), Event Tokenizer and AMBER-enabled snapshot-resolution scaling become more compute-efficient because tokenization is asynchronous and its outputs are cached (Sec. 5.4).**

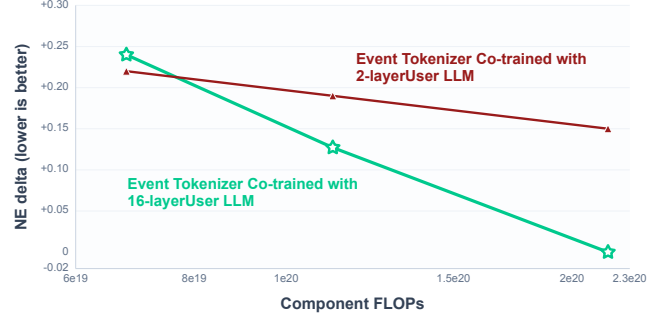
Snapshot resolution improves the compute-quality frontier. Moving from Semantic IDs and CU embeddings to item features and then full features consistently improves NE (Figure 1). This result indicate model scaling cannot recover information absent from the input representation. AMBER further improves the frontier by moving full-feature materialization off the online serving path.

Training-optimal scaling is not system-optimal. When training FLOPs are considered, increasing User LLM capacity provides the strongest scaling trend (Figure 5, left). After serving compute is included, event-side scaling becomes more efficient because the User LLM runs for every request, whereas Event Tokens are computed once and cached (Figure 5, right). Scaling decisions based only on training compute can therefore be suboptimal under total compute.

Event Token quality governs downstream scaling. We freeze Event Tokens produced by tokenizers co-trained with either a 2-layer or 16-layer User LLM, and scale a new downstream User LLM on future data. Tokens co-trained with the 16-layer model improve more rapidly with downstream compute (Figure 6), showing that stronger co-training produces representations that larger downstream models can exploit more effectively.

5.5 Drift Mitigation Evaluation

We compare DANN with MMD, CORAL, and L_2 regularization over a 20-day checkpoint gap, a stricter and less short-term-sensitive

**Figure 6: Scaling downstream User LLMs with frozen Event Tokens. We tokenize future data using Event Tokenizers co-trained with either a 2-layer or 16-layer User LLM, freeze the resulting tokens, and scale a downstream User LLM. Tokens co-trained with the 16-layer model exhibit a stronger scaling trend (Sec. 5.4).****Table 6: Drift mitigation over a 20-day checkpoint gap. Higher $\cos_mean$ and lower knn_acc indicate better alignment; NE Δ is relative to no regularization (Sec. 5.5).**

Method	$\cos_mean$ ↑	knn_acc ↓	NE Δ (%) vs. no reg. ↓
DANN	0.956	0.856	−0.02
MMD	0.955	0.870	+0.03
CORAL	0.932	0.899	+0.36
L_2	0.900	0.939	+1.64

drift test. Cosine similarity compares the same events across encoders; k-NN accuracy independently measures how well a nonlinear classifier separates their representations, reducing dependence on the optimized alignment objective.

DANN provides the best alignment without sacrificing predictive quality, and its slight NE improvement may reflect reduced overfitting to the current distribution.

5.6 Large-Scale Evaluation

We integrate an Event Tokenizer (3.84M dense parameters; ~200 GB of sparse embedding tables) on a full-traffic Facebook surface, supporting online encoding for billions of events per day. We update it through AMBER’s recurrent-training framework and tokenize each logged event once to maintain Event Token sequences. The Incumbent consumes these sequences as additional historical features.

On our internal datasets, an NE reduction of 0.02% is considered statistically significant and sufficient to yield measurable revenue impact. AMBER's Event Tokens reduce NE by 0.06%. This improvement demonstrates cross-architecture transfer and indicates that increasing history snapshot resolution provides predictive signal absent from the Incumbent.

6 Conclusion

We show that increasing historical event snapshot resolution improves the compute-quality Pareto frontier, but increases high feature-materialization compute during serving. To overcome this, AMBER asynchronously compresses heterogeneous event features into compact Event Tokens. By caching these representations, AMBER enables high-resolution event histories without online feature materialization.

Specifically, AMBER provides three primary advantages: (1) end-to-end joint training with the LLM enables efficient downstream scaling; (2) scaling both the LLM and Event Tokenizer reveals that optimizing total training and serving FLOPs, rather than component-level training FLOPs, is important for system-level scaling decisions; and (3) domain-adversarial training combined with EMA updates mitigates representation drift during recurrent serving. Separately, large-scale evaluation yields a 0.06% NE improvement.

Future Work. We show that downstream performance scales with Event Tokenizer capacity; future work will explore Mixture-of-Experts (MoE) for more efficient scaling. On the serving side, asynchronous tokenization introduces some feature delay (Appendix E). A promising direction is to maintain a high-resolution sequence with very short retention (e.g., 10 minutes), enabling on-demand tokenization of events not yet cached. Finally, because Event Tokens are aligned with the LLM embedding space, they may be jointly modeled with text, effectively treating the user as a new LLM modality. We plan to explore this direction toward personalized LLMs that combine language understanding with recommendation signals.

References

- [1] Jean-Baptiste Alayrac et al. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*. Curran Associates, Inc., 23716–23736. arXiv:2204.14198
- [2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR* abs/1308.3432 (2013). arXiv preprint. arXiv:1308.3432
- [3] Jake Bruce et al. 2024. Genie: Generative Interactive Environments. *CoRR* abs/2402.15391 (2024). arXiv:2402.15391
- [4] Junyi Chen et al. 2025. LCRec: Towards Generative Recommendation with Language-Based Collaborative Representations. *CoRR* abs/2502.14927 (2025). arXiv:2502.14927
- [5] Junyi Chen, Lu Chi, Bingyue Peng, and Zehuan Yuan. 2024. HLLM: Enhancing Sequential Recommendations via Hierarchical Large Language Models for Item and User Modeling. *CoRR* abs/2409.12740 (2024). arXiv:2409.12740
- [6] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data (DLP-KDD)*. ACM. arXiv:1905.06874
- [7] Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. 2025. OneRec: Unifying Retrieve and Rank with Generative Recommender and Iterative Preference Alignment. *CoRR* abs/2502.18965 (2025). arXiv:2502.18965
- [8] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research* 17, 59 (2016), 1–35.
- [9] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*. ACM, New York, NY, USA, 299–315.
- [10] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A Kernel Two-Sample Test. *Journal of Machine Learning Research* 13 (2012), 723–773.
- [11] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering Diverse Domains through World Models. *CoRR* abs/2301.04104 (2023). arXiv:2301.04104
- [12] Ruining He, Lukasz Heldt, Lichan Hong, Raghunandan H. Keshavan, Shifan Mao, Nikhil Mehta, Zhengyang Su, Alicia Tsai, Yueqi Wang, Shao-Chuan Wang, Xinyang Yi, et al. 2025. PLUM: Adapting Pre-trained Language Models for Industrial-scale Generative Recommendations. *CoRR* abs/2510.07784 (2025). arXiv:2510.07784
- [13] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising (ADKDD)*. ACM, 5:1–5:9. doi:10.1145/2648584.2648589
- [14] Shali Jiang, Hua Zheng, Boyang Liu, Laming Chen, Huayu Li, et al. 2026. LoopFM: Learning from Historical Representations of Foundation Model for Recommendation. *CoRR* abs/2605.29280 (2026). arXiv:2605.29280
- [15] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *IEEE International Conference on Data Mining (ICDM)*. 197–206.
- [16] Kirill Khrylchenko, Artem Matveev, Sergei Makeev, and Vladimir Baikalov. 2026. Scaling Recommender Transformers to One Billion Parameters. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. arXiv:2507.15994
- [17] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Sapra, Prateek Jain, Ali Farhadi, and Sham Kakade. 2022. Matryoshka Representation Learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*. Curran Associates, Inc., 30233–30249. arXiv:2205.13147
- [18] Riwei Lai et al. 2024. Matryoshka Representation Learning for Recommendation. *CoRR* abs/2406.07432 (2024). arXiv:2406.07432
- [19] Yann LeCun. 2022. A Path Towards Autonomous Machine Intelligence. *OpenReview* (2022). <https://openreview.net/pdf?id=BZ5a1r-kVsf>
- [20] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set Transformer: A Framework for Attention-based Permutation-Invariant Input. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 3744–3753. arXiv:1810.00825
- [21] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*. PMLR, 19730–19742. arXiv:2301.12597
- [22] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*. Curran Associates, Inc., 34892–34916. arXiv:2304.08485
- [23] Dat Mai. 2024. StockGPT: A GenAI Model for Stock Prediction and Trading. *CoRR* abs/2404.05101 (2024). arXiv:2404.05101
- [24] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Mahesh Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*. Curran Associates, Inc., 10935–10959.
- [25] Baochen Sun, Jiashi Feng, and Kate Saenko. 2016. Return of Frustratingly Easy Domain Adaptation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*. 2058–2065.
- [26] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. *CoRR* abs/2402.17152 (2024). arXiv:2402.17152
- [27] Buyun Zhang, Liang Luo, Xi Liu, Jay Li, Zeliang Chen, Weilin Zhang, Xiaohan Wei, Yuchen Hao, Michael Tsang, Wenjun Wang, Yang Liu, Huayu Li, Yasmine Badr, Jongsoo Park, Jiyan Yang, Dheevatsa Mudigere, and Ellie Wen. 2022. DHEN: A Deep and Hierarchical Ensemble Network for Large-Scale Click-Through Rate Prediction. *CoRR* abs/2203.11014 (2022). arXiv:2203.11014
- [28] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, 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 (KDD)*. ACM, 1059–1068. arXiv:1706.06978

A Total Computational Cost and Sensitivity Analysis

We model a ranking system over a common evaluation period. Training uses logged impressions, heavily samples negatives, and retains only a small fraction of consideration events. Ranking inference instead runs on every request without downsampling and scores every candidate passed to the ranking stage. This gives the workload decomposition

$$m_u = 1 + r_{\text{traffic}} \frac{K_{\text{serve}}}{K_{\text{train}}} r_{\text{eff}}, \quad (11)$$

where r_{traffic} is the ratio of served to retained training impressions, $K_{\text{serve}}/K_{\text{train}}$ captures full-candidate scoring versus sampled training examples, and r_{eff} accounts for the lower utilization of latency-constrained inference relative to batched training. Empirical workload analysis places m_u on the order of 30–100×; we use the lower end in all primary plots. This range is ranking-specific and is not presented as an industry-wide constant; for retrieval workloads, this amplification is generally more limited.

The plots compute C_{user} and C_{event} from active parameter FLOPs over the full training run; sparse embedding gathers are excluded from FLOPs. For AMBER, the Event Tokenizer is invoked asynchronously and its outputs are cached, so $m_e \ll m_u$. An uncached full-feature sequence instead sets $m_e = m_u$, reflecting repeated event-side computation on the ranking path.

Feature-materialization compute depends on the serving stack. We map it to the common FLOP axis using an empirical power ratio between feature serving and GPU serving. Because this ratio is infrastructure-dependent, we report only coarse ranges rather than absolute measurements. We model its first-order scaling as

$$C_{\text{mat}} \propto N_{\text{served}} H \sum_g n_g c_g, \quad (12)$$

where N_{served} is the number of served ranking requests, H is the materialized history length, n_g is the number of features in group g per event, and c_g is its empirically calibrated per-value materialization compute on the common axis. These coefficients may differ across serving stacks.

For the common compute axis, we normalize full heterogeneous-event materialization to one and empirically calibrate each schema. Semantic IDs with CU embeddings fall in the low-single-digit percentage range of this reference compute, while item features fall in the high-single-digit range because they contain more values and a larger floating-point payload. AMBER replaces both with a fixed-size cached token. These ranges affect only horizontal compute coordinates; all NE values are measured directly.

B Baseline Encoder Architectures

Figure 7 compares the Event Tokenizer with the Concat+FFN baseline used in the encoder architecture study (§5.3.2). Unlike the Event Tokenizer, Concat+FFN omits self-attention and limits inter-feature interactions to the feed-forward network.

Table 7 reports the comparison. DHEN (×12) matches the four-layer bidirectional Transformer at comparable FLOPs, while PMA and Concat+FFN trail by 0.20% and 0.22% NE, respectively. We select the Transformer because it is more flexible and achieves slightly

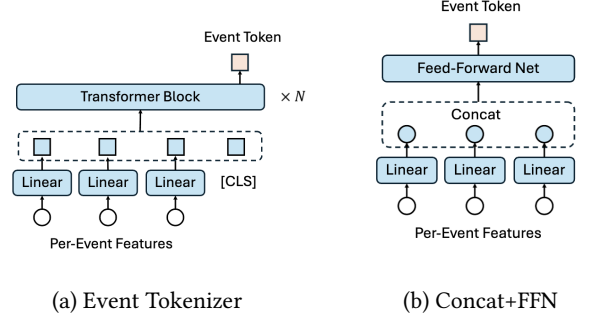


Figure 7: Event encoder architectures. (a) The Event Tokenizer processes projected per-event features and learnable [CLS] tokens with N bidirectional Transformer layers, then maps the [CLS] output to an Event Token. (b) Concat+FFN directly concatenates projected features and applies a feed-forward network without self-attention.

Table 7: Encoder architecture comparison. NE Δ is relative to the default four-layer bidirectional Transformer; positive values indicate degradation.

Encoder	NE Δ ↓
Bi-Transformer (×4)	—
Bi-Transformer (×2)	+0.09%
Bi-Transformer (×1)	+0.19%
DHEN (×12) [27]	0.00%
PMA (×4) [20]	+0.20%
Concat+FFN	+0.22%

higher practical efficiency through FlashAttention. Shrinking it to ×2 and ×1 increases NE by 0.09% and 0.19%, respectively.

C HSTU-Style Input Baseline

AMBER’s Event Tokenizer is agnostic to the downstream model: it can feed an HSTU-style sequential backbone as readily as the User LLM, and prior work finds these backbones perform comparably at scale [16]. The informative comparison with HSTU is therefore over the *input schema*, not the architecture. This ablation holds AMBER’s User LLM backbone fixed and changes only the input schema to match HSTU’s: each event retains only item-related categorical features, and user-side features are supplied once at the front of the sequence rather than encoded per event. To avoid handicapping it, we are generous with these user-side inputs, including all user float, embedding, and ID-list features (the reference implementation places only a few user categorical features at the front). This schema results in 1.0% higher NE relative to AMBER (+0.6% vs. Incumbent (fair)).

D Pointwise Baseline

This baseline isolates the benefit of maintaining high snapshot resolution throughout history from the training-efficiency benefit

of autoregressive modeling. Comparing AMBER directly with a conventional pointwise architecture would confound these effects because autoregressive training processes the full sequence in one pass, whereas pointwise training repeatedly encodes history for each impression. We therefore reproduce the pointwise input pattern within the same User LLM backbone and retain single-pass autoregressive training.

Each event is represented by three tokens: a high-resolution query token A containing the full event features, an item-only history token B containing the complete item-feature set, and an outcome token C . Although B is lower-resolution than the full event, it is a strong upper bound whose feature count would still be computationally costly to materialize throughout the online history. The attention mask enforces the pointwise information flow. For event i , A_i attends only to prior B and C tokens, so its prediction uses a high-resolution current query over an item-only history. Neither B nor C can attend to any A token. This restriction prevents high-resolution information from entering B or C and propagating indirectly to future predictions; the persistent history therefore remains strictly item-only.

This construction keeps the backbone, optimization objective, compute budget, and autoregressive training efficiency matched to AMBER, while changing only whether high-resolution event information is retained in history. Pointwise improves over Incumbent (fair) by 0.1% NE, isolating the benefit of autoregressive training, but remains 0.3% behind AMBER. The remaining difference therefore supports the benefit of preserving high snapshot resolution throughout history rather than only at the current query.

E Robustness to Feature Materialization Latency

In realistic serving settings, a delay of 1–5 minutes between an event’s occurrence and its feature materialization is typical. We experimented with simulating this delay during training by randomly masking recent events within a time window. Contrary to expectation, delay simulation degraded NE by 0.2% even when evaluation accounted for the same offset, suggesting that the model is inherently robust to feature materialization latency and that the artificial masking removes useful training signal.

F Representation Drift Visualization



Figure 8: t-SNE visualization of Event Tokens from two encoder checkpoints (red vs. blue) encoding the same events. (a) With a 1-day gap, the two distributions are well-mixed. (b) With a 20-day gap, color separation emerges within clusters, indicating accumulated drift.

G Recurrent Training Stability

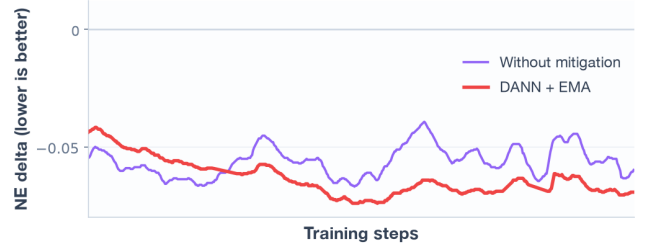


Figure 9: NE trajectory during recurrent training. Without drift mitigation (purple), NE shows higher variance across training steps. With DANN + EMA (red), the smoother trajectory indicates more stable downstream predictions. Both runs use identical hyperparameters.

H Pre-trained Initialization Dynamics

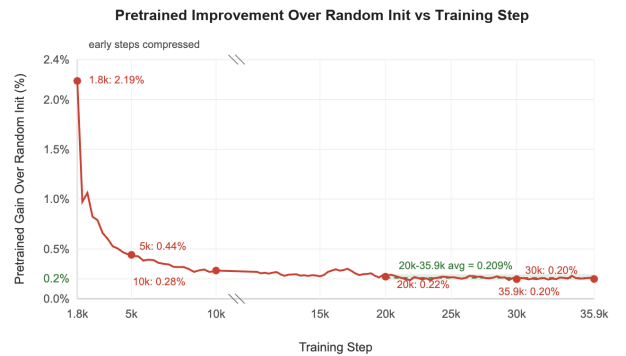


Figure 10: NE advantage of pre-trained LLM initialization over random initialization as a function of training steps. The advantage decays from ~2.2% early in training to a stable ~0.2% after 20K steps.