

OneLA: Scaling Linear-Attention Decoding to Large Beams in Generative Recommendation

Xiangrui Yang*
The University of Hong Kong
China
xiangruiyang@connect.hku.hk

Cheng Peng*
Kuaishou Technology
China
pengcheng09@kuaishou.com

Yunfeng Zhao
Kuaishou Technology
China
zhaoyunfeng@kuaishou.com

Liang Zeng
Kuaishou Technology
China
zengliang03@kuaishou.com

Ao Hu
Kuaishou Technology
China
huao@kuaishou.com

Jiawei Yang†
Peking University
China
yangjiawei26@stu.pku.edu.cn

Shengzhe Wang
Kuaishou Technology
China
wangshengzhe@kuaishou.com

Jingshan Lv
Kuaishou Technology
China
lvjingshan@kuaishou.com

Xiao Liang
Kuaishou Technology
China
liangxiao03@kuaishou.com

Chen Yang
Kuaishou Technology
China
yangchen28@kuaishou.com

Jiaqiang Liu‡
Kuaishou Technology
China
liujiaqiang@kuaishou.com

Yiming Qiu‡
The University of Hong Kong
China
yimingq@hku.hk

Abstract

Generative recommendation (GR) relies on large-beam decoding to generate hundreds of candidate items, creating a new scaling challenge for recurrent linear attention. Existing linear attention serving systems either materialize a full recurrent state for every beam or repeatedly replay shared history, incurring substantial memory and traffic overhead.

To address this, we present OneLA, a linear-attention decoding framework that exploits the shared prompt and short divergent suffixes of GR workloads. Specifically, OneLA represents all beam states using a single shared prompt-derived state and compact, append-only records of their divergent transitions. Using this representation, OneLA computes only the state information required at each decoding step, without reconstructing a full recurrent state for every beam. Furthermore, OneLA uses a lightweight ancestry index to track the transition records that make up each beam’s history, allowing beams to be updated without moving or copying existing records. A fused GPU kernel further reuses the shared state across beams. Our analysis shows that OneLA achieves 1.54–2.46× end-to-end decode speedups while substantially reducing recurrent-state memory use and data movement.

Keywords: recommendation systems, attention mechanisms

1 Introduction

Recommendation systems are central to modern online services, and generative recommendation (GR) has emerged as a practical paradigm for large-scale applications [1, 3, 9, 20, 24]. The widely adopted approach represents each item as a short sequence of discrete semantic IDs (SIDs), thereby converting recommendation into sequence generation while retaining semantic and collaborative structure [4, 11, 14–16, 22]. Recent industrial systems such as OneRec and OneReason illustrate both the adoption and continued extension of this formulation [9, 24]. Unlike conventional LLM serving, GR must produce hundreds of candidate items rather than a single sequence and therefore relies on large-beam decoding, where candidate SID sequences are expanded and globally selected at every step [12]. The resulting workload combines a long shared prompt, short generated suffixes, and a large, dynamically evolving beam.

Processing such long user histories [2] with conventional attention incurs quadratic computation in sequence length, which motivates the use of linear attention in GR [6]. These modules compress token history into a fixed-size recurrent state, substantially reducing the dependence of computation and memory on context length. We use Gated DeltaNet (GDN) [18], widely adopted in frontier LLMs [10], as our representative instance. However, large-beam GR turns this fixed-size state into a new scaling problem: an independently evolving state for every live beam makes storage and memory traffic grow sharply with beam width.

*Xiangrui Yang and Cheng Peng contributed equally to this work.

†This work was completed at Kuaishou.

‡Jiaqiang Liu and Yiming Qiu are the corresponding authors.

General-purpose LLM serving systems optimize KV-cache management and attention execution for conventional decoding, where each sequence owns an independent state [7, 8, 19, 23]. Their GDN execution paths inherit this assumption: a beam either persists its own complete recurrent state or reconstructs one from its own history, and in both cases the state is bound to the logical beam that owns it [7, 8, 19]. This binding is particularly costly in GR, where global beam selection reorders and duplicates beams at every decoding step, forcing states to be copied, reconstructed, or recomputed as the beam tree evolves [12].

We observe that full per-beam recurrent state materialization is fundamentally redundant. In large-beam GR, all candidate beams within a request originate from an identical prompt-derived recurrent state and diverge only through a brief sequence of decode-time state transitions. Each transition can be represented compactly, allowing a beam state to be expressed as one shared post-prefill state together with a short sequence of branch-specific transition records. This trades computation for memory, with replay depth bounded by the fixed-length SID sequence. Moreover, beam selection changes only which historical records a surviving beam refers to; it does not modify the records themselves. These records can therefore remain immutable and append-only, while lightweight ancestry metadata tracks the dynamically evolving beam tree.

Based on these observations, we present OneLA, an efficient linear-attention decoding framework for large-beam GR. OneLA stores the post-prefill recurrent state once per request and represents beam-specific evolution with compact GDN Transition Records (GTRs). Rather than using these records to reconstruct each beam’s recurrent state, OneLA replays them onto the two state projections that each decoding step actually uses. Replay therefore costs vector rather than matrix processing, and no per-beam recurrent matrix is ever materialized or written back. A lightweight ancestry index decouples dynamic beam evolution from physical record placement, while a fused shared-context GPU kernel reuses the shared state across beams and performs replay entirely on chip. We also implement the OneLA prototype and evaluate it on an industrial GR workload. The results show that OneLA achieves 1.54–2.46× end-to-end decode speedups over existing GDN execution paths. In summary, this paper makes the following contributions:

- We identify beam width as a new scaling dimension for recurrent linear attention in GR, exposing the inefficiency of per-beam state management.
- We design OneLA with a shared context state, compact append-only GDN Transition Records (GTRs), and lightweight beam ancestry, and develop a fused shared-context GPU kernel that performs on-chip projection replay with cross-beam context-state reuse.

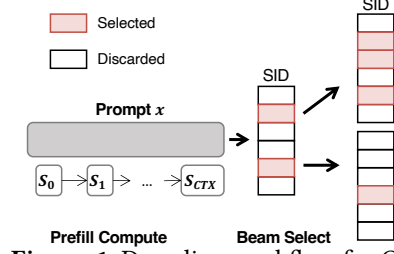


Figure 1. Decoding workflow for GR.

- We implement and evaluate OneLA, which achieves 1.54–2.46× end-to-end decode speedups over existing GDN execution paths.

2 Background and Motivation

2.1 Generative Recommendation

Generative recommendation (GR) reformulates candidate generation as autoregressive sequence generation [11, 21, 24]. Recent systems, such as OneRec, OneReason, PLUM, and S-GRec, have established SID-based generation as a common GR formulation [3, 5, 9, 24]: each item is represented by a short sequence of discrete semantic IDs (SIDs), which the model generates. Formally, an item i is represented as $s_i = (s_i^1, \dots, s_i^T)$. Given an input prompt $\mathbf{x}$, the model generates its SID autoregressively: $\prod_{t=1}^T P(s_i^t | \mathbf{x}, s_i^{<t})$.

After the generation process, the complete SID sequence is mapped back to an item [11]. Unlike conventional text generation, GR must produce hundreds of candidate items rather than a single best sequence [12]. It therefore employs large-beam decoding to explore SID sequences.

At each decoding step, all active beams are expanded over the SID codebook, and the global top- k candidates are retained [12]. Multiple surviving children may originate from the same parent, while other parents may disappear. Consequently, the parent mapping $\pi_t(b)$ of beam b changes across steps, forming a dynamic beam tree. The resulting workload combines a long shared prompt, a short generated SID sequence, and a large dynamically branching beam [12]. That is to say, the prompt length and beam width are much greater than the SID sequence length, i.e., $L_{prompt} \gg T$ and $W_t \gg T$ [12]. Figure 1 summarizes this workflow: prefill computes the shared prompt, after which beam selection expands and retains candidate SID sequences.

2.2 Gated Delta Network

The long prompts of this workload are costly for conventional attention, whose computation is quadratic in sequence length and whose KV cache grows linearly with it [6, 13]. Linear attention, such as GDN, instead uses a fixed-size recurrent state: each head compresses its preceding history into $S_t \in \mathbb{R}^{d_v \times d_k}$, updated once per token, where d_v and d_k denote the value and key dimensions. Specifically, GDN processes token t from state S_{t-1} as follows [18]:

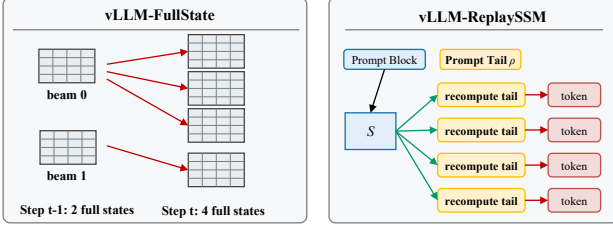


Figure 2. State management in an existing framework.

$$\begin{aligned} S_t &= \alpha_t S_{t-1} + \beta_t (v_t - \alpha_t S_{t-1} k_t) k_t^T, \\ o_t &= S_t q_t, \end{aligned} \quad (1)$$

where α_t and β_t control state decay and correction, respectively, and q_t and k_t denote the query and key. We refer to one such update, $S_{t-1} \rightarrow S_t$, as a *GDN transition step*. The state S_t summarizes the history up to token t , and the output o_t reads it out along the query q_t .

2.3 GDN Decoding for Generative Recommendation

Linear attention like GDN is widely adopted in frontier LLMs [10] and works well for GR models that need to absorb long user contexts. However, existing LLM serving frameworks handle GDN-based large-beam GR inefficiently. The difficulty arises when the compact recurrent representation of GDN is combined with the large and dynamically branching beam in GR.

Among existing serving frameworks, only vLLM [7] provides native end-to-end support for GDN-based beam search. As shown in Figure 2, the FullState path in vLLM maintains a complete recurrent state for every live beam, resulting in many huge full-state matrices. The ReplaySSM path in vLLM reduces state persistence through prefix replay but does not preserve replay history across the independent requests created by the native beam-search path. Specifically, ReplaySSM does not explicitly distinguish the long shared prompt from the short decode phase: it materializes a recurrent-state checkpoint only at completed prompt-block boundaries, potentially leaving a large prompt tail to be replayed during decoding. Moreover, ReplaySSM treats each beam as an independent request, requiring every beam to separately load the cached state checkpoint and replay its subsequent tokens. Other frameworks, including SGLang, FlashInfer, and TensorRT-LLM, provide operator-level support for GDN modules but do not offer end-to-end GDN serving with beam search. Overall, existing systems either materialize recurrent state independently for each beam or lack native end-to-end support for this workload.

Existing state-management approaches become inefficient under this workload for two main reasons. First, although GDN eliminates sequence-length-dependent KV storage [6, 18], the large-beam search decoding process of GR introduces a new scaling dimension [12]. Maintaining an independent state for w live beams requires $\Theta(wd_v d_k)$ persistent

state capacity. The per-beam dimension also amplifies recurrent state reads and writebacks during every decoding step. Second, dynamic beam evolution becomes expensive when materialized states are bound directly to logical beams. At every decoding step, global beam selection may reorder or duplicate beams and change the beam width [12]. Therefore, when materialized states are bound directly to individual beams, these dynamic parent-child reassignments require state copying, reconstruction, or recomputation.

These limitations expose a mismatch between logical beam evolution and existing materialized-state management. Efficient large-beam GDN decoding should therefore avoid materializing a complete recurrent state for every beam and decouple logical beam evolution from materialized state placement. We next identify the key observations that enable such a representation.

2.4 Key Insight

The above limitations stem from the assumption that each logical beam needs to materialize a recurrent state or be treated as an independent request. However, we derive two key insights that enable more efficient state management.

First, the recurrent states of different beams share the same prompt-derived origin and diverge only through a short sequence of GDN transition steps. Moreover, each GDN transition step can be represented by a compact tuple whose size is substantially smaller than the full recurrent state. More importantly, each such step consumes the recurrent state only through two projections, so the state itself never has to be reconstructed.

Second, the beam selection process only keeps references to prefix beams rather than changing the original prefix data. The selection process may reorder or duplicate parents but does not modify previously generated recurrent updates. Therefore, these records can remain append-only, and lightweight ancestry information can specify which records belong to each selected beam, avoiding state or record relocation as beam selection proceeds. This decouples logical beam selection from materialized state placement and enables efficient sharing across dynamically branching beams.

These insights motivate a recurrent-state management approach that separates the shared prompt-derived state, branch-specific compact records, and logical beam ancestry, as we show in the following sections.

3 Design

We present OneLA, a GDN attention decode framework for industrial GR. Following the above key observation that all beams within a request share an identical prompt-derived state and differ only in their short decode suffixes, OneLA leverages the insights that GDN transitions are fully captured by compact tuples and that dynamic beam evolution

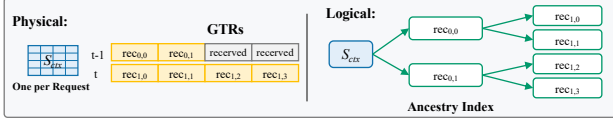


Figure 3. OneLA Overview

can be tracked via lightweight ancestry over immutable prefix records. OneLA therefore decomposes state management into a single shared post-prefill context alongside append-only branch records, evaluating subsequent tokens via projection replay without materializing per-beam recurrent matrices.

3.1 Design Overview

For the GR decoding process, we use t to denote the current decoding step, W_t the number of active beams at step t , and $b \in \{0, \dots, W_t - 1\}$ a beam at that step. For $t > 0$, $\pi_t(b)$ denotes the parent of beam b selected from the preceding step. We denote by S_{ctx} the recurrent state produced after prompt prefill, which is shared by all beams within the same request. For exposition, $S_{t,b}$ denotes the conceptual recurrent state of beam b after decode step t , which is not materialized in OneLA. Unless noted, we describe one request and one GDN head and omit request and head indices for clarity.

As shown in Figure 3, OneLA represents each beam as the shared post-prefill state S_{ctx} together with a short sequence of branch-specific state transitions, instead of materializing an independent recurrent state $S_{t,b}$ for every beam. Based on this representation, OneLA consists of two key components. First, it retains S_{ctx} and reconstructs the state-dependent computation of each beam by replaying only its short sequence of branch-specific transitions. Second, OneLA decouples logical beam evolution from physical state storage through lightweight beam ancestry. Together, these components allow OneLA to share the prompt-derived state across beams while maintaining only lightweight branch-specific information as the beam tree evolves dynamically.

3.2 OneLA State Representation

To avoid materialization of a full recurrent state for every beam, OneLA represents each beam’s recurrent evolution using a shared post-prefill state together with a short sequence of compact GDN Transition Records (GTRs). During decoding, OneLA replays these records directly on the state projections, reproducing the original GDN computation. For clarity, we omit the beam index b throughout this subsection.

Compact update records. In OneLA, at decoding step t , the GDN recurrent update can be written as

$$\begin{aligned} S_t &= \alpha_t S_{t-1} + \beta_t (v_t - \alpha_t S_{t-1} k_t) k_t^\top \\ &= \alpha_t S_{t-1} + \delta_t k_t^\top, \end{aligned} \quad (2)$$

where S_{t-1} is the state of the selected parent beam, with the shared post-prefill state S_{ctx} serving as the initial state. We refer to $r_t = (\alpha_t, \delta_t, k_t)$ as a **GDN Transition Record (GTR)**,

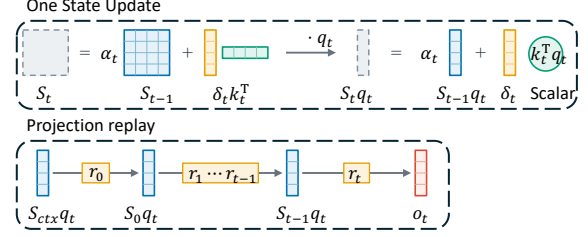


Figure 4. Projection replay in vector form.

in which $\delta_t = \beta_t (v_t - \alpha_t S_{t-1} k_t)$. Given the parent state, a GTR completely determines the transition to the child state.

In our design, OneLA stores S_{ctx} only once and records each GDN transition step by appending one GTR for each generated beam candidate, instead of persistently materializing a full $d_v \times d_k$ recurrent matrix ($O(d_v d_k)$) for every active beam as conventional FullState designs do. Because a GTR consists only of a scalar decay factor, a d_v -dimensional update vector, and a d_k -dimensional key vector, each transition requires merely $O(d_v + d_k)$ storage. We store δ_t rather than the underlying (β_t, v_t) because replay never needs the state matrix itself, only its projections onto the current query and key, to which a record contributes δ_t scaled by a single dot product. Keeping (β_t, v_t) instead leaves the contribution dependent on the parent state S_{t-1} , which would have to be reconstructed before the record could be used, defeating the purpose of the record.

Projection replay Although representing beams as GTR sequences eliminates persistent matrix storage, computing the subsequent decode step still requires the historical state context. Crucially, a GDN layer interacts with the prior state S_{t-1} solely through two vector contractions: the key projection $S_{t-1} k_t$ (for the update δ_t) and the query projection $S_{t-1} q_t$ (for the output o_t). Rather than reconstructing the intermediate $d_v \times d_k$ matrix on the fly, OneLA performs *projection replay* to evaluate these two projections directly across the ancestral chain of GTRs, as illustrated in Figure 4.

At decode step t , consider the sequence of GTRs $r_0, \dots, r_{t-1}$ along the ancestry of the current beam. For the current query q_t and key k_t , we define a *query-projection accumulator* $u_q^{(j)} \equiv S_j q_t$ and a *key-projection accumulator* $u_k^{(j)} \equiv S_j k_t$, where j denotes the replay depth along the beam ancestry. Specifically, right-multiplying the state transition in Equation 2 by q_t or k_t gives the following recurrences. Starting from the shared context,

$$\begin{aligned} u_q^{(-1)} &= S_{\text{ctx}} q_t, & u_q^{(j)} &= \alpha_j u_q^{(j-1)} + \delta_j (k_j^\top q_t), \\ u_k^{(-1)} &= S_{\text{ctx}} k_t, & u_k^{(j)} &= \alpha_j u_k^{(j-1)} + \delta_j (k_j^\top k_t), \end{aligned} \quad 0 \leq j < t. \quad (3)$$

After replaying through $j = t - 1$, the accumulators directly provide the two projections required by the current GDN computation. The current GDN update and output can

therefore be evaluated as

$$\begin{aligned}\delta_t &= \beta_t \left(v_t - \alpha_t u_k^{(t-1)} \right), \\ o_t &= \alpha_t u_q^{(t-1)} + \delta_t (k_t^\top q_t).\end{aligned}\quad (4)$$

Thus, projection replay requires only two d_v -dimensional accumulators per beam and value head, without materializing any intermediate $d_v \times d_k$ recurrent state.

3.3 Dynamic Beam Ancestry Index

OneLA’s state representation uses a shared context state S_{ctx} together with a short sequence of ancestral GTRs to replay a candidate beam. Under dynamic beam search in the real world, however, global beam selection may prune, reorder, or fan out beams, so a logical beam can hardly be bound to a fixed set of physical records. As pointed out previously, these operations do not modify the previously generated transition records and only update which records are referenced by each surviving beam. OneLA therefore keeps GTRs append-only in their original physical slots and tracks dynamic beam evolution separately through a lightweight *ancestry index*.

Specifically, for beam b at decoding step t , we define $h_t(b, j)$ as the physical slot for the GTR at decode step j , $j < t$, along the ancestry of b . Let $\pi_t(b)$ denote the parent of beam b . The ancestry index is updated as follows:

$$h_t(b, j) = \begin{cases} h_{t-1}(\pi_t(b), j), & 0 \leq j < t-1, \\ \pi_t(b), & j = t-1. \end{cases} \quad (5)$$

Each child reuses the ancestry indices of its selected parent and records the parent’s physical slot for the immediately preceding decoding step. Newly generated GTRs are written once to the physical slots of the current decoding step and are never relocated afterward. The logical beam tree is therefore represented entirely by the ancestry indices, while the underlying GTRs remain immutable.

This indirection decouples logical beam evolution from physical record placement. Specifically, beam pruning discards references, reordering changes their order, and fan-out duplicates only ancestry indices rather than copying the corresponding GTR chains. Consequently, dynamic beam selection requires no movement or duplication of historical GTRs. The projection replay can directly retrieve the record at depth j as $r_{j, h_t(b, j)}$, obtaining the exact GTR sequence associated with each current beam.

3.4 Fused Shared-Context Attention Kernel

The compact state representation removes per-beam full-state persistence, but its memory benefits must also be preserved during GPU execution. In particular, S_{ctx} is shared across beams and should not be repeatedly fetched from HBM, while the transient projections used during replay should not be materialized in memory. We therefore design a fused GDN attention kernel that exploits cross-beam reuse of S_{ctx} and keeps projection replay entirely on chip.

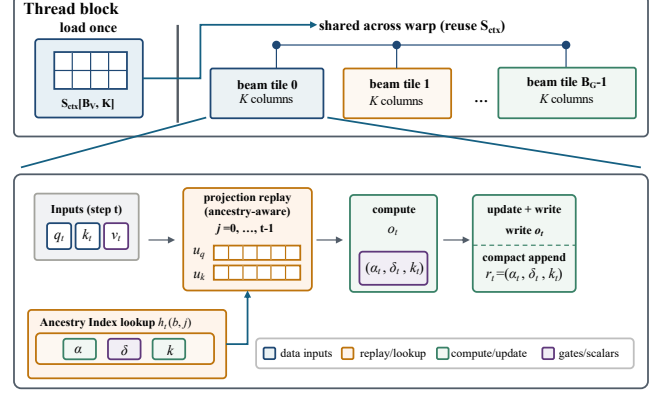


Figure 5. Shared-Context Attention Kernel

To exploit cross-beam reuse, OneLA assigns each GPU thread block a B_V -row tile in the value dimension together with a group of beams. As shown in Figure 5, the block loads the corresponding $S_{\text{ctx}}[B_V, d_k]$ tile once into on-chip storage and reuses it across the beam tiles processed by the block. This amortizes the cost of reading S_{ctx} from HBM across multiple beams instead of incurring the same state read independently for every beam.

Within each thread block, the kernel fuses shared-context projection, ancestry-aware replay, and the current GDN update into one execution flow. For the current beam tile, the block first loads the current query, key, value, and gate inputs, and initializes two projection accumulators, $u_q = S_{\text{ctx}} q_t$ and $u_k = S_{\text{ctx}} k_t$, using the context state tile $S_{\text{ctx}}[B_V, d_k]$. It then replays the ancestral GTRs selected by the ancestry index in the following steps. First, at replay depth j , the ancestry index $h_t(b, j)$ maps the logical ancestry position of beam b to the corresponding physical GTR slot, from which the kernel gathers $r_j = (\alpha_j, \delta_j, k_j)$. Second, the replay is applied directly to the two projection accumulators, updating them without ever reconstructing the intermediate recurrent state matrices S_j . Third, the accumulators hold the required projections $S_{t-1} q_t$ and $S_{t-1} k_t$, which are then used immediately to compute the current transition record r_t and output o_t . Since the above steps are fused into a single kernel and no intermediate state matrix is reconstructed, the intermediate values remain on chip without being materialized in HBM.

Finally, the kernel appends only the compact GTR $r_t = (\alpha_t, \delta_t, k_t)$ for each newly generated beam, rather than writing back a full recurrent state $S_t \in \mathbb{R}^{d_v \times d_k}$. This eliminates the large per-beam state-writeback cost and preserves the compact state representation throughout decoding.

4 Evaluation

Our evaluation answers three questions: (Q1) How much does OneLA improve the end-to-end full-model decode performance? (Q2) Does the GR GDN attention kernel maintain its latency and state-capacity advantages across workloads

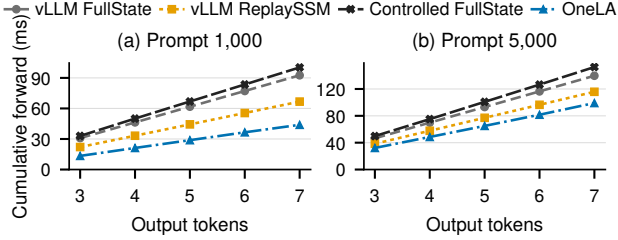


Figure 6. Cumulative full-model decode-forward GPU time.

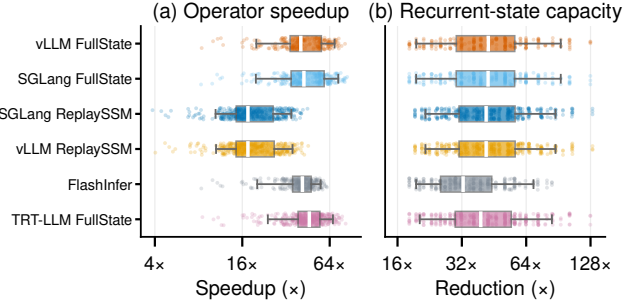


Figure 7. Operator speedup and persistent recurrent-state capacity reduction.

and model configurations? (Q3) How efficient is OneLA’s replay execution, and how much does it reduce memory traffic and consumption?

4.1 Methodology

We first describe the workload notation and experimental configurations used in the evaluation. Let R , W , and O denote the request count, fixed logical beam width, and total number of output tokens, respectively. Prefill selects the first output token, leaving $T = O - 1$ decode calls. The full-model experiment uses the 0.8B Qwen3.5 model, with 6 full-attention layers and 18 GDN layers. We set R to 4, W to 256, the prompt length to 1K/5K, and O to 3–7. The operator suite covers 405 shapes over $R = 4/8/16$, $W = 128/256/512$, $T = 2-6$, and nine recurrent geometries with $H_K = 4-32$, $H_V = 4-64$, $K = 128-256$, and $V = 128-512$. On identical inputs and reorder-and-duplicate lineages, we pair OneLA with the unmodified vLLM/SGLang FullState and ReplaySSM kernels, FlashInfer, and TensorRT-LLM FullState on their natively supported shapes [7, 8, 19, 23].

4.2 Full-Model Decode-Forward Performance

To show the end-to-end performance improvement when using OneLA, Figure 6 reports the decode-forward GPU time of the complete model across the two prompt lengths. We compare four execution paths. vLLM FullState and vLLM ReplaySSM serve as native framework references. Controlled FullState uses the same decode pipeline as OneLA while retaining conventional per-beam FullState execution, whereas OneLA is our full system.

Controlled FullState remains within 6.9–9.3% of native vLLM FullState, validating that our controlled implementation closely reproduces the framework baseline. Replacing its

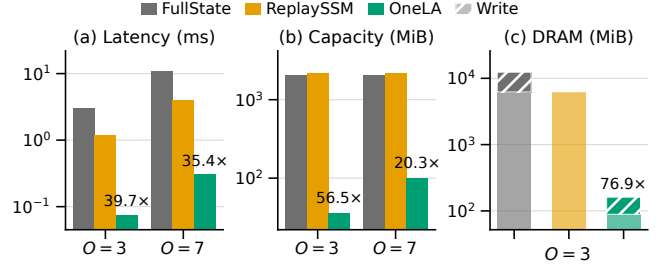


Figure 8. Recurrent-state efficiency breakdown.

recurrent state computation with OneLA yields a 1.54–2.46 $\times$ reduction in cumulative decode-forward time across all evaluated prompt and output lengths, and OneLA consistently outperforms both vLLM execution paths. These results show that OneLA’s recurrent-state optimization translates into substantial full-model decode-forward performance gains.

4.3 GR GDN Attention Generality

We next isolate the GDN attention operator to evaluate whether OneLA’s latency and recurrent-state capacity advantages persist across diverse workloads and recurrent-state configurations. For each paired workload, Figure 7 reports baseline/OneLA ratios for captured latency and persistent recurrent-state capacity in GPU memory. The evaluation covers all workloads natively supported by each baseline, where boxes indicate the median and interquartile range, and whiskers span the 5th–95th percentiles.

Figure 7(a) shows that OneLA consistently reduces GDN attention latency across the supported workloads. Across all paired shapes, median speedups are 40.5 $\times$ and 42.5 $\times$ over vLLM and SGLang FullState, 17.5 $\times$ over both ReplaySSM paths, and 41.5 $\times$ and 46.3 $\times$ over FlashInfer and TensorRT-LLM. At the largest Qwen geometry ($R = 16$, $W = 512$, $H_K = H_V = 16$, $K = V = 128$), OneLA takes 0.212 ms at $O = 3$ and 0.990 ms at $O = 7$, yielding 16.0–56.5 $\times$ speedups over the four baselines supporting this shape. Across all 405 OneLA cases, the maximum absolute output error against FP32 FullState is 4.88×10^{-4} , confirming numerical accuracy.

Figure 7(b) shows similarly consistent reductions in persistent recurrent-state capacity. Across the paired workloads, OneLA reduces the median required capacity by up to 42.5 $\times$ relative to the baselines, demonstrating that GTRs substantially reduce persistent GPU-memory requirements across different workload and recurrent-state configurations. Together, these results show that OneLA’s GDN attention latency and recurrent-state capacity advantages generalize across request counts, beam widths, decode lengths, and all nine evaluated recurrent-state geometries.

4.4 Sources of Performance Improvement

At the Qwen geometry with $R = 4$ and $W = 512$, we isolate the complete recurrent-state path and break down its performance through three complementary measurements: recurrent-path latency, persistent-state capacity and logical

writes, and physical DRAM traffic. Together, these measurements connect OneLA’s compact state representation to the reductions in memory movement that drive its execution-time improvement.

Figure 8(a) first shows the resulting recurrent-state-path latency, including each path’s explicit beam-state maintenance. At $O = 3$ and $O = 7$, OneLA takes only 0.0755 and 0.308 ms, whereas vLLM ReplaySSM takes 1.20 and 4.06 ms and vLLM FullState 3.00 and 10.9 ms, corresponding to 39.7× and 35.4× reductions relative to FullState.

Further, Figure 8(b) shows that the reductions in persistent state and state writeback contribute to the improvement. OneLA requires only 36.3 and 101 MiB of persistent recurrent-state capacity at $O = 3$ and $O = 7$, corresponding to 56.5× and 20.3× reductions relative to FullState, respectively. This reduction follows from replacing per-beam full-state matrices with one request-level S_{ctx} together with compact branch-specific GTRs. The same representation also substantially reduces state writes: OneLA writes only 32.3 and 96.8 MiB at $O = 3$ and $O = 7$, respectively, achieving a 127× reduction relative to FullState at both endpoints by appending compact GTRs instead of writing back full states.

Finally, Figure 8(c) shows that the reduced state storage and writeback translate into substantially lower physical DRAM traffic. At $O = 3$, OneLA moves only 161 MiB of data, exhibiting 76.9× and 39.9× reductions relative to FullState and ReplaySSM, respectively. This reduction comes from cross-beam reuse of S_{ctx} , which avoids repeated state reads and full-state writeback. Consequently, OneLA reaches an arithmetic intensity [17] of 40.0 FLOP/byte, 45.6× and 20.6× higher than FullState and ReplaySSM, respectively, directly explaining the recurrent-state-path latency improvement.

5 Conclusion

We present OneLA, an efficient linear-attention decoding framework for large-beam generative recommendation. By replacing per-beam recurrent states with a shared post-prefill state, compact transition records, and lightweight ancestry, OneLA enables on-chip projection replay and cross-beam state reuse. OneLA achieves 1.54–2.46× end-to-end speedups and substantial operator-level gains, demonstrating the effectiveness of compact state representation for dynamic large-beam decoding.

References

- [1] Ben Chen, Xian Guo, Siyuan Wang, Zihan Liang, Yufei Ma, Yue Lv, Chenyi Lei, Yuqing Ding, Wenwu Ou, Han Li, and Kun Gai. 2026. OneSearch: A Preliminary Exploration of the Unified End-to-End Generative Framework for E-commerce Search. In *Proceedings of the 43rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 306)*. PMLR. <https://openreview.net/forum?id=JKGgHY9FKa>
- [2] 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 *Proceedings of the ACM Web Conference 2026*. ACM, New York, NY, USA, 7989–7998. doi:10.1145/3774904.3792811
- [3] Ruining He, Lukasz Heldt, Lichan Hong, Raghunandan Keshavan, Shifan Mao, Nikhil Mehta, Zhengyang Su, Alicia Tsai, Yueqi Wang, Shao-Chuan Wang, et al. 2026. Plum: Adapting pre-trained language models for industrial-scale generative recommendations. In *Proceedings of the ACM Web Conference 2026*. 8093–8104.
- [4] Yupeng Hou, Jianmo Ni, Zhankui He, Naveen Sachdeva, Wang-Cheng Kang, Ed H. Chi, Julian McAuley, and Derek Zhiyuan Cheng. 2025. ActionPiece: Contextually Tokenizing Action Sequences for Generative Recommendation. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 24004–24024. <https://proceedings.mlr.press/v267/hou25f.html>
- [5] Jie Jiang, Hongbo Tang, Wenjie Wu, Yangru Huang, ZhenMao Li, Qian Li, Changping Wang, Jun Zhang, and Huan Yu. 2026. S-GRec: Personalized Semantic-Aware Generative Recommendation with Asymmetric Advantage. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (Republic of Korea) (KDD ’26)*. Association for Computing Machinery, New York, NY, USA, 7465–7476. doi:10.1145/3770855.3818473
- [6] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 5156–5165. <https://proceedings.mlr.press/v119/katharopoulos20a.html>
- [7] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165
- [8] NVIDIA. 2026. TensorRT-LLM. <https://github.com/NVIDIA/TensorRT-LLM> Evaluated tag v1.3.0rc23 (d41ab33).
- [9] OneRec Team et al. 2026. OneReason Technical Report. doi:10.48550/ARXIV.2606.06260
- [10] Qwen Team. 2025. Qwen3-Next: Towards Ultimate Training and Inference Efficiency. Qwen Blog. <https://qwen.ai/blog?id=qwen3-next> Accessed August 2026.
- [11] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc. doi:10.52202/075280-0452
- [12] Qingxiao Sun, Tongxuan Liu, Shen Zhang, Siyu Wu, Peijun Yang, Haotian Liang, Menxin Li, Xiaolong Ma, Zhiwei Liang, Ziyi Ren, Minchao Zhang, Yifan Wang, Xinyu Liu, Ke Zhang, Hailong Yang, and Depei Qian. 2025. xGR: Efficient Generative Recommendation Serving at Scale. doi:10.48550/ARXIV.2512.11529
- [13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin.

2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 5998–6008. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
- [14] Huimu Wang, Xingzhi Yao, Yiming Qiu, Qinghong Zhang, Haotian Wang, Yufan Cui, Songlin Wang, Sulong Xu, and Mingming Li. 2026. Towards Efficient and Generalizable Retrieval: Adaptive Semantic Quantization and Residual Knowledge Transfer. doi:10.48550/ARXIV.2602.23978
- [15] Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2024. Learnable Item Tokenization for Generative Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. ACM, New York, NY, USA, 2400–2409. doi:10.1145/3627673.3679569
- [16] Ye Wang, Jiahao Xun, Minjie Hong, Jieming Zhu, Tao Jin, Wang Lin, Haoyuan Li, Linjun Li, Yan Xia, Zhou Zhao, and Zhenhua Dong. 2024. EAGER: Two-Stream Generative Recommender with Behavior-Semantic Collaboration. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 3245–3254. doi:10.1145/3637528.3671775
- [17] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (2009), 65–76. doi:10.1145/1498765.1498785
- [18] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. 2025. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=r8H7xhYPwz>
- [19] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. doi:10.48550/ARXIV.2501.01005
- [20] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 58484–58509. <https://proceedings.mlr.press/v235/zhai24a.html>
- [21] Shuqi Zhao, Jingsong Su, Xiang Liu, Xingzhi Yao, Yiming Qiu, Huimu Wang, Liang Lin, Pengbo Mo, Mingming Li, Jiao Dai, Jizhong Han, and Songlin Hu. 2026. HoloRec: Holistic Encoding and Interleaved Reasoning for Generative Recommendation. doi:10.48550/ARXIV.2606.15331
- [22] Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. 2024. Adapting Large Language Models by Integrating Collaborative Semantics for Recommendation. In *2024 IEEE 40th International Conference on Data Engineering*. IEEE, 1435–1448. doi:10.1109/ICDE60146.2024.00118
- [23] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2023. SGLang: Efficient Execution of Structured Language Model Programs. doi:10.48550/ARXIV.2312.07104
- [24] Guorui Zhou, Jiaxin Deng, Jinghao Zhang, Kuo Cai, Lejian Ren, Qiang Luo, Qianqian Wang, Qigen Hu, Rui Huang, Shiyao Wang, et al. 2025. OneRec Technical Report. doi:10.48550/ARXIV.2506.13695