

COMPUTE-EFFICIENT SCALING OF LARGE SEQUENTIAL RECOMMENDATION MODELS UNDER DYNAMIC USER BEHAVIOR

AoWei Shen

University of Washington, Seattle, WA 98195, USA.

Abstract: Transformer-based sequential recommenders are increasingly scaled up and retrained as interaction logs grow, yet user behavior drifts between retraining cycles, and repeated from-scratch training of large models consumes compute that rarely yields proportional accuracy. This paper studies how model scale, update strategy, and training compute interact under a streaming protocol in which a model trained on earlier blocks of a chronologically ordered log serves the next block. On MovieLens-1M and Amazon Grocery, widening a SASRec-style model from 6.6K to 397K non-embedding parameters raises training compute by 4.4-13.3x relative to width 32 without a consistent accuracy gain, and models that are not refreshed lose 36% and 82% of their NDCG@10, respectively. Building on these observations, we propose SPROUT, a compute-aware update framework that warm-starts each update from a recency-aware replay stream, expands capacity with an exactly function-preserving width-doubling operator for pre-LayerNorm Transformers, and accepts growth only when a paired validation test shows a gain larger than one standard error. Over four update cycles, SPROUT stays within 1.3% of the best periodic-retraining accuracy on MovieLens-1M with 44% less cumulative compute than the retrained small model and 93% less than the retrained large model, and it achieves the highest accuracy on Amazon Grocery, 5.0% above the best retraining baseline, with 47% less compute than retraining the large model. Its growth-free configuration offers the most economical operating point, reaching 99.1% and 102.1% of the retraining accuracy with 33% and 51% of its compute.

Keywords: Sequential recommendation; Scaling laws; Model growth; Continual learning; Concept drift; Compute efficiency

1 INTRODUCTION

Sequential recommendation predicts the next item a user will interact with from the order of past interactions. Early neural approaches modeled sessions with recurrent networks and convolutional filters [1-3]; self-attentive models such as SASRec and BERT4Rec then became the standard backbone, and later work added time-interval awareness and self-supervised objectives [4-8]. In parallel, the scaling behavior of Transformers in language modeling, where test loss follows power laws in parameters, data, and compute, has motivated larger recommendation models, and power-law trends have been reported for click-through-rate models and for general-purpose user encoders [9-13].

Two properties of deployed recommenders complicate the transfer of these results. First, interaction logs are non-stationary. New users arrive, catalogs change, and item popularity shifts, so a model trained on the past is always evaluated on a future that differs from its training distribution [14]; offline protocols that ignore this ordering can change the ranking of competing systems [15]. Second, a deployed model is refreshed periodically, which means that training compute is paid at every update cycle rather than once. Full retraining on the entire history remains the most common practice, and its cost grows with both the volume of the log and the size of the model. Scaling studies typically assume a static corpus and a single training run, whereas incremental update methods for recommenders assume a fixed architecture [10-12, 16-17]. How to scale a sequential recommender across update cycles under drifting behavior with a bounded compute budget has therefore received little attention.

This paper studies that question empirically and methodologically. We build a streaming protocol that splits a chronologically ordered log into equal-volume blocks, trains a model on all blocks preceding stage s , and evaluates it on every interaction of block s with full-catalog ranking. Every training run is charged its forward and backward FLOPs, so that strategies can be compared on a common compute axis. Applying this protocol to MovieLens-1M and Amazon Grocery yields three observations. (i) Width scaling saturates early: across a 60-fold range of non-embedding parameters, accuracy either peaks at width 32–64 or stays flat within 0.0015 NDCG@10, while compute grows by more than an order of magnitude. (ii) Staleness is severe and is not mitigated by scale; a larger model that is not updated degrades more than a smaller one, in line with findings for language models [18]. (iii) Warm-started updates with replay recover the accuracy of full retraining at a fraction of its cost.

These observations motivate SPROUT (Scaling by PRogressive grOwth Under Temporal drift), a compute-aware update framework. SPROUT warm-starts every update from the deployed model and trains it on a recency-aware replay stream that covers all users active in the newest slice of the log plus a decayed sample of earlier users. Capacity is added through a width-doubling operator that preserves the network function exactly, including through LayerNorm, so that a grown model can replace the deployed one without any accuracy regression. Growth is proposed when the log has

doubled and is accepted only if a paired test on held-out users shows a gain larger than its standard error, which prevents compute from being spent on capacity that the data cannot yet support.

The contributions of this paper are as follows.

- 1) A streaming evaluation and compute-accounting protocol for sequential recommendation, together with a characterization of width scaling and staleness of SASRec-style models on two public datasets with contrasting drift profiles.
- 2) An exactly function-preserving width-doubling operator for pre-LayerNorm Transformer recommenders with tied item embeddings; unlike existing Transformer growth operators, it requires no approximation at LayerNorm [19].
- 3) SPROUT, which combines recency-aware warm-start updates with statistically gated growth, and an evaluation showing that it attains retraining-level or higher accuracy with 7–53% of the compute of retraining the large model, see Figure 1.

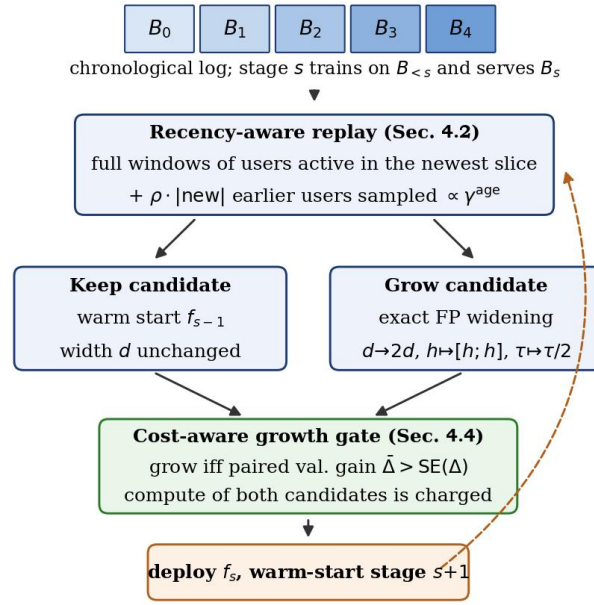


Figure 1 Overview of One SPROUT Update at Stage s

Note: A recency-aware replay stream feeds a keep candidate and, when growth is proposed, a grow candidate obtained by exact function-preserving widening. The growth gate accepts the widened model only if its paired validation gain exceeds one standard error; the compute of both candidates is charged to the update.

2 RELATED WORK

2.1 Sequential Recommendation

GRU4Rec introduced recurrent networks for session-based recommendation [1]. Caser applied horizontal and vertical convolutions to embedded sequences, and NextItNet used dilated residual convolutions [2-3]. SASRec adopted causal self-attention and became a widely used baseline, while BERT4Rec trained a bidirectional encoder with a cloze objective [4-5]. TiSASRec encoded the time intervals between interactions, and S3-Rec and CL4SRec introduced self-supervised pre-training and contrastive objectives [6-8]. These models are commonly evaluated with leave-one-out splits in which the last interaction of every user is held out regardless of its timestamp, so that the training data of one user may postdate the test data of another [15]. We use a global temporal split instead, and our backbone follows SASRec with a pre-LayerNorm arrangement of the Transformer blocks [9].

2.2 Scaling Laws and Model Growth

Kaplan *et al.* showed that language-model loss decreases as a power law in parameters, data, and compute, and Hoffmann *et al.* revised the compute-optimal allocation toward more data per parameter [10-11]. For recommendation, Ardalani *et al.* observed power-law-plus-constant behavior for DLRM-style click-through-rate models and found diminishing returns from parameter scaling, and Shin *et al.* reported power-law improvements for a contrastively trained user encoder [12-13]. Model growth reuses a trained smaller network to initialize a larger one. Net2Net introduced function-preserving widening and deepening, progressive stacking accelerated BERT training by duplicating layers, StackRec transferred layer stacking to very deep sequential recommenders, and bert2BERT extended function-preserving initialization to Transformers while accepting a small mismatch introduced by LayerNorm [19-22]. Our operator removes that mismatch by duplicating the residual stream and compensating the logit temperature.

2.3 Recommendation under Dynamic Behavior

Temporal dynamics have long been modeled explicitly in collaborative filtering [14]. For periodic retraining, SML learns a transfer component that converts the previous model into a new one using only new data, and ADER replays exemplars with an adaptive distillation loss to limit forgetting in session-based recommendation [16-17]. Both build on general continual-learning techniques such as Learning without Forgetting and experience replay [23-24]. Ash and Adams showed that warm-started networks can generalize worse than freshly initialized ones, and Lazaridou *et al.* found that increasing the size of language models does not remove their degradation on future data [18, 25]. Our work differs from these studies in treating model size as a decision variable across update cycles and in charging every training run, including discarded candidates, to the update.

3 PROBLEM FORMULATION

3.1 Streaming Protocol

Let $D = \{(u, i, t)\}$ be the interaction log sorted by timestamp and split into K equal-volume blocks $B_0, \dots, B_{K-1}$; we use $K = 5$. At stage $s \in \{1, \dots, K-1\}$, a model f_s is trained on $B_{<s}$ and serves block B_s . The last 10% of B_{s-1} forms a validation slice V_s used for early stopping and for the growth gate; training targets end at the validation cut v_s , but validation interactions remain part of user histories at test time. For each interaction $(u, i, t) \in B_s$, the model receives the complete history of u before t , including earlier interactions of the same block, because parameters are frozen while a block is served. It ranks every item observed before v_s , excluding items the user has already consumed; targets whose item was never observed in training are skipped. The interactions of a user are scored in chunks of 25 consecutive targets with one causal pass over the 50 preceding items, so that each target sees at least 26 history items when available. Ranking quality is measured by HR@10 and NDCG@10, averaged over the test interactions T_u of each user and then over users,

$$\text{NDCG@}K = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|T_u|} \sum_{t \in T_u} \frac{\mathbf{1}[r_t \leq K]}{(r_t + 1)} \quad (1)$$

where r_t is the rank of the target among all candidates and HR@ K replaces the discount by 1.

3.2 Update Strategies and Compute Accounting

An update strategy maps the deployed model f_{s-1} and the log $B_{<s}$ to f_s . Its cost is measured in training FLOPs. Following the standard approximation that a backward pass costs twice a forward pass, the cost per processed (padded) position of an L -layer model of width d is [10]:

$$C_{\text{tok}} = 3 [L(24d^2 + 4Td) + 2d(K + 1)] \quad (2)$$

where T is the padded sequence length and K the number of sampled negatives; the terms cover the query, key, value, output, and feed-forward projections, the two attention products, and the output scores. Embedding lookups and element-wise operations are omitted. The cumulative cost of a strategy over stages 1–4 includes its stage-1 cold start, and when a strategy trains several candidates, every candidate is charged.

4 THE SPROUT FRAMEWORK

4.1 Backbone

The recommender is a causal Transformer with $L = 2$ pre-LayerNorm blocks, width d , $d/32$ attention heads of dimension 32 (one head for $d = 16$), a feed-forward width of $4d$ with GELU activations, and dropout on embeddings and sub-layer outputs. Item embeddings $\mathbf{E} \in \mathbb{R}^{|I| \times d}$ are tied with the output layer, and learned positional embeddings $\mathbf{P} \in \mathbb{R}^{T \times d}$ are right-aligned so that the most recent item always occupies the last position. Given the final hidden state $\mathbf{h}$, the next-item distribution is

$$p(i|S_{u, <t}) = \frac{\exp(\tau \mathbf{h}_t^\top \mathbf{e}_i)}{\sum_{j \in J} \exp(\tau \mathbf{h}_t^\top \mathbf{e}_j)} \quad (3)$$

with a learnable temperature τ . Training minimizes a sampled softmax in which $K = 512$ negatives N are drawn uniformly per batch, shared by all positions, and corrected for the sampling rate [26]:

$$\mathcal{L} = - \sum_t w_t \log \frac{e^{\tau \mathbf{h}_t^\top \mathbf{e}_{y_t}}}{e^{\tau \mathbf{h}_t^\top \mathbf{e}_{y_t}} + \sum_{j \in N} e^{\tau \mathbf{h}_t^\top \mathbf{e}_j + \log \frac{|J|-1}{K}}} \quad (4)$$

where $w_t \in \{0, 1\}$ selects the supervised positions. The correction term turns the sampled normalizer into an unbiased estimate of the full one, and sampling removes the $O(|J|)$ term from the training cost; evaluation always scores the full catalog.

4.2 Recency-Aware Replay

An update at stage s learns from the slice of interactions between the previous and the current validation cuts, $[v_{s-1}, v_s)$. Every user with at least one target in this slice contributes its most recent training window of up to $T = 50$ items, and the loss is applied to all positions of the window. Because the forward pass already covers the older positions, supervising

them replays past behavior at no additional cost. In addition, $\rho|U_{\text{new}}|$ users without new activity are sampled at every epoch, where U_{new} is the set of users active in the slice, with probability

$$\Pr(u) \propto \gamma^{a(u)}, \quad a(u) = (s-1) - b_{\text{last}}(u) \quad (5)$$

where $b_{\text{last}}(u)$ is the block of the last training interaction of u . The decay γ concentrates replay on recently active users while still revisiting older behavior; we use $\rho = 0.5$ and $\gamma = 0.5$. The model is warm-started from f_{s-1} with a fresh optimizer state, and the pre-update model is kept as the initial early-stopping candidate, so that an update never selects a model that is worse on validation than the deployed one.

4.3 Exact Function-Preserving Widening

The growth operator G maps a width- d model to width $2d$. Let $\mathbf{1}_2 = [1]^\top$ and $\mathbf{J}_2 = \mathbf{1}_2 \mathbf{1}_2^\top$. G sets

$$\mathbf{E}' = \mathbf{1}_2^\top \otimes \mathbf{E}, \quad \mathbf{W}' = \frac{1}{2} (\mathbf{J}_2 \otimes \mathbf{W}), \quad \mathbf{b}' = \mathbf{1}_2 \otimes \mathbf{b}, \quad \tau' = \frac{\tau}{2} \quad (6)$$

for the item and positional embeddings ($\mathbf{E}$, $\mathbf{P}$), for every linear map $\mathbf{W}$ and bias $\mathbf{b}$ inside a block, and for the LayerNorm gains and biases, which are duplicated. The query, key, and value maps are transformed separately, which doubles the number of heads while keeping their dimension, and the feed-forward width becomes $8d$.

Proposition 1: For every input sequence, the widened model produces the residual state $[\mathbf{h}_\ell; \mathbf{h}_\ell]$ after every layer ℓ and exactly the same logits as the original model for every item.

Proof: The input embedding of the widened model is $[\mathbf{x}; \mathbf{x}]$. The mean and variance of a duplicated vector equal those of the original vector, and every widened map receives two copies of its input, each multiplied by $W/2$:

$$\text{LN}([\mathbf{x}; \mathbf{x}]) = [\text{LN}(\mathbf{x}); \text{LN}(\mathbf{x})], \quad [\mathbf{z}; \mathbf{z}] \mathbf{W}' = [\mathbf{z}\mathbf{W}; \mathbf{z}\mathbf{W}] \quad (7)$$

Hence the query, key, and value vectors of the $2H$ heads are two copies of the original H heads; attention scores and weights are unchanged because the head dimension is unchanged, and the attention output and its projection are duplicated. The feed-forward block applies an element-wise activation to a duplicated vector and therefore also returns a duplicated output. Residual additions preserve duplication, so by induction every layer outputs $[\mathbf{h}_\ell; \mathbf{h}_\ell]$ and the final LayerNorm returns $[\mathbf{z}; \mathbf{z}]$, where $\mathbf{z}$ is the original final state. Since the tied output embeddings are duplicated as well,

$$\tau' \mathbf{h}'^\top \mathbf{e}'_i = \frac{\tau}{2} [\mathbf{z}; \mathbf{z}]^\top [\mathbf{e}_i; \mathbf{e}_i] = \tau \mathbf{z}^\top \mathbf{e}_i \quad (8)$$

which proves the claim.

The operator is exact up to floating-point rounding; the largest absolute logit difference measured in our implementation after one and two widenings is below 10^{-6} . bert2BERT, in contrast, expands LayerNorm without preserving its statistics and accepts the resulting discrepancy [19]. After widening, dropout and a Gaussian perturbation with standard deviation $0.1\sigma_0$ added to every weight matrix ($\sigma_0 = 0.02$ is the initialization scale) break the symmetry between duplicated units so that they can specialize, as in Net2Net [20].

4.4 Cost-Aware Growth Gate

Growth is proposed whenever the interaction log has doubled since the previous proposal, which with equal-volume blocks corresponds to stages 2 and 4. Two candidates are then trained on the same replay stream with identical seeds: a keep candidate at width d and a grow candidate initialized with G . For each validation user u , let Δ_u be the difference between the NDCG@10 of the grow and keep candidates. The grow candidate is accepted if and only if

$$\bar{\Delta} = \frac{1}{n} \sum_{u=1}^n \Delta_u > \text{SE}(\Delta) = \frac{\hat{\sigma}_\Delta}{\sqrt{n}} \quad (9)$$

where $\hat{\sigma}_\Delta$ is the sample standard deviation over the n validation users. This is a paired form of the one-standard-error rule: the larger model must exceed the smaller one by more than the resolution of the validation set [27]. The training compute of both candidates is charged to the update. Algorithm 1 summarizes one update.

Algorithm 1. SPROUT update at stage s

Input: deployed model f_{s-1} of width d ; log $B < s$; validation slice V_s ; ρ , γ

1: $W \leftarrow$ latest windows of users with targets in $[v_{s-1}, v_s]$, all positions supervised

2: $O \leftarrow$ remaining users; $a(u) \leftarrow$ blocks since last activity

3: stream R : each epoch uses $W \cup \text{sample}(O, \rho|W|, \Pr \propto \gamma a(u))$

4: **if** the log has doubled since the last proposal **then**

5: $f_{\text{keep}} \leftarrow \text{Train}(f_{s-1}, R)$; $f_{\text{grow}} \leftarrow \text{Train}(G(f_{s-1}) + \varepsilon, R)$

6: $\Delta_u \leftarrow \text{NDCGu}(f_{\text{grow}}) - \text{NDCGu}(f_{\text{keep}})$ for every user of V_s

7: $f_s \leftarrow f_{\text{grow}}$ if $\text{mean}(\Delta) > \text{SE}(\Delta)$, otherwise f_{keep}

8: **else** $f_s \leftarrow \text{Train}(f_{s-1}, R)$

9: **return** f_s ; the compute of every Train call is charged to stage s

$\text{Train}(f, R)$: Adam from f ; keep the best validation checkpoint, including f itself; stop after 3 epochs without improvement.

4.5 Update Cost

Full retraining at stage s processes the windows of all users with training history in every epoch and starts from random weights. A SPROUT update processes at most $(1 + \rho)|U_{\text{new}}|$ windows per epoch and starts from the deployed model, which is already accurate for most users, so that it needs fewer epochs. When growth is proposed, the grow candidate

adds roughly four times the per-position cost of the keep candidate, because the dominant terms of (2) scale with d^2 . The gate therefore spends a bounded exploration budget to decide whether additional capacity is justified by the data.

5 EXPERIMENTAL SETUP

5.1 Datasets

We use two public datasets with timestamps. MovieLens-1M contains 1,000,209 ratings by 6,040 users on 3,706 movies between April 2000 and February 2003 [28]; every rating is treated as an interaction. The 5-core version of Amazon Grocery and Gourmet Food contains 151,254 reviews by 14,681 users on 8,713 products between August 2000 and July 2014 [29]. Table 1 reports the drift profile of the five blocks. On MovieLens-1M, most active users of blocks 1–3 are new (79–86%) while the catalog is stable (at most 1.1% of the interactions involve unseen items), and the Jensen–Shannon divergence between the item-popularity distributions of consecutive blocks stays between 0.011 and 0.027. Amazon Grocery exhibits strong catalog drift: 56.4% of the interactions of block 1 involve items never seen before, and the divergence ranges from 0.317 to 0.582. The test sets contain 196,009–199,287 interactions of 1,435–1,794 users per stage on MovieLens-1M and 10,604–25,123 interactions of 4,622–7,790 users on Amazon Grocery.

Table 1 Drift Profile of the Equal-Volume Blocks

Block	Period	New users	Unseen items	JS
MovieLens-1M				
B ₀	2000-04 – 2000-07	1.000	–	–
B ₁	2000-07 – 2000-08	0.858	0.010	0.027
B ₂	2000-08 – 2000-11	0.789	0.011	0.022
B ₃	2000-11 – 2000-12	0.833	0.001	0.011
B ₄	2000-12 – 2003-02	0.359	0.001	0.016
Amazon Grocery				
B ₀	2000-08 – 2011-04	1.000	–	–
B ₁	2011-04 – 2012-08	0.400	0.564	0.582
B ₂	2012-08 – 2013-05	0.324	0.300	0.477
B ₃	2013-05 – 2013-12	0.180	0.221	0.375
B ₄	2013-12 – 2014-07	0.095	0.131	0.317

Note: New users: share of active users whose first interaction lies in the block. Unseen items: share of interactions on items absent from all earlier blocks. JS: Jensen–Shannon divergence (base 2) between the item-popularity distributions of the block and the previous block.

5.2 Compared Strategies

FR-d retrains a model of width d from scratch on $B_{<s}$ at every stage, with $d \in \{32, 128\}$ at all stages and $d \in \{16, 64\}$ at stages 1 and 4 for the scaling analysis. *Static-d* deploys the stage-1 *FR-d* model without updates. *FT-128* fine-tunes the previous width-128 model with the loss restricted to new targets. *FT+KD-128* (MovieLens-1M) adds a distillation term toward the previous model's softmax over the sampled candidates, in the spirit of Learning without Forgetting and the distillation used by ADER, with weight 0.5 and temperature 1 [17, 23]. *SPROUT* starts from the *FR-32* stage-1 model; *SPROUT-lite* is *SPROUT* without growth proposals. Three ablations grow at stages 2 and 4 without the gate: *always grow*, *random widening*, which keeps the old weights in the leading block and initializes the new rows and columns randomly, so that the function is not preserved; and *no replay*, which restricts the loss to new targets.

5.3 Implementation Details

All models are implemented in JAX and trained with Adam (learning rate 10^{-3} , $\beta = (0.9, 0.999)$), batches of 32 sequences, maximum length $T = 50$, and $K = 512$ sampled negatives [30]. Dropout is 0.2 on MovieLens-1M and 0.5 on Amazon Grocery, following the settings of SASRec [4]. Batches are bucketed by length (8, 16, 32, 50) to limit padding. Validation NDCG@10 is computed after every epoch. Cold-start runs train for at least 1,500 steps and stop when validation has not improved for max(3 epochs, 400 steps), with at most 120 epochs; warm-started updates stop after 3 epochs without improvement, with at most 20 epochs. All runs use one fixed seed and a single Intel Xeon CPU core, and compute is reported with (2).

6 RESULTS

6.1 Width Scaling under Temporal Drift

Table 2 and Figure 2 report the accuracy and cost of full retraining at four widths. On MovieLens-1M at stage 1, accuracy peaks at $d = 32$ (NDCG@10 of 0.0787); $d = 64$ and $d = 128$ are 9.7% and 23.5% worse while consuming $4.2\times$ and $10.9\times$ more compute. At stage 4, with four times as much data, $d = 64$ matches $d = 32$ (0.1055 versus 0.1053) and the deficit of $d = 128$ shrinks to 8.1%. The accuracy-optimal width thus grows with the volume of the log, consistent

with compute-optimal analyses in which the best model size increases with data, but in this regime compute grows far faster than accuracy [11]. On Amazon Grocery, stage-1 accuracy is flat across widths (0.0143–0.0158), and at stage 4 $d = 32$ is the most accurate (0.0276), while $d = 128$ costs $4.4\times$ more and is 15.2% worse. Larger models therefore do not repay their compute when they are retrained from scratch at every cycle.

Table 2 Full Retraining at Four Widths: Test NDCG@10 / Training TFLOPs

d	Non-emb. params	MovieLens-1M		Amazon Grocery	
		Stage 1	Stage 4	Stage 1	Stage 4
16	6,560	0.0748 / 0.41	0.0927 / 0.64	0.0157 / 0.08	0.0254 / 0.40
32	25,408	0.0787 / 0.79	0.1053 / 1.53	0.0154 / 0.15	0.0276 / 1.12
64	99,968	0.0711 / 3.29	0.1055 / 4.66	0.0143 / 0.58	0.0217 / 1.43
128	396,544	0.0602 / 8.63	0.0968 / 10.00	0.0158 / 1.99	0.0234 / 4.92

Note: Total parameters including embeddings range from 66,704 to 877,696 on MovieLens-1M and from 146,816 to 1,518,592 on Amazon Grocery. Best value per column in bold.

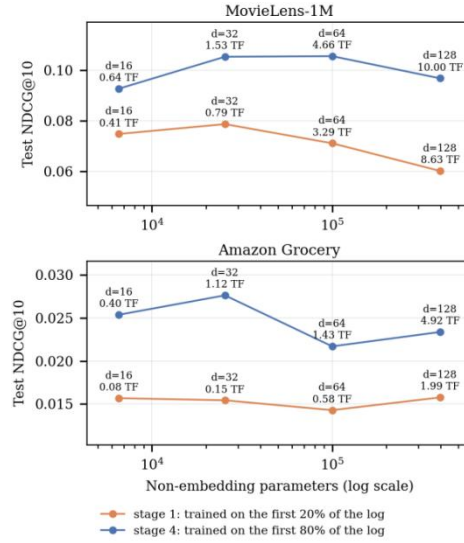


Figure 2 Test NDCG@10 of Full Retraining Versus Non-Embedding Parameters at Stage 1 (Trained on the First 20% of the Log) and Stage 4 (First 80%)

Note: Point labels give the width and the training compute.

6.2 Staleness

Figure 3 shows how models degrade when they are not refreshed. Averaged over stages 2–4, Static-32 loses 35.9% of the NDCG@10 of FR-32 on MovieLens-1M and 81.6% on Amazon Grocery. Scale does not protect against drift: Static-128 loses 45.6% and 88.3% relative to FR-128. The severity follows the drift profile in Table 1. On Amazon Grocery, where the catalog changes quickly, the stale models fall to NDCG@10 values between 0.0020 and 0.0053, less than one quarter of those of the refreshed models. Periodic updates are therefore indispensable, and their cumulative cost is the quantity to optimize.

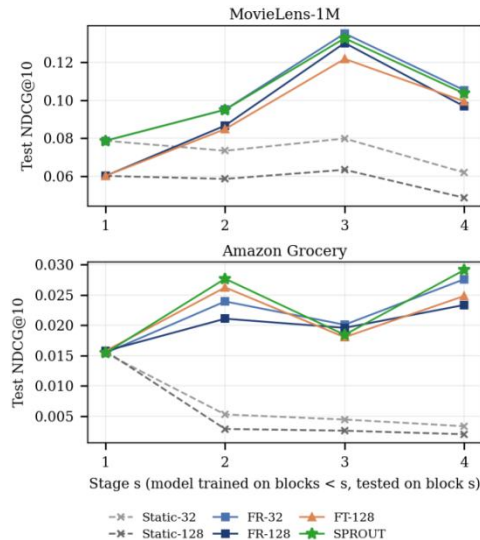


Figure 3 Test NDCG@10 per Stage

Note: Static models are trained once at stage 1; FR models are retrained from scratch at every stage; FT-128 and SPROUT are warm-started updates.

6.3 Accuracy–Compute Trade-Off

Table 3 and Figure 4 compare all strategies. On MovieLens-1M, FR-32 reaches the highest mean NDCG@10 (0.1118) with 5.92 TFLOPs. SPROUT is 1.3% lower (0.1104) while using 3.30 TFLOPs (−44%) and 249 s of training time instead of 371 s. Compared with FR-128, SPROUT is 5.6% more accurate with 7.1% of the compute and 6.6× less training time. Fine-tuning reduces the cost of the large model by 68% (FT-128) at a loss of 2.4% relative to FR-128, and distillation lowers accuracy further (0.0986), because its target encodes the preferences of the previous period. On Amazon Grocery, SPROUT attains the best mean NDCG@10 (0.0251) and HR@10 (0.0449), 5.0% above FR-32, 17.8% above FR-128, and 8.7% above FT-128, while using 53% of the compute of FR-128. Its cost exceeds that of FR-32 (6.26 versus 1.72 TFLOPs) mainly because the gate trained and rejected a width-128 candidate at stage 4 (3.48 TFLOPs).

SPROUT-lite provides the lowest-cost operating point on both datasets: it reaches 99.1% of the accuracy of FR-32 with 33% of its compute on MovieLens-1M and 102.1% with 51% of its compute on Amazon Grocery. In Figure 4, the Pareto front consists of SPROUT variants on Amazon Grocery and of SPROUT-lite and FR-32 on MovieLens-1M; every strategy that trains or fine-tunes a width-128 model is dominated. The savings come from two sources. Warm starts shorten each update, and the replay stream is smaller than the full training set on MovieLens-1M (2,200–2,641 windows per epoch versus 2,548–5,306 for full retraining), whereas on Amazon Grocery, where most users are active in every slice, the savings stem mainly from the shorter updates.

Table 3 Streaming Results: Test NDCG@10 per Stage, Means over Stages 2–4, Cumulative Training Compute, and Training Time

Method	s=2	s=3	s=4	Mean	HR@10	TFLOPs	Time (s)
MovieLens-1M							
Static-32	0.0734	0.0798	0.0620	0.0717	0.1405	0.79	78
Static-128	0.0586	0.0634	0.0487	0.0569	0.1126	8.63	357
FR-32	0.0951	0.1352	0.1053	0.1118	0.2147	5.92	371
FR-128	0.0867	0.1302	0.0968	0.1045	0.2043	46.49	1637
FT-128	0.0848	0.1218	0.0994	0.1020	0.1997	14.78	603
FT+KD-128	0.0803	0.1164	0.0991	0.0986	0.1932	18.96	719
SPROUT-lite	0.0950	0.1327	0.1046	0.1108	0.2131	1.94	166
SPROUT	0.0950	0.1327	0.1036	0.1104	0.2129	3.30	249
Amazon Grocery							
Static-32	0.0053	0.0045	0.0034	0.0044	0.0085	0.15	29
Static-128	0.0029	0.0026	0.0020	0.0025	0.0054	1.99	77
FR-32	0.0239	0.0201	0.0276	0.0239	0.0443	1.72	127
FR-128	0.0211	0.0196	0.0234	0.0213	0.0393	11.79	345
FT-128	0.0263	0.0180	0.0248	0.0231	0.0425	8.61	287
SPROUT-lite	0.0273	0.0187	0.0270	0.0244	0.0447	0.87	97
SPROUT	0.0277	0.0184	0.0291	0.0251	0.0449	6.26	234

Note: Static models are listed with their single stage-1 training cost. TFLOPs and time are cumulative over stages 1–4 and include the stage-1 cold start of the model from which each strategy starts. Best values among updated models in bold.

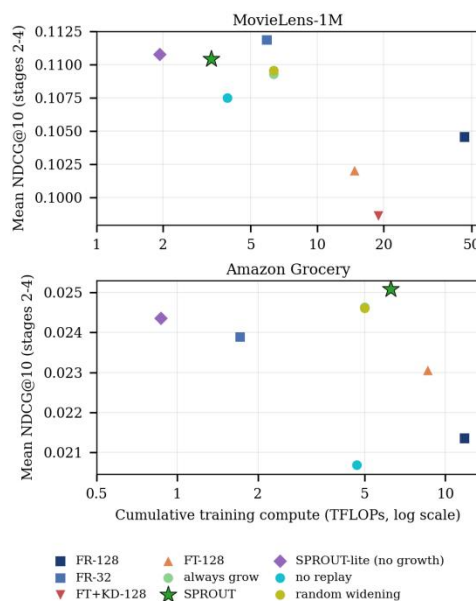


Figure 4 Mean NDCG@10 over Stages 2–4 Versus Cumulative Training Compute over Stages 1–4 (Log Scale)

Note: Never-updated models are omitted for readability (see Table 3).

6.4 Ablation Study

Table 4 isolates the components of SPROUT. Gating matters for both accuracy and cost: always growing lowers the mean NDCG@10 by 1.0% on MovieLens-1M while nearly doubling the compute (6.36 versus 3.30 TFLOPs), and it lowers the mean NDCG@10 by 2.0% on Amazon Grocery. On MovieLens-1M the gate rejected the stage-2 growth, which always-grow accepts at a cost in stage-2 NDCG@10 (0.0924 versus 0.0950); on Amazon Grocery it rejected the stage-4 growth to width 128, which always-grow accepts at a cost in stage-4 NDCG@10 (0.0278 versus 0.0291). Growth itself pays off where the drift is strong: removing it (SPROUT-lite) costs 2.8% on Amazon Grocery and is neutral on MovieLens-1M (+0.4%), whose stable catalog is served well at width 32. Replay is the most important component under drift. Without it, NDCG@10 falls by 1.6% on MovieLens-1M and by 15.9% on Amazon Grocery, where each user contributes only about four new interactions per update and the stage-4 value drops from 0.0278 to 0.0192. Function preservation leaves the converged accuracy unchanged (0.1093 versus 0.1095 and 0.0246 versus 0.0246), because both initializations are trained until validation stops improving; its benefit appears at the moment of growth, as analyzed next.

Table 4 Ablation: Mean NDCG@10 over Stages 2–4 and Cumulative TFLOPs

Variant	Growth	MovieLens-1M		Amazon Grocery	
		NDCG@10	TFLOPs	NDCG@10	TFLOPs
SPROUT (full)	gated	0.1104	3.30	0.0251	6.26
w/o growth (SPROUT-lite)	none	0.1108	1.94	0.0244	0.87
w/o gate (always grow)	s = 2, 4	0.1093	6.36	0.0246	5.01
always grow, random widening	s = 2, 4	0.1095	6.36	0.0246	5.01
always grow, w/o replay	s = 2, 4	0.1075	3.93	0.0207	4.68

6.5 Growth Dynamics

Figure 5 traces the validation NDCG@10 during the updates that follow a growth step. With function-preserving widening, the first evaluation after growth reproduces the quality of the pre-growth model up to the symmetry-breaking perturbation, whereas random widening starts lower. On MovieLens-1M, the validation NDCG@10 immediately after growth is 0.0558 versus 0.0458 at stage 2 (−17.9% for random widening) and 0.1151 versus 0.0959 at stage 4 (−16.7%); on Amazon Grocery it is 0.0125 versus 0.0108 at stage 4 (−13.3%). The best validation value reached after training is also higher with function preservation in all four growth events (0.0849 versus 0.0839, 0.1292 versus 0.1253, 0.0988 versus 0.0973, and 0.0402 versus 0.0398). Function preservation therefore turns growth into a zero-regression operation: the widened model can be deployed as soon as it is created, and subsequent training only adds to its quality.

Table 5 lists the decisions of the gate. The accepted growths correspond to gains of 1.92 and 1.10 standard errors, and the rejected ones to 0.87 and 0.32 standard errors. The rejection at stage 4 on Amazon Grocery kept a model with 661K parameters in place of one with 1.52M parameters, whose validation advantage (0.0402 versus 0.0397) lay well within the resolution of the validation set.

Table 5 Growth-Gate Decisions of SPROUT

Dataset	s	Proposal	Val. keep	Val. grow	Δ / SE	Decision
MovieLens-1M	2	32→64	0.0819	0.0849	0.87	keep
MovieLens-1M	4	32→64	0.1260	0.1294	1.92	grow
Amazon Grocery	2	32→64	0.0972	0.0988	1.10	grow
Amazon Grocery	4	64→128	0.0397	0.0402	0.32	keep

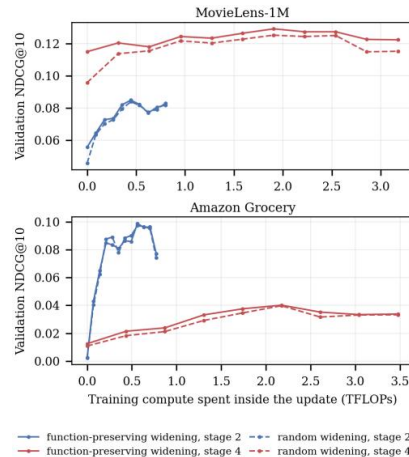


Figure 5 Validation NDCG@10 During the Updates That Follow a Growth Step, for Function-Preserving and Random Widening (Always-Grow Schedule)

Note: The first point of each curve is the model immediately after widening.

6.6 Discussion

The results carry three practical implications for recommenders that are refreshed periodically. First, capacity should follow the data rather than be fixed at a large value: the accuracy-optimal width grows only slowly with the volume of the log, and retraining a large model at every cycle is both the most expensive option and among the least accurate ones. Second, the dominant saving comes from the update rule: warm starting with recency-aware replay recovers retraining accuracy with fewer users per epoch and fewer epochs per update. Third, capacity changes should be tested statistically and executed with function-preserving operators, so that growth is accepted only when validation evidence supports it and never causes a regression at deployment time. The cost of the gate is bounded by one additional candidate per proposal and can be reduced further by proposing growth less often or by training the grow candidate on a subsample of the replay stream.

7 CONCLUSION

This paper examined the scaling of Transformer-based sequential recommenders under dynamic user behavior with an explicit account of training compute. A streaming protocol on MovieLens-1M and Amazon Grocery showed that widening models beyond a small size multiplies training cost without consistent accuracy gains, and that stale models lose most of their accuracy regardless of their size. The proposed SPROUT framework combines recency-aware warm-start updates, an exactly function-preserving widening operator, and a one-standard-error growth gate. It matches periodic retraining within 1.3% on MovieLens-1M with 44–93% less compute and exceeds all retraining baselines on Amazon Grocery. Future work will extend the operator family to depth growth with identity-initialized blocks, apply the gate to larger industrial logs and catalogs, and learn when to propose growth from the observed drift.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

- [1] Hidasi B, Karatzoglou A, Baltrunas L, et al. Session-based recommendations with recurrent neural networks. In: *Proceedings of International Conference Learn. Represent. (ICLR)*, 2016.
- [2] Tang J, Wang K. Personalized top-N sequential recommendation via convolutional sequence embedding. In: *Proceedings of ACM International Conference Web Search Data Mining (WSDM)*, 2018: 565-573.
- [3] Yuan F, Karatzoglou A, Arapakis I, et al. A simple convolutional generative network for next item recommendation. In: *Proceedings of ACM International Conference Web Search Data Mining (WSDM)*, 2019: 582-590.
- [4] Kang W-C, McAuley J. Self-attentive sequential recommendation. In: *Proceedings of IEEE International Conference Data Mining (ICDM)*, 2018: 197-206.
- [5] Sun F, Liu J, Wu J, et al. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In: *Proceedings of ACM International Conference Inf. Knowl. Manage. (CIKM)*, 2019: 1441-1450.
- [6] Li J, Wang Y, McAuley J. Time interval aware self-attention for sequential recommendation. In: *Proceedings of ACM International Conference Web Search Data Mining (WSDM)*, 2020: 322-330.
- [7] Zhou K, Wang H, Zhao W X, et al. S3-Rec: Self-supervised learning for sequential recommendation with mutual information maximization. In: *Proceedings of ACM International Conference Inf. Knowl. Manage. (CIKM)*, 2020.
- [8] Xie X, Sun F, Liu Z, et al. Contrastive learning for sequential recommendation. In: *Proceedings of IEEE International Conference Data Eng. (ICDE)*, 2022: 1259-1273.
- [9] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [10] Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.
- [11] Hoffmann J, Borgeaud S, Mensch A, et al. Training compute-optimal large language models. *arXiv:2203.15556*, 2022.
- [12] Ardalani N, Wu C-J, Chen Z, et al. Understanding scaling laws for recommendation models. *arXiv:2208.08489*, 2022.
- [13] Shin K, Kwak H, Kim S Y, et al. Scaling law for recommendation models: Towards general-purpose user representations. *arXiv:2111.11294*, 2021.
- [14] Koren Y. Collaborative filtering with temporal dynamics. In: *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.
- [15] Meng Z, McCreddie R, Macdonald C, et al. Exploring data splitting strategies for the evaluation of recommendation models. In: *Proceedings of ACM Conference Recommender Syst. (RecSys)*, 2020: 681-686.

- [16] Zhang Y, Feng F, Wang C, et al. How to retrain recommender system? A sequential meta-learning method. In: Proceedings of International ACM SIGIR Conference Res. Develop. Inf. Retrieval (SIGIR), 2020: 1479-1488.
- [17] Mi F, Lin X, Faltings B. ADER: Adaptively distilled exemplar replay towards continual learning for session-based recommendation. In: Proceedings of ACM Conference Recommender Syst. (RecSys), 2020: 408-413.
- [18] Lazaridou A, Kuncoro A, Gribovskaya E, et al. Mind the gap: Assessing temporal generalization in neural language models. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2021: 29348-29363.
- [19] Chen C, Yin Y, Shang L, et al. bert2BERT: Towards reusable pretrained language models. In: Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL), 2022: 2134-2148.
- [20] Chen T, Goodfellow I, Shlens J. Net2Net: Accelerating learning via knowledge transfer. In: Proceedings of International Conference Learn. Represent. (ICLR), 2016.
- [21] Gong L, He D, Li Z, et al. Efficient training of BERT by progressively stacking. In: Proceedings of International Conference Machine Learning (ICML), PMLR 97, 2019: 2337-2346.
- [22] Wang J, Yuan F, Chen J, et al. StackRec: Efficient training of very deep sequential recommender models by iterative stacking. In: Proceedings of International ACM SIGIR Conference Res. Develop. Inf. Retrieval (SIGIR), 2021.
- [23] Li Z, Hoiem D. Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, 40(12): 2935-2947.
- [24] Rolnick D, Ahuja A, Schwarz J, et al. Experience replay for continual learning. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2019: 350-360.
- [25] Ash J T, Adams R P. On warm-starting neural network training. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2020, 33: 3884-3894.
- [26] Jean S, Cho K, Memisevic R, et al. On using very large target vocabulary for neural machine translation. In: Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL), 2015: 1-10.
- [27] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning. 2nd ed. New York, NY, USA: Springer, 2009.
- [28] Harper F M, Konstan J A. The MovieLens datasets: History and context. ACM Transactions on Interactive Intelligent Systems, 2015, 5(4): 1-19.
- [29] He R, McAuley J. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In: Proceedings of International Conference World Wide Web (WWW), 2016: 507-517.
- [30] Kingma D P, Ba J. Adam: A method for stochastic optimization. In: Proceedings of International Conference Learn. Represent. (ICLR), 2015.