Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Patch-Routed Mixture-of-Experts for Continual MOBA Draft Recommendation

Mohammadreza Mohammadnejad
Semnan University
Morteza Dorrigiv
Dorrigiv@semnan.ac.ir
Semnan University
Farzin Yaghmaee
Semnan University

Research Article

Keywords: Continual learning, Mixture of experts, Concept drift, Sequential recommendation, Parameter isolation, MOBA games

Posted Date: August 25th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10581875/v1

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

Additional Declarations: No competing interests reported.

A Patch-Routed Mixture-of-Experts for Continual MOBA Draft Recommendation

Anonymous Author(s)[1*]

[1*]Affiliation withheld for double-anonymous review.

Corresponding author(s). E-mail(s): anonymous@example.com;

Abstract

Draft-phase recommendation in multiplayer online battle arena (MOBA) games must adapt to balance patches that repeatedly shift the game meta. In shared-weight continual learning, adapting to a new patch can overwrite parameters that remain important for earlier patches, creating a fundamental stability–plasticity trade-off. We propose a patch-routed mixture-of-experts architecture that isolates adaptation across patches while retaining a shared representation. Each patch is assigned a dedicated expert selected through its patch identity, which is naturally available at deployment in versioned games. This patch-aware routing prevents cross-patch interference while allowing new patches to adapt without modifying earlier patch-specific knowledge. Across our experiments, the proposed architecture matches sequential fine-tuning's plateau at the first patch transition while reaching it from roughly half as many post-patch samples, and achieves zero forgetting on earlier patches at 63% of the per-patch storage required to maintain an independent model for every patch. The zero-forgetting property is verified by weight-identity checks against saved models at every patch and seed. The speed advantage holds over the first two patch transitions and is lost as the stream lengthens, scoping the architecture to streams short relative to the frozen trunk's useful lifetime.

Keywords: Continual learning, Mixture of experts, Concept drift, Sequential recommendation, Parameter isolation, MOBA games

1 Introduction

Hero drafting is among the most-studied decision problems in Multiplayer Online Battle Arena (MOBA) games. Two teams alternately ban and pick heroes, and each

choice constrains the synergies a side can still build and the counters it faces. Because a match's outcome is predictable from in-game state well before it ends [2], and the draft is the earliest such state, picking well has real competitive value, and a line of work recommends picks from archives of historical matches [6, 9, 16, 24]. These recommenders are fit to a historical distribution and perform well on it, but that distribution is not stationary. The developer patches the game deliberately and often, and a recommender trained on pre-patch matches loses accuracy on the matches that follow.

This degradation is systematic and dated. A balance patch rewrites hero base statistics, ability values, and item costs on a known release date, and such hero-level factors register at the population level in match outcomes [42]. Because synergy is a first-class concept in MOBA play, *which* heroes changed matters as much as by how much, since one buff can re-rank an entire family of compositions [22]. Retraining from scratch after every patch is costly and discards the knowledge the patch left untouched [51]; the problem is adaptation at the patch boundary, not adaptation in general.

Continual learning (CL) is the natural frame, since the patch stream is domain-incremental, the task (recommend the next pick) fixed while the input distribution moves [31, 41, 47]. Its regularization branch, exemplified by elastic weight consolidation [20], anchors parameters against catastrophic forgetting, and its rehearsal branch keeps an explicit memory of past data [28, 35].

Shared-weight continual learning faces a fundamental problem in this setting: adapting to a new patch modifies parameters that may still encode knowledge useful for earlier patches, forcing different patches' adaptations to compete for the same parameter space. Rather than asking which shared parameters an update should touch, we ask whether the patch itself can determine which parameters are used and updated.

In a deployed MOBA system, patch identity is available as explicit game-version metadata. The active patch is therefore known at both training and inference, and does not have to be inferred from the interaction stream as a task label would.

Earlier patches remain worth retaining because several patch-specific distributions can be in play at once. Competitive events operate on a fixed game version, historical interactions are queried for retrospective analysis, and strategies established under an earlier patch can become relevant again after a later update. A deployed model must therefore serve multiple patch-specific distributions over its lifetime.

Maintaining a separate model per patch is an obvious zero-forgetting solution, but at substantially higher storage cost. This model zoo serves as the zero-forgetting anchor throughout the paper. The question a parameter-isolation architecture must answer is not whether it can reach zero forgetting, which the zoo also does, but what it costs to store relative to a zoo that reaches it trivially.

We propose *PatchExpertFFN*, a patch-routed mixture-of-experts architecture that assigns each patch a dedicated feed-forward expert while retaining a shared backbone. Routing is hard and keyed on patch identity, so a sequence associated with patch t is always processed by expert t and never by a later one. Each new expert is initialized from its predecessor rather than from the shared backbone, and the backbone is frozen

after patch 0. Because no later patch's training updates an earlier patch's expert, backward transfer is zero by construction, at a fraction of the zoo's per-patch storage.

We make the following contributions:

1. **PatchExpertFFN: a patch-routed expert architecture.** We replace the feed-forward sublayer of every backbone layer with a library of per-patch experts, hard-routed by patch identity known at deployment; an optional per-patch output head extends the same routing to the classifier. No learned gate and no task-inference step are required.
2. **Storage against a zero-forgetting anchor.** We verify by weight-identity check that the implementation delivers the architecture's zero-backward-transfer property at every patch and seed, and quantify the trade-off this buys against a per-patch model zoo that reaches the same retention trivially: 63% of its per-patch storage for 98.7% of its mean plateau HR@10.
3. **Expert-drift instrumentation.** We introduce an expert-drift measure, the per-layer and aggregate displacement of a trained expert from its copy source, and use it to quantify how much of a patch's adaptation the feed-forward block absorbs beyond what its warm start already encodes.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 states the preliminaries and the problem formulation, and reports the displacement analysis that locates where patch adaptation lands. Section 4 defines PatchExpertFFN and its training schedule. Section 5 describes the experimental setup and Section 6 reports the results. Section 7 discusses the findings and their limits. Finally, Section 8 concludes the paper and outlines directions for future work.

2 Background and Related Work

2.1 Mixture-of-Experts and Parameter-Isolation Continual Learning

A separate branch of continual learning avoids forgetting by isolating parameters instead of regularizing or rehearsing them. Progressive Networks [36] instantiate a new column per task and freeze every earlier one, giving an explicit zero-forgetting guarantee at the cost of a column's full parameter count per task; routing to a column requires knowing the task at inference. Expert Gate [3] relaxes that requirement by learning a gating autoencoder per expert, so a new input is routed by reconstruction error instead of by a given task label, trading the parameter-isolation guarantee's precondition (a known task id) for an inferred one. Recent mixture-of-experts work for continual learning of pre-trained models follows the same pattern: MoTE [26] filters and fuses task-specific adapters at inference to resolve dimensional mismatch across tasks; a reframing of the pre-trained attention block itself as a mixture of experts motivates a learned residual gate over prefix-tuned experts [23]; SEED [37] selects a single expert per class by distributional similarity and fine-tunes only that expert; and MoE-Adapters [48] expands a library of LoRA experts behind a learned out-of-distribution router so that unseen inputs fall back to the frozen backbone. Outside the pre-trained-model setting, DriftMoE [5] co-trains a neural router with a pool of

incremental experts directly on a drifting stream, and a recent theoretical account [25] formalizes why a mixture of experts separates tasks better than a single shared model under continual training, and shows that the gate should stop updating once enough tasks have been seen, which is the limiting case our routing occupies from the start.

These methods differ in where the routing decision comes from. Almost all of them recover the identity from the input itself, whether by reconstruction error, distributional similarity, or a learned gate. Patch identity is neither inferred this way nor stipulated by a benchmark protocol; it is supplied by the environment's own versioning (Section 1). PatchExpertFFN routes directly on that signal.

2.2 In-Game Draft Recommendation

Recommending picks from historical matches is well studied. Logistic and tree-search models predict picks and outcomes directly from match archives [10, 39], rule-mined synergy and counter scores feed a win-probability network [16], and the Art of Drafting [9] combines Monte Carlo tree search with a learned evaluator for team-oriented recommendation. DraftRec [24] casts the problem as transformer sequence modeling over the ban/pick order, the same family our BERT4Rec backbone belongs to, and League of Legends systems add graph-based [8] and classical [6] variants. Sequential recommendation has also been applied elsewhere in the same game, to in-match item purchases [11]. Match outcome is predictable from drafted composition well before a match ends [2], which establishes the downstream value of drafting accuracy. These systems are trained on a fixed historical snapshot and evaluated under a stationarity assumption. None treats the patch as a distribution shift the model must track, and none measures a trained recommender's accuracy across a patch boundary.

2.3 Continual Learning for Recommenders Under Concept Drift

Recommender systems deployed over time face concept drift [13, 29], and a body of work adapts continual learning to this setting: incremental recommendation methods [47, 49] update a model as new interactions arrive, including designs that absorb the incoming flow by re-detecting communities in a user and item graph rather than by retraining the scoring model [15], drift-detection methods flag a shift once enough post-shift data accumulate [1, 21], and streaming approaches characterize how much of a distributional change a model should retain [4, 7, 12, 50]. Sequential recommenders also model shifting intent inside a single user's interaction history [44], but that shift is a property of one trajectory rather than a dated, global change to the item semantics that every user faces simultaneously. Across this literature, drift is a statistical property of the incoming data stream: unlabeled, of unknown onset, and detected, if at all, after the fact from the model's own degrading predictions. A balance patch instead has a labeled, developer-announced boundary at a known release timestamp. Methods designed to infer a boundary provide no mechanism for exploiting one that is given, and the mixture-of-experts thread above targets this case only through benchmark-stipulated task indices, which are artifacts of how a dataset was partitioned rather than labels a deployed system carries. Where the increments are

supplied rather than inferred, as in continual knowledge-graph embedding over successive snapshots [46], the response is still regularization or distillation over a shared parameter set, so retention is traded against plasticity rather than guaranteed.

This paper studies parameter isolation in a setting where two properties hold jointly. The drift events are real and labeled, and the identity of each event is given at deployment rather than inferred. A per-patch expert can therefore be routed on an authored key, and its retention guarantee verified directly against saved models.

2.4 Positioning

Table 1 places the three threads above against this work. *Boundary identity* is how a method knows which increment an input belongs to: inferred from the input, or given from outside the model. The *routing signal* column records what is actually routed on, since a benchmark task or snapshot index disappears when the model is deployed, whereas a patch identifier is attached to every match record the game emits. *Isolation and retention* is exact when it is structural and verifiable against saved weights, and approximate when it holds only up to a router's errors or a shared trunk. *Per-increment cost* decides whether exact isolation survives a long stream.

One prior row, Progressive Nets, also pairs a given boundary with exact retention, but under conditions that do not transfer: the identity is a benchmark partition index rather than a label a deployed system carries, and the isolation costs a full network column per increment. The combination this paper occupies, an environment-supplied boundary with structural retention at sublayer cost, is unoccupied.

3 Preliminaries and Problem Formulation

This section defines balance patches as the source of drift, formalizes drafting as sequential recommendation, states the resulting continual-learning problem and its evaluation metrics, describes the dataset, and measures where patch adaptation lands inside the backbone, which determines where the architecture of Section 4 places its adaptable parameters. Throughout, f_θ denotes the backbone and θ its learnable parameters.

3.1 Dota 2 Balance Patches and the Competitive Meta

Dota 2 receives periodic balance *patches* $P = \{P_0, \ldots, P_K\}$, with $K = 4$ for this paper's stream, giving five patches and four transitions. Each patch is an authored revision of hero base statistics, ability values, item costs, or hero interactions, published as a dated release with an exact version number. A patch is a discrete, developer-announced event with a known boundary rather than a gradual trend, and every match played after it is tagged with the version under which it was drafted. This paper's stream (Section 3.4) covers five consecutive DOTA-Draft version ids, 54–58, corresponding to Dota 2 patches 7.35–7.39. Each revises enough heroes that the drafting meta measurably shifts, as documented by the widening or narrowing of pick and win rates across the boundary [32].

Table 1 Prior work and PatchExpertFFN on three axes. Column definitions are given in the text. MOBA draft recommenders do not represent increments at all. Mixture-of-experts continual learning mostly infers the boundary; the one method handed an exact boundary, Progressive Nets, reads it from a benchmark partition and pays a full network column per task. Drift-aware recommenders that see a labeled boundary still retain by regularizing or distilling shared parameters. Only the last row combines a boundary supplied by the deployed environment with structural retention at sublayer cost.

Work	**Boundary identity**	**Routing signal**	**Isolation / retention**	**Per-increment cost**
MOBA draft recommenders				
Conley & Perry [10]	None	n/a	None	n/a
Summerville et al. [39]	None	n/a	None	n/a
Hanke & Chaimowicz [16]	None	n/a	None	n/a
Art of Drafting [9]	None	n/a	None	n/a
DraftRec [24]	None	n/a	None	n/a
MOBARec-GCNFP [8]	None	n/a	None	n/a
Błaszczyk & Szajerman [6]	None	n/a	None	n/a
Parameter-isolation / mixture-of-experts CL				
Progressive Nets [36]	Given	Benchmark task index	Exact	Network column
Expert Gate [3]	Inferred	Reconstruction error	Approx.	Expert model
MoTE [26]	Inferred	Expert filtering and fusion	Approx.	Adapter set
MoE-Prompt-CL [23]	Inferred	Learned residual gate	Approx.	Prefix-tuned expert
SEED [37]	Inferred	Distributional similarity	Approx.	Expert branch
MoE-Adapters [48]	Inferred	Out-of-distribution router	Approx.	LoRA adapter
DriftMoE [5]	Inferred	Co-trained router	Approx.	Incremental expert
Drift-aware recommenders (shared parameters)				
Continual RecSys, survey [47]	Inferred	Post-hoc detection	None	Shared
Incremental RS, survey [49]	Inferred	Post-hoc detection	None	Shared
Dynamic trust/graph RS [15]	None	Continuous update	None	Shared
Drift detection + replay [21]	Inferred	Drift detector	None	Shared
Continual KGE, snapshots [46]	Given	Benchmark snapshot index	None	Shared
PatchExpertFFN (ours)	**Given**	**Environment version identifier**	**Exact**	**FFN sublayer and head**

Two properties follow. First, the boundary is dated and labeled rather than inferred, unlike the concept-drift setting most continual-learning methods assume (Section 2). Second, the patch a sequence was drafted under is available to the system at inference, since a client reports its own version at connection time. The second property is what the architecture of Section 4 routes on.

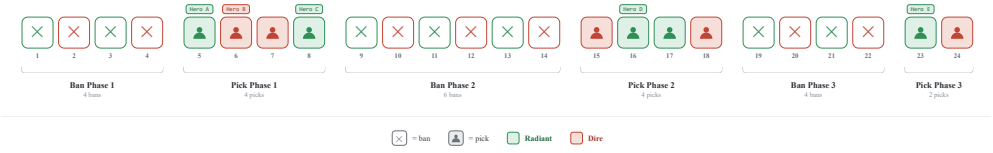


Fig. 1 Captain's Mode drafting comprises a 24-turn alternating ban/pick sequence (14 bans, 10 picks) over three phases. Team A (first pick) opens the draft and makes the first pick (turn 8), while Team B (second pick) receives an extra opening ban and the final pick (turn 24). The candidate pool shrinks after every ban and pick.

3.2 MOBA Drafting as Sequential Recommendation

We study the drafting phase of Dota 2 in *Captain's Mode*, the standard competitive format [32]. Two teams (Radiant and Dire) alternate *ban* phases, removing a hero from the shared pool, and *pick* phases, committing a hero to a line-up. Both teams see the full ban/pick history, so every choice is conditioned on all prior actions, and Captain's Mode fixes this as a $T = 24$ step alternating sequence (14 bans, 10 picks; Figure 1) whose candidate pool shrinks with each action. Strong synergy and counter dependencies make drafting a sequential decision problem. Formally, a match is a fixed-length sequence

$$m = (h_1, \ldots, h_T), \qquad h_t \in H, \tag{1}$$

where H is the hero vocabulary. The output table has a fixed $|H| = 127$ rows, the 126 heroes appearing anywhere in the stream plus one reserved index; heroes introduced by a later patch occupy rows that receive no gradient until that patch is first trained on (Section 3.4). At step t the draft state is the prefix $s_t = (h_1, \ldots, h_{t-1})$, and draft recommendation models the conditional $p(h_t \mid s_t)$.

To capture the bidirectional, high-order dependencies among picks and bans we adopt a masked bidirectional Transformer f_θ following BERT4Rec [40]. A subset of positions $M_{\text{mask}} \subset \{1, \ldots, T\}$ is masked to form the corrupted sequence $\tilde{m}$, and f_θ minimizes the masked negative log-likelihood

$$\mathcal{L}_{\text{rec}} = -\sum_{t \in M_{\text{mask}}} \log p\big(h_t = h_t^\star \mid \tilde{m}; \theta\big), \tag{2}$$

in which $h_t^\star$ is the ground-truth hero at masked position t, so the model is trained to recover each masked choice from the full surrounding draft context rather than from a left-to-right prefix alone. The per-position distribution over candidate heroes is

$$p(h_t = v \mid \tilde{m}) = \big[\text{softmax}\big(W x_t^L + b\big)\big]_v, \tag{3}$$

where $x_t^L \in \mathbb{R}^d$ is the final-layer state at position t, d the hidden dimension, $W \in \mathbb{R}^{|H| \times d}$ the output projection, and b a bias. Equation (3) defines the output head that the architecture of Section 4 optionally replicates per patch.

3.3 Continual Learning under Patch-Induced Drift: Problem Formulation

Adaptation across patches is a *domain-incremental* continual-learning problem [41]: the objective (predict the next pick) is constant while the input distribution drifts from patch to patch. Each patch P_k contributes one experience D_k, the matches played under it, and the learner sees the experiences in order $\{D_0, D_1, \ldots, D_K\}$ (labeled Exp000–Exp004 in this paper), with

$$D_k \sim p_k, \qquad p_k \neq p_{k-1} \quad (k \geq 1). \tag{4}$$

Training proceeds over $k = 0, \ldots, K$ with no access to later patches, so the environment is non-stationary by construction. This exposes the *stability–plasticity* tension [45]: f_θ must take on each new patch (plasticity) yet retain what earlier patches established (stability). Both sides are quantified from the matrix R of per-patch held-out scores, where $R_{i,j}$ is the score on experience j after training through experience i, following Lopez-Paz and Ranzato [28]. *Backward transfer* is the mean change on earlier experiences between the score attained immediately after training on them and the score retained at the end of the stream,

$$\mathrm{BWT} = \frac{1}{K} \sum_{j=0}^{K-1} \big(R_{K,j} - R_{j,j}\big), \tag{5}$$

with negative values indicating forgetting.

Held-out ranking quality is measured with HitRate@10 and NDCG@10 [18] on the masked-target task, averaged over all held-out mask positions in an experience's test split (Appendix E). Both are top-N cut-off metrics, the standard form of recommender evaluation, reported here at the conventional $N = 10$ [15]. Plasticity after a patch boundary is reported as a *restoration-time* pair, a *plateau* and a *speed* [38], defined in full in Appendix F.

3.4 Dataset: DOTA-Draft

All experiments use DOTA-Draft [32], a public, RecBole-compatible [52] benchmark of professional Dota 2 pro-league matches recorded as per-turn draft interactions. The January 2026 release[1] holds 52,905 matches and 1,269,720 interactions over the five balance versions of Section 3.1, across 126 heroes (Table 2). Every record carries a patch tag, which yields the stream $\{D_0, \ldots, D_4\}$ of Section 3.3 directly. Exp000 is trained once and the resulting model shared, so every arm of Section 6 enters the first drift from an identical state.

3.5 Where Patch Adaptation Lands

Which block to replicate per patch depends on where adaptation lands inside f_θ. Distance travelled in parameter space is an established probe of what fine-tuning

[1] Obtained from the public repository of [32], https://github.com/DeusExMachina1993/Dota-Draft.

Table 2 Composition of the five-patch stream. Each row gives the Dota 2 version, the date range over which matches carrying that version tag occur, and the resulting match and interaction counts. Every patch is divided 80/10/10 into training, validation, and test partitions. A date range can extend a few days past the following release because client updates roll out gradually rather than at a single instant.

Exp.	Version	Coverage (from–to)		Matches	Interactions
Exp000	7.35	2024-01-01	2024-05-23	11,920	286,080
Exp001	7.36	2024-05-23	2024-08-05	6,057	145,368
Exp002	7.37	2024-08-05	2024-12-31	10,707	256,968
Exp003	7.38	2025-02-19	2025-05-22	6,912	165,888
Exp004	7.39	2025-05-22	2025-12-16	17,309	415,416
Total				52,905	1,269,720

alters [33] and of which parts of a network drive forgetting under sequential training [34]. For a parameter group c and a transition $t : P_{t-1} \to P_t$, the *relative displacement* of the sequential fine-tuning state is

$$\Delta_c^t = \frac{\|\theta_c^t - \theta_c^{t-1}\|_2}{\|\theta_c^{t-1}\|_2}, \tag{6}$$

scale-free by construction, so groups of unequal size are comparable. Measurements are over three seeds and the three transitions for which consecutive sequential fine-tuning states were retained.

The feed-forward sublayers and the attention stack move by indistinguishable amounts at every transition, their separation falling at or below the seed spread, and both exceed the per-hero embedding rows and the normalization layers throughout (Figure 2). Adaptation is therefore not localized to one sublayer type, and the per-hero embedding rows, the surface a patch note describes most directly, are not where a shared-weight model places it.

Within the attention stack, movement is dense. The documented expectation is the opposite: a minority of heads carries most of a trained transformer's function and the remainder can be pruned at little cost [30, 43], which would make head-level isolation the cheapest form of per-patch parameter isolation available. Patch adaptation does not distribute that way here. Equation (6) applied at head granularity, with θ_h the head's query, key, and value row blocks of the input projection together with its column block of the output projection, gives a participation ratio [14] of 0.995 ± 0.001 over the 40 heads and a Gini coefficient [17] of 0.040 ± 0.006, against 1 and 0 under perfect uniformity (Figure 3). A mild gradient toward the earliest layers concentrates movement nowhere targetable.

Displacement records the trajectory sequential fine-tuning happened to take, not the minimal subset required to fit the incoming patch: a loosely constrained group can move far without mattering, and a group can move little while remaining necessary. Adam's per-parameter step normalization also flattens relative displacement, so the head-level uniformity holds for the optimizer actually used rather than reporting an

intrinsic property of the heads. That the same measure separates the normalization layers from the two sublayer types by roughly a factor of three indicates the flattening is partial rather than total. What the measurement establishes is negative. No sparse head subset exists, which removes head selection as an option, and no sublayer type is singled out, so it does not by itself favor any block. The choice of the feed-forward sublayer rests on two other grounds. It is a self-contained per-layer block, so it can be replicated per patch while leaving the shared trunk intact, and at an equal single-sublayer budget it is the more productive block to leave trainable (Section 6.2). Whether the resulting frozen trunk is adequate is settled empirically in Section 6 rather than assumed here. Section 4 builds on this block.

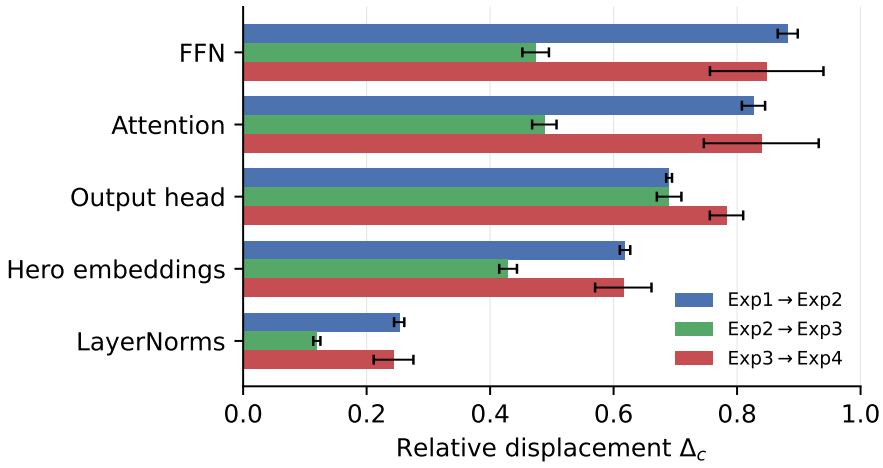


Fig. 2 Relative displacement by component (Equation (6)) under sequential fine-tuning, mean ± std over three seeds, per patch transition. The feed-forward sublayers and the attention stack are indistinguishable at every transition and exceed the per-hero embedding rows throughout; the normalization layers move least. Non-trainable groups (the fixed context embeddings) are omitted, their displacement being exactly zero.

4 Patch-Routed Expert Isolation

Section 3.5 establishes that the feed-forward sublayers and the attention stack absorb a patch's adaptation in comparable measure. The feed-forward sublayer is selected as the isolated block for two reasons: it is a self-contained per-layer module, so per-patch copies replace it without altering the shared trunk, and of the two it retains more of the fully-trainable model's plasticity when left trainable alone (Section 6.2). PatchExpertFFN accordingly replaces every backbone layer's feed-forward sublayer with a library of per-patch experts, hard-routed by patch identity and paired with an optional per-patch output head.

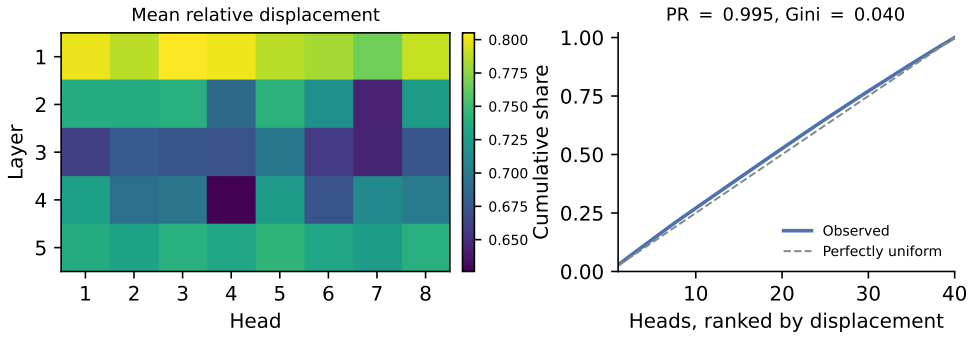


Fig. 3 Per-head relative displacement of attention parameters (Equation (6)) under sequential fine-tuning, mean over three seeds and the transitions Exp1→Exp2, Exp2→Exp3, and Exp3→Exp4. Left: displacement by layer and head. Right: cumulative share over heads ranked by displacement, against the perfectly uniform reference. Movement is dense across the 40 heads, with a mild gradient toward the earliest layers. No sparse head subset absorbs the attention stack's adaptation.

4.1 Architecture

Figure 4 shows the architecture. The embeddings and, within each backbone layer, the self-attention sublayer and its norm form the *shared trunk*, whose identical parameters serve every patch, trained once at patch 0 and frozen afterward. The feed-forward sublayer and its norm are *routed*, with a library holding one complete copy per patch and the routing switch $r(t)$ selecting which copy processes a given sequence. The output head follows the same pattern, one classifier per patch behind the same switch. Figure 5 gives the resulting lifecycle across the five-patch stream, formalized in Section 4.2.

4.2 Formal Definition, Training Schedule, and Parameter Accounting

Let θ_{trunk} denote the parameters shared across patches (token and position embeddings, and, in every layer $l = 1, \ldots, 5$, the self-attention weights and the pre-attention norm norm_1), and let $E^k = \{E_l^k\}_{l=1}^5$ and H^k denote the feed-forward expert library and output head belonging to patch k. A layer's forward pass is

$$x \leftarrow x + SA_l(\text{norm}_1(x)), \tag{7}$$

$$x \leftarrow x + E_l^{r(t)}(x), \tag{8}$$

where the routed expert is

$$E_l^k(x) = \text{drop}\big(W_2^{l,k}\,\phi(W_1^{l,k}\,\text{norm}_2^{l,k}(x))\big), \tag{9}$$

with ϕ the feed-forward nonlinearity, drop dropout, and $\text{norm}_2^{l,k}$ the expert's own pre-norm, which belongs to E_l^k rather than to the trunk. The output head applies the

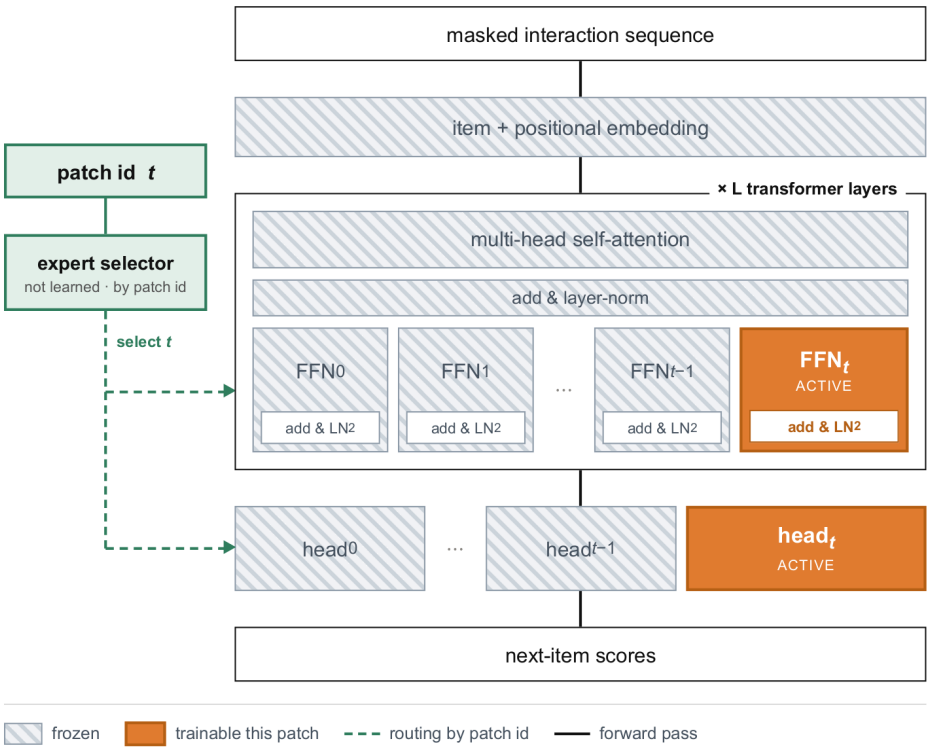


Fig. 4 PatchExpertFFN architecture. The item and positional embeddings and, within each of the L transformer layers, the multi-head self-attention sublayer and its add-and-layer-norm form the shared trunk, trained once at patch 0 and frozen afterward. The feed-forward sublayer is a library of per-patch experts $\{\mathrm{FFN}_0, \ldots, \mathrm{FFN}_t\}$, each with its own second layer-norm, and the output head follows the same routed pattern. The expert selector routes on patch identity t alone, with no learned gate: at patch t only FFN_t and head_t receive gradient updates, while every earlier expert and head remains frozen.

same routing,

$$\text{logits} = H^{r(t)}(x^L) = W_H^{r(t)}\, x^L + b_H^{r(t)}, \tag{10}$$

with x^L the final layer's state. Routing is hard and given rather than learned,

$$r(t) = t, \tag{11}$$

the patch a sequence was drafted under, available at inference as argued in Section 1. The training schedule differs at patch 0 from every later patch. At $t = 0$, θ_{trunk}, E^0, and H^0 are randomly initialized and trained jointly on D_0, as in a standard single-domain fit. For $t \geq 1$,

$$(E^t, H^t) \leftarrow \text{copy}(E^{t-1}, H^{t-1}), \tag{12}$$

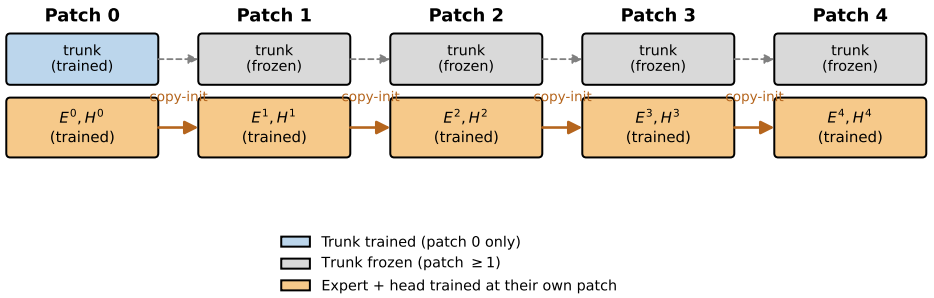


Fig. 5 Expert lifecycle across the patch stream. The trunk trains once, at patch 0, and is frozen for every later patch. Each patch's expert and head are copy-initialized from the immediately preceding patch's expert and head, then trained on that patch's data alone.

θ_{trunk} and every (E^j, H^j) for $j < t$ are frozen, and training minimizes $\mathcal{L}_{\text{rec}}$ (Section 3.2) over (E^t, H^t) alone on D_t.

Once patch j's training concludes, no later training step writes to the trunk, to E^j, or to H^j. Copy-initialization only reads an earlier patch's expert and head, and training at every later patch is restricted to that patch's own pair. The function applied to patch-j inputs under routing $r = j$ is therefore unchanged by any subsequent training, and backward transfer on patch j is zero by construction rather than by optimization. This is a design property, and Section 6 verifies that the implementation delivers it, by weight-identity check against saved models. A model zoo shares the same property for the same reason, since each saved model is never written to again, so Section 6 reports the architecture against that alternative rather than in place of it.

Parameter accounting. The quantity of interest is not the absolute size of any one model but the rate at which storage grows with the number of patches, since both PatchExpertFFN and the zero-forgetting model zoo grow linearly in K and differ only in their slope. A model zoo stores one full model per patch, a slope of 1; PatchExpertFFN stores one frozen trunk once plus one expert library and head per patch. At the configuration used here ($d = 128$, feed-forward width $4d$), the per-patch increment is a fixed fraction $\rho = 0.628$ of a full model, so the asymptotic storage ratio is ρ independently of K and of the backbone's absolute scale. Because ρ is set by the ratio of the routed sublayers to the whole architecture rather than by the parameter count itself, it is preserved under uniform width scaling: the same isolation applied to a backbone an order of magnitude larger stores the same fraction per patch, and the absolute saving grows with the backbone. Appendix C gives the per-component breakdown at this configuration.

Expert drift. Write $\text{vec}(E_l^t)$ for the concatenation of expert l's parameters at patch t flattened into a single vector ($W_1^{l,t}$, $W_2^{l,t}$, their biases, and $\text{norm}_2^{l,t}$'s scale and

shift), and $\mathrm{vec}(H^t)$ for the head's, including b_H^t. For $t \geq 1$, expert drift is

$$D_t^l = \left\| \mathrm{vec}(E_l^t) - \mathrm{vec}(E_l^{t-1}) \right\|_2, \quad D_t = \Big(\sum_{l=1}^{5} \big(D_t^l\big)^2 \Big)^{1/2}, \tag{13}$$

with the analogous quantity $\|\mathrm{vec}(H^t) - \mathrm{vec}(H^{t-1})\|_2$ for the head.

4.3 Routing Semantics and Implementation

The routing switch is set once per training or evaluation call and held fixed for its duration. Training routes through the experience currently being trained, $r = t$, while evaluation routes each experience through its own expert, $r = j$ (Section 5.2).

The architecture wraps the BERT4Rec backbone of Section 3.2, run as an Avalanche strategy [27] over the stream of Section 3.4; Appendices A and B give the training-loop hooks and the optimization configuration. Under degenerate routing, with the switch pinned to a single fixed expert for every patch and no copy-chain initialization, the architecture reproduces the sequential fine-tuning baseline of Section 6 exactly, confirming that it adds nothing beyond routing and copy-chain initialization.

5 Experimental Setup

5.1 Model and Training Configuration

All arms share the BERT4Rec backbone [40] and the optimizer settings of Appendix A. No arm uses a drift signal to steer training; routing is by patch identity alone (Section 4).

5.2 Evaluation Protocol

Metrics are reported per patch transition (Exp1–Exp4), Exp0 being a shared initial state rather than a drift transition. Retention is measured two ways for the routed arm. The *routed* measurement scores each past experience through its own expert and is the measurement the architecture makes a claim about. The *unrouted* measurement scores every past experience through the final patch's expert, isolating the retention the architecture would show with routing removed but the frozen trunk retained. Both are computed post-hoc from the saved per-patch models and cross-checked by weight-identity assertion; Appendix D gives the procedure.

5.3 Conditions

Core arms (Table 3) cover the full stream (Exp1–Exp4, resumed from a shared Exp0 model state), three seeds each. The model zoo is not separately trained, since its plateau and BWT follow analytically from Base's own per-patch diagonal and the definition of a never-retrained model (Section 6.1.2). *Shared-weight reference arms* (Table 6) cover the Exp0→Exp1 transition only, three seeds each. The degenerate-routing check of Section 4.3 is a correctness test on the implementation, not a reported table row.

A head-pinned PatchExpertFFN variant, with a shared rather than per-patch output head, falls short of the three-seed standard above and is left to future work (Section 8).

5.4 Metrics

HR@10, NDCG@10, and the (speed, plateau) restoration pair are defined in Section 3.3, which also defines backward transfer; expert drift is defined in Section 4.2.

5.5 Statistical Protocol

Every arm uses three seeds $(1, 2, 3)$, reported as mean ± standard deviation. A difference is treated as distinguishable from noise when the two arms' seed ranges do not overlap; any gap falling within a method's own seed spread is treated as noise.

6 Results

6.1 Core Arms: Plasticity, Retention, and Storage

Table 3 compares three arms over the full stream: sequential fine-tuning (*Base*), the zero-forgetting *model zoo* (Section 6.1.2), and *PatchExpertFFN+head*.

Table 3 Core arms, held-out test stream, mean ± std over 3 seeds. Plateau HR@10/NDCG@10 and recovery speed (post-patch training samples to 95% of a method's own plateau) per patch transition. BWT is a single value per run; PatchExpertFFN+head's is the routed measurement, verified against saved per-patch models (Section 6.1.1). Model-zoo plateau/speed are identical to Base's own diagonal by construction (Section 6.1.2); its BWT is 0 by construction, not measured.

Arm	Metric	Exp1	Exp2	Exp3	Exp4	BWT
Base (sequential FT)	HR@10	0.6665±0.0031	0.7088±0.0027	0.7062±0.0026	0.7173±0.0018	-0.2928 ± 0.0017
	NDCG@10	0.4409±0.0033	0.4816±0.0026	0.4812±0.0024	0.4973±0.0027	
	Recovery (samples)	190,179±11,645	219,469±12,000	119,125±8,277	366,264±84,869	
Model zoo	HR@10	= Base, per patch (Section 6.1.2)				0 (by construction)
	NDCG@10	= Base, per patch				
PatchExpertFFN+head	HR@10	0.6695±0.0019	0.6997±0.0034	0.6938±0.0023	0.6985±0.0008	0 (routed; verified)
	NDCG@10	0.4417±0.0018	0.4716±0.0023	0.4664±0.0038	0.4763±0.0014	
	Recovery (samples)	92,501±7,192	180,565±13,636	155,395±3,128	358,781±21,019	

Plasticity. The pattern is not uniform across the stream. At Exp1 the two arms' plateaus are indistinguishable, falling within combined seed noise, while PatchExpertFFN+head restores that plateau from roughly half as many post-patch samples, with disjoint seed ranges. From Exp2 on the plateau gap widens and reverses sign, each deficit exceeding the combined seed spread. The speed advantage persists at Exp2, again with disjoint ranges, then reverses at Exp3 and is indistinguishable at Exp4.

Read against the frozen trunk, these plateaus are a sufficiency result. The routed arm matches sequential fine-tuning at Exp1 and holds 97 to 99% of its plateau at Exp2–Exp4, so a trunk fitted at patch 0 and never updated carries enough of the representation for each later patch to be absorbed by its expert and head alone. Adequacy of the shared trunk is therefore measured rather than assumed (Section 3.5).

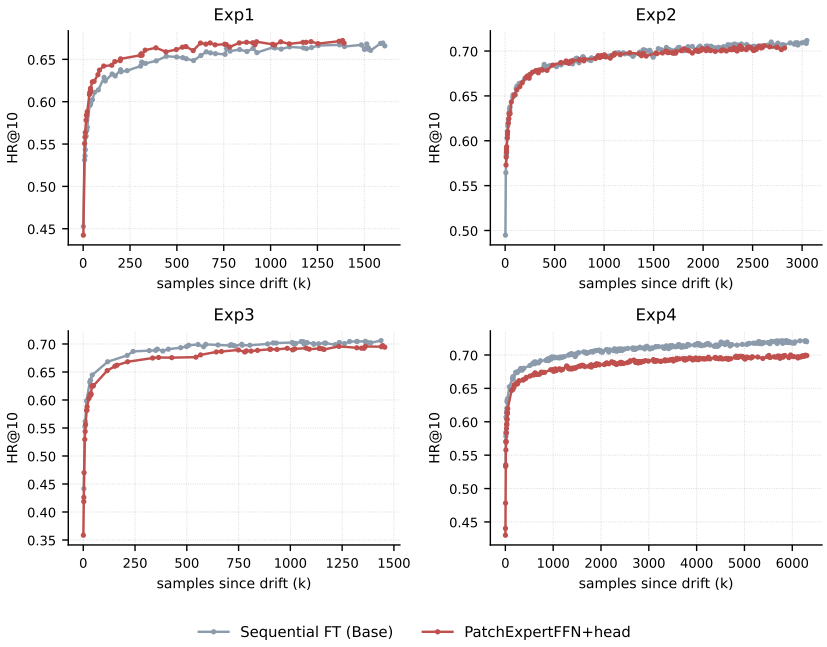


Fig. 6 Recovery curves (held-out HR@10 on the just-trained patch vs. training samples since drift), Base vs. PatchExpertFFN+head, seed 1 of 3, one panel per patch transition. The routed arm reaches its own plateau faster at Exp1 and Exp2; from Exp2 on it converges to a slightly lower plateau, and its speed advantage reverses at Exp3 (Section 7 discusses the growing plasticity tax).

The widening Exp2–Exp4 deficit is a scope condition rather than a defect of routing. Since the trunk is frozen at its Exp0-era state, each new expert absorbs both its own patch's shift and the growing distance between that fixed trunk and the current distribution, a load that grows with patch index (Section 6.1.3).

6.1.1 Backward Transfer: Routed vs. Unrouted

Table 4 reports the measurement the architecture makes a claim about. Verified by weight-identity check against the saved per-patch models, routed BWT is exactly zero at every patch and seed rather than approximately so, and the post-hoc pass reproduces the training-time diagonal of Table 3 to four decimal places.

The unrouted ablation isolates the contribution of freezing. Removing routing while retaining the frozen trunk yields no improvement over Base's retention, so freezing the 37% of the model that the trunk occupies recovers nothing on its own. The retention result is therefore produced by *routing*, not by *freezing*.

The unrouted per-patch delta has a recency gradient. Patch 3, the direct copy source of the final expert, is displaced about half as much as patches 0–2, which are one or more copy-chain hops removed. Forgetting through the copy chain, where routing does not prevent it, therefore grows with copy-chain distance, mirroring the expert-drift structure of Section 6.1.3.

Table 4 Backward transfer under routing vs. without it, PatchExpertFFN+head, mean ± std over 3 seeds. Both rows are computed post-hoc from the saved per-patch models in a single pass at the end of the stream. Routed: each past patch scored by its own expert/head (weight-identity verified against saved models). Unrouted: every past patch scored by the final (patch-4) expert/head.

	BWT	**Per-patch delta,** $j \in \{0, \ldots, 3\}$
Routed	0 (exact)	0, 0, 0, 0
Unrouted	-0.3057 ± 0.0003	-0.349, -0.350, -0.332, -0.192

6.1.2 Model Zoo: The Zero-Forgetting Anchor

A model zoo, one full model saved per patch and never retrained, shares PatchExpertFFN's zero-BWT property for the same reason (Section 4.2). Its per-patch plateau is identical by construction to Base's own diagonal, so no additional evaluation was run. What differs is storage, given in Table 5.

Table 5 Cumulative resident parameters after training through patch T. Sequential FT keeps one evolving model; the model zoo keeps one frozen copy per patch; PatchExpertFFN+head keeps one frozen trunk (401,024) plus one frozen expert+head (676,223) per patch. $N_{\text{model}} = 1{,}077{,}247$ (Appendix C). The figures are the storage a deployment retains, one expert being allocated per patch as that patch ships. The reference implementation instead allocates the expert library eagerly at construction, so its checkpoints hold the full 3,782,139 parameters at every T and match this column only at T=4.

T	Sequential FT	Model zoo	PatchExpertFFN+head
0	1.08M	1.08M	1.08M
1	1.08M	2.15M	1.75M
2	1.08M	3.23M	2.43M
3	1.08M	4.31M	3.11M
4	1.08M	5.39M	3.78M

The per-patch storage ratio is $\rho = 0.628$, which is also the asymptotic slope since both arms grow linearly beyond patch 0; the cumulative ratio at T=4 is looser because the trunk is paid for once regardless of T. Against the zero-forgetting anchor, the routed architecture gives up roughly one percentage point of mean plateau for a 37% reduction in per-patch storage (Figure 7). What transfers beyond this backbone is the slope rather than the absolute counts, since ρ is set by the routed sublayers' share of the architecture (Section 4.2), so the divergence widens without bound as a stream lengthens and scales with the backbone.

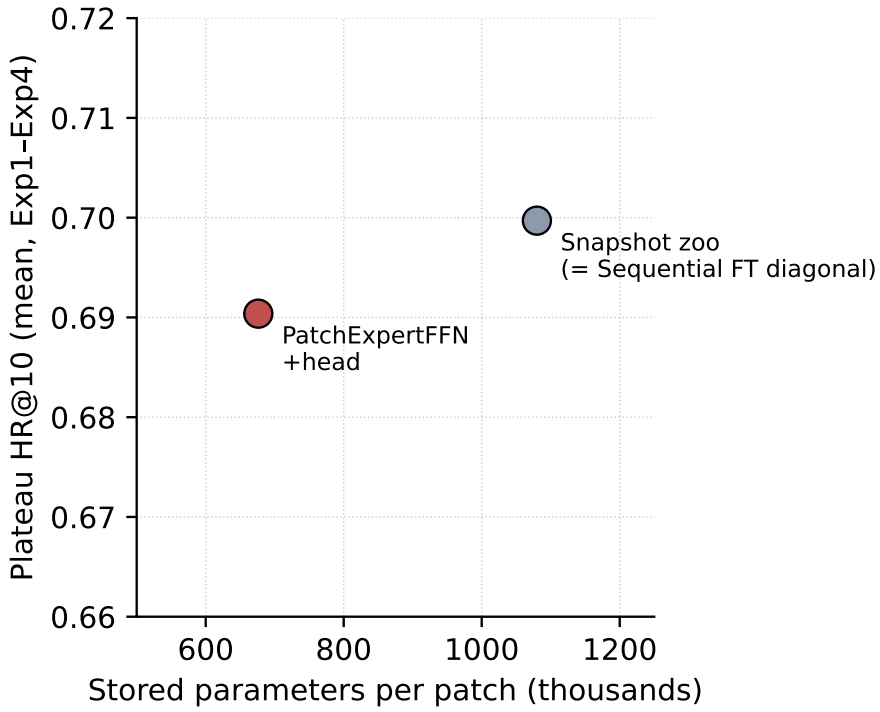


Fig. 7 Stored parameters per patch vs. mean plateau HR@10 (Exp1–Exp4), PatchExpertFFN+head against the model zoo.

6.1.3 Expert Drift

Total expert drift D_t grows over the stream, non-monotonically at Exp3 but net increasing, which is the evidence that each boundary carries a real and growing adaptation load. Figure 8 shows this growth is layer-uniform: per-layer drift at Exp4 varies little across the five layers relative to the spread from Exp1 to Exp4 within any single layer, so depth is not where the load concentrates. Head drift grows monotonically, and at Exp4 the feed-forward block's per-parameter RMS displacement is roughly twice the head's; Section 7 discusses what this does and does not establish about the head's role.

6.1.4 Cost Accounting Beyond Storage

Wall-clock training time is cost-neutral, differing by under 3% between the two arms with no consistent direction across matched seeds. Freezing the trunk skips its weight-gradient computation and optimizer step, but backpropagation still traverses the full graph and evaluation and data loading dominate at this model scale. The gain is therefore in samples rather than compute, a data-efficiency claim.

Training-time memory improves for the same reason storage does, since Adam holds two state tensors per *trainable* parameter, so a patch's optimizer state scales

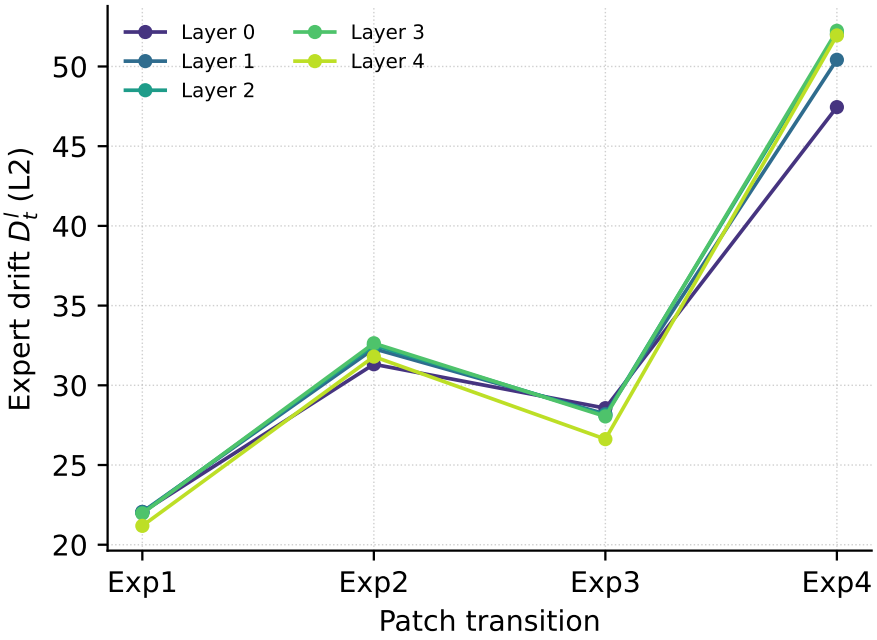


Fig. 8 Per-layer expert drift D_t^l (Eq. (13)), mean over 3 seeds, by patch transition. All 5 layers move together; no layer specializes in absorbing more of a patch's shift than the others.

with the per-patch footprint at ratio ρ. Inference cost is unaffected, since routing is an integer index rather than a learned gate and pays no gating-network overhead.

6.2 Supplementary Comparison: Shared-Weight Reference Arms

Table 6 reports a narrower comparison on the Exp0→Exp1 transition only; extension to the full stream is future work (Section 8). *FreezeExceptFFN* trains only the feed-forward sublayers, with attention frozen after Exp0, and *FreezeExceptAttn* is its complement. Neither reaches the plasticity of the fully-trainable arm, and the two are closer to each other than either is to it, consistent with feed-forward and attention absorbing a patch's adaptation in comparable measure (Section 3.5). The ordering between them is nonetheless asymmetric: at the same single-sublayer budget the feed-forward sublayer is the more productive one to leave trainable, which is the empirical basis for isolating it rather than the attention stack. The gap is small relative to the combined seed spreads and is read as directional support rather than a decisive margin.

These arms answer a narrower question than Table 3 does and should not be read as the same result. They compare two blocks at a matched budget under a single shared feed-forward sublayer and a single shared head, and so establish an ordering, not a sufficiency claim. Whether a frozen trunk suffices is settled by the routed arm of Table 3, which pairs the same frozen trunk with a per-patch expert and head and recovers the plateau to within 1 to 3%.

Table 6 Shared-weight reference arms, Exp0→Exp1 only, mean ± std over 3 seeds. Both are compared against the fully-trainable sequential fine-tuning arm of Table 3 at the same transition.

Strategy	HR@10	NDCG@10	Recovery (samples)
FreezeExceptFFN	0.6375±0.0062	0.4132±0.0056	181,987±24,690
FreezeExceptAttn	0.6282±0.0034	0.4081±0.0026	161,507±28,255

7 Discussion

What the head contributes. The head moves less per parameter than the expert (Section 6.1.3), but displacement magnitudes do not establish a functional division of labor, and no conclusion is drawn from them. Whether the per-patch head is *necessary* for the plasticity of Table 3 or merely *sufficient* alongside a shared head remains undetermined, since no head-pinned or head-only arm has been run at three seeds. A separate argument for a per-patch head is independent of the feed-forward result. The output projection faces heroes that first appear partway through the stream (Section 3.2), and so faces label shift the trunk does not.

The trade-off against the zero-forgetting anchor. No retention improvement over the zoo is available and none is claimed; the contribution is a storage argument at matched retention (Section 6.1.2). Because the guarantee follows structurally from the freezing and routing schedule rather than from optimization, the verification in Section 6.1.1 tests the implementation rather than the claim.

The plasticity tax and its scope. The plateau gap grows from indistinguishable at Exp1 to a widening deficit by Exp4. Because the trunk is fixed after patch 0, every later expert absorbs both its own patch's shift and the growing distance between that fixed trunk and the current distribution, a load that grows with patch index though not monotonically (Section 6.1.3). The design is therefore scoped to streams short relative to the interval over which the trunk stays adequate. The natural response is a maintenance policy rather than an architectural change: a periodic trunk refresh that retrains the shared trunk and restarts the expert chain, a deliberate and scheduled break of the guarantee. This policy is not evaluated here.

Limitations. An expert for a patch that has not yet arrived does not exist, so a client requesting a recommendation under an unseen patch has no defined routing target; Section 8 lists learned or soft routing for that case. Storage grows linearly in the number of patches by design (Table 5), which is the axis traded against the anchor's own linear growth rather than a cost unique to this method, with expert pruning or merging the mitigation. Evidence rests on one dataset, one game, one backbone, and five patches, at three seeds per arm with no formal significance test (Section 5.5), so the size of the plasticity tax should be read as specific to this stream even where its direction is not. The growing tax is attributed to a trunk fitted before the stream began, but a trunk fitted on one patch's data and a trunk that has aged relative to the current distribution are distinct failure modes with the same symptom here, and only the second is addressed by a refresh policy. Correspondingly, the adequacy of the frozen

trunk established in Section 6 is a sufficiency claim over the five patches observed, and the deficit's trajectory gives no basis for extrapolating it to longer streams.

Generality. The mechanism requires only that a task identity be available at deployment without inference, satisfied here by a game client reporting its own patch version and, more generally, by any system whose environment discretely and identifiably versions itself, such as software releases or catalog schema versions. It does not require the environment to document *what* changed, only *that* a new, identifiable era began.

8 Conclusion

This paper asked whether a MOBA game's own patch identity, a value a deployed client reports rather than infers, is a usable key for isolating a continual recommender's parameters by patch. PatchExpertFFN answers architecturally, with a per-patch library of feed-forward experts hard-routed by patch identity, warm-started from the preceding patch's expert, and a shared trunk frozen after patch 0. On a five-patch Dota 2 benchmark the architecture matches sequential fine-tuning's plateau at the first patch transition while restoring it from roughly half as many post-patch samples, at 63% of a zero-forgetting model zoo's per-patch storage for 98.7% of that zoo's mean plateau HR@10. Backward transfer is exactly zero, and the unrouted ablation attributes this to routing rather than to freezing. The plateau match does not extend past the first transition; the widening deficit over later patches is the cost of a trunk fixed after patch 0, which scopes the design to streams short relative to the trunk's useful lifetime.

Three extensions follow from where the evidence stops. A ladder of cheaper per-patch parameterizations, bias-only, head-only, and low-rank experts, would locate how much plasticity survives a smaller slot and settle whether the per-patch head is necessary or merely sufficient. Regressing expert drift on an external per-boundary severity signal, with patch index as a control, would separate boundary severity from trunk age, which the measures reported here confound. A scheduled trunk refresh that restarts the expert chain would trade a bounded, deliberate break of the retention guarantee against relief from the accumulating plasticity tax, at a cadence this stream is too short to determine. Routing for patches no expert has yet been trained for, and bounding storage growth by merging experts across similar patches remain open.

Declarations

Funding, competing interests, author contributions, and acknowledgements

These declarations identify the authors and are therefore supplied separately through the submission system, as required for double-anonymous peer review. They are omitted here and will be restored in the accepted version.

Data availability

The data analysed in this study are drawn entirely from DOTA-Draft, a previously published, publicly available benchmark of professional Dota 2 pro-league drafts [32]. That benchmark is distributed without access restriction from the repository https://github.com/DeusExMachina1993/Dota-Draft, and its dataset publication carries the persistent identifier https://doi.org/10.22044/jadm.2026.16888.2819. This study uses the January 2026 release, whose per-patch composition is reported in Table 2, and generated no new data: the five experiences are derived from that release by the deterministic splitting procedure given in Online Resource 1.

The full experimental configuration, the per-seed results behind every reported table, and the training and verification routines are given in Online Resource 1. The per-experience checkpoints of the sequential fine-tuning and routed arms (three seeds, four post-drift experiences), their parameter-level SHA-256 digests, and the per-step recovery records behind Figure 6 are available for review at https://anonymous.4open.science/r/supplementary-files-9FDD/, which permits the zero-forgetting verification of Online Resource 1 to be repeated without retraining. The source code is available from the corresponding author on reasonable request and is deposited under a permanent DOI on acceptance.

Ethics approval

Not applicable. This study uses publicly available match records containing no personally identifying information and involves no human participants or animal subjects.

Consent to participate

Not applicable.

Consent for publication

Not applicable.

Appendix A Architecture and Optimization

Every arm uses the same BERT4Rec backbone [40]: hidden dimension d_{model}=128, N_{layers}=5, feed-forward width $4d$=512, sequence length T=24 (the complete Captain's Mode draft), and a hero vocabulary of 127 entries (126 heroes plus one reserved index; Section 4.2). Optimization is Adam [19] with learning rate 0.001. Optimizer state is reset at every patch boundary for the routed and shared-weight arms; the sequential fine-tuning anchor carries its moment estimates across boundaries. The stream is built in Avalanche [27] on RecBole-derived data [52].

Appendix B Training Schedule

Algorithm 1 summarises the training procedure of Section 4.

Algorithm 1 PatchExpertFFN training over the patch stream.

Require: patch stream $D_0, \ldots, D_K$
Ensure: trunk θ_{trunk}; experts $\{E^t\}$; heads $\{H^t\}$
initialize $\theta_{\text{trunk}}, E^0, H^0$ at random
train $\theta_{\text{trunk}}, E^0, H^0$ jointly on D_0
freeze θ_{trunk}
for $t = 1$ **to** K **do**
    $(E^t, H^t) \leftarrow \text{copy}(E^{t-1}, H^{t-1})$ ▷ copy-chain initialization
    freeze (E^j, H^j) for all $j < t$
    train (E^t, H^t) on D_t, routing all inputs to expert t
end for

Appendix C Per-Patch Parameter Breakdown

With d=128 and feed-forward width $4d$=512, one layer's expert holds $W_1 \in \mathbb{R}^{d\times 4d}$, $W_2 \in \mathbb{R}^{4d\times d}$, their biases, and norm$_2$'s scale and shift, giving $2(d{\cdot}4d) + 4d + d + 2d =$ 131,968 parameters; five layers give 659,840 per patch. The head, over the 127-entry vocabulary, adds $d \times 127 + 127 =$ 16,383, so a patch's trainable and stored footprint is

$$659{,}840 + 16{,}383 = 676{,}223. \tag{C1}$$

The remainder of the model is the shared trunk. Within a layer the trunk is the self-attention block ($4(d^2{+}d) =$ 66,048) and its pre-norm ($2d = 256$), i.e. 66,304 per layer and 331,520 over five layers. Adding the five per-patch experts counted in (C1) (659,840) gives 198,272 per layer and 991,360 for the layer stack as a whole. The embedding stage is hierarchical: a static hero embedding and a patch-offset hero embedding, each over 128 rows (padding, 126 heroes, and the mask index), at $(126{+}2)d =$ 16,384 apiece, fused by a linear map $\mathbb{R}^{2d} \to \mathbb{R}^d$ contributing $2d^2{+}d =$ 32,896. Three fixed, non-trainable embeddings encode draft context: team and action ($2d = 256$ each) and draft order ($24d =$ 3,072). With the embedding norm (256) this gives 69,504 in the embedding stage. Adding the trunk layers and the head yields a full-model total of

$$N_{\text{model}} = 69{,}504 + 991{,}360 + 16{,}383 = 1{,}077{,}247, \tag{C2}$$

of which the frozen trunk is 401,024. The per-patch storage fraction is therefore $\rho = 676{,}223/1{,}077{,}247 = 0.628$. Of the trunk, 3,584 parameters are the fixed context embeddings, which never receive gradients in any arm.

Appendix D Evaluation Procedure

Held-out metrics are computed with the model in evaluation mode (dropout disabled), restored to training mode immediately after. The routed and unrouted retention measurements of Section 5.2 are single post-hoc passes over the per-patch models saved after each patch. The weight-identity assertion checks that the saved expert and head for patch j are bit-identical between the model taken right after patch j and the final model after patch 4, for every j and every seed.

Appendix E Ranking Metrics

For a held-out mask position with ground-truth hero $h^\star$ and top-10 ranked candidates $\hat{R}_{10}$ under f_θ,

$$\text{HR@10} = \mathbf{1}[h^\star \in \hat{R}_{10}], \tag{E3}$$

$$\text{NDCG@10} = \frac{\mathbf{1}[h^\star \in \hat{R}_{10}]}{\log_2(1 + \text{rank}(h^\star))}, \tag{E4}$$

each averaged over all held-out mask positions in an experience's test split [18].

Appendix F Restoration-Time Formalization

Section 3.3 defines the (speed, plateau) restoration pair informally; this is its precise form, instantiating the general restoration-time notion for non-stationary streams [13, 38]. For patch e, let $t_i = n_i - n_{\text{start}(e)}$ be the training samples elapsed since the patch boundary (n_i the cumulative samples trained). The held-out HR@10 curve is smoothed by a centered moving average of window w=3 (giving $\tilde{h}_i$); the plateau L_e is the mean $\tilde{h}_i$ over the last fraction ξ=0.25 of the drift-relative span, and the target is $\tau_e = \zeta L_e$ with ζ=0.95. Recovery time is the first t_i at which the smoothed curve reaches τ_e and holds it for p=3 consecutive evaluations,

$$\text{RT}_e = \min\left\{ t_i \;:\; \tilde{h}_{i+k} \geq \tau_e \text{ for all } k = 0, \ldots, p-1 \right\}, \tag{F5}$$

undefined if no such point exists. Table 3 reports this per patch transition ($e \in \{1, 2, 3, 4\}$; Exp0 excluded as initial convergence, not a drift). Because τ_e is relative to each run's own plateau, a lower-plateauing run reaches an easier target, which is why speed is always reported alongside its plateau.

References

[1] Agrahari S, Singh AK (2022) Concept drift detection in data stream mining: A literature review. Journal of King Saud University - Computer and Information Sciences 34(10):9523–9540. https://doi.org/10.1016/j.jksuci.2021.11.006

[2] Akhmedov K, Phan AH (2021) Machine learning models for Dota 2 outcomes prediction. URL https://arxiv.org/abs/2106.01782, arXiv:2106.01782

[3] Aljundi R, Chakravarty P, Tuytelaars T (2017) Expert gate: Lifelong learning with a network of experts. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp 3366–3375, arXiv:1611.06194

[4] Ashrafee A, Kozal J, Woźniak M, et al (2025) Holistic continual learning under concept drift with adaptive memory realignment. URL https://arxiv.org/abs/2507.02310, arXiv:2507.02310

[5] Aspis M, Cajas Ordoñez SA, Suárez-Cetrulo AL, et al (2025) DriftMoE: A mixture of experts approach to handle concept drifts. URL https://arxiv.org/abs/2507.18464, arXiv:2507.18464

[6] Błaszczyk KW, Szajerman D (2023) Champion recommendation in League of Legends using machine learning. In: Mikyška J, de Mulatier C, Paszynski M, et al (eds) Computational Science – ICCS 2023. Springer Nature Switzerland, Cham, pp 155–170

[7] Caroprese L, Pisani FS, Veloso BM, et al (2025) Modelling concept drift in dynamic data streams for recommender systems. ACM Transactions on Recommender Systems 3(2). https://doi.org/10.1145/3707693

[8] Chen C, Mo F, Fan X, et al (2023) MOBARec-GCNFP: Champion recommendation for multi-player online battle arena games using graph convolution network with fewer parameters. In: Proc. 8th IEEE Int. Conf. on Big Data Analytics (ICBDA), pp 147–153, https://doi.org/10.1109/ICBDA57405.2023.10104995

[9] Chen Z, Nguyen THD, Xu Y, et al (2018) The art of drafting: A team-oriented hero recommendation system for multiplayer online battle arena games. In: Proc. 12th ACM Conf. on Recommender Systems (RecSys), pp 200–208

[10] Conley K, Perry D (2013) How does he saw me? A recommendation engine for picking heroes in Dota 2. CS229 course report, Stanford University, URL https://api.semanticscholar.org/CorpusID:15859842

[11] Dallmann A, Kohlmann J, Zoller D, et al (2022) Sequential item recommendation in the MOBA game Dota 2. URL https://arxiv.org/abs/2201.08724, arXiv:2201.08724

[12] Du J, Zhang X, Shen H, et al (2024) Drift to remember. URL https://arxiv.org/abs/2409.13997, arXiv:2409.13997

[13] Gama J, Žliobaitė I, Bifet A, et al (2014) A survey on concept drift adaptation. ACM Computing Surveys 46(4). https://doi.org/10.1145/2523813

[14] Gao P, Trautmann E, Yu B, et al (2017) A theory of multineuronal dimensionality, dynamics and measurement. bioRxiv https://doi.org/10.1101/214262, URL https://www.biorxiv.org/content/10.1101/214262v2

[15] Hamidi H, Moradi R (2024) Design of a dynamic and robust recommender system based on item context, trust, rating matrix and rating time using social networks analysis. Journal of King Saud University - Computer and Information Sciences 36(2):101964. https://doi.org/10.1016/j.jksuci.2024.101964

[16] Hanke L, Chaimowicz L (2017) A recommender system for hero line-ups in MOBA games. In: Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital

Entertainment (AIIDE), pp 43–49, https://doi.org/10.1609/aiide.v13i1.12938

[17] Hurley N, Rickard S (2009) Comparing measures of sparsity. IEEE Transactions on Information Theory 55(10):4723–4741. https://doi.org/10.1109/TIT.2009.2027527

[18] Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4):422–446. https://doi.org/10.1145/582415.582418

[19] Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proc. Int. Conf. on Learning Representations (ICLR), arXiv:1412.6980

[20] Kirkpatrick J, Pascanu R, Rabinowitz N, et al (2017) Overcoming catastrophic forgetting in neural networks. Proc National Academy of Sciences (PNAS) 114(13):3521–3526. ArXiv:1612.00796

[21] Korycki Ł, Krawczyk B (2021) Class-incremental experience replay for continual learning under concept drift. In: Proc. IEEE/CVF CVPR Workshops

[22] Kritz J, Gaina R (2025) When 1+1 does not equal 2: Synergy in games. URL https://arxiv.org/abs/2502.10304, arXiv:2502.10304

[23] Le M, Nguyen A, Nguyen H, et al (2024) Mixture of experts meets prompt-based continual learning. In: Advances in Neural Information Processing Systems (NeurIPS), arXiv:2405.14124

[24] Lee H, Hwang D, Kim H, et al (2022) DraftRec: Personalized draft recommendation for winning in multi-player online battle arena games. In: Proc. ACM Web Conference 2022 (WWW), pp 3428–3439, https://doi.org/10.1145/3485447.3512278, arXiv:2204.12750

[25] Li H, Lin S, Duan L, et al (2025) Theory on mixture-of-experts in continual learning. In: Proc. Int. Conf. on Learning Representations (ICLR), arXiv:2406.16437

[26] Li L, Wu Z, Ji Y (2025) MoTE: Mixture of task-specific experts for pre-trained model-based class-incremental learning. URL https://arxiv.org/abs/2506.11038, arXiv:2506.11038

[27] Lomonaco V, Pellegrini L, Cossu A, et al (2021) Avalanche: An end-to-end library for continual learning. Proc IEEE/CVF CVPR Workshops (CLVision)

[28] Lopez-Paz D, Ranzato M (2017) Gradient episodic memory for continual learning. In: Advances in Neural Information Processing Systems (NeurIPS)

[29] Lu J, Liu A, Dong F, et al (2018) Learning under concept drift: A review. IEEE Transactions on Knowledge and Data Engineering 31(12):2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

[30] Michel P, Levy O, Neubig G (2019) Are sixteen heads really better than one? In: Advances in Neural Information Processing Systems (NeurIPS), URL https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html

[31] Mitchell R, Alliegro A, Camoriano R, et al (2025) Continual learning should move beyond incremental classification. URL https://arxiv.org/abs/2502.11927, arXiv:2502.11927

[32] Mohammadnejad M, Dorrigiv M, Yaghmaee F (2026) DOTA-Draft: A dataset for in-game recommendation in multiplayer online battle arenas. Journal of AI and Data Mining https://doi.org/10.22044/jadm.2026.16888.2819, URL https://jad.shahroodut.ac.ir/article_3801.html, code and data: https://github.com/DeusExMachina1993/Dota-Draft

[33] Neyshabur B, Sedghi H, Zhang C (2020) What is being transferred in transfer learning? In: Advances in Neural Information Processing Systems (NeurIPS), URL https://proceedings.neurips.cc/paper/2020/hash/0607f4c705595b911a4f3e7a127b44e0-Abstract.html, arXiv:2008.11687

[34] Ramasesh VV, Dyer E, Raghu M (2021) Anatomy of catastrophic forgetting: Hidden representations and task semantics. In: Proc. Int. Conf. on Learning Representations (ICLR), URL https://arxiv.org/abs/2007.07400, arXiv:2007.07400

[35] Rolnick D, Ahuja A, Schwarz J, et al (2019) Experience replay for continual learning. Advances in Neural Information Processing Systems (NeurIPS)

[36] Rusu AA, Rabinowitz NC, Desjardins G, et al (2016) Progressive neural networks. URL https://arxiv.org/abs/1606.04671, arXiv:1606.04671

[37] Rypeść G, Cygert S, Khan V, et al (2024) Divide and not forget: Ensemble of selectively trained experts in continual learning. In: Proc. Int. Conf. on Learning Representations (ICLR), arXiv:2401.10191

[38] Shaker A, Hüllermeier E (2015) Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. Neurocomputing 150:250–264. https://doi.org/10.1016/j.neucom.2014.09.076

[39] Summerville A, Cook M, Steenhuisen B (2016) Draft-analysis of the ancients: Predicting draft picks in DotA 2 using machine learning. In: Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), pp 100–106, https://doi.org/10.1609/aiide.v12i2.12899

[40] Sun F, Liu J, Wu J, et al (2019) BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In: Proc. 28th ACM Int. Conf. on Information and Knowledge Management (CIKM), pp 1441–1450

[41] van de Ven GM, Tuytelaars T, Tolias AS (2022) Three types of incremental learning. Nature Machine Intelligence 4:1185–1197

[42] Viggiato M, Bezemer CP (2020) Trouncing in Dota 2: An investigation of blowout matches. In: Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), pp 294–300, https://doi.org/10.1609/aiide.v16i1.7444

[43] Voita E, Talbot D, Moiseev F, et al (2019) Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In: Proc. 57th Annual Meeting of the Association for Computational Linguistics (ACL), pp 5797–5808, URL https://aclanthology.org/P19-1580/

[44] Wang C, Wang M, Wang X, et al (2024) IPSRM: An intent perceived sequential recommendation model. Journal of King Saud University - Computer and Information Sciences 36(9):102206. https://doi.org/10.1016/j.jksuci.2024.102206

[45] Wang L, Zhang X, Su H, et al (2024) A comprehensive survey of continual learning: Theory, method and application. IEEE Trans Pattern Analysis and Machine Intelligence (TPAMI)

[46] Wang M, Wang P, He M, et al (2026) TADG: Topology-aware and distillation-guided framework for continual knowledge graph embedding. Journal of King Saud University Computer and Information Sciences 38(6):329. https://doi.org/10.1007/s44443-026-00727-1

[47] Yoo H, Kang S, Tong H (2025) Continual recommender systems. URL https://arxiv.org/abs/2507.03861, arXiv:2507.03861

[48] Yu J, Zhuge Y, Zhang L, et al (2024) Boosting continual learning of vision-language models via mixture-of-experts adapters. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), arXiv:2403.11549

[49] Zhang P, Kim S (2023) A survey on incremental update for neural recommender systems. URL https://arxiv.org/abs/2303.02851, arXiv:2303.02851

[50] Zhang Ss, Liu Jw, Zuo X (2021) Adaptive online incremental learning for evolving data streams. Applied Soft Computing 105:107255. https://doi.org/10.1016/j.asoc.2021.107255

[51] Zhang Y, Feng F, Wang C, et al (2020) How to retrain a recommender system: A sequential meta-learning method. Proc ACM SIGIR

[52] Zhao WX, Mu S, Hou Y, et al (2021) RecBole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In: CIKM. ACM, pp 4653–4664

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- onlineresource1.pdf