

OneTrans-V2: Unifying Retrieval, Pre-rank, and Fine-rank with One Transformer in Industrial Recommender

ByteDance Global E-Commerce Recommendation Foundation Team

Abstract

Industrial recommendation systems typically operate as a *cascade* of retrieval, pre-rank, and fine-rank, but these stages are usually trained and served as separate models, causing repeated user-sequence encoding, isolated optimization, and duplicated engineering effort. Building on OneTrans’ model-level unification, we present OneTrans-V2, one Transformer that unifies the entire cascade. It encodes the user behavior sequence once as a shared context while preserving stage-specific candidate features and computation. Joint training lets the three stages reinforce one another and enables in-model knowledge distillation from fine-rank to pre-rank. We scale the shared backbone with sparse mixture-of-experts (MoE), which increases capacity with bounded activated computation, and stabilize scaling with μ P-style parameterization. To consolidate objective-specific retrieval channels, we introduce Decision-Conditioned Generative Retrieval (DCGR). DCGR predicts a decision prefix describing the upcoming interaction and generates items conditioned on it, allowing business objectives to steer a single generative process. Finally, Sequence-Native Training (SNT) organizes training around each user’s lifelong behavior sequence and amortizes its encoding across exposures. Deployed across all three stages of a large-scale industrial recommendation system, OneTrans-V2 improves gross merchandise value (GMV) by 9.74% and, with a co-designed serving stack, delivers 3.2 $\times$ the throughput of the cascade it replaces under the same hardware budget.

Date: September 25, 2026

1 Introduction

Modern industrial recommendation systems operate as a *cascade* to meet stringent latency constraints. Retrieval selects thousands of candidates from a billion-scale catalog, pre-rank reduces them to hundreds with inexpensive per-candidate scoring, and fine-rank produces the final order using richer candidate features, including user-item cross features. This progressive reduction makes billion-scale recommendation feasible within a few hundred milliseconds.

In conventional cascades, however, retrieval, pre-rank, and fine-rank are designed, trained, and served independently, and retrieval may further maintain multiple models for different business objectives. This fragmentation is costly in four ways. 1) Engineering effort is duplicated across stages. A new user feature, a longer behavior sequence, or a model-scaling improvement must be implemented, tuned, validated, and deployed separately for each stage. 2) The same user behavior sequence is repeatedly encoded by every stage, wasting substantial computation. 3) Stages optimized in isolation cannot directly share learned representations or supervision, which can lead to inconsistencies across the cascade. 4) Model capacity is fragmented across specialized models, limiting the ability to scale the cascade as a whole.

Individual ranking models once suffered from a similar fragmentation. They traditionally relied on specialized modules for feature interaction and behavior modeling. Our previous work, OneTrans [53], showed that these components can be unified within a single causal Transformer. However, this *model-level* unification remains confined to a single ranking stage, leaving retrieval, pre-rank, and fine-rank as separate models. An alternative route, represented by OneRec [23],

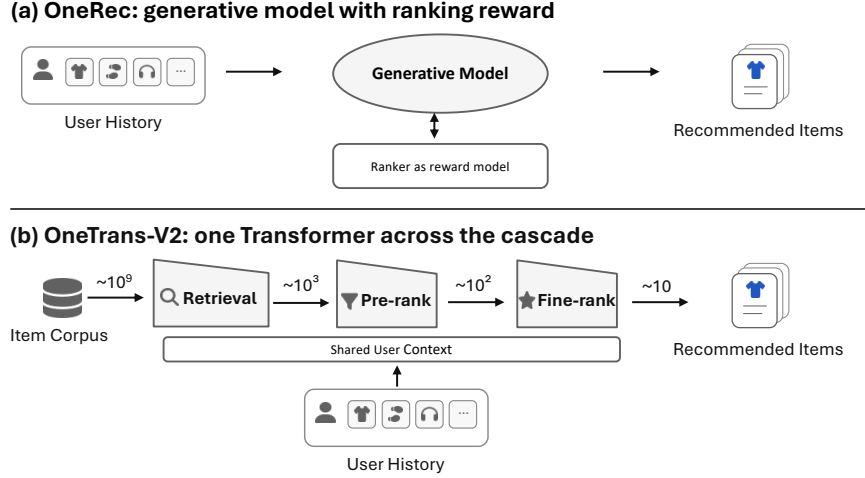


Figure 1 Two routes to unifying an industrial recommendation cascade. (a): End-to-end generative recommendation, exemplified by OneRec, replaces the cascade with one generative model and uses a ranker as its reward model. (b): OneTrans-V2 keeps the retrieval, pre-rank, and fine-rank cascade and its candidate-set reduction, but runs the three stages as tasks of one Transformer over a shared, cached user context.

eliminates the conventional cascade with an end-to-end generative model (Figure 1(a)). While this provides a direct path to architectural unification, industrial ranking stages rely heavily on rich candidate features. Removing these features can substantially degrade recommendation quality [13]. We therefore pursue a *cascade-preserving* approach (Figure 1(b)). Rather than eliminating the cascade, we retain each stage’s native candidate modeling while unifying the candidate-independent user context shared across stages.

Building on OneTrans, we present OneTrans-V2. It extends Transformer-based unification from a single ranking model to the whole recommendation cascade by representing retrieval, pre-rank, and fine-rank as three tasks of a single causal Transformer. Specifically, in OneTrans-V2 the candidate-independent user behavior sequence is encoded into a shared user context whose states are reused across stages. Each stage appends its own tokens to this context and uses token-specific parameters under OneTrans’s mixed parameterization to model its heterogeneous features. The behavior sequence is therefore encoded once per request rather than once per stage. The three stages are also trained jointly, allowing them to improve the shared user context rather than being optimized in isolation. A stage visibility mask (§4.1) controls how stage-specific tokens access the shared user context. Each stage can read the shared user context, but its tokens remain isolated from tokens of other stages and cannot modify it. This isolation allows each stage to choose its token budget independently according to its latency and feature requirements. Pre-rank remains lightweight with a single token over a small feature set, whereas fine-rank uses multiple tokens to model its richer candidate features.

Beyond computation reuse, unification pools the capacity previously dispersed across specialized models into a single backbone that can be scaled as a whole. Dense scaling, however, increases model capacity together with computation and memory traffic [4, 28]. We therefore scale the pooled backbone with sparse mixture-of-experts (MoE) [9, 35], decoupling total model capacity from activated computation and per-token memory traffic. As the backbone grows, we further adopt μ P-style parameterization to stabilize optimization across model scales. Unification also simplifies cross-stage knowledge distillation. Pre-rank and fine-rank are conventionally optimized as separate models and can produce inconsistent candidate orderings [46, 54]. Existing distillation methods [17, 29, 42] address this inconsistency but require a separate fine-rank teacher to be trained, scored, and refreshed independently. In OneTrans-V2, the two ranking stages are tasks of the same Transformer and therefore share the user context and update in the same training step. Fine-rank can therefore directly supervise pre-rank within the model, improving consistency between the two stages.

Retrieval, however, poses a distinct challenge to unification. In our production system, different business objectives call for different candidate sets. The main channel targets clicks and orders, while dedicated channels promote discovery or advertising. Retrieval has therefore evolved into multiple objective-specific channels, each with its own model and decoding process. The difficulty is that these channels differ not merely in what they predict, but in what they retrieve. In pre-rank and fine-rank, candidates are already given, so multiple objectives can share one forward pass through separate

prediction heads. In retrieval, the objective determines which candidates are generated in the first place. Keeping a separate decoding process for each objective would therefore preserve the very fragmentation we seek to remove. A unified retriever must instead express multiple business objectives within a single generation process.

We resolve this with Decision-Conditioned Generative Retrieval (DCGR). Instead of generating an item directly from the user context, DCGR introduces an intermediate decision step before item generation. The model first predicts a decision prefix that represents a user-conditional distribution over the upcoming interaction. The decision dimensions capture objective-related outcomes, such as whether the user converts, at what value, and how new the category is to the user. The model then generates the item’s semantic identifier (SID) conditioned on this prefix. In this way, decision-conditioned generation provides a compact, non-linguistic analogue of chain-of-thought reasoning. The model represents intermediate decisions as decision tokens rather than in natural language and uses them to guide subsequent item generation. This decision space also provides a common interface for business objectives. Instead of assigning each objective its own retrieval channel, we apply a business offset to steer the predicted decision distribution toward the desired objective. The offset changes how decision prefixes are prioritized while leaving the item-generation model shared across objectives. Its strength is exposed as an online parameter. The same model can therefore trade off different business objectives without retraining or adding a new retrieval channel.

Once the three stages share one backbone, encoding the long user behavior sequence becomes its dominant training cost. We therefore introduce Sequence-Native Training (SNT). SNT organizes exposures around each user’s lifelong behavior sequence and amortizes sequence encoding across the user’s exposures. Each exposure retains its own causal view of the shared behavior sequence while reusing the same user-side computation. This yields a 4.4× training speedup.

We deploy OneTrans-V2 across retrieval, pre-rank, and fine-rank in a large-scale industrial recommendation system. It improves gross merchandise value (GMV) by 9.74%. With a co-designed serving stack, OneTrans-V2 delivers 3.2× the queries per second (QPS) of the cascade it replaces under the same hardware budget.

In summary, our contributions are as follows:

- **Cascade-level One Transformer.** We formulate retrieval, pre-rank, and fine-rank as three tasks of one jointly trained Transformer, reusing the candidate-independent user context across stages while preserving each stage’s native candidate features, heterogeneous objectives, and serving budgets. This unification enables computation reuse, cross-stage knowledge transfer, and capacity scaling of the whole cascade within a single backbone. We show empirically that joint training improves all three stages over their separately trained counterparts, while distillation within the model improves consistency between pre-rank and fine-rank.
- **Objective consolidation in retrieval.** In our production system, retrieval remains split into objective-specific channels because it generates rather than scores candidates. We introduce DCGR, which predicts a user-conditional decision distribution from the behavior sequence and conditions identifier generation on the resulting decision prefix. Business objectives steer that distribution through a business offset, so one generative model covers multiple objectives without an external routing rule.
- **Production-efficient training, scaling, and validation.** We use sparse MoE to scale model capacity with bounded activated computation, adopt μ P-style parameterization to stabilize optimization across model scales, and introduce SNT to keep the unified backbone within a production training budget. We validate OneTrans-V2 in a large-scale industrial system, where it improves GMV by 9.74%. With a co-designed serving stack, OneTrans-V2 delivers 3.2× the end-to-end QPS of the cascade it replaces.

2 Related Work

2.1 Model-Level Unification in Ranking

Industrial ranking models traditionally combine specialized components for feature interaction and user behavior modeling, including feature-interaction architectures such as DeepFM [12] and DCN V2 [45], and behavior encoders such as DIN [55] and SIM [27]. Recent work increasingly integrates these functions within unified token-based or shared-backbone architectures. RankMixer [57] emphasizes hardware-efficient feature interaction through token mixing, while MTGR [13] incorporates rich candidate features into scalable sequence modeling. TokenFormer [56] and OneTrans [53] more explicitly unify heterogeneous feature interaction and user behavior modeling within a single

ranking model. Together, these works demonstrate the value of reducing architectural fragmentation within an individual ranking stage. OneTrans-V2 extends this direction from *model-level* unification within a stage to *cascade-level* unification across retrieval, pre-rank, and fine-rank.

2.2 From Cross-Stage Collaboration to Cascade-Level Unification

Cross-stage methods have progressively evolved from transferring information between independently optimized models toward directly sharing representations and computation across stages. Early approaches such as RankFlow [29] and cooperative retriever and ranker (CoRR) [17] improve cross-stage consistency through supervision propagation, adaptive sampling, and distillation, while largely retaining separate parameters, user-side computation, and serving paths.

OneRec [23] represents one route toward unifying the cascade. It reformulates recommendation around generative retrieval, where ranking is absorbed into the generation process through ranking-oriented supervision or policy signals rather than served as a separate online stage. Concurrent work such as Sona [37] further extends this direction toward retrieval, pre-rank, and fine-rank within a unified encoder-decoder architecture. Meanwhile, feature-rich industrial rankers such as MTGR [13] continue to rely on stage-specific candidate features, motivating designs that keep such heterogeneous features while sharing reusable computation.

A second route preserves the cascade while more tightly coupling retrieval and ranking. UniPinRec [20] jointly designs retrieval and lightweight ranking across representation, training, and serving, while OneRanker [39] and UniSGR [40] connect generative retrieval with downstream ranking through shared representations and ranking-oriented supervision. UniR² [50] further places user context, the SID generation trajectory, and ranking-side item feature tokens in a single heterogeneous decoder-only sequence. Its ranking query directly attends to the generation trajectory, which serves as a representation bridge from retrieval to ranking. Shared base attention parameters, ranking-specific FFNs, and low-rank adaptation (LoRA) balance representation sharing with task adaptation and optimization isolation.

OneTrans-V2 takes this second route but couples the stages through the shared user context and through a distillation loss between its two ranking stages. Unlike UniR², whose ranking query attends to the generation trajectory, the stage-specific tokens of OneTrans-V2 remain isolated across stages in the forward pass rather than consuming intermediate representations produced by another stage.

2.3 Decision-Conditioned Generative Retrieval

Generative retrieval represents items with semantic identifiers and autoregressively generates candidates conditioned on user context, as exemplified by TIGER [31] and OneRec [23]. Recent work further incorporates objective-related signals into generation. PinRec [2] conditions retrieval on desired outcome representations, while UniSGR [40] prepends learned task-aware tokens for click, add-to-cart, and purchase.

OneTrans-V2 differs in that DCGR does not take the objective as an input token. Instead, it predicts the decision dimensions from the user context and uses the inferred decision prefix to condition subsequent SID generation. In this way, DCGR jointly learns decision inference and decision-conditioned retrieval, allowing one generative model to serve multiple business objectives.

Steering what a generative model recommends has two approaches. One attaches a reward to what the model generates and optimizes it with reinforcement learning, as OneRec [23] does with a ranker serving as the reward model. The other writes the outcome into the sequence itself and trains the model by ordinary next-token prediction. Decision Transformer [3] and the outcome-conditioned methods that follow it [10, 33] take this second approach. They place the outcome of a sequence of actions in front of those actions, so the desired outcome can be written in at inference and the actions follow from it.

DCGR is the single-step case of that idea. A decision describes the one interaction at hand rather than the total reward collected over a long sequence of actions, so its labels are read directly from the log and nothing has to be estimated. Two costs of the first approach disappear with it. No reward model has to score what the model generates, and the logged samples need not be reweighted to account for having been produced by an earlier model rather than by the one being trained. Being single-step also addresses a known limitation of outcome conditioning. Given only sequences of actions that are individually poor, it cannot assemble a better one out of their good parts, as methods that plan several steps ahead can. DCGR predicts one decision and then the item, so there is no sequence to assemble and the limitation does not apply.

The closest prior method at inference is Multi-Game Decision Transformer [19], which does not need the desired outcome to be supplied either. It predicts a distribution over outcomes, shifts that distribution toward the better ones, and then generates an action. The business offset of §4.2.1 is that shift, applied to structured decisions rather than to a single number. That model draws one outcome from the shifted distribution and conditions on it, whereas DCGR scores every decision prefix. Each prefix carries its shifted score into one beam search over the decision and the SID codes together. A prefix that scores well on its own but leads to no good item is therefore outranked instead of being chosen.

2.4 Sequence-Centric Training and Compute Reuse

Recent work increasingly reorganizes recommendation training and inference around shared sequence or request contexts to reduce repeated encoding of the user behavior sequence. HSTU [51] formulates recommendation as sequential transduction, and its M-FALCON inference algorithm reuses cached history states across candidate microbatches and requests. Request-level batching (RLB) [11], SORT [44], and MTGR [13] amortize user-side computation by grouping the candidates of a request or a user around one shared context.

SNT applies this reuse principle to the unified cascade. Within each user window, exposures from different requests and cascade stages share one encoding of the user’s behavior sequence (§4.4).

3 Background

OneTrans [53] is a Transformer-based ranking model that jointly models sequential user behaviors and heterogeneous non-sequential features. It projects each behavior event into a sequential token, forming a behavior sequence $\mathcal{S}$, while non-sequential features are converted into a set of tokens $\mathcal{NS}$ through a dedicated tokenizer.

The key design of OneTrans is its *mixed parameterization*. Sequential tokens follow a homogeneous representation scheme and therefore share the same attention ($Q/K/V$) and feed-forward (FFN) parameters, whereas heterogeneous non-sequential tokens use *non-shared* (token-specific) $Q/K/V$ and FFN parameters. A causal attention mask places the non-sequential tokens after the behavior sequence, allowing them to attend to the entire preceding behavior sequence without violating causality. The resulting non-sequential token states are then fed into task heads to produce ranking predictions.

This decomposition separates candidate-independent behavior-sequence modeling from heterogeneous non-sequential feature interaction, providing the architectural basis for OneTrans-V2, since the former can be shared across cascade stages while the latter remains stage-specific.

4 OneTrans-V2

Conventional cascades cut along the stage boundary, so every stage has its own model and re-encodes the common user behavior sequence for itself. OneTrans-V2 (Figure 2) instead cuts between what repeats and what is stage-specific, encoding that sequence once into a single user context and letting every stage append its own tokens to it, so each stage keeps its native candidate-side modeling. OneTrans-V2 therefore preserves the cascade and its stage-specific modeling. Yet it needs only one jointly trained causal Transformer, whereas the conventional design needs a separate model for every stage and, within retrieval, for every business objective.

4.1 One Transformer across the Cascade

The inputs to OneTrans-V2 fall into two parts. The candidate-independent user behavior sequence $\mathcal{S}$ is shared across the cascade. The non-sequential feature sets $\mathcal{NS}_r$, $\mathcal{NS}_p$, and $\mathcal{NS}_f$ are specific to retrieval, pre-rank, and fine-rank; each is tokenized only by its own stage into tokens that follow the shared user context. Following OneTrans (§3), OneTrans-V2 applies *mixed parameterization* to the two token types within every Transformer block (TransBlock). The behavior tokens in $\mathcal{S}$ share one set of attention ($Q/K/V$) and FFN parameters, whereas the non-sequential tokens of each stage, which we call *stage-specific tokens*, use their own non-shared, token-specific $Q/K/V$ and FFN parameters. This allows all three stages to operate within one Transformer while preserving their heterogeneous feature representations; the internal design of each block is given in §4.3.

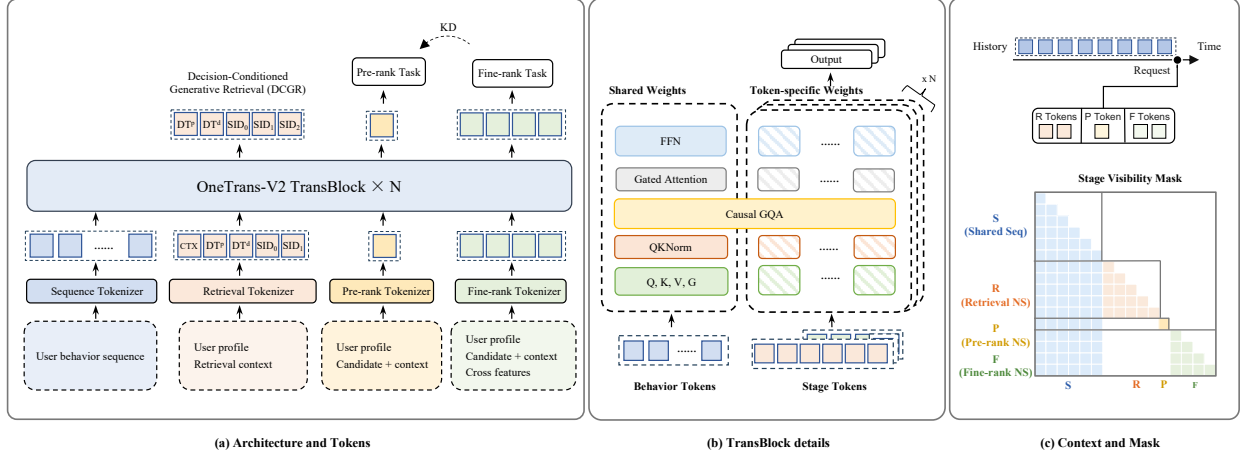


Figure 2 Architecture of OneTrans-V2, with N Transformer blocks. (a) Shared user behavior tokens and stage-specific non-sequential (NS) tokens are produced by separate tokenizers and processed by the unified backbone. DCGR uses the retrieval context token CTX, parallel and dependent decision tokens DT^p and DT^d , and three SID codes SID_0 – SID_2 to generate items. Pre-rank and fine-rank heads score candidates; the dashed KD arrow denotes knowledge distillation from the fine-rank teacher to the pre-rank student. (b) Behavior tokens share parameters, whereas stage-specific tokens use parameters specific to each token. Q , K , and V denote query, key, and value projections. G denotes the gate of gated attention. (c) S denotes the shared behavior sequence, and R , P , and F denote the retrieval, pre-rank, and fine-rank tokens, respectively. The request timestamp anchors the tokens of each stage to its visible behavior prefix. In the mask, rows are queries and columns are keys; colored cells permit attention and white cells mask it. Behavior tokens attend causally within S . Each stage-specific token attends to its visible behavior prefix, itself, and preceding tokens in the same stage and exposure; tokens from other stages or exposures are masked.

Shared Causal User Context. The user behavior sequence $\mathcal{S} = (s_1, \dots, s_L)$ of a user u , also written $\mathcal{S}_u$, lists behavior events in the order in which they arrive, with each new event appended to the sequence. To avoid future-information leakage, we follow OneTrans [53] and model $\mathcal{S}$ with a causal attention block, so that the hidden state at each position summarizes only the behaviors that precede it in the sequence.

Stage Visibility Mask. A *request* is one recommendation call from a user, served by all three stages in turn. The stage-specific tokens of an exposure belong to a request made at some timestamp. We *anchor* them at the last behavior that had arrived by that timestamp, so a stage-specific token attends to the behavior prefix up to its anchor and not to any behavior that arrives later. Within the same exposure and stage, stage-specific tokens further follow causal attention according to their predefined token order. Tokens from different stages or different exposures are mutually masked, and behavior tokens never attend to stage-specific tokens, so the user context can be encoded once and read by every stage. Although stage-specific tokens are isolated from one another, all three tasks jointly optimize the shared user context, allowing retrieval, pre-rank, and fine-rank to share user modeling without mixing their non-sequential features. How exposures from different requests share one encoded sequence under this mask is the subject of §4.4.

4.2 Task Formulations and Joint Optimization

Given the shared user context and stage-specific tokens described above, OneTrans-V2 jointly optimizes three heterogeneous tasks, namely autoregressive generation for retrieval and discriminative prediction for pre-rank and fine-rank.

4.2.1 Decision-Conditioned Generative Retrieval

In generative retrieval, the model does not score given candidates. Instead, it autoregressively generates item identifiers conditioned on the user context. To merge the objective-specific channels of §1 into one such model, we need two things. First, items need a structured identifier that can be generated. Second, the model needs an internal representation of the decision that each channel builds in, so that business objectives can steer it. Semantic identifiers provide the first, and Decision-Conditioned Generative Retrieval (DCGR) provides the second.

Semantic Identifiers. Following prior generative retrieval work [23, 31], we represent each item with a three-level semantic identifier (SID) (SID_0, SID_1, SID_2) obtained by applying residual-quantized k-means (RQ-KMeans) [22] to its multimodal embedding. The three codes form a coarse-to-fine hierarchy, and the model generates them autoregressively in that order. Retrieval is implemented as a beam search of width k over this code tree rather than as a scoring pass over a fixed candidate set. Because multiple items may share the same codes, every item mapped to a retrieved SID is returned.

Decision-Conditioned Generative Retrieval. Plain generative retrieval goes straight from the user context to the SID codes, with nothing observable in between. DCGR inserts an intermediate step. Before generating the SID codes, the model first predicts a few *decisions* about the upcoming interaction, namely its *purchase level*, *spending level*, *discovery level*, and *supply type* (whether the user will buy, at what value, how new the category is to the user, and whether the item is sponsored). These predictions form a *decision prefix*, a few decision tokens placed before the SID codes. Chain-of-thought prompting [47] shows how much a language model gains from intermediate steps, and DCGR adds such steps to retrieval, supervised by the log rather than generated freely. A *decision* here is an outcome that the model predicts, not a choice that the user makes explicitly. The four dimensions span a *decision space*, and each business objective corresponds to a region of this space rather than to a separate retrieval channel.

Consider a single request. A user arrives, and before generating any item the model predicts what kind of interaction is coming. It might predict a purchase at the middle spending level, a high discovery level, and organic supply. It then decodes the SID codes conditioned on that prediction, so the items it returns are the ones that fit an interaction of that shape. Had it predicted no purchase at a low discovery level, the same user context would have produced different items. This is why one model can serve several objectives. The objective is not a routing decision made outside the model, but a region of the prefix the model itself predicts, and a business offset moves the prediction toward that region.

Decisions come in two types. *Parallel decisions* depend only on the user context and are predicted at the same position. *Dependent decisions* also depend on earlier decisions and therefore take a later, *chained* position. The retrieval-specific features NS_r are summarized into a context token CTX that follows the shared behavior sequence S . Conditioned on $\mathcal{H} = (S, CTX)$, the model generates from the CTX position onward, one position after another,

$$[\underbrace{DT^P}_{\text{parallel decisions}}, \underbrace{DT^d}_{\text{dependent decisions}}, \underbrace{SID_0, SID_1, SID_2}_{\text{item}}]. \quad (1)$$

The four decision dimensions take their labels directly from the behavior logs.

Parallel decisions. Three dimensions can be predicted from the user context alone. *Purchase level* (z_{oc} , short for order count) says whether the interaction ends in no purchase, one purchase, or several. Because $z_{oc}=0$ already marks a non-converting interaction, no separate purchase indicator is needed. *Discovery level* (z_{disc} , short for discovery) says how new the candidate category is relative to the user’s recent behavior. *Supply type* (z_{ad} , short for advertising) says whether the item is organic or sponsored. Each of the three is predicted from the hidden state at the CTX position by its own lightweight decision head. Their decision embeddings are summed into one parallel decision token DT^P , which takes the next position. We sum rather than chain them for two reasons. Chaining would force us to pick a decoding order among dimensions that have no dependence to order them by. That choice is not neutral, since whichever dimension is decoded first is served best (§7.5). Summing also keeps the decision prefix short.

Dependent decisions. The only dependent decision in this paper is the *spending level* (z_{aov} , short for average order value), the value range of each order. We deliberately do not model transaction value as a single GMV label, because such a label cannot tell a basket of several cheap orders from one expensive purchase. With transaction value split into purchase level and spending level, the spending level is meaningful only once the purchase level is known. It is therefore predicted after DT^P and represented by the dependent decision token DT^d . The SID codes are then generated one by one, conditioned on the user context and the predicted decisions.

DCGR Objective. DCGR models the joint distribution of the decisions and the item as

$$P(\mathbf{z}, \text{Item} \mid \mathcal{H}) = \underbrace{P(\mathbf{z} \mid \mathcal{H})}_{\text{Decision Modeling}} \cdot \underbrace{P(\text{Item} \mid \mathcal{H}, \mathbf{z})}_{\text{Decision-Conditioned Retrieval}}, \quad (2)$$

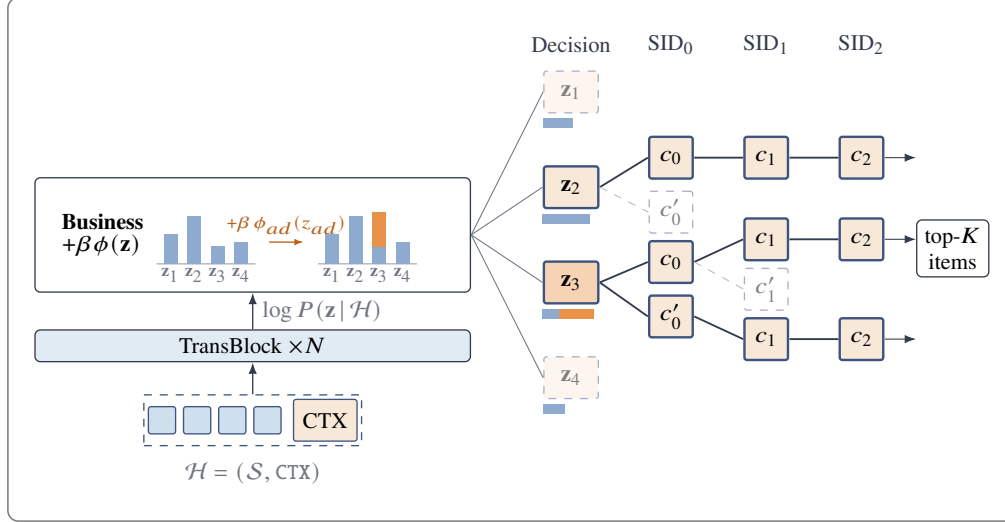


Figure 3 DCGR at inference. The model scores every combination of the decision dimensions $\mathbf{z}$ over the cached context $\mathcal{H}$, which is the behavior sequence $\mathcal{S}$ followed by the CTX token. A business offset $\beta \phi(\mathbf{z})$ is added to these decision prefix scores only. In the steering box, the offset $\beta \phi_{ad}(z_{ad})$ goes to prefixes that carry the sponsored value, here $\mathbf{z}_3$ (orange). The offset raises its score and with it the chance that the beam search keeps it, while $\beta=0$ would leave the model’s own ranking unchanged. Each prefix then carries its score into the beam search over the SID codes, which returns the top- K items to pre-rank. In the tree, c_ℓ is a code at level ℓ , a prime marks a competing code, and dashed nodes are pruned by the beam. Changing β online therefore changes which decisions the model acts on, without adding a retrieval channel.

where $\mathbf{z}^P = (z_{oc}, z_{disc}, z_{ad})$ denotes the parallel decisions and $\mathbf{z} = (\mathbf{z}^P, z_{aov})$ the complete decision prefix. With the two types of decisions organized as above,

$$\begin{aligned}
 P(\mathbf{z} | \mathcal{H}) &= \prod_{j \in \{oc, disc, ad\}} P(z_j | \mathcal{H}) \cdot P(z_{aov} | \mathcal{H}, \mathbf{z}^P), \\
 P(\text{Item} | \mathcal{H}, \mathbf{z}) &= \prod_{\ell=0}^2 P(\text{SID}_\ell | \mathcal{H}, \mathbf{z}, \text{SID}_{<\ell}).
 \end{aligned} \tag{3}$$

The first line predicts the decisions from the user context. The second line generates the item given the decisions. All decision and SID predictions are trained with cross-entropy in the autoregressive order of Eq. 1, with the ground-truth decisions supplied at their positions (teacher forcing). The resulting retrieval loss is $\mathcal{L}_r$.

Business Steering. Decoding is a beam search over the decision prefix and the SID codes, so which items are returned depends on how the decision prefixes score against one another. A business objective enters at decoding time by changing that ranking. Decoding proceeds in three steps. First, every valid combination $\mathbf{z}$ of the decisions is scored over the cached user context, which gives $\log P(\mathbf{z} | \mathcal{H})$ for every decision prefix (Figure 3). Second, a business offset $\beta \phi(\mathbf{z})$ is added to each of these scores. Third, beam search extends the prefixes into the SID codes, ranking each prefix together with its codes by

$$\underbrace{\log P(\mathbf{z} | \mathcal{H}) + \beta \phi(\mathbf{z})}_{\text{steered decision prefix}} + \sum_{\ell=0}^2 \log P(\text{SID}_\ell | \mathcal{H}, \mathbf{z}, \text{SID}_{<\ell}). \tag{4}$$

Here ϕ_j assigns an offset to each value of dimension j . A positive offset on the highest spending level, for instance, steers retrieval toward more valuable orders. $\phi(\mathbf{z})$ sums these offsets over the prefix, and the single scalar β sets how strong the whole offset is.

The offset is a soft preference rather than a rule. The first term of Eq. 4 corresponds to the distribution $q(\mathbf{z}) \propto P(\mathbf{z} | \mathcal{H}) e^{\beta \phi(\mathbf{z})}$, which is the solution of $\max_q \mathbb{E}_q[\phi(\mathbf{z})] - \frac{1}{\beta} \text{KL}(q \| P(\cdot | \mathcal{H}))$, the KL-regularized improvement step used

in offline reinforcement learning [26]. In words, the offset raises the expected business value of the decisions, a KL penalty keeps them close to what the model learned from the log, and β decides how far they may move. Multi-Game Decision Transformer [19] applies the same idea to a single outcome score. Because ϕ is a sum over dimensions, the exponential factors as $e^{\beta\phi(\mathbf{z})} = \prod_j e^{\beta\phi_j(z_j)}$, one factor per objective. Each business objective therefore receives its own offset on its own dimension instead of one global weight.

Three things follow. First, the offset applies only to the decision part of Eq. 4. It changes which decisions win, while the item term $P(\text{Item} \mid \mathcal{H}, \mathbf{z})$ of Eq. 2 is left alone, so under a given decision the model still returns the items it finds most likely. Second, $\beta=0$ gives the model’s own ranking, and a larger β moves further away from it, so the offset adjusts that ranking instead of overriding it. Third, β is set online. Steering toward an objective that the existing dimensions already express therefore needs no retraining. A new objective becomes a new decision dimension, which adds one lightweight decision head to the same model instead of a new retrieval channel with its own model. If the new dimension has no conditional dependence, it is summed into DT^P and the decision prefix does not get longer.

Guidance on the Item Side. The business offset decides which decision wins. The same design leaves room for a second control, which would decide how strongly the generated items follow that decision. Classifier-free guidance [16], a technique first developed for diffusion models and later applied to language models [32], can provide it. The idea is to score each SID code twice with the same model, once with the decision prefix and once without it, and to treat the difference between the two scores as the effect of the decision.

To make the second score possible, the decision prefix is replaced with a special null token $\emptyset$ on a small fraction of training samples, so that the model also learns to predict the SID codes when it is told nothing about the decision. At decoding time, each code would then be scored by

$$\log P(\text{SID}_\ell \mid \mathcal{H}, \emptyset, \text{SID}_{<\ell}) + w \left[\log P(\text{SID}_\ell \mid \mathcal{H}, \mathbf{z}, \text{SID}_{<\ell}) - \log P(\text{SID}_\ell \mid \mathcal{H}, \emptyset, \text{SID}_{<\ell}) \right]. \quad (5)$$

The bracket measures how much knowing the decision $\mathbf{z}$ changes the score of a code, and the weight w scales that change. With $w=1$ the score equals the ordinary item term of Eq. 4. With $w>1$ the effect of the decision is amplified, so the retrieved items follow it more closely. With $w<1$ the effect is weakened, and with $w=0$ the decision is ignored. Because both scores come from the same model, their difference is largely unaffected by how well either score is calibrated. Computing the second score requires a second pass of the decoder, but both passes read the same cached user context and differ only at the prefix and SID positions, so the extra work is confined to the beam search. Steering would then have two online parameters. The offset β decides which decision is made, and the weight w decides how closely the items follow it. The interventions of §7.4 suggest that the decoder responds to its prefix strongly enough for w to have an effect. This extension is not yet deployed, and we leave its evaluation to future work.

4.2.2 Discriminative Ranking Tasks

Pre-rank. Pre-rank scores every candidate that retrieval returns, so its per-candidate budget is the tightest in the cascade, and it can neither read the full non-sequential feature set nor afford many tokens for each candidate. We therefore restrict pre-rank to a lightweight feature subset $\mathcal{NS}_p \subset \mathcal{NS}_f$ and let its tokenizer compress the whole subset into a single candidate token. This token attends to the shared user context under the stage visibility mask. The pre-rank heads read out its resulting state and predict the pre-rank target set $\mathcal{T}_p$ of CTR and CVR, each optimized by a binary cross-entropy loss $\mathcal{L}_p$,

$$\mathcal{L}_p = - \sum_{t \in \mathcal{T}_p} [y_p^t \log \hat{y}_p^t + (1 - y_p^t) \log(1 - \hat{y}_p^t)], \quad (6)$$

where $y_p^t \in \{0, 1\}$ and $\hat{y}_p^t$ denote the label and the predicted probability of target t , respectively.

Fine-rank. Fine-rank scores only the candidates that survive pre-rank and determines the final order. Its main constraint therefore moves from per-candidate throughput to ranking fidelity, and pooling its rich candidate features into one token would erase the feature heterogeneity that this fidelity depends on. Following OneTrans [53], we let fine-rank consume the full non-sequential set $\mathcal{NS}_f$ and keep it as multiple tokens rather than the single token that pre-rank uses. These tokens attend to the shared user context under the same stage visibility mask, and their resulting states are read out by the

fine-rank heads, which predict the fine-rank targets. Unlike pre-rank, fine-rank predicts many more objectives; we focus on the core ones, CTR and CVR, denoted $\mathcal{T}_f$, again with a binary cross-entropy loss $\mathcal{L}_f$,

$$\mathcal{L}_f = - \sum_{t \in \mathcal{T}_f} \left[y_f^t \log \hat{y}_f^t + (1 - y_f^t) \log(1 - \hat{y}_f^t) \right], \quad (7)$$

where y_f^t and $\hat{y}_f^t$ are the label and the predicted probability of target t as in Eq. 6.

Knowledge Distillation. Sharing the user context gives pre-rank and fine-rank a common representation, but each stage still learns its own scoring function, and the two can order the same candidates differently. This is problematic, as a candidate that pre-rank discards can never be recovered by fine-rank, so every disagreement between the two stages permanently removes items that fine-rank would have ranked at the top. To address this, we adopt knowledge distillation [15] between the two ranking stages, with the fine-rank task as the teacher and the pre-rank task as the student. The stage with the richer candidate features thus supervises the ordering produced by the cheaper one. We follow the mean-centering step of logit standardization [41] but replace the standard deviation by a fixed temperature τ . For each pre-rank target $t \in \mathcal{T}_p$, the teacher logit o_t^T and the student logit o_t^S are centered by their respective means μ_t^T and μ_t^S , and the resulting student probability is trained against the teacher probability with a binary cross-entropy loss $\mathcal{L}_{kd}$ summed over the targets,

$$\begin{aligned} p_t^T &= \sigma\left(\frac{o_t^T - \mu_t^T}{\tau}\right), & p_t^S &= \sigma\left(\frac{o_t^S - \mu_t^S}{\tau}\right), \\ \mathcal{L}_{kd} &= - \sum_{t \in \mathcal{T}_p} \left[p_t^T \log p_t^S + (1 - p_t^T) \log(1 - p_t^S) \right], \end{aligned} \quad (8)$$

where σ is the sigmoid function. To prevent the distillation loss from affecting the teacher, we detach the teacher logit, allowing it to be shaped only by the fine-rank objective while the alignment moves the student alone. Because of the one-model design of OneTrans-V2, this distillation requires neither a separately trained teacher nor an offline distillation stage. We optimize $\mathcal{L}_{kd}$ together with the task losses in the same backward pass, so the student keeps improving as the teacher itself grows stronger.

4.2.3 Overall Objective

OneTrans-V2 is optimized via $\mathcal{L}$, a weighted sum of the three task objectives and the distillation loss $\mathcal{L}_{kd}$,

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_p \mathcal{L}_p + \lambda_f \mathcal{L}_f + \lambda_{kd} \mathcal{L}_{kd}, \quad (9)$$

where λ_r , λ_p , λ_f , and λ_{kd} control the relative contributions of the four objectives.

4.3 Scaling and Stabilization

Scaling OneTrans-V2 requires increasing model capacity while bounding activated computation and maintaining effective attention and stable optimization. To this end, we combine parameter-efficient scaling, stabilization, and optimization techniques, and revisit which input designs remain beneficial at scale.

TransBlock Design. Figure 2(b) expands the internal structure of a TransBlock. Each block follows the pre-norm layout, in which an RMSNorm [52] precedes the attention and FFN sub-layers and a residual connection follows each. The branch output is scaled by a residual multiplier $\gamma = 1/\sqrt{2N}$ for N blocks, so the residual stream stays bounded as the model deepens. In the attention sub-layer, queries and keys are RMS-normalized per head (QKNorm [14]), and gated attention [30] modulates the attention output before the output projection. The FFN sub-layer is a *sparse mixture-of-experts* (MoE), which routes each token to a small subset of experts. This grows the parameter count while keeping the activated compute per token roughly constant. Algorithm 1 summarizes the forward pass of a block, and we detail each component below.

Algorithm 1 TransBlock forward pass

Require: hidden states X , stage visibility mask $\mathcal{M}$, residual multiplier $\gamma = 1/\sqrt{2N}$

- 1: $\tilde{X} \leftarrow \text{RMSNorm}(X)$ ▷ pre-norm
 - 2: $Q, K, V, G \leftarrow \text{split}(\tilde{X} W^{QKVG})$ ▷ one fused projection
 - 3: $\tilde{Q}, \tilde{K} \leftarrow \text{RMSNorm}_{\text{head}}(Q), \text{RMSNorm}_{\text{head}}(K)$ ▷ QKNorm
 - 4: $A \leftarrow \text{Attn}(\tilde{Q}, \tilde{K}, V; \mathcal{M})$
 - 5: $X \leftarrow X + \gamma (A \odot \sigma(G)) W^O$ ▷ gated attention
 - 6: $\tilde{X} \leftarrow \text{RMSNorm}(X)$ ▷ pre-norm
 - 7: $X \leftarrow X + \gamma \text{MoE}(\tilde{X})$ ▷ sparse MoE
 - 8: **return** X
-

Grouped-Query Attention. On the attention side, we adopt grouped-query attention (GQA) [1]. The query heads are partitioned into groups, and every group shares a single key/value head, so a layer keeps far fewer key/value heads than query heads. The saving matters because the key/value cache of the shared behavior sequence is computed once and then read by the stage-specific tokens of all three tasks. During retrieval it is further broadcast across the beams of the decoder. Its size therefore sets both the memory footprint and the memory traffic of every cross-attention step. GQA shrinks this cache in proportion to the group size without loss in quality (§7.6). Only the number of key/value heads and the head dimension must agree between the stage-specific tokens and the shared behavior sequence. The two are therefore free to use different numbers of query heads over the same cached keys and values.

Sparse MoE. Following DeepSeekMoE [5], the FFN in OneTrans-V2 is a sparse MoE built from *fine-grained experts*, SwiGLU networks [34] with a narrow intermediate width. One *shared expert* is always active, and the rest form a set of routed experts. For each input, a lightweight router produces one score per routed expert, and a fixed number of the highest-scoring routed experts are selected. Their scores are renormalized to sum to one and multiplied by a fixed routed-expert scaling factor. The sub-layer output is the shared expert’s output plus the weighted sum of the selected routed experts. Inspired by DeepSeek-V3 [7], we adopt a sigmoid router rather than a softmax one. A softmax couples the experts, since raising the score of one necessarily lowers all the others. This pushes the router toward winner-take-all routing and ties the scale of every score to the total number of experts. A sigmoid scores each expert’s relevance to the input independently and on a fixed scale. Several experts can then be judged relevant at once, and the scores remain comparable as experts are added, which eases training at larger expert counts.

Gated Attention. We adopt gated attention [30], which applies a head-specific, input-dependent sigmoid gate to the attention output. The gate logits are produced from the normalized attention input by an extra slice of the fused QKVG projection. The softmax-weighted values of every attention head are multiplied element-wise by the sigmoid of these gate logits before the output projection. The gate therefore controls what each head writes back to the residual stream rather than the attention distribution itself. Sitting between the value and output projections, it adds a non-linearity that keeps the two from collapsing into a single linear map. It also yields input-dependent sparsity, so a head that is uninformative for the current token can be switched off instead of placing its attention mass on a sink position. This suppresses attention sinks and massive activations and improves training stability.

Stabilization Techniques. As the model grows in width and depth, training becomes harder. The scales of the activations, gradients, and per-step weight updates all drift with model size, so a configuration that is stable at one size can diverge at the next. Each scaling step would then call for a fresh hyperparameter search, at a scale where every trial is expensive. We therefore parameterize the backbone so that its training dynamics stay invariant as the model grows, following the maximal update parameterization ($\mu\mathbf{P}$) [48] for width and **Depth- $\mu\mathbf{P}$** [49] for depth. Both rest on one principle. Scaling changes how many terms each layer sums. The initialization, the multipliers, and the learning rate must therefore be chosen together. The goal is that every hidden activation and its per-step change stay at a scale that neither grows nor vanishes with model size, which we write as $\Theta(1)$. For width, consider a hidden layer $W\mathbf{x}$ with input $\mathbf{x} \in \mathbb{R}^d$ and weight matrix W , where d is the width. Fan-in initialization, *i.e.*, entries of W with variance $1/d$, already keeps $W\mathbf{x}$ at $\Theta(1)$, since its d terms are uncorrelated at initialization. An Adam step with learning rate η , however, changes every entry of W by roughly η . The resulting change $\Delta W \mathbf{x}$ sums d terms that are correlated with $\mathbf{x}$, so it grows as $\Theta(\eta d)$. $\mu\mathbf{P}$ cancels this growth by scaling the Adam learning rate of every hidden weight by $1/d$. For depth, let N be the number of blocks,

Hidden-layer quantity	Base model	Width rule (μ P)	Depth rule (Depth- μ P)
Initialization variance of W	$1/d_0$	$\times 1/m_d$ (i.e., $1/d$)	unchanged
Residual multiplier γ	$1/\sqrt{2N_0}$	unchanged	$\times 1/\sqrt{m_N}$ (i.e., $1/\sqrt{2N}$)
Adam learning rate	η_0	$\times 1/m_d$	$\times 1/\sqrt{m_N}$

Table 1 μ P and Depth- μ P rules for the hidden weights of the backbone, relative to a base model of width d_0 and depth N_0 . The two rules compose multiplicatively, e.g., the hidden learning rate becomes $\eta_0/(m_d\sqrt{m_N})$.

each adding an attention branch and an FFN branch to the residual stream. The stream sums $2N$ branch outputs that are nearly uncorrelated at initialization, so a $1/\sqrt{2N}$ residual multiplier keeps the forward signal at $\Theta(1)$. The updates of different branches, however, are correlated through the shared loss and add up rather than cancel, so the total update grows as N . The residual multiplier already cancels one factor of $\sqrt{N}$ of this growth, and under Adam the learning rate must be scaled by $1/\sqrt{N}$ to cancel the other. Table 1 lists the resulting rules for the hidden layers relative to a base model of width d_0 and depth N_0 , in terms of the width multiplier $m_d = d/d_0$ and the depth multiplier $m_N = N/N_0$.

Two further quantities drift during training rather than with model size. The attention logits grow with the norms of the query and key projections, so we apply **QKNorm** [14] to the queries and keys. This holds the logits in a fixed range and keeps the softmax from collapsing onto a single position. The weight norms themselves grow under Adam, and the effective learning rate shrinks with them. We therefore optimize the dense parameters with **AdamW**, whose decoupled **weight decay** [21] holds the norms at a stable level.

Multimodal Embedding. Capacity also changes how an item’s multimodal embedding should enter the model. The established practice, *SimTier* [36], does not feed the multimodal embedding itself. For every behavior sequence, it computes the cosine similarity between the candidate and each item of that sequence, summarizes those scores into a histogram, and feeds the histogram to the model as a dense feature. OneTrans-V2 instead feeds the multimodal embedding directly, as side information on every behavior item and on the candidate. This choice is beneficial only with sufficient capacity, as §7.8 shows.

4.4 Sequence-Native Training

A larger backbone makes every training step more expensive, and the largest part of that cost is encoding the long behavior sequence. Conventional recommendation training pays this cost many times over, because each stage builds its own training samples, one per exposure or one per request, and every sample carries its own copy of the sequence. A user with a hundred exposures therefore has that sequence copied and encoded up to a hundred times. Methods such as RLB [11] and MTGR [13] reduce this redundancy by grouping exposures around a shared user context. SNT extends this reuse across the cascade. The sequence is encoded once, and every exposure reads from that encoding, whichever request and whichever stage it comes from.

Two properties make this possible. First, the behavior sequence is append-only and is encoded with causal attention [53]. The hidden state at each position therefore summarizes only the behaviors up to that position, and the encoding of the whole sequence contains the encoding of every earlier prefix unchanged. Second, each exposure keeps its own request identity, timestamp, task identity, stage-specific features, and labels, but it no longer carries a copy of the behavior sequence. In the model, an exposure appears as its stage-specific tokens attached to the shared encoding under the stage visibility mask, and it reads only the prefix up to its request time, as the next paragraph describes. To make this sharing work in practice, SNT also changes how the training samples are grouped. Exposures are grouped by user, so that the exposures of one user within a time window (a *user window*) are stored and read together with one copy of that user’s behavior sequence. The length of that window is a configuration choice (§7.9).

What Each Exposure Sees. Formally, exposure e is anchored at the position a_e of the last behavior in $\mathcal{S}_u$ that had arrived by its request time (§4.1). Under the stage visibility mask it attends only to the prefix up to a_e . Exposures from the same request share one anchor. Each exposure therefore sees exactly what the model saw online at that request, even though the sequence was encoded once for all of them. We call this the causal view of an exposure, and Figure 4 illustrates it for two requests.

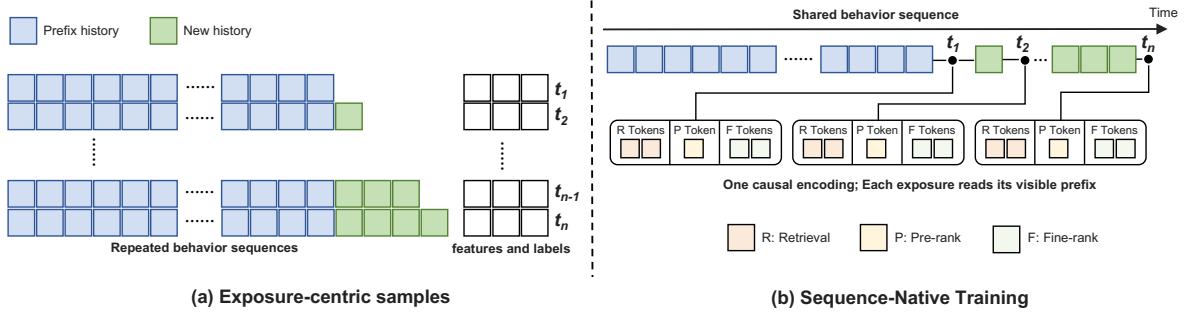


Figure 4 Sample organization in SNT. (a) Exposure-centric samples are stored separately for every exposure at every cascade stage, each with its own copy of the behavior sequence. (b) SNT stores and causally encodes the behavior sequence once per user window. Each exposure keeps its stage-specific features and labels and reads only the prefix available at its request time.

The views stay valid as the sequence grows because it is append-only. Let $S_u^{(n)}$ denote the sequence seen by the n -th request. The sequence seen by the next request is

$$S_u^{(n+1)} = S_u^{(n)} \parallel \Delta S_u^{(n+1)}, \quad (10)$$

where $\parallel$ denotes concatenation and $\Delta S_u^{(n+1)}$ contains the behaviors that arrive between the two requests. Because new behaviors are appended and never inserted into an earlier prefix, the causal states and anchors defined on $S_u^{(n)}$ remain valid in $S_u^{(n+1)}$.

Request-Relative Time Encoding. The anchor decides which behaviors an exposure can see, but not how old they are. A behavior from yesterday matters differently to a request made today than to one made a month later. Yet both requests read the same encoded sequence. We therefore make the attention between stage-specific tokens and the behavior sequence depend on the time difference between the request and the behavior. We use a timestamp-based variant [24] of rotary position encoding [38]. The query of an exposure at request time t_e is rotated by t_e , and the key of a behavior at event time t_i is rotated by t_i . For the temporal channels, the attention term then satisfies

$$\left(R(t_e) \mathbf{q}_e^{\text{time}} \right)^\top \left(R(t_i) \mathbf{k}_i^{\text{time}} \right) = (\mathbf{q}_e^{\text{time}})^\top R(t_i - t_e) \mathbf{k}_i^{\text{time}}, \quad (11)$$

where $R(t)$ denotes the rotary transformation for timestamp t . The term depends only on $t_i - t_e$, not on where the behavior sits in the sequence. Tokens of the same exposure share one request time, so their relative rotation cancels. Behavior self-attention is left unchanged. We use multiple fixed temporal periods to capture both short- and long-term dynamics. The rotation of a key depends only on the behavior’s own timestamp. The rotated keys are therefore computed once and reused by every request and stage, and each exposure contributes only its own query and anchor.

Compared with exposure-centric samples, SNT yields a $4.4\times$ training speedup under the same hardware budget (§7.9).

5 Serving and System Optimization

5.1 Serving Pipeline

Although OneTrans-V2 is jointly trained as one Transformer, serving still follows the original retrieval $\rightarrow$ pre-rank $\rightarrow$ fine-rank cascade, with each stage executing only its own candidate-side computation.

Shared User Context. For an online request from user u , the behavior sequence S_u is first processed by the shared causal Transformer to produce a reusable user-context cache C_u . The same C_u is then consumed by retrieval, pre-rank, and fine-rank, so the long behavior sequence is encoded once per request rather than once per stage. Retrieval consumes C_u together with the retrieval-specific features $\mathcal{NS}_r$ and performs DCGR decoding, applying the business offset of §4.2.1 to the prefix scores, followed by beam search over the SID hierarchy. Pre-rank and fine-rank are non-autoregressive candidate-scoring passes over the same C_u . Pre-rank uses the lightweight feature set $\mathcal{NS}_p$ to score the retrieved

candidates, while fine-rank applies the richer NS_f only to the surviving candidates. The candidate set therefore continues to shrink through the cascade even though the user context is shared.

5.2 Decoding and Kernel Optimization

Retrieval Decoding Optimization. Retrieval first generates the decision prefix and then decodes the SID codes with beam search. Both are decoded step by step, so their latency is set by the number of sequential steps and the work per step. We reduce the former for the prefix and the latter for SID decoding. For the decision prefix, every dimension has a small vocabulary. There are three values for the purchase level z_{oc} , two for the supply type z_{ad} , and a few each for the discovery level z_{disc} and the spending level z_{aov} . Their product is small enough to score in a single forward pass. Rather than decoding the parallel and chained positions one after another, we therefore enumerate all valid combinations of $\mathbf{z}$ in Eq. 3 and score them in one forward pass over the cached user context. This collapses the prefix from multiple sequential steps to a single scoring step, which is where the business offset acts. For SID decoding, write k for the beam width and K for the number of items the search finally returns to pre-rank. We replace the global top- k over all beam-token pairs with a two-stage top- k , which first retains the k best extensions of each beam and then selects the global top- k among them. This yields exactly the same top- k set. The intermediate candidate pool shrinks from the beam width times the vocabulary size to k per beam, which cuts the work per step accordingly.

Kernel Fusion. Beyond the decoding changes above, a set of kernel-level optimizations raises the model FLOPs utilization (MFU) of every forward pass in both training and serving. On modern GPUs, utilization is governed by the size and number of GEMMs and by the memory traffic of the element-wise operations between them. We therefore make the GEMMs larger and fewer and fuse whatever lies in between. The attention between the stage-specific tokens and the shared behavior sequence runs on a FlashAttention-style kernel [6] whose built-in stage visibility mask covers all candidates of a user in one call, without padding or materializing the attention matrix. Within a TransBlock, the query, key, value, and gate projections are issued as one bias-free general matrix multiplication (GEMM) whose output is consumed in place. QKNorm is applied by an RMSNorm kernel that addresses the query and key slices through offsets rather than copies. The gate and up projections of SwiGLU are likewise merged into one GEMM. Each fused kernel is written to exploit the efficient instructions of modern GPUs. For example, the RMSNorm kernel guarantees that every token’s hidden vector starts at a 16-byte-aligned address, so the compiler can emit 128-bit vectorized loads that fetch eight 16-bit values per instruction.

Low-Precision Serving. Unlike LLMs, recommendation models consume many floating-point features whose value ranges and precision must be preserved, so serving runs in FP16 by default. The Transformer backbone itself is more tolerant, since the RMSNorm before every sub-layer and on the queries and keys keeps its activations on a bounded scale. We therefore push the SwiGLU GEMMs, which account for about two thirds of the FLOPs of a TransBlock, further down to FP8. Serving also admits one fusion that training cannot. With no backward pass that needs the intermediate activations, the merged gate-and-up GEMM, the down GEMM, and the scaled residual update of SwiGLU run as a single kernel.

6 Experimental Results

6.1 Experimental Setup

Dataset. We evaluate OneTrans-V2 offline on production logs from a large-scale industrial recommendation system. The logs are collected with the same logging, feature-snapshot, and labeling pipeline as OneTrans [53] but over a different time window, so absolute numbers are not comparable with those reported there. The production logs are kept in chronological order, and every feature is recorded at exposure time to avoid temporal leakage and to keep offline evaluation consistent with online serving. Labels such as clicks and orders are accumulated over the fixed windows used in production. All logs are collected in compliance with strict privacy policies, with personally identifiable information anonymized and hashed.

Tasks and Metrics. We evaluate all three stages. Retrieval is generative and decodes SID codes with beam search rather than scoring a fixed candidate set. Following UniSGR [40], we measure it with HitRate@ M (HR@ M), the fraction of

requests whose ground-truth item has its SID among the top- M SIDs that beam search returns, and report $M=1$ and $M=10$. Pre-rank and fine-rank score a given candidate set, and we report AUC and UAUC (exposure-weighted user-level AUC) for both CTR and CVR. Both stages and their baselines are trained and evaluated on the same exposures and labels, without unexposed candidates or sampled negatives.

Following the next-batch protocol of OneTrans [53], we stream the data chronologically and score each mini-batch before it updates the model, so no prediction is made on data the model has already trained on. Metrics are computed daily and macro-averaged over the evaluation period. We also report serving *GFLOPs*, the floating-point cost that a model spends on one request under the deployed configuration, summed over the stages it serves.¹ This is the cost actually executed **after the optimizations described in OneTrans**. For example, the candidates that a stage scores for a request share one encoding of the behavior sequence, so the encoding is counted once rather than once per candidate.

Implementation. We evaluate OneTrans-V2 (§4) at two scales. Our default, **OneTrans-V2_L** (L for large), stacks the TransBlocks of §4.3 and costs 662 GFLOPs per request. **OneTrans-V2_S** (S for small) is a lighter variant of the same design that costs 239 GFLOPs per request, about 36% of OneTrans-V2_L. For the retrieval task, we quantize each item into three SID codes with a codebook of 8192 entries per level. The decision prefix covers four dimensions (z_{oc} , z_{disc} , z_{ad} , z_{aov}) laid out over two token positions.

During training, we use a dual-optimizer strategy. Sparse embeddings are optimized with AdaGrad [8], with a learning rate of 0.1 and an initial accumulator value of 1.0. Dense parameters are optimized with AdamW [21]² (Adam $\beta_1=0$, $\beta_2=0.99999$, $\epsilon=10^{-5}$, weight decay 0.01). With $\beta_1=0$ this optimizer carries no momentum and reduces to RMSProp with bias correction of the second moment and decoupled weight decay. We keep the name AdamW for the implementation it uses. Beyond the stabilization techniques of §4.3, we use global gradient-norm clipping [25] for both dense and sparse layers, and we train in BF16 with a per-GPU batch size of 2048 on 16 GPUs. In the training objective (Eq. 9), the retrieval, pre-rank, and fine-rank losses carry weights $\lambda_r=0.1$, $\lambda_p=1$, and $\lambda_f=1$, and the distillation loss carries $\lambda_{kd}=1$ at a temperature of $\tau=0.5$. The centering means of Eq. 8 are maintained as exponential moving averages over training batches with momentum 0.9999.

Baselines. We compare OneTrans-V2 against the following baselines, all trained with the same features:

- **OneTrans-GR** is our deployed generative retrieval (GR) model, a standalone SID-only model without the decision prefix. It represents each item by a three-level SID (§4.2.1) and autoregressively decodes its codes with a Transformer decoder. It is the most recently developed GR model in our system, built after OneTrans [53] and at a much larger scale. This makes it a strong baseline rather than a lightly tuned reference model.
- **OneTrans-Lite** is our deployed pre-rank model, a compact variant of OneTrans that fits the tighter compute budget of pre-rank.
- **OneTrans** [53] is our deployed fine-rank model, which jointly models a user’s behavior sequences and non-sequential features within a single Transformer (§3).
- **UniSGR** [40] and **UniPinRec** [20] are two recent frameworks that unify generative retrieval with a downstream ranking component. In our reproduction, we map UniPinRec’s lightweight ranking task to pre-rank and UniSGR’s multi-objective ranking task to fine-rank, and we decode UniSGR’s retrieval following its original implementation, in which the task token is set externally rather than predicted from the behavior sequence. Since each couples only two stages, we compare each model on the stages represented by this mapping. UniPinRec’s retrieval is the one exception, as it is not generative but retrieves by nearest-neighbor search over item embeddings, and we therefore do not compare it against the SID-based generative retrieval models reported here.

6.2 Offline Performance

Table 2 summarizes the performance of OneTrans-V2 and the corresponding baselines across the three stages. The existing cascade, *i.e.*, OneTrans-GR for retrieval, OneTrans-Lite for pre-rank, and OneTrans for fine-rank, together costs

¹OneTrans [53] reported training FLOPs. Here we report serving FLOPs instead, since the cost of serving a request is what bounds the QPS and latency of the deployed cascade.

²Weight decay follows the convention of modern LLM training, *i.e.*, a decoupled shrinkage $W \leftarrow W - \eta\omega W$ with weight decay coefficient ω , scaled by the learning rate η rather than a learning-rate-independent step $W \leftarrow W - \omega W$, as in TensorFlow’s legacy AdamW.

Type	Model	Retrieval		Pre-rank				Fine-rank				Efficiency
		HR@1	HR@10	CTR		CVR		CTR		CVR		GFLOPs
				AUC	UAUC	AUC	UAUC	AUC	UAUC	AUC	UAUC	
(a) Single Model	OneTrans-GR	0.0545	0.3112	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	202
	OneTrans-Lite	N.A.	N.A.	0.7965	0.7080	0.9102	0.7365	N.A.	N.A.	N.A.	N.A.	4
	OneTrans	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	0.8084	0.7393	0.9139	0.7428	69
(b) Unified Models	UniPinRec	N.A.	N.A.	0.8046	0.7212	0.9113	0.7506	N.A.	N.A.	N.A.	N.A.	325
	UniSGR	0.0483	0.2964	N.A.	N.A.	N.A.	N.A.	0.7976	0.7209	0.9130	0.7328	117
	OneTrans-V2 _S	0.0660	0.3441	0.8053	0.7372	0.9128	0.7534	0.8171	0.7532	0.9163	0.7589	239
	OneTrans-V2 _L	0.0923	0.4053	0.8089	0.7409	0.9145	0.7596	0.8182	0.7564	0.9184	0.7682	662

Table 2 Performance across the three stages. Part (a) reports our deployed production baselines, each specialized for one stage; part (b) reports models that unify multiple stages. “N.A.” marks a stage that a model does not cover, as well as UniPinRec’s retrieval, which we exclude because it is not generative. GFLOPs is the total arithmetic cost of one request for each model in isolation, including all candidates that stage scores, thousands for pre-rank and hundreds for fine-rank. It is the cost actually executed after the optimizations described in OneTrans [53], *e.g.*, the user-side sequence that these candidates share is encoded once rather than once per candidate. It is not a throughput proxy, since achieved throughput also depends on MFU. The best result per quality metric is shown in **bold**.

275 GFLOPs per request. This is the reference for the comparisons below.

OneTrans-V2_S serves all three stages at 239 GFLOPs, slightly under that reference, yet surpasses each of the three baselines on the stage it serves, by +21.10% on retrieval HR@1 and +2.17% on fine-rank CVR UAUC, for example. OneTrans-V2_L spends 2.4× as much as the cascade and turns the extra compute into larger gains on all ten metrics. It lifts retrieval HR@1 by 69.36% over OneTrans-GR, pre-rank CTR UAUC by 4.65% and CVR UAUC by 3.14% over OneTrans-Lite, and fine-rank CTR UAUC by 2.31% and CVR UAUC by 3.42% over OneTrans. Both variants also surpass UniPinRec and UniSGR, two recent models that unify multiple stages, on every stage they cover. The part of these gains that comes from joint training alone, under matched architecture and parameter budget, is isolated in §7.1.

6.3 Online A/B Performance

We assess the business impact of OneTrans-V2 with online A/B tests in a large-scale industrial recommendation system. The *control group* is served by the existing cascade of OneTrans-GR, OneTrans-Lite, and OneTrans. In the *treatment group*, a single OneTrans-V2_L, served with the stack of §5.2, replaces all three. Traffic is split at the user level with hashing-based randomization, with 50% of traffic assigned to each group.

OneTrans-V2 lifts GMV per user by 9.74% over the control group. It also raises user active days by 0.25%, so the items it recommends bring users back more often. These gains are measured against a strong cascade of three separately trained and separately served models, and they show the potential of cascade-level unification.

OneTrans-V2 also serves more efficiently. We define QPS as the number of requests completed per second through all three stages. Under the same hardware budget, the treatment group is served at 3.2× the QPS of the control group, even though OneTrans-V2 executes more FLOPs per request. The higher QPS comes from the serving stack and from encoding the behavior sequence once per request rather than once per stage, and §7.9 quantifies both.

7 Analysis

In this section, we analyze what each component of OneTrans-V2 contributes. Unless noted otherwise, ranking results are reported on CTR, and every improvement represents a relative change against the corresponding reference configuration instead of an absolute difference in AUC or UAUC.

7.1 Effect of Joint Training

We analyze whether training the three tasks jointly outperforms training each of them separately. Throughout this subsection, the joint model and its separately trained counterparts share the same architecture, parameter budget, training data, and training objective, with the loss weights of §6.1 unchanged. This includes the distillation loss of §4.2.2, which

the separately trained pre-rank receives from the separately trained fine-rank. Any gain therefore reflects joint training rather than added capacity or distillation. We first analyze merging pre-rank with fine-rank, and then analyze merging all three stages together.

Combining Pre-rank and Fine-rank. We start with the two ranking tasks, letting their tokens read one shared user context and optimizing them together. Table 3 shows that both tasks improve over their separately trained counterparts. Pre-rank gains +0.09% CTR AUC and +0.17% CTR UAUC, and fine-rank gains +0.12% CTR AUC and +0.21% CTR UAUC. The two tasks therefore reinforce each other rather than compete.

Task	CTR AUC	CTR UAUC
Pre-rank	+0.09%	+0.17%
Fine-rank	+0.12%	+0.21%

Table 3 Improvement from jointly training pre-rank and fine-rank over training each task separately.

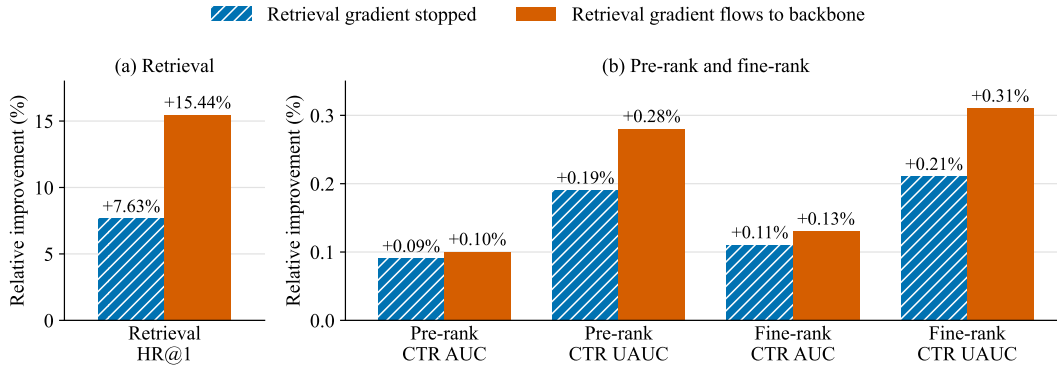


Figure 5 Effect of jointly training all three tasks, before and after letting the retrieval task’s gradient flow into the shared user context. (a): Improvement of HR@1 over a separately trained retrieval model of the same architecture and parameter budget. (b): Improvement of pre-rank and fine-rank CTR AUC/UAUC over training each task separately.

Combining All Three Stages. We then add the retrieval task and train all three together, and Figure 5 compares every task against its separately trained counterpart. With the retrieval gradient into the shared user context stopped, the retrieval task³ reuses the shared user context already shaped by the ranking tasks and gains +7.63% on HR@1. Pre-rank and fine-rank gain +0.19% and +0.21% CTR UAUC.⁴ We then let the retrieval gradient flow into the shared user context, so that its next-token prediction objective also shapes the user context the ranking tasks read. This roughly doubles the retrieval task’s gain to +15.44% and raises pre-rank and fine-rank to +0.28% and +0.31% CTR UAUC. Unifying all three tasks therefore improves every task over training them separately, with the retrieval objective further strengthening the shared user context for ranking.

7.2 Distillation and Consistency between the Ranking Stages

Knowledge Distillation. The previous subsection shows that joint training helps all three tasks. For the two ranking tasks, OneTrans-V2 adds a second mechanism, the knowledge distillation of §4.2.2, an extra loss $\mathcal{L}_{kd}$ in which the fine-rank logits supervise the pre-rank ones. Figure 6(a) shows what this loss contributes. This ablation is orthogonal to the joint-training comparison, where both settings keep $\mathcal{L}_{kd}$; here the joint model drops it. Removing $\mathcal{L}_{kd}$ from OneTrans-V2 while holding the architecture, data, and every other hyperparameter fixed costs pre-rank 0.55% CTR AUC and 0.97% CTR UAUC. This shows how much distillation contributes to closing the gap between pre-rank and fine-rank. Fine-rank itself moves by at most 0.04%, as the detached teacher logit intends, so the student’s gain is not paid for by the teacher.

³In this experiment the retrieval task of the joint model and its separately trained counterpart share the same backbone and compute budget as OneTrans-GR, so that the comparison isolates joint training from capacity.

⁴These differ slightly from Table 3 because the embedding tables remain shared, so the retrieval task still has a small effect on pre-rank and fine-rank even with its gradient into the shared user context stopped.

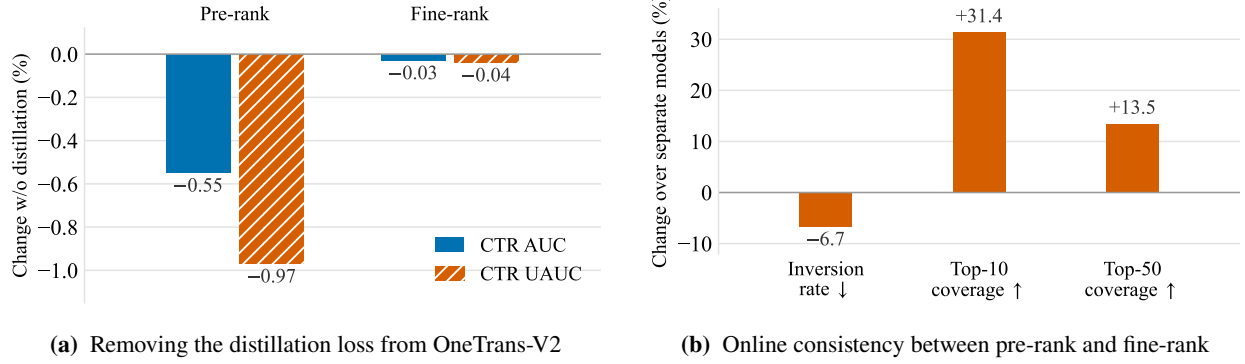


Figure 6 Distillation between the two ranking tasks and their consistency. (a): Relative change in CTR AUC and CTR UAUC of each ranking task when $\mathcal{L}_{kd}$ is removed from OneTrans-V2 with everything else held fixed. (b): Relative change of OneTrans-V2 over the separately trained pre-rank and fine-rank of §7.1 on each consistency metric, where the arrow in a label gives the direction of improvement.

Cross-Stage Consistency. We further analyze how consistently pre-rank and fine-rank order the same candidates online. Both the shared user context and the distillation loss pull the two stages together, and Figure 6(b) measures their combined effect with two metrics, the *inversion rate* and *top-10 and top-50 coverage*. The inversion rate is the fraction of item pairs that the two tasks order differently, and top-10 coverage is the fraction of pre-rank’s top 10 that also appears in fine-rank’s top 10, and likewise for top-50. Both are computed over the candidates that pre-rank forwards to fine-rank. Compared with the separately trained pre-rank and fine-rank of §7.1, OneTrans-V2 reduces the inversion rate by 6.7% and raises top-10 and top-50 coverage by 31.4% and 13.5%, all as relative changes. The two ranking stages therefore disagree less often. More importantly, pre-rank forwards far more of fine-rank’s top candidates, so fewer of the items that fine-rank would place at the top are discarded before fine-rank ever sees them.

7.3 Objective Coverage of DCGR

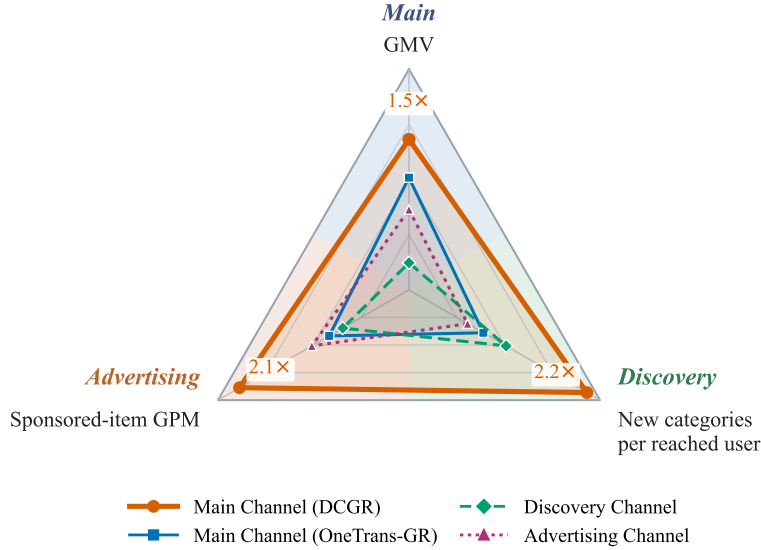


Figure 7 Coverage of the three retrieval objectives. In this retrieval-stage A/B test at equal traffic, the treatment group unifies the three channels into OneTrans-V2’s retrieval task, shown as DCGR, while the control group keeps the three channels. All four are compared on the three objectives, one per axis. On every axis DCGR exceeds the channel built for that objective, by the multiple marked at that vertex. Each exposure is credited to exactly one channel by the retrieval-source field of the serving log, and the radius is a power scale.

As described in §1, retrieval in our production system is split into specialized channels for different business objectives. The main channel, served by OneTrans-GR, targets clicks and orders, the discovery channel promotes categories new to the user, and the advertising channel retrieves sponsored items. DCGR represents these objectives within a single

model through corresponding decision dimensions. Specifically, the purchase level z_{oc} captures the main channel’s click-and-order objective, the discovery level z_{disc} captures the discovery objective, and the supply type z_{ad} distinguishes organic from sponsored items for advertising. This moves the three objectives from separate retrieval channels into one shared decision space. We next ask whether **a single DCGR can still cover what each specialized channel was built to provide**.

We test this with an online retrieval-stage A/B test, with traffic split equally at the user level. In the treatment group, OneTrans-V2’s retrieval task, *i.e.*, DCGR, replaces OneTrans-GR in the main channel, and the discovery and advertising channels are switched off, so the three channels are unified by a single model. The control group keeps all three channels as deployed. Because the three channels serve different purposes, we evaluate each objective with its native metric, namely GMV for the main objective, the number of categories new to the user per reached user for discovery, and GMV per thousand exposures (GPM) on sponsored items for advertising.

Figure 7 shows that DCGR covers all three objectives within one model. It produces 1.5× the GMV of the control group’s main channel, reaches 2.2× as many categories new to the user per reached user as the discovery channel, and achieves 2.1× the sponsored-item GPM of the advertising channel. Thus, a single DCGR matches or exceeds the specialized channel on the metric native to each objective.

Main Channel Analysis. The 1.5× GMV gain on the main objective comes primarily from broader exposure rather than higher value per exposure. Table 4(a) shows that DCGR increases exposures by 69.3% and clicks by 74.6% over the control group’s main channel. Part of that volume is what the discovery and advertising channels served in the control group, since DCGR now serves it as well. Its per-exposure conversion is lower. Specifically, orders per thousand exposures decrease by 34.0%, and GMV per thousand exposures by 13.0%. The larger exposure volume nevertheless more than offsets these declines, increasing total orders by 11.4% and total GMV by 48.0%. CTR per exposure also increases by 3.2%. The average price of an exposed item decreases by 4.4%, indicating that the additional exposures are concentrated more heavily on lower-priced items.

<i>(a) Main channel, treatment against control</i>	
Exposures	+69.3%
Clicks	+74.6%
Orders	+11.4%
GMV	+48.0%
Click-through rate per exposure	+3.2%
Orders per thousand exposures	−34.0%
GMV per thousand exposures	−13.0%
Average price of an exposed item	−4.4%
<i>(b) Against the discovery channel</i>	
Categories new to the user, per reached user	2.2×
Share of the other channel’s discovery items also reached	76.0% / 14.0%
Share of the other channel’s user–category pairs also reached	11.7% / 1.2%
User–category pairs formed per day	9.4×
Conversion of the users only this channel reaches	2.8×
<i>(c) Against the advertising channel, on sponsored items</i>	
Exposures	1.8×
GMV per thousand exposures	2.1×
Orders per exposure	1.7×
Conversion rate	2.6×
Average order value	1.2×

Table 4 The retrieval-stage A/B test of Figure 7 in detail. (a) Relative changes of DCGR in the treatment group over the control group’s main channel. The GMV row of (a) is the figure’s main axis. (b) and (c) Ratios of DCGR to the discovery and the advertising channel, respectively. The first row of (b) and the second row of (c) are the figure’s discovery and advertising axes. Each pair of percentages in (b) gives DCGR’s coverage of the discovery channel, then the reverse.

These measurements compare one channel against another rather than one system against another. DCGR serves in the

treatment group the exposures that all three channels served in the control group, so its margin over the main channel alone is not the value it adds to the system. Figure 7 therefore measures which objectives DCGR can cover, not how much it adds. The A/B test in §6.3 provides the latter over all traffic.

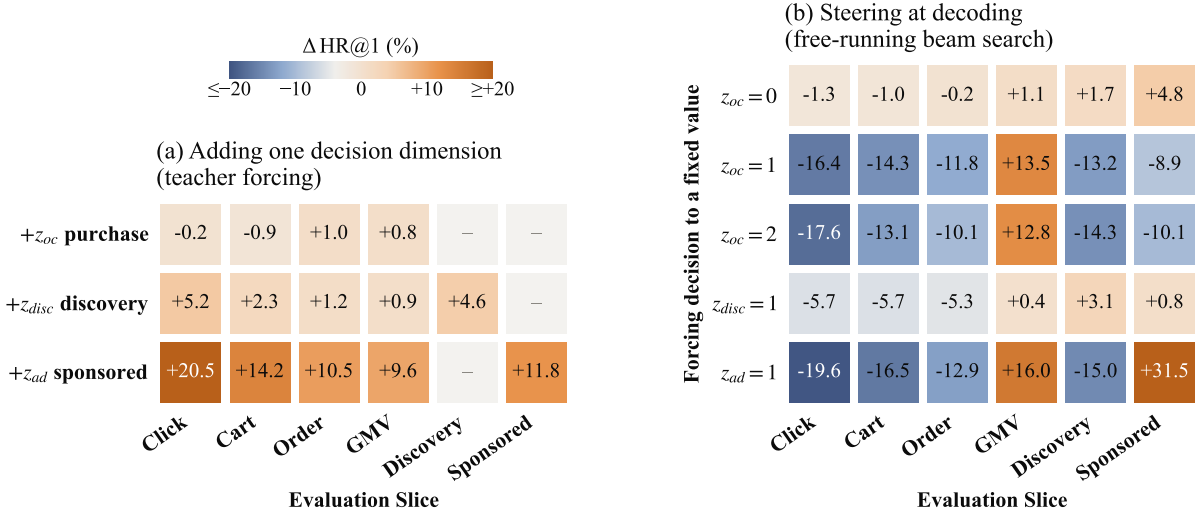


Figure 8 Ablation of decision dimensions, as the relative change in HR@1 (in %). (a): Adding one decision dimension to a SID-only generative retrieval model, measured under teacher forcing on the evaluation slice named in each column; a dash marks a slice we did not measure for that model. (b): Forcing one decision dimension to a value at decoding time, with the rest of the prefix decoded autoregressively, relative to decoding the whole prefix that way. Cells carry the exact value; the color scale is clipped at ± 20 .

Discovery and Advertising Analysis. We next examine the discovery and advertising results in more detail. Table 4(b, c) breaks down DCGR’s comparison with the corresponding specialized channels.

For discovery, DCGR reaches 4.81 categories new to the user per reached user, compared with 2.15 for the specialized discovery channel, yielding the 2.2 $\times$ gain in Figure 7. More importantly, the difference lies primarily in which users receive these categories rather than in which items are retrieved. DCGR covers 76.0% of the discovery channel’s items but only 11.7% of its user–category pairs; in the reverse direction, the corresponding coverages are 14.0% and 1.2%. DCGR also forms 9.4 $\times$ as many user–category pairs per day, and users reached only by DCGR convert 2.8 $\times$ as efficiently as those reached only by the discovery channel. The two retrievers therefore draw on substantially overlapping items but expose them to very different users.

For advertising, DCGR also takes on a substantially larger role in sponsored-item retrieval. Sponsored items account for 33.7% of its exposures, compared with 4.7% for the main channel in the control group. On sponsored items, DCGR serves 1.8 $\times$ as many exposures as the specialized advertising channel and achieves 2.1 $\times$ its GPM. This gain comes from stronger downstream conversion and value rather than clicks. DCGR achieves 1.7 $\times$ the orders per exposure, 2.6 $\times$ the conversion rate, and 1.2 $\times$ the average order value, despite a click-through rate about one third lower. Sponsored items retrieved by DCGR are therefore clicked less often but purchased more often and at higher value.

Together, these results show that a single DCGR can cover the objectives represented by z_{oc} , z_{disc} , and z_{ad} , matching or exceeding the corresponding specialized channel on its native metric. This demonstrates the potential to consolidate these retrieval objectives within one model.

7.4 DCGR Decision Dimensions Analysis

The previous subsection shows that a single DCGR retriever can cover multiple specialized objectives. We next ask whether the decision prefix itself carries useful retrieval signals and whether it controls item generation.

Intervention on $\mathbf{z}$	CTR	Orders	AOV	GMV
None (predicted prefix)	—	—	—	—
Force $z_{oc}=0$ (click)	+8.2%	−10.4%	+5.8%	−5.1%
Force z_{aov} to the highest value	−8.1%	−33.5%	+38.9%	−7.7%
Business offset, weak β	+2.2%	+4.9%	+2.5%	+7.5%
Business offset, medium β	−0.9%	−3.0%	+13.9%	+10.4%
Business offset, strong β	−5.4%	−1.2%	+28.3%	+26.7%

Table 5 Steering the same trained model at decoding time in an online retrieval-stage A/B test. Values are relative changes against decoding with the predicted prefix, over the exposures credited to OneTrans-V2’s retrieval task as in §7.3. Orders counts main orders, one per checkout however many sub-orders it splits into, and AOV is GMV per main order. Exposure counts were logged only for the row that forces z_{aov} to its highest value, so the table has no exposure column. In that row, the exposures credited to OneTrans-V2’s retrieval task fall by 19.4%.

Decision Dimensions Carry Retrieval Signal. Figure 8(a) adds one decision dimension at a time to a SID-only GR model and supplies its ground-truth value through teacher forcing. We measure the resulting change in HR@1 across click, cart, order, GMV, discovery, and sponsored-item slices. Because the supplied decision describes the interaction being predicted, this experiment measures the information provided by a correct decision rather than deployable end-to-end retrieval accuracy.

We also discover these dimensions carry different amounts of item information. The purchase level z_{oc} changes no slice by more than 1%, whereas z_{disc} improves all measured slices and z_{ad} produces the largest gains, ranging from 9.6% to 20.5%. This difference reflects how strongly each decision constrains item identity. For example, z_{ad} restricts a sponsored interaction to the sponsored subset, which accounts for only 28% of interactions, whereas z_{oc} says relatively little about which item was chosen. The gains also extend beyond the corresponding objective. In particular, z_{ad} improves click, cart, order, and GMV retrieval in addition to sponsored-item retrieval. No measured slice decreases by more than 1%.

Decision Dimensions Control Generation. Teacher forcing shows that a correct decision is informative, but serving relies on decisions predicted by the model. We therefore intervene on one predicted dimension during free-running beam search. If the SID decoder ignored the prefix, forcing a decision would have little effect. Figure 8(b) shows the opposite. Forcing z_{ad} to sponsored raises sponsored-item HR@1 by 31.5% and GMV HR@1 by 16.0%, while reducing click HR@1 by 19.6%. Forcing a multi-order z_{oc} raises GMV HR@1 by 12.8% but reduces click HR@1 by 17.6%. Forcing z_{disc} toward discovery raises its own slice by 3.1%. Changing a decision therefore changes the retrieved items in the expected direction.

The intervention size also depends on how often the forced value already occurs. Sponsored items account for 28% of interactions, whereas interactions in a category new to the user already make up roughly three quarters, leaving fewer requests for discovery forcing to change. The small offline discovery movement also does not contradict its larger online gain in §7.3, nor does the slight drop in click HR@1 under $z_{oc}=0$ contradict its CTR gain online in Table 5. Offline HR@1 rewards reproducing the logged item, whereas online metrics reward exposures that users click or that reach valuable categories new to the user, even when the retrieved item differs from the log.

7.5 DCGR Decision Steering and Prefix Design

We next examine how the decision dimensions should be steered and organized in practice.

Predicted vs. Supplied Decisions. A natural alternative to predicting the decision prefix is to supply the desired decision directly, as objective-conditioned methods do [40]. Table 5 compares this hard conditioning with DCGR’s business offset in an online retrieval-stage A/B test. We either force $z_{oc} = 0$ toward clicks, force z_{aov} to its highest value, or apply the offset at increasing strengths. Hard forcing approximates the extreme of objective steering, whereas the offset gradually tilts the model’s predicted distribution.

Hard forcing behaves much like an objective-specific channel. Table 5 shows that forcing $z_{oc} = 0$ raises CTR by 8.2%

but lowers GMV by 5.1%, while forcing the highest z_{aov} value raises AOV by 38.9% but lowers orders by 33.5% and GMV by 7.7%. The business offset instead provides a gradual trade-off. As its strength increases from weak to strong, GMV rises from +7.5% to +26.7%, while CTR moves from +2.2% to -5.4%. The offset therefore steers the model toward a business objective without replacing its predicted decision with a fixed value.

Table 6 presents a second retrieval-stage A/B test, which confirms the same trade-off at the system level. Unlike Table 5, it measures all exposures per user rather than only those credited to DCGR. Steering z_{oc} alone increases orders by 2.4% but reduces GMV by 0.6%. Adding z_{aov} trades some orders for higher AOV, turning the GMV change positive and reaching +3.0% at the stronger setting. The two dimensions therefore control complementary aspects of business value.

Offset applied	Clicks	Orders	AOV	GMV
Purchase level z_{oc} only	+8.8%	+2.4%	-2.9%	-0.6%
+ spending level z_{aov} , weak	+11.4%	-5.2%	+5.9%	+0.4%
+ spending level z_{aov} , strong	+5.1%	-9.4%	+13.7%	+3.0%

Table 6 A second retrieval-stage A/B test of the business offset, as relative changes per user against the control group. Unlike Table 5, the metrics cover all of a user’s exposures, not only those credited to DCGR, so they show the offset’s effect on the whole system. The last two rows keep the purchase-level offset unchanged and add a spending-level offset.

Organizing the Decision Prefix. The complementary roles of z_{oc} and z_{aov} motivate modeling purchase quantity and value separately. Replacing them with a single GMV token reduces transaction-value-weighted HR@1 by 9%. A separate purchase indicator is also redundant because $z_{oc} = 0$ already represents a non-converting interaction, and adding one leaves HR@1 unchanged.

Independent decision dimensions should not be chained arbitrarily because chaining requires choosing a decoding order. Table 7 tests this effect on an earlier configuration with three binary dimensions for order, cold-start, and discovery. The order objective performs best when decoded first, and the order-first configurations also achieve the highest weighted HR@128. The decoding order therefore biases which objective is favored. DCGR avoids this bias by combining independent dimensions into the parallel decision token DT^p and chaining only genuinely dependent decisions, with z_{aov} following z_{oc} in DT^d . This design also shortens the prefix, but its main advantage is avoiding ordering bias, since the summed dimensions share one position and have no decoding order.

Position of the order dimension	Weighted	Order	Cold-start	Discovery
First (acd, adc)	0.3287	0.4590	0.0352	0.1437
Second (cad, dac)	0.3267	0.4548	0.0348	0.1437
Third (cda, dca)	0.3252	0.4520	0.0371	0.1446

Table 7 Effect of the decoding order of chained decision tokens, on an earlier configuration with three binary dimensions (a = order, c = cold-start, d = discovery). HR@128 with the prefix decoded autoregressively, averaged over the two permutations that place the order dimension at each position; the weighted HR@128 column weights hits by their transaction value.

7.6 Architecture Analysis

We now analyze the architecture components introduced in §4.3, namely GQA, gated attention, and MoE. Figure 9 reports what gated attention and MoE contribute to OneTrans-V2_L on the fine-rank task, and then what each design choice inside the MoE block contributes.

Replacing full multi-head attention with GQA leaves fine-rank quality effectively unchanged, so GQA’s smaller key/value cache costs no accuracy. Figure 9(a) further shows that gated attention contributes 0.08% CTR AUC and 0.15% CTR UAUC over the conventional attention it replaces.

MoE contributes more than gated attention, gaining 0.13% CTR AUC and 0.30% CTR UAUC over a compute-matched dense FFN. Figure 9(b) looks inside the MoE block. Fine-grained experts contribute 0.01% CTR AUC and 0.04% CTR UAUC over fewer, wider experts of the same total parameter count, and the shared expert contributes 0.02% and 0.07%

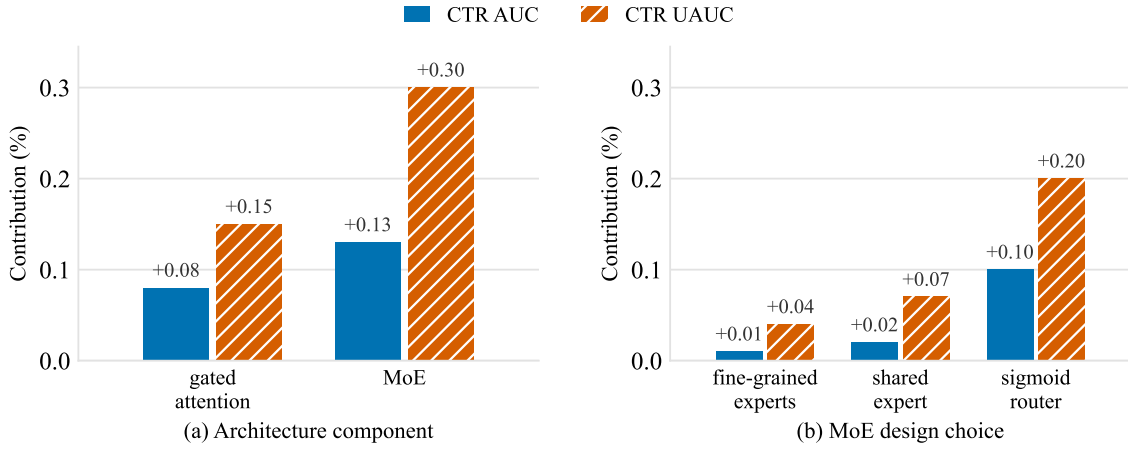


Figure 9 Contribution of each architecture component and each MoE design choice, measured on the fine-rank task. Each bar is the relative improvement in CTR AUC or CTR UAUC that a single component or design choice brings over its conventional alternative, with everything else in OneTrans-V2_L held fixed. The bars are therefore separate single-component ablations rather than a decomposition of the MoE gain. (a): The architecture components of §4.3. (b): The design choices inside the MoE block.

over the same MoE without a shared expert. The sigmoid router contributes more than either, with 0.10% CTR AUC and 0.20% CTR UAUC over a softmax router.⁵

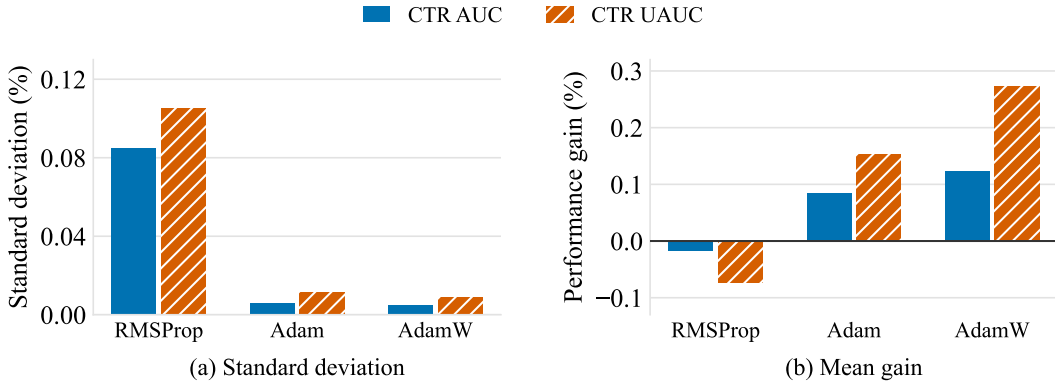


Figure 10 Performance gain from widening OneTrans-V2, measured over three runs. (a): Standard deviation across different runs. (b): Averaged gain across different runs.

7.7 Stabilization Analysis

We analyze the effectiveness of our stabilization methods introduced in §4.3. Each experiment below removes one technique from OneTrans-V2 and measures how much the same widening of the model gains without and with that technique, each against a narrow model built the same way. The widening gain therefore differs from paragraph to paragraph because the reference differs, so gains are comparable within one figure but not across figures. The gain of widening the full configuration is the one reported for AdamW in Figure 10.

AdamW and Weight Decay. RMSProp [43] is the optimizer commonly used to train industrial recommendation models, and the RMSProp baseline here uses the learning rate, ϵ , and decay coefficient tuned for our deployed models. As Figure 10 shows, it yields a large standard deviation across repeated runs, and widening the model brings no gain. We

⁵Repeated runs of the same configuration differ by about 0.006% in CTR AUC and 0.012% in CTR UAUC (§7.7), so a difference larger than this should be read as meaningful.

then replace it with Adam [18] under the same learning rate and ϵ , with β_2 set to the RMSProp decay coefficient and $\beta_1=0$ (§6.1) so that no momentum is added. The only difference between the two optimizers is then the bias correction of the second-moment estimate. This correction prevents the inflated effective learning rate of the early updates. It cuts the standard deviation of CTR AUC from 0.085% to 0.006% and of CTR UAUC from 0.105% to 0.012%, both estimated from the same three runs. Widening then becomes effective, turning the negative outcome under RMSProp into a gain of 0.15% CTR UAUC. Adding decoupled weight decay on top, *i.e.*, AdamW, raises that gain further to nearly 0.3% CTR UAUC while the standard deviation stays at the same low level.

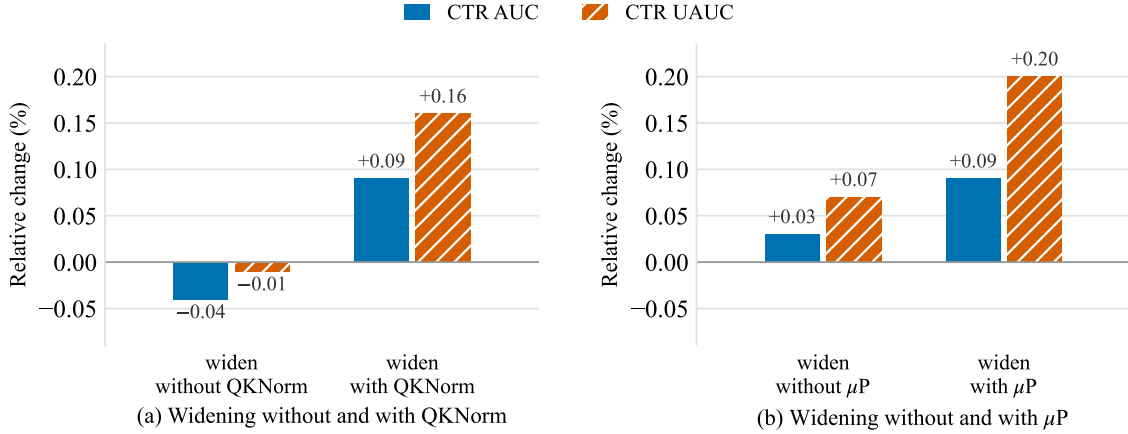


Figure 11 Effect of QKNorm and μP when widening OneTrans-V2, measured on the fine-rank task. In each panel, both bars are measured against a narrow model built without that technique, which for μP means standard parameterization with its own initialization and learning rate. Bars are therefore comparable within a panel but not across panels or with Figure 10. (a): Widening without and with QKNorm. (b): Widening without and with μP .

QKNorm. Figure 11(a) shows that widening the model without QKNorm falls short of the expected gain, changing fine-rank by -0.04% CTR AUC and -0.01% CTR UAUC. Once QKNorm is introduced, the same widening gains $+0.09\%$ CTR AUC and $+0.16\%$ CTR UAUC.

μP . Figure 11(b) shows that widening the model without μP , keeping the hyperparameters tuned at the original width, brings only a slight improvement of $+0.03\%$ CTR AUC and $+0.07\%$ CTR UAUC. Under standard parameterization these hyperparameters are no longer the right ones once the model is widened, so every widening would call for a new search. μP instead keeps the optimal hyperparameters invariant to width, *i.e.*, hyperparameter transfer [48]. The same widening then yields $+0.09\%$ CTR AUC and $+0.20\%$ CTR UAUC without any retuning.

Depth- μP . The experiments above widen the model; deepening it is governed by Depth- μP (§4.3), which scales both the residual multiplier and the learning rate by $1/\sqrt{m_N}$, and we test it here. Table 8 shows that adding blocks to OneTrans-V2 gains only 0.04% CTR AUC and 0.05% CTR UAUC on its own, whereas the same deepening under Depth- μP gains 0.08% and 0.11%. Depth- μP therefore turns additional depth into a larger gain.

Setting	CTR AUC	CTR UAUC
Base depth	—	—
deepened, w/o Depth- μP	+0.04%	+0.05%
deepened, w/ Depth- μP	+0.08%	+0.11%

Table 8 Effect of Depth- μP when deepening OneTrans-V2 on the fine-rank task, relative to the base depth.

7.8 Multimodal Embedding and Model Capacity

We verify the multimodal design of §4.3 by comparing SimTier against the multimodal embedding at two model scales. One is our default OneTrans-V2_L; the other is a much smaller variant, OneTrans-V2_{XS} (XS for extra small), which is roughly 46× cheaper to train.

As shown in Figure 12, the two scales call for opposite designs. At the smaller scale, only SimTier helps, contributing

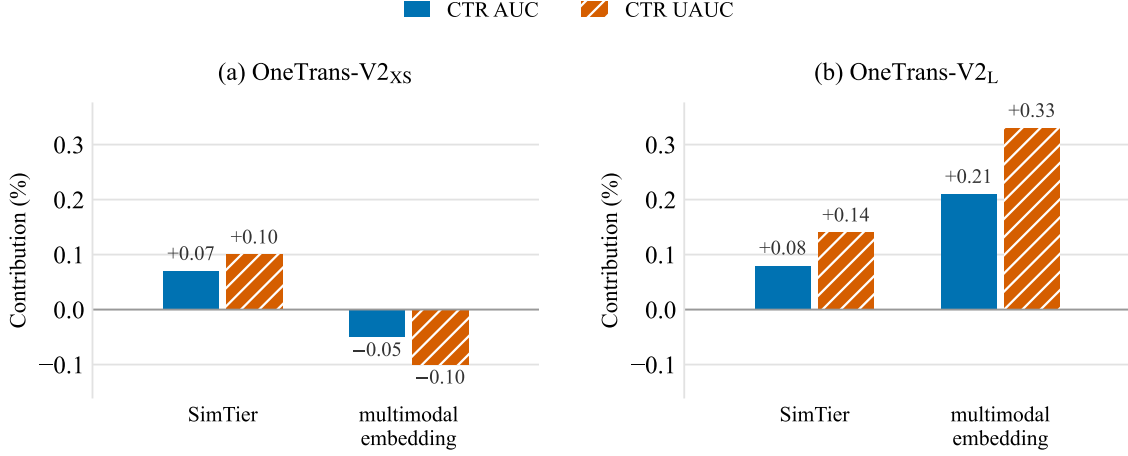


Figure 12 Contribution of each way of injecting the multimodal embedding, measured at two model scales on the fine-rank task. Each bar is the relative change from adding that input alone to a model that receives no multimodal embedding, so the two bars within a panel are separate experiments rather than a decomposition. (a): OneTrans-V2XS. (b): OneTrans-V2L.

0.07% CTR AUC and 0.10% CTR UAUC. The multimodal embedding contributes -0.05% and -0.10% , so feeding the embedding directly hurts a model of this size. At the larger scale both ways help, and the multimodal embedding becomes the more valuable choice, contributing 0.21% against 0.08% CTR AUC and 0.33% against 0.14% CTR UAUC without any precomputed feature. The multimodal embedding and the model’s own learned item embeddings live in different spaces, and mapping one onto the other is something the model must learn. We hypothesize that a small model lacks the capacity to learn this mapping, so the similarity has to be precomputed for it. OneTrans-V2L learns the mapping on its own and gains more from the multimodal embedding than from a precomputed summary of it. How the multimodal embedding should enter a ranking model is therefore not a fixed design choice but a function of capacity, and a practice established at one scale need not be the best one at the next.

7.9 Training and Inference Speedup

We first analyze the training speedup brought by SNT (§4.4), and then the inference speedup brought by the serving optimizations of §5.2, first those specific to retrieval decoding and then those that apply to the whole model.

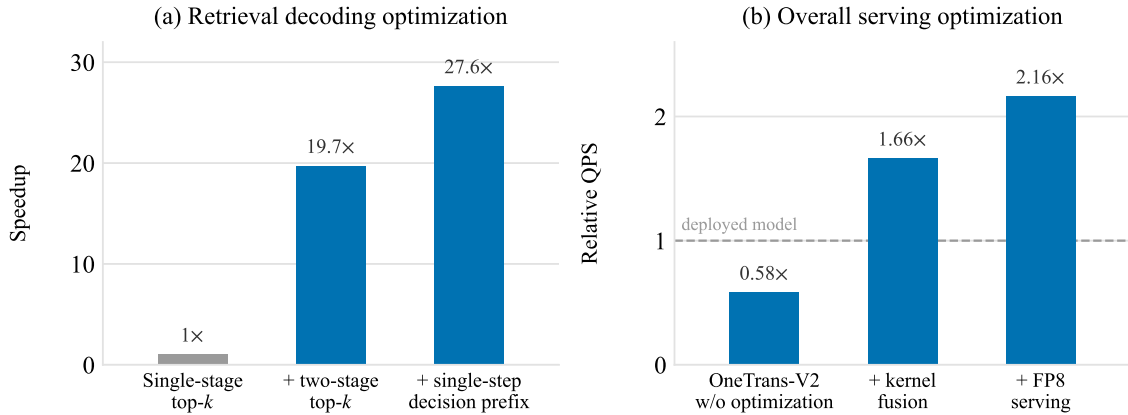


Figure 13 Speedup from the serving-side optimizations. (a): Decoding speedup of OneTrans-V2’s retrieval task, measured as throughput under the same latency budget, with the two decoding optimizations of §5.2 added cumulatively. (b): Serving throughput of the fine-rank task relative to the deployed fine-rank model under the same latency budget, with kernel fusion and FP8 serving added cumulatively.

Sequence-Native Training. We compare SNT against the exposure-centric samples of Figure 4(a), in which every exposure is a separate sample that encodes its own copy of the behavior sequence. Sharing one encoding across all exposures of a user window yields a 4.4 $\times$ training speedup under the same hardware budget. In these experiments a mini-batch is organized by user windows, each holding the exposures of one user within a window of time. We cut the windows at day boundaries, so that no window straddles an evaluation date and the daily metrics of §6.1 stay comparable with those of the baselines. This choice caps the speedup, since a window can share the sequence encoding only among the exposures of one day. Windows cut by week or by month raise the speedup further, and we leave that configuration to future work. SNT also saves storage. Each user’s behavior sequence is stored once instead of once per exposure, which cuts sequence feature storage by 95%. The exposures of a user are stored next to each other, which makes their non-sequential features compress better and cuts that storage by half.

Retrieval Decoding Optimization. We further analyze the two decoding optimizations on the retrieval task. For SID decoding, Figure 13(a) shows that the two-stage top- k raises throughput by 19.7 $\times$ under the online latency budget, against our own single-stage implementation rather than the deployed retrieval baseline. Per request, the same change cuts beam-search latency by 2.3 $\times$. The throughput gain exceeds the latency gain because throughput is measured at the largest batch that fits the latency budget. Shrinking the candidate pool from the beam width times the vocabulary size to k per beam lets each GPU serve a much larger batch within that budget. Optimizing this step matters because, before this change, 94% of the retrieval task’s inference time was spent in the top- k . Scoring the decision prefix in a single step then removes the sequential prefix steps, so each remaining step carries a larger batch and maps better onto the GPU. This cuts the measured latency by about 30% against decoding the prefix step by step and raises the cumulative decoding speedup to 27.6 $\times$.

Overall Serving Optimization. The decoding optimizations above are specific to retrieval, whereas kernel fusion and FP8 serving apply to the whole model. Figure 13(b) measures them on the fine-rank task, relative to the deployed fine-rank model under the same latency budget. Without them the unified model serves at 0.58 $\times$ the throughput of the deployed model, since its fine-rank task spends more compute per request. Kernel fusion raises this to 1.66 $\times$ and FP8 serving to 2.16 $\times$, so the unified model serves the fine-rank stage 2.16 $\times$ faster than the specialized model it replaces despite its larger compute. Sharing one user context across the three stages (§5.1) encodes the behavior sequence once per request rather than once per stage, which adds a further 1.53 $\times$ on top of all per-stage optimizations, measured over the whole cascade. The two together yield the end-to-end 3.2 $\times$ QPS of §6.3.

8 Conclusion

We present OneTrans-V2, one Transformer that serves retrieval, pre-rank, and fine-rank in an industrial recommendation cascade system. OneTrans-V2 builds on three key ideas. First, it unifies retrieval, pre-rank, and fine-rank within one Transformer model by sharing the candidate-independent user context across stages while preserving each stage’s native candidate features and computation. Second, DCGR consolidates objective-specific retrieval models by representing business objectives as decisions within a single generative process. Third, SNT reorganizes training around each user’s lifelong behavior sequence and amortizes sequence encoding across exposures. Together, these designs unify the cascade across stages and retrieval objectives while making the resulting shared backbone practical to train at production scale. This unification enables computation reuse, joint learning, and in-model knowledge distillation. It also pools previously fragmented capacity into a shared backbone. We scale this backbone with sparse MoE to increase capacity with bounded activated computation, and use μ P-style parameterization to stabilize scaling. In a large-scale industrial deployment, OneTrans-V2 improves GMV by 9.74% and, together with a co-designed serving stack, delivers 3.2 $\times$ QPS of the cascade it replaces under the same hardware budget. We believe OneTrans-V2 provides a production-scale foundation for the broader transition from model-per-stage recommendation toward a unified one-model recommendation architecture.

References

- [1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training generalized multi-query Transformer models from multi-head checkpoints. In *Proc. of EMNLP*, pages 4895–4901.
- [2] Edoardo Botta, Jaewon Yang, Yi-Ping Hsu, Laksh Bhasin, Yilin Chen, Prabhat Agarwal, Anirudhan Badrinath, Jiajing Xu, and Charles Rosenberg. 2025. PinRec: Unified generative retrieval for Pinterest recommender systems. *arXiv preprint arXiv:2504.10507*.
- [3] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. In *Proc. of NeurIPS*.
- [4] Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Marc’Aurelio Ranzato, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. 2022. Unified scaling laws for routed language models. In *Proc. of ICML*.
- [5] Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proc. of ACL*, pages 1280–1297.
- [6] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Proc. of NeurIPS*.
- [7] DeepSeek-AI. 2024. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.
- [8] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.
- [9] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.
- [10] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. 2021. Learning to reach goals via iterated supervised learning. In *Proc. of ICLR*.
- [11] Lin Guan, Jia-Qi Yang, Zhishan Zhao, Beichuan Zhang, Bo Sun, Xuanyuan Luo, Jinan Ni, Xiaowen Li, Yuhang Qi, Zhifang Fan, Hangyu Wang, Qiwei Chen, Yi Cheng, Feng Zhang, and Xiao Yang. 2026. Make it long, keep it fast: End-to-end 10k-sequence modeling at billion scale on Douyin. In *Proc. of WWW*, pages 7989–7998.
- [12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A factorization-machine based neural network for CTR prediction. In *Proc. of IJCAI*, pages 1725–1731.
- [13] Ruidong Han, Bin Yin, Shangyu Chen, He Jiang, Fei Jiang, Xiang Li, Chi Ma, Mincong Huang, Xiaoguang Li, Chunzhen Jing, Yueming Han, Menglei Zhou, Lei Yu, Chuan Liu, and Wei Lin. 2025. MTGR: Industrial-scale generative recommendation framework in Meituan. In *Proc. of CIKM*, pages 5731–5738.
- [14] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. 2020. Query-key normalization for Transformers. In *Findings of EMNLP*, pages 4246–4253.
- [15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.
- [16] Jonathan Ho and Tim Salimans. 2021. Classifier-free diffusion guidance. In *NeurIPS Workshop on Deep Generative Models and Downstream Applications*.
- [17] Xu Huang, Defu Lian, Jin Chen, Zheng Liu, Xing Xie, and Enhong Chen. 2023. Cooperative retriever and ranker in deep recommenders. In *Proc. of WWW*, pages 1150–1161.
- [18] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.
- [19] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, and Igor Mordatch. 2022. Multi-game decision transformers. In *Proc. of NeurIPS*.
- [20] Hanyu Li, Yi-Ping Hsu, Aditya Mantha, Prabhat Agarwal, Laksh Bhasin, Jialu Wang, Hongtao Lin, Bella Huang, Yaxin Li, Xinyi Li, Chuxi Wang, Kousik Rajesh, Hooshmand Shokri Razaghi, Shun Yao Li, Zongyue Qin, Jaewon Yang, James Li, Dhruvil Deven

- Badani, Jiajing Xu, and Charles Rosenberg. 2026. UniPinRec: Unifying generative retrieval and ranking at Pinterest scale. *arXiv preprint arXiv:2606.00422*.
- [21] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proc. of ICLR*.
- [22] Xinchen Luo, Jiangxia Cao, Tianyu Sun, Jinkai Yu, Rui Huang, Wei Yuan, Hezheng Lin, Yichen Zheng, Shiyao Wang, Qigen Hu, Changqing Qiu, Jiaqi Zhang, Xu Zhang, Zhiheng Yan, Jingming Zhang, Simin Zhang, Mingxing Wen, Zhaojie Liu, Kun Gai, and Guorui Zhou. 2024. QARM: Quantitative alignment multi-modal recommendation at Kuaishou. *arXiv preprint arXiv:2411.11739*.
- [23] OneRec Team. 2025. OneRec technical report. *arXiv preprint arXiv:2506.13695*.
- [24] David Pardoe, Neil Daftary, Miro Furtado, Aditya Ayier, Yu Wang, Liuqing Li, Tao Song, Lars Hertel, Young Jin Yun, Senthil Radhakrishnan, Zhiwei Wang, Tommy Li, Khai Tran, Ananth Nagarajan, Ali Naqvi, Yue Zhang, Renpeng Fang, Avi Romascanu, Arjun Kulothungun, Deepak Kumar, Praneeth Boda, Fedor Borisyuk, and Ruoyan Wang. 2026. CADET: Context-conditioned ads CTR prediction with a decoder-only Transformer. *arXiv preprint arXiv:2602.11410*.
- [25] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proc. of ICML*, pages 1310–1318.
- [26] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. 2019. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.
- [27] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *Proc. of CIKM*, pages 2685–2692.
- [28] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling Transformer inference. In *Proc. of MLSys*.
- [29] Jiarui Qin, Jiachen Zhu, Bo Chen, Zhirong Liu, Weiwen Liu, Ruiming Tang, Rui Zhang, Yong Yu, and Weinan Zhang. 2022. RankFlow: Joint optimization of multi-stage cascade ranking systems as flows. In *Proc. of SIGIR*, pages 814–824.
- [30] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *Proc. of NeurIPS*.
- [31] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender systems with generative retrieval. In *Proc. of NeurIPS*.
- [32] Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. 2024. Stay on topic with classifier-free guidance. In *Proc. of ICML*.
- [33] Jürgen Schmidhuber. 2019. Reinforcement learning upside down: Don’t predict rewards – just map them to actions. *arXiv preprint arXiv:1912.02875*.
- [34] Noam Shazeer. 2020. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*.
- [35] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.
- [36] Xiang-Rong Sheng, Feifan Yang, Litong Gong, Biao Wang, Zhangming Chan, Yujing Zhang, Yueyao Cheng, Yong-Nan Zhu, Tiezheng Ge, Han Zhu, Yuning Jiang, Jian Xu, and Bo Zheng. 2024. Enhancing Taobao display advertising with multimodal representations: Challenges, approaches and insights. In *Proc. of CIKM*, pages 4858–4865.
- [37] Sona Team. 2026. Sona technical report. *arXiv preprint arXiv:2608.11015*.
- [38] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2024. RoFormer: Enhanced Transformer with rotary position embedding. *Neurocomputing*, 568:127063.
- [39] Dekai Sun, Yiming Liu, Jiafan Zhou, Xun Liu, Chenchen Yu, Yi Li, Jun Zhang, Huan Yu, and Jie Jiang. 2026. OneRanker: Unified generation and ranking with one model in industrial advertising recommendation. *arXiv preprint arXiv:2603.02999*.
- [40] Jiawei Sun, Jun Yang, Ziyue Guo, Dongyue Xu, Jianan Yan, Lifang Deng, and Xiaoyi Zeng. 2026. UniSGR: Unified framework for semantic ID generation and ranking. *arXiv preprint arXiv:2607.04068*.
- [41] Shangquan Sun, Wenqi Ren, Jingzhi Li, Rui Wang, and Xiaochun Cao. 2024. Logit standardization in knowledge distillation. In *Proc. of CVPR*, pages 15731–15740.

- [42] Jiayi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proc. of KDD*.
- [43] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.
- [44] Chunqi Wang, Bingchao Wu, Taotian Pang, Jiahao Wang, Jie Yang, Jia Liu, Hao Zhang, Hai Zhu, Lei Shen, Shizhun Wang, Bing Wang, and Xiaoyi Zeng. 2026. SORT: A systematically optimized ranking Transformer for industrial-scale recommenders. *arXiv preprint arXiv:2603.03988*.
- [45] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. 2021. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proc. of WWW*, pages 1785–1797.
- [46] Zhe Wang, Liqin Zhao, Biye Jiang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. COLD: Towards the next generation of pre-ranking system. *arXiv preprint arXiv:2007.16122*.
- [47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.
- [48] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. 2021. Tensor Programs V: Tuning large neural networks via zero-shot hyperparameter transfer. In *Proc. of NeurIPS*.
- [49] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. 2024. Tensor Programs VI: Feature learning in infinite-depth neural networks. In *Proc. of ICLR*.
- [50] Ruochen Yang, Shuang Wen, Pengbo Xu, Yusheng Huang, Jiangxia Cao, Shuang Yang, Zhaojie Liu, Jiawei Sheng, and Tingwen Liu. 2026. UniR²: Unifying generative recall and multi-objective ranking in a single decoder-only sequence. *arXiv preprint arXiv:2607.24439*.
- [51] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proc. of ICML*, pages 58484–58509.
- [52] Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Proc. of NeurIPS*.
- [53] Zhaoqi Zhang, Haolei Pei, Jun Guo, Yufei Feng, Hui Sun, Shaowei Liu, and Aixin Sun. 2026. OneTrans: Unified feature interaction and sequence modeling with One Transformer in industrial recommender. In *Proc. of WWW*, pages 8162–8170.
- [54] Zhishan Zhao, Jingyue Gao, Yu Zhang, Shuguang Han, Siyuan Lou, Xiang-Rong Sheng, Zhe Wang, Han Zhu, Yuning Jiang, Jian Xu, and Bo Zheng. 2023. COPR: Consistency-oriented pre-ranking for online advertising. In *Proc. of CIKM*, pages 4974–4980.
- [55] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proc. of KDD*, pages 1059–1068.
- [56] Yifeng Zhou, Yuehong Hu, Zhixiang Feng, Junwei Pan, Kaihui Wu, Hanyong Li, Shangyu Zhang, Shudong Huang, Zhangbin Zhu, Chengguo Yin, Haijie Gu, and Jie Jiang. 2026. TokenFormer: Unify the multi-field and sequential recommendation worlds. *arXiv preprint arXiv:2604.13737*.
- [57] Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, Huizhi Yang, Zheng Chai, Zhe Chen, Yuchao Zheng, Qiwei Chen, Feng Zhang, Xun Zhou, Peng Xu, Xiao Yang, Di Wu, and Zuotao Liu. 2025. RankMixer: Scaling up ranking models in industrial recommenders. In *Proc. of CIKM*.

A Full Author List

Foundation Team: Hannan Cao, Jun Guo, Haolei Pei, Zhaoqi Zhang, Tianyu Wang, Ziyang Wang, Youchen Sun, Yue Xue, Yucheng Mao, Lintao Yan, Yufei Feng, Shaowei Liu*

Architecture Team: Rongkun Xing, Feiling Gong, Xinyu Chenli, Cong Xu, Mingge Zhang, Yunjia Zhu, Yajing Zhang, Pengfei Ren, Yue Lin

*: Corresponding author, liushaowei.nphard@bytedance.com