# FReS: Factorized Recurrent State for Efficient Inference with Gated Delta-Rule Linear Attention

**Anonymous authors**
Paper under double-blind review

## Abstract

Recurrent gated delta-rule linear attention has been widely adopted in recent hybrid language models, such as Gated DeltaNet (GDN) in the Qwen3 series and Kimi Delta Attention (KDA) in Kimi K3. By replacing the context-length-dependent key–value cache with a fixed-size recurrent state table, these architectures enable efficient long-context inference. However, these models maintains a dense $d_k \times d_v$ recurrent state in each of its layers and heads for every active sequence. As the number of concurrent sequences increases, the replicated recurrent states lead to aggregate memory usage and traffic to grow linearly with concurrency, making GPU memory capacity and bandwidth critical bottlenecks for high-concurrency serving. To address these challenges, we propose **FReS**, a training-free inference method. Building on the observation that recurrent states exhibit low effective ranks, which indicates the potential for compression. FReS leverages low-rank matrix factorization to represent recurrent states compactly, reducing memory storage and traffic. FReS further employs an online, subspace-aware state maintenance to support incremental decoding while preserving most state insights and limiting error accumulation. Experiments show that FReS largely preserves dense-state quality while achieving up to $\mathbf{1.91}\times$ the throughput of the dense-state baseline under high concurrency.

## 1 Introduction

Recurrent gated delta-rule linear attention has recently been adopted in practical hybrid language models (Yang et al., 2023; De et al., 2024). Qwen3.5-3.8 series combines Gated DeltaNet (GDN) (Yang et al., 2025) with gated-attention layers, while Kimi K3 (Team et al., 2026) employs Kimi Delta Attention (KDA) (Team et al., 2025), a fine-grained extension of the gated delta rule. These architectures replace the context-length-dependent key-value cache of softmax attention (Vaswani et al., 2017; Sun et al., 2023) with a fixed-size recurrent state (Katharopoulos et al., 2020), enabling efficient long-context inference while introducing a persistent state for every active sequence.

Despite their differences in gating granularity, both GDN and KDA maintain a dense $d_k \times d_v$ recurrent state at every layer and attention head. Under concurrent serving, this per-sequence state is replicated across requests, causing aggregate state storage and recurrent-state traffic to grow linearly with concurrency. Consequently, recurrent-state storage and movement can become important capacity and bandwidth bottlenecks for high-concurrency inference.

Prior work has reported low-rank structure in the recurrent states of delta-rule linear-attention architectures (Parnichkun et al., 2025; Sun et al., 2026). We confirm and extend these observations across GDN- and KDA-based models, finding strong spectral concentration. Our experiments further show that preserving the components that capture most of the state's energy largely maintain task accuracy, whereas retaining arbitrary or low-energy components causes substantial degradation. These suggests that dense states can be represented more compactly. However, exploiting this structure during inference is nontrivial: a practical compression method must preserve dominant state information, support token-wise updates, control errors accumulated through repeated recompression, and avoid excessive decoding overhead (Nazari & Rusch, 2026).



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address these challenges, we propose **FReS**, a training-free inference method that leverages low-rank matrix factorization to represent recurrent states compactly, reducing persistent state storage and recurrent-state traffic. FReS further employs online, subspace-aware state maintenance to support incremental decoding while preserving dominant state information and limiting error accumulation.

We evaluate **FReS** against the corresponding dense-state implementations of GDN and KDA. FReS largely preserves the generation quality of the dense baseline across the evaluated benchmarks. On the sampled ShareGPT workload, FReS extends the serviceable capacity beyond the dense baseline and achieves up to $\mathbf{1.91}\times$ the throughput of the dense implementation under high concurrency.

Our contributions are:

- We characterize recurrent states in GDN- and KDA-based models across different context lengths, revealing strong spectral concentration and showing that preserving dominant spectral directions is critical for maintaining task accuracy.
- We introduce **FReS**, a training-free method for efficient recurrent-state inference. FReS maintains a factorized state representation that supports direct rank-one updates during decoding and applies randomized subspace recompression to bound the state rank without reconstructing the dense state.
- We evaluate FReS across multiple GDN- and KDA-based language models and workloads. FReS largely preserves dense-state quality while achieving up to $91\%$ higher throughput on sampled ShareGPT workloads under high concurrency.

## 2 Background

### 2.1 Gated Delta-Rule Linear Attention

Gated DeltaNet (GDN) and Kimi Delta Attention (KDA) are recurrent linear-attention mechanisms that maintain a matrix-valued state instead of storing all previous key-value pairs (Schlag et al., 2021). Given the query, key, and value vectors $q_t, k_t \in \mathbb{R}^{d_k}$ and $v_t \in \mathbb{R}^{d_v}$, the recurrent state $H_t \in \mathbb{R}^{d_k \times d_v}$ is updated as

$$H_t = \left(I - \beta_t k_t k_t^\top\right) D_t H_{t-1} + \beta_t k_t v_t^\top, \qquad o_t = H_t^\top q_t, \tag{1}$$

where $\beta_t$ controls the delta update and $D_t$ applies state decay. GDN uses a scalar decay, whereas KDA uses a channel-wise diagonal decay (Team et al., 2025). Despite this difference, both mechanisms maintain a fixed-size $d_k \times d_v$ recurrent state and write a rank-one update $\beta_t k_t v_t^\top$ at each decoding step (Dao & Gu, 2024).

### 2.2 Mathematical Tools for Low-Rank Approximation

**Singular Value Decomposition.** For $X \in \mathbb{R}^{m\times n}$, the singular value decomposition is

$$X = U\Sigma V^\top. \tag{2}$$

Keeping the largest $r$ singular values gives the rank-$r$ approximation

$$X_r = U_r \Sigma_r V_r^\top, \tag{3}$$

which is optimal under the Frobenius norm by the Eckart–Young theorem (Eckart & Young, 1936):

$$\|X - X_r\|_F^2 = \sum_{i>r} \sigma_i^2. \tag{4}$$

**Randomized Subspace Iteration.** To efficiently approximate the dominant singular subspace, randomized subspace iteration (Halko et al., 2011; Martinsson & Tropp, 2020) starts with

$$Y_0 = X\Omega, \qquad \Omega \in \mathbb{R}^{n\times r}, \tag{5}$$

and applies $q$ power iterations:

$$Y_{j+1} = \mathrm{orth}(XX^\top Y_j), \qquad j = 0, \ldots, q-1. \tag{6}$$

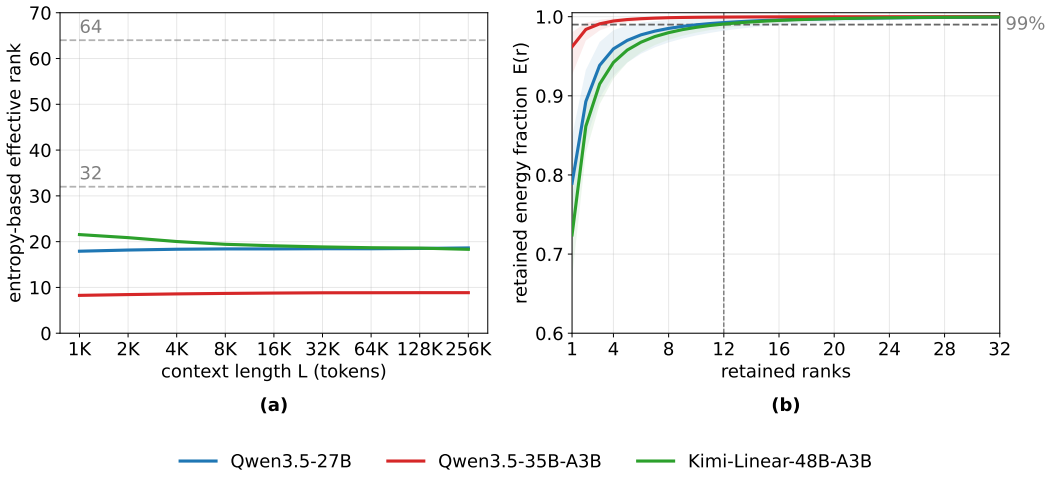



Figure 1: Low effective rank and concentrated spectral energy in recurrent states whose maximum rank is 128. **(a)** Effective rank across context lengths, averaged over heads and layers within each document. **(b)** Retained energy versus retained rank at 256K tokens: median and 40th–60th percentiles over states from all heads, layers, and documents.

Letting $Q = \mathrm{orth}(Y_q)$, the matrix can be projected as

$$X \approx QQ^\top X. \tag{7}$$

The power iterations enhance the dominant singular directions while avoiding the cost of computing an exact SVD.

## 3 Spectral and Functional Analysis of Recurrent States

Although GDN and KDA replace the context-length-dependent KV cache with a fixed-size recurrent state, each active sequence still maintains a dense $d_k \times d_v$ matrix in every layer and head. Under high-concurrency inference, these states remain a source of storage and memory-traffic overhead. This raises the question of whether the dense representation contains substantial redundancy relative to the task-relevant information.

We examine whether recurrent states can be compressed while preserving task-relevant information. We first characterize their spectral concentration and required approximation rank, and then evaluate how the choice of retained singular components affects task accuracy. Together, these analyses provide the empirical motivation for FReS.

### 3.1 Spectral Concentration and Approximation Rank

**Low effective rank across models and contexts.** For singular values $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$, we measure spectral concentration using the entropy-based effective rank,

$$r_{\mathrm{eff}}(H) = \exp\left(-\sum_{i=1}^{d} p_i \log p_i\right), \qquad p_i = \frac{\sigma_i}{\sum_j \sigma_j}. \tag{8}$$

**Rank required for energy retention.** Effective rank characterizes the spectrum, whereas an approximation budget requires a specified reconstruction tolerance. We therefore measure the retained energy and the minimum rank needed to reach a target $\eta$:

$$E_H(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{d} \sigma_i^2}, \qquad r_\eta(H) = \min\{r : E_H(r) \geq \eta\}. \tag{9}$$

We analyze recurrent states from 60 documents sampled from five domains in LongBench-v2 (Bai et al., 2025) across several GDN- and KDA-based models and context lengths. The mean effective

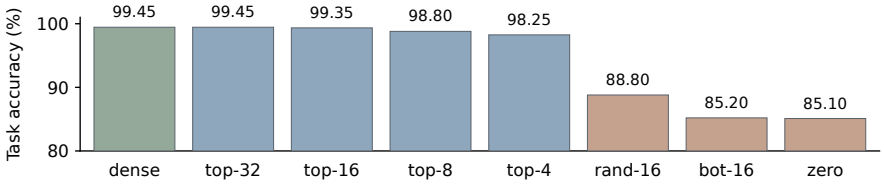



Figure 2: Task accuracy of Kimi-Linear-48B-A3B on the *niah_multivalue* task of RULER-16K after one-time recurrent-state truncation. *dense* is the unmodified baseline; *top-$r$*, *rand-16*, *bot-16*, and *zero* retain the $r$ largest, 16 random, 16 smallest, and no recurrent state components, respectively.

ranks are approximately 8–22, far below the full dimension of 128, and change only modestly as context length increases (Figure 1(a)). At 256K tokens, the median rank required to retain 99% of the spectral energy is 11, 3, and 12 for Qwen3.5-27B, Qwen3.5-35B-A3B, and Kimi-Linear-48B-A3B, respectively. Rank 16 retains more than 99.5% of the median energy in all models, while rank 32 retains more than 99.9% (Figure 1(b)). These results indicate that the recurrent states admit compact approximation ranks.

### 3.2 Task Accuracy under State Truncation

To assess the functional importance of the retained singular components, we perform a one-time state truncation experiment on a multi-value retrieval task. As shown in Figure 2, top-32 matches the dense baseline and performance declines progressively at top-8 and top-4. In contrast, random or bottom-component selection, as well as removing the recurrent state, causes substantial degradation.

These results show that effective compression must preserve dominant singular components. Since the experiment applies truncation only once, an online method must additionally support token-wise updates, control repeated compression error, and maintain a bounded state size during decoding.

## 4 FReS: Factorized Recurrent State

The low-rank structure of recurrent states suggests replacing the dense state with a bounded-rank approximation. FReS implements this idea while preserving token-wise recurrent updates without reconstructing the dense state. As illustrated in Figure 3, FReS factorizes the terminal recurrent state at the prefill-to-decode transition, and maintains the state in factorized form during decoding. Each new token appends one factor pair, and periodic recompression restores the factor width to the target rank.

### 4.1 Factorized State Representation and Online Updates

After the prefill-to-decode transition, FReS represents the recurrent state at step $t$ as

$$\widehat{H}_t = U_t W_t^\top, \qquad U_t \in \mathbb{R}^{d_k \times r_t}, \quad W_t \in \mathbb{R}^{d_v \times r_t}, \tag{10}$$

where $r_t$ is the current number of active factor columns. This representation also allows the recurrent output to be computed directly in factor space:

$$\widehat{o}_t = \widehat{H}_t^\top q_t = W_t(U_t^\top q_t). \tag{11}$$

Our design objective is to keep $r$ small while preserving the dominant subspace of $H$ and controlling the approximation error introduced by rank reduction (Ghashami et al., 2016). We will discuss it in Section 4.2 later.

**Rank-One Update.** Figure 4 illustrates how one decoding token updates the two factors. The decay is applied to the left factor, while the right factor remains unchanged. The new delta correction is then appended as one column to each factor.

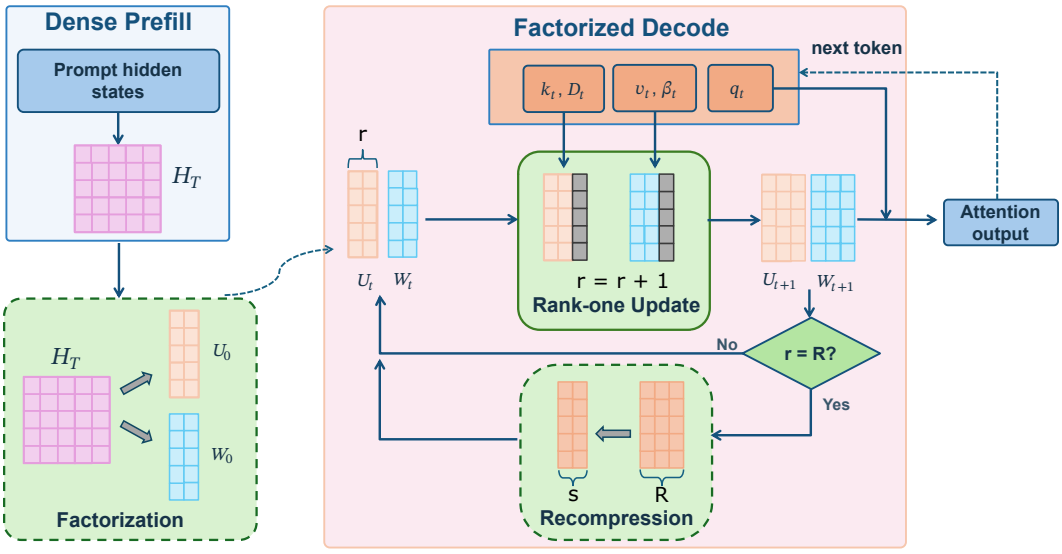



Figure 3: Overview of FReS. The dense recurrent state produced during prefill is factorized before decoding. During factorized decoding, each token is incorporated through a rank-one update. When the active factor width reaches the rank budget $R$, it will be recompressed to $s$.

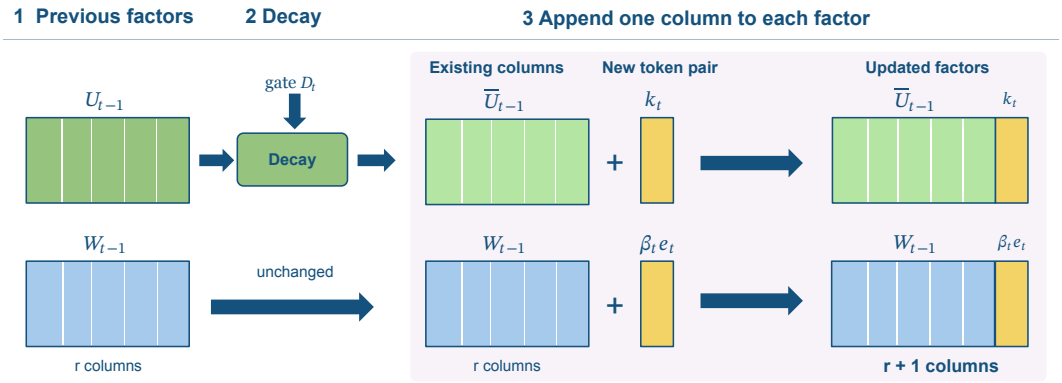



Figure 4: Factorized token update. The decay operator is applied to the existing left factor, and the new rank-one delta correction is appended as one column to each factor, increasing the active width from $r$ to $r + 1$.

Let the approximate recurrent state before processing token $t$ be $\widehat{H}_{t-1} = U_{t-1}W_{t-1}^{\top}$. After applying the decay operator, the left factor becomes

$$\overline{U}_t = D_t U_{t-1}. \tag{12}$$

The delta-rule correction can then be computed entirely in factor space as

$$e_t = v_t - W_{t-1}(\overline{U}_t^{\top} k_t). \tag{13}$$

Consequently, the updated state is

$$\widehat{H}_t = [\overline{U}_t,\ k_t][W_{t-1},\ \beta_t e_t]^{\top}. \tag{14}$$

Thus, each decoding step incorporates the new token by appending one column to each factor, without reconstructing the dense recurrent state. The factor width grows by one after every update, and is periodically reduced by the recompression procedure described below. The full derivation and dimensional verification are provided in Appendix B.1.

### 4.2 Factor Construction and Online Recompression

FReS represents the recurrent state using a pair of low-rank factors. Although truncated SVD provides the optimal rank-$s$ approximation, it remains computationally expensive when invoked repeatedly during decoding. We therefore use randomized subspace iteration as an efficient approximation. The resulting projection is evaluated directly in factor space.

**Randomized Factor-Space Recompression.** Following the randomized subspace iteration described in Section 2.2, FReS approximates the dominant subspace of the recurrent state without computing an exact SVD. For a factorized state $X = UW^\top$, the required matrix products are evaluated directly from the factors:

$$X\Omega = U(W^\top\Omega), \qquad X^\top Q = W(U^\top Q). \tag{15}$$

Given the estimated rank-$s$ subspace $Q \in \mathbb{R}^{d_k \times s}$, the recompressed state is represented as

$$U' = Q, \qquad W' = W(U^\top Q). \tag{16}$$

This factor-space implementation avoids explicitly constructing the dense state while preserving its dominant subspace. The same operation is used both to initialize the factorized state after prefill and to recompress the active factors during decoding.

### 4.3 Phase-Asymmetric Inference Procedure

FReS leaves the original prefill computation unchanged and applies factorization to the persistent recurrent state used during decoding. This phase-asymmetric design avoids modifying the prefill path while reducing the state storage and update cost in the decode phase.

**Prefill-to-Decode Transition.** After the prompt has been processed by the original recurrent computation, FReS projects the terminal state to rank $s$ using the factor-space recompression operator described in Section 4.2. The resulting factors initialize the persistent state for decoding.

**Bounded-Rank Factorized Decode.** During decoding, FReS updates the recurrent state directly in factor space. Each token appends one factor pair, increasing the active factor width by one. When the width reaches the rank budget $R$, FReS recompresses the active factors to width $s$, leaving $R - s$ slots for subsequent updates. The persistent state therefore requires $O(R(d_k + d_v))$ storage per request, layer, and head.

## 5 Experiment

### 5.1 Experimental Setup

**Models and Baselines.** We evaluate FReS on three pretrained hybrid language models: Qwen3.5-35B-A3B, Qwen3.5-27B, and Kimi-Linear-48B-A3B, covering different model scales, recurrent-attention types, and proportions of GDN- or KDA-based layers. We compare FReS with **Dense**, the original full-rank implementation, and **FIFO**, a factorized-state baseline that overwrites existing factor columns with new ones instead of recompressing the state. Exact SVD recompression is used only as a reference in the ablation study. Detailed model specifications are provided in Appendix A.1.

**Benchmarks and Workloads.** We evaluate generation quality on WikiText-2, RULER (Hsieh et al., 2024), LongBench (Bai et al., 2025), MMLU-Pro (Wang et al., 2024), GPQA-Diamond (Rein et al., 2023), MATH-500 (Hendrycks et al., 2021), and GSM8K (Cobbe et al., 2021). We report perplexity on WikiText-2, token-level F1 on LongBench, and accuracy on the remaining benchmarks. For serving evaluation, we sample variable-length conversations from ShareGPT while preserving the original prompt and response length of each request. Detailed evaluation protocols and workload configurations are provided in Appendix A.2.

Table 1: Generation quality across models and benchmarks.

| | Perplexity | Long-Context | | General Reasoning | | Math Reasoning | |
|---|---|---|---|---|---|---|---|
| **Method** | **WikiText-2↓** | **RULER↑** | **LongBench↑** | **MMLU-Pro↑** | **GPQA-D↑** | **MATH-500↑** | **GSM8K↑** |
| *Qwen3.5-35B-A3B* | | | | | | | |
| Dense | 6.52 | 96.53 | 58.83 | 78.86 | 80.81 | 85.6 | 95.6 |
| FIFO | 9.25 | 83.83 | 29.06 | 64.57 | 52.02 | 73.2 | 88.8 |
| **FReS** | **6.83** | **96.86** | **58.57** | **74.57** | **75.76** | **85.4** | **95.6** |
| *Qwen3.5-27B* | | | | | | | |
| Dense | 6.82 | 94.27 | 60.79 | 78.00 | 82.32 | 88.2 | 96.4 |
| FIFO | 7.63 | 80.01 | 48.38 | 74.57 | 67.17 | 81.2 | 95.2 |
| **FReS** | **7.39** | **96.81** | **61.12** | **78.86** | **79.80** | **84.2** | **96.0** |
| *Kimi-Linear-48B-A3B* | | | | | | | |
| Dense | 7.45 | 95.65 | 55.53 | 58.86 | 37.37 | 69.0 | 90.8 |
| FIFO | 11.35 | 94.22 | 51.94 | 21.43 | 15.66 | 18.6 | 46.0 |
| **FReS** | **7.60** | **95.89** | **54.59** | **58.57** | **45.45** | **69.6** | **90.0** |

**Implementation Details.** All experiments use bfloat16 precision and vLLM 0.18.0 (Kwon et al., 2023). Quality and serving experiments run on two NVIDIA A800-SXM4-80GB GPUs with tensor parallelism of degree two, while operator microbenchmarks use a single A800 GPU. FReS uses a rank budget of $R = 32$ and retains $s = 16$ factor columns after recompression.

### 5.2 Quality Evaluation

Table 1 compares Dense, FIFO, and FReS in three hybrid language models and seven benchmarks. FReS consistently outperforms FIFO in all models evaluated, substantially reducing the degradation caused by naive state truncation. This degradation is particularly severe on Kimi Linear, which is consistent with KDA's channel-wise gating: unlike the scalar decay in GDN, it does not guarantee that the oldest factor columns are the most heavily decayed, making FIFO replacement less reliable.

Compared with Dense, FReS achieves higher RULER scores on all three models, while its LongBench scores remain within 0.94 percentage points of the dense baseline. Its perplexity also changes by less than 0.57 points across models. On reasoning and mathematical benchmarks, FReS remains competitive with Dense and improves over FIFO by large margins. Overall, FReS offers a favorable quality–memory trade-off under the same rank budget.

### 5.3 Operator Performance

We benchmark the GDN decode kernel in a standalone microbenchmark using the per-GPU configuration corresponding to Qwen3.5-35B-A3B under tensor parallelism of degree two. As shown in Figure 5, FReS consistently achieves lower decoding latency than dense implementation. Its speedup increases with the batch size, from $1.66\times$ at batch size 64 to $1.92\times$ at 512. This result shows that factorized recurrent states reduce state-movement overhead and become increasingly beneficial as the number of concurrently decoded sequences grows.

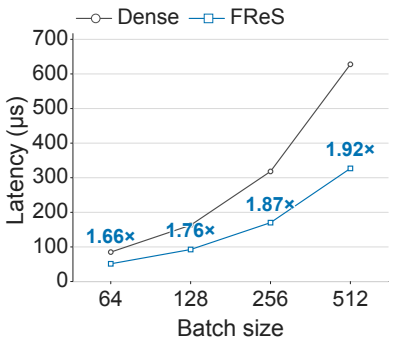



Figure 5: GDN decode-kernel microbenchmark comparing the latency of Dense and FReS across batch sizes. The annotations indicate the speedup of FReS over Dense.

### 5.4 Serving Performance

We evaluate end-to-end serving throughput on a sampled ShareGPT workload. We vary the target concurrency $C$ over 64, 128, 256, 512, and 1,024, where $C$ denotes the maximum number of si-

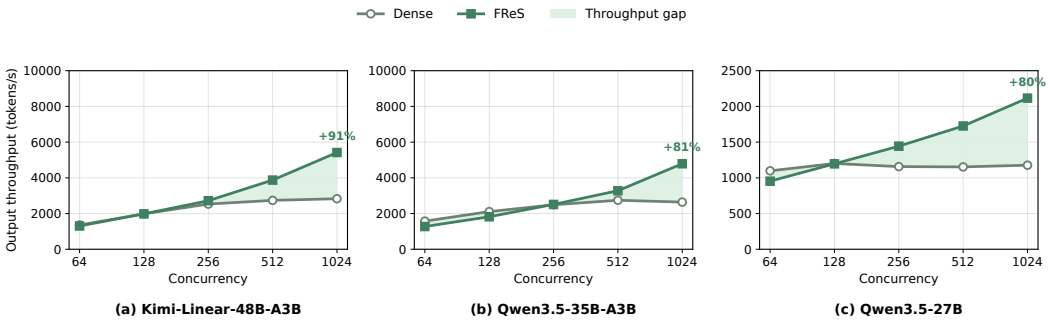



Figure 6: End-to-end output throughput on the sampled ShareGPT workload. Each panel compares the dense recurrent-state baseline with FReS across concurrency levels using variable-length requests. The shaded regions indicate the throughput gap between the two methods.

multaneously active requests. We set the request rate to infinity, yielding a closed-loop workload in which a new request is issued whenever an active request finishes, subject to the concurrency limit $C$. Before collecting measurements, we warm up the server with $2C$ requests at each concurrency level; these warm-up requests are excluded from the reported throughput. we then process $4C$ variable-length conversations sampled from ShareGPT, preserving the original prompt and response lengths and enabling natural end-of-sequence termination. All methods use identical workload and serving configurations, and we report output throughput in tokens per second.

Figure 6 reports the results for the sampled ShareGPT workload. At low concurrency, FReS incurs a modest throughput penalty because the overhead of factor-space updates and periodic recompression has not yet been offset by the reduction in recurrent-state traffic. As concurrency increases, FReS consistently outperforms the dense-state baseline, achieving maximum throughput gains of $91\%$, $81\%$, and $80\%$ on Kimi-Linear-48B-A3B, Qwen3.5-35B-A3B, and Qwen3.5-27B, respectively. These results show that FReS provides the greatest benefit in the high-concurrency regime, where recurrent-state memory traffic becomes a dominant serving bottleneck.

Table 2: Relative recurrent-state and output errors of GDN-1.3B under different sequence lengths.

| $T$ | State Error | | | Output Error | | |
|---|---|---|---|---|---|---|
| | **FIFO** | **FReS** | **SVD** | **FIFO** | **FReS** | **SVD** |
| 128 | 0.965 | 0.174 | **0.160** | 0.706 | 0.124 | **0.117** |
| 256 | 0.954 | 0.272 | **0.252** | 0.768 | 0.154 | **0.139** |
| 512 | 0.919 | 0.341 | **0.319** | 0.797 | 0.205 | **0.193** |
| 1024 | 0.947 | 0.426 | **0.410** | 0.834 | 0.268 | **0.251** |
| 2048 | 0.956 | 0.467 | **0.444** | 0.857 | 0.305 | **0.283** |

## 5.5 Ablation Studies

We conduct controlled ablations to examine the effectiveness of subspace-preserving recompression and the impact of the rank budget. These experiments use the lightweight GDN-1.3B model, which has $K = V = 256$ and four recurrent heads per layer, together with an FP32 reference recurrence. Since exact SVD provides an optimal approximation, we use it as a reference method in these ablations.

Table 3: Relative recurrent-state error under different rank budgets at $T = 2048$.

| $R$ | **FIFO** | **FReS** | **SVD** |
|---|---|---|---|
| 32 | 0.956 | 0.467 | **0.444** |
| 64 | 0.920 | 0.354 | **0.335** |
| 128 | 0.888 | 0.270 | **0.222** |

**Recompression Strategy Comparison.** Table 2 compares FIFO, FReS, and exact SVD recompression. FReS consistently achieves substantially lower recurrent- state and output errors than FIFO, while remaining close to the SVD reference. At $T = 2048$, FReS obtains state and output errors of 0.467 and 0.305, compared with 0.956 and 0.857 for FIFO and 0.444 and 0.283 for SVD. This result shows that preserving the dominant state subspace is more effective than directly discarding old factor columns.

**Rank Budget Ablation.** We further evaluate different rank budgets with $s = R/2$. As shown in Table 3, increasing $R$ consistently reduces the recurrent-state error for all methods. FReS reduces its error from 0.467 at $R = 32$ to 0.270 at $R = 128$, while remaining close to SVD and substantially outperforming FIFO at every rank. These results demonstrate that FReS provides an effective quality–rank trade-off for factorized recurrent states.

## 6 Related Work

**Linear Attention and Gated Delta Rules.** Linear attention reformulates attention as a recurrent computation over a matrix-valued state, enabling linear-time sequence processing and constant-size inference memory with respect to sequence length (Gu & Dao, 2023). DeltaNet (Yang et al., 2024) further introduces error-correcting updates (Schlag et al., 2021), while Gated DeltaNet combines delta rules with input-dependent forgetting for more expressive and efficient recurrent modeling. Our work does not modify the recurrence or model architecture, but focuses on reducing the serving cost of the persistent state in pretrained Gated DeltaNet models.

**Rank Structure and Recurrent-State Compression.** Recent studies suggest that the nominal size of recurrent states can substantially exceed their effective memory utilization. Effective state-size and state-rank analyses (Parnichkun et al., 2025; Sun et al., 2026) reveal low-rank or rank-stratified structures in linear-attention states. Other work reduces recurrent-state cost through structured key/query pruning (Nazari & Rusch, 2026) or mixed-precision quantization (Zhang et al., 2026). These methods either permanently change the state dimension or reduce the numerical precision of state elements. Our method instead compresses the state through dynamic low-rank factorization entirely at inference time, without training or fine-tuning. It preserves the pretrained model's weights and architecture, maintaining a bounded-rank state for each request through periodic recompression during decoding.

## 7 Conclusion

We introduced FReS, a training-free method for low-rank inference in GDN and KDA based language models. Motivated by their strong spectral concentration, FReS combines factorized rank-one updates with randomized subspace recompression to maintain a bounded-rank state without reconstructing the dense matrix. This reduces persistent state storage from $O(d_k d_v)$ to $O(R(d_k + d_v))$. Experiments demonstrate competitive quality relative to dense-state baselines, consistent improvements over FIFO truncation, and up to $63\%$ higher end-to-end serving throughput on the evaluated workloads. These results highlight the importance of preserving dominant state directions and establish factorized recurrent states as a practical approach to memory-efficient, high-concurrency inference.

### AI use statement

In this work, we used AI to aid and polish the writing, assist with the retrieval and discovery of related work, and draft selected sections of the paper. AI tools were used to improve the grammar, clarity, and organization of the manuscript, identify potentially relevant literature, and prepare initial drafts of certain sections.

All AI-assisted materials were critically reviewed, revised, and verified by the authors. We take full responsibility for the final content of this work, including all text, claims, citations, and other artifacts produced with the aid of AI tools.

## References

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53 (2):217–288, 2011.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joey Gonzalez, Hao Zhang, and Ion Stoica. vllm: Easy, fast, and cheap llm serving with pagedattention. *See https://vllm. ai/(accessed 9 August 2023)*, 2023.

Per-Gunnar Martinsson and Joel A Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

Philipp Nazari and T Konstantin Rusch. The key to state reduction in linear attention: A rank-based perspective. *arXiv preprint arXiv:2602.04852*, 2026.

Rom N Parnichkun, Neehal Tumma, Armin W Thomas, Alessandro Moro, Qi An, Taiji Suzuki, Atsushi Yamashita, Michael Poli, and Stefano Massaroli. Quantifying memory utilization with effective state-size. *arXiv preprint arXiv:2504.19561*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Ao Sun, Hongtao Zhang, Heng Zhou, Yixuan Ma, Yiran Qin, Tongrui Su, Yan Liu, Zhanyu Ma, Jun Xu, Jiuchong Gao, et al. State rank dynamics in linear attention llms. *arXiv preprint arXiv:2602.02195*, 2026.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Tao Zhang, Jianchao Tan, Pingwei Sun, Yanqi Yu, Zixu Jiang, Yuchen Xie, Xunliang Cai, and Ziqian Zeng. Damp: Decay-aware mixed-precision recurrent-state quantization. *arXiv preprint arXiv:2608.27513*, 2026.

# A Additional Experimental Details

## A.1 Model Configurations

In our experiments, three hybrid linear-attention models are evaluated. All three interleave linear-recurrent layers with a smaller number of interspersed full-attention layers; FReS is applied only to the recurrent layers, whose per-head state is $d_k \times d_v = 128 \times 128$ in every model. Table 4 summarizes the configurations.

## A.2 Evaluation Protocol

**Benchmarks and metrics.** Table 5 lists the evaluation suites, sample counts, few-shot settings, and generation budgets. Perplexity is measured on WikiText-2 (raw); long-context ability on RULER-8192 and LongBench (16k); general reasoning on MMLU-Pro and GPQA-Diamond; and math reasoning on MATH-500 and GSM8K. RULER is reported as the macro-average over its 12 sub-tasks excluding `cwe` (case-insensitive gold-substring recall, $N$=50 per sub-task); LongBench uses the normalized token-F1 with the maximum taken over multiple references.

**Sampling and answer extraction.** All generation uses greedy decoding (temperature $= 0$), the model-specific chat template with `enable_thinking=False`, and a minimum of 64 new tokens. Answer extraction is strict for the multiple-choice and boxed-answer suites—a response with no parseable marker is scored incorrect: MMLU-Pro and GPQA-Diamond require an "answer is (X)" marker and MATH-500 a complete `\boxed{}` (compared after LaTeX normalization, not symbolic equivalence). GSM8K takes the last "#### N", falling back to the last number in the output, compared at $10^{-6}$ tolerance. The generation budgets in Table 5 are enlarged relative to standard harnesses so that long chains of thought are not truncated before an answer marker is emitted; the GPQA budget is raised to 12288 for the Qwen models to keep the comparison truncation-fair. For Kimi-Linear the score does not improve at 12288 (its long-CoT non-parse rate stays ≈97/198, a genuine capability ceiling rather than truncation), so the 8192 budget is used to save compute.

| | Qwen3.5-27B | Qwen3.5-35B-A3B | Kimi-Linear-48B-A3B |
|---|---|---|---|
| Total / active params | 27B / 27B | 35B / 3B (MoE) | 48B / 3B (MoE) |
| Total layers | 64 | 40 | 27 |
| – recurrent layers | 48 | 30 | 20 |
| – full-attention layers | 16 | 10 | 7 |
| Recurrent op | Gated DeltaNet | Gated DeltaNet | Kimi Delta Attention |
| Full-attention op | GQA | GQA | MLA |
| Hidden size | 5120 | 2048 | 2304 |
| Recurrent key/value heads | 16 / 48 | 16 / 32 | 32 / 32 |
| Recurrent head dim ($d_k$=$d_v$) | 128 | 128 | 128 |
| Per-layer recurrent state ($H_v d_k d_v$) | $48\times128^2$ | $32\times128^2$ | $32\times128^2$ |
| Vocab size | 248,320 | 248,320 | 163,840 |

Table 4: Base model configurations.

| Suite | #Samples | Shots | Max new tokens | Metric |
|---|---|---|---|---|
| WikiText-2 (raw) | 16 windows | — | teacher-forced | decode-incremental PPL |
| RULER-8192 | 650 | 0 | data-driven ($\geq$64) | substr. recall (12-task, excl. cwe) |
| LongBench (16k) | 75 | 0 | data-driven ($\geq$64) | normalized token-F1 |
| MMLU-Pro | 350 | 5 | 2048 | exact-match |
| GPQA-Diamond | 198 | 0 | 8192$^\dagger$ | exact-match |
| MATH-500 | 500 | 0 | 4096 | \boxed{} match |
| GSM8K | 250 | 8 | 400 | numeric match |

Table 5: Quality evaluation suites and per-suite generation settings. $^\dagger$GPQA-Diamond uses 12288 max-new tokens for the Qwen models.

**Perplexity (decode path).** Since FIFO and FReS both run a dense prefill and differ only in the decode-time recurrent update, a prefill-only PPL collapses every scheme onto the dense value. We therefore report a *decode-incremental* perplexity: each window prefills a 64-token prefix and teacher-forces the remaining tokens one at a time through the decode kernel (reading each ground-truth log-probability before forcing that token), so every FIFO eviction and periodic recompression fires as in generation. We report the median per-window PPL at $T = 2048$ over 16 windows.

### A.3 Implementation and Serving Configuration

**Hardware and engine.** All experiments run on one node with 2×NVIDIA A800-SXM4-80GB, using tensor parallelism (TP= 2) on a vLLM 0.18.0 fork. Weights and activations are bf16 and the recurrent state is fp32; we use `gpu-memory-utilization`= 0.75, `--disable-custom-all-reduce`, and enabled CUDA graphs (piecewise for Kimi-Linear, whose MLA prefill cannot be captured as a single graph). Quality evaluation uses `max-num-seqs` 8–32 with `max-model-len` set per suite (8192, up to 32768 for RULER-16384); the serving benchmark uses `max-model-len`= 8192 with the batch bounded by the target concurrency.

**Recurrent-state schemes.** The three methods share identical base weights and differ only in the recurrent-state configuration (Table 6). Both low-rank schemes use a dense prefill compressed once at the hand-off to rank $r$=32; FIFO keeps the buffer as a ring with no recompression, while FReS periodically recompresses to $s$=16 retained directions.

**Serving benchmark.** End-to-end serving is measured on the ShareGPT dataset (V3 unfiltered, cleaned split) with vLLM's benchmark client in closed-loop mode (`--max-concurrency` with `--request-rate inf`), sweeping concurrency $\{64, 128, 256, 512, 1024\}$ with `num-prompts` $= \max(4 \times \text{concurrency}, 2000)$ per point. Requests use the real ShareGPT prompt and reply lengths (no fixed output length), with `--seed 0` and `max-model-len`= 8192. Each (model, scheme) point is collected serially on a dedicated GPU pair, with servers launched and torn down as isolated process groups so that no orphaned worker retains GPU memory.

| Scheme | Prefill | Rank $r$ | Keep $s$ | Decode update | Recompression |
|---|---|---|---|---|---|
| Dense | dense | full | — | exact recurrence | — |
| FIFO | dense | 32 | 0 | low-rank ring | none |
| FReS | dense | 32 | 16 | low-rank | periodic (randomized) |

Table 6: Recurrent-state configuration; identical for the GDN (Qwen) and KDA (Kimi) layers.

# B Additional Method Details

## B.1 Detailed Derivation of the Factorized Rank-One Update

We provide the detailed derivation of the factorized rank-one update used by FReS. Let the approximate recurrent state before processing token $t$ be represented as

$$\widehat{H}_{t-1} = U_{t-1} W_{t-1}^{\top}, \tag{17}$$

where

$$U_{t-1} \in \mathbb{R}^{d_k \times r_{t-1}}, \qquad W_{t-1} \in \mathbb{R}^{d_v \times r_{t-1}}. \tag{18}$$

Applying the decay operator gives

$$\overline{H}_t = D_t \widehat{H}_{t-1}. \tag{19}$$

Since the decay operator acts on the key dimension, it can be applied directly to the left factor:

$$\overline{H}_t = D_t U_{t-1} W_{t-1}^{\top} = \overline{U}_t W_{t-1}^{\top}, \qquad \overline{U}_t = D_t U_{t-1}. \tag{20}$$

Applying the delta-rule update to the decayed state yields

$$\begin{aligned} \widehat{H}_t &= \left(I - \beta_t k_t k_t^{\top}\right) \overline{H}_t + \beta_t k_t v_t^{\top} \\ &= \overline{H}_t - \beta_t k_t k_t^{\top} \overline{H}_t + \beta_t k_t v_t^{\top} \\ &= \overline{H}_t + \beta_t k_t \left(v_t - \overline{H}_t^{\top} k_t\right)^{\top}. \end{aligned} \tag{21}$$

Define the correction vector

$$e_t = v_t - \overline{H}_t^{\top} k_t. \tag{22}$$

Using Eq. 20, this vector can be evaluated without reconstructing the dense state:

$$\begin{aligned} e_t &= v_t - \left(\overline{U}_t W_{t-1}^{\top}\right)^{\top} k_t \\ &= v_t - W_{t-1} \left(\overline{U}_t^{\top} k_t\right). \end{aligned} \tag{23}$$

Substituting the correction vector into the delta-rule update gives

$$\widehat{H}_t = \overline{U}_t W_{t-1}^{\top} + k_t (\beta_t e_t)^{\top}. \tag{24}$$

The two terms can be combined by appending one column to each factor:

$$\widehat{H}_t = [\overline{U}_t,\ k_t][W_{t-1},\ \beta_t e_t]^{\top}. \tag{25}$$

For completeness, the dimensions of the appended factors are

$$\begin{aligned} \overline{U}_t \in \mathbb{R}^{d_k \times r_{t-1}}, \qquad k_t \in \mathbb{R}^{d_k}, \\ W_{t-1} \in \mathbb{R}^{d_v \times r_{t-1}}, \quad \beta_t e_t \in \mathbb{R}^{d_v}. \end{aligned} \tag{26}$$

Therefore,

$$[\overline{U}_t,\ k_t] \in \mathbb{R}^{d_k \times (r_{t-1}+1)}, \qquad [W_{t-1},\ \beta_t e_t] \in \mathbb{R}^{d_v \times (r_{t-1}+1)}, \tag{27}$$

and their product remains in $\mathbb{R}^{d_k \times d_v}$, matching the shape of the recurrent state.

Equation 25 shows that each token update increases the factor width by exactly one. The update is exact with respect to the current approximate state and does not require materializing $\widehat{H}_t$. Approximation is introduced only when the initial dense state is projected or when the active factors are periodically recompressed to satisfy the rank budget.

### B.2 Detailed Derivation of Factor-Space Recompression

Let a recurrent state be represented in factorized form as

$$X = UW^\top, \qquad U \in \mathbb{R}^{d_k \times r}, \quad W \in \mathbb{R}^{d_v \times r}. \tag{28}$$

Randomized subspace iteration first forms a random sketch

$$Y_0 = X\Omega, \qquad \Omega \in \mathbb{R}^{d_v \times s}, \tag{29}$$

and refines the sketch through power iterations:

$$Y_{j+1} = \mathrm{orth}(XX^\top Y_j). \tag{30}$$

The estimated dominant subspace is obtained as

$$Q = \mathrm{orth}(Y_q), \qquad Q \in \mathbb{R}^{d_k \times s}. \tag{31}$$

The sketch and power-iteration products can be evaluated directly from the factors. Specifically,

$$X\Omega = U(W^\top \Omega), \tag{32}$$

and

$$XX^\top Y_j = U(W^\top W)(U^\top Y_j). \tag{33}$$

The projection coefficients are also computed in factor space:

$$X^\top Q = W(U^\top Q). \tag{34}$$

Using the orthonormal projection $QQ^\top$, the rank-$s$ approximation of $X$ is

$$\widehat{X} = QQ^\top X. \tag{35}$$

Substituting $X = UW^\top$ gives

$$\begin{aligned} \widehat{X} &= QQ^\top UW^\top \\ &= Q\left(W(U^\top Q)\right)^\top. \end{aligned} \tag{36}$$

Therefore, the recompressed state can be stored as

$$U' = Q, \qquad W' = W(U^\top Q), \tag{37}$$

where

$$U' \in \mathbb{R}^{d_k \times s}, \qquad W' \in \mathbb{R}^{d_v \times s}. \tag{38}$$

All matrix products required for recompression are thus evaluated from the factor matrices, and the dense state $X$ is never explicitly reconstructed. The recompression reduces the active factor width from $r$ to $s$ while retaining an approximation to the dominant left singular subspace of the recurrent state.

## C Baseline Details

### C.1 FIFO Truncation

At each decoding step, the existing recurrent state is first transformed by the decay operator before the new rank-one update is appended. Consequently, factor columns that have been present for longer undergo more decay steps than recently appended columns, and their contributions tend to be progressively attenuated.

This observation motivates a simple FIFO truncation strategy. When the active factor width reaches the budget $R$ at decode step $t$, FIFO overwrites the oldest factor columns with the newly appended columns and retains the most recent factors:

$$U = [u_{t-R+1}, \ldots, u_t], \qquad W = [w_{t-R+1}, \ldots, w_t]. \tag{39}$$

FIFO truncation introduces negligible recompression overhead and provides a matched-budget baseline. However, it selects factors according to their age rather than their contribution to the current recurrent state. It may therefore discard important directions and lead to substantial error accumulation over repeated updates.

### C.2 TRUNCATED SVD

For a represented state $X \in \mathbb{R}^{d_k \times d_v}$, truncated SVD computes

$$X \approx X_s = U_s \Sigma_s V_s^\top, \tag{40}$$

where the $s$ largest singular values and their associated singular vectors are retained. The corresponding factorization is given by

$$U = U_s \sqrt{\Sigma_s}, \qquad W = V_s \sqrt{\Sigma_s}, \tag{41}$$

such that

$$X_s = U W^\top. \tag{42}$$

By the Eckart–Young theorem, $X_s$ is the best rank-$s$ approximation to $X$ under the Frobenius norm. Truncated SVD therefore serves as an ideal reference for evaluating the error introduced by rank reduction.

However, repeatedly applying truncated SVD during decoding incurs substantial overhead, even when the state is maintained in factorized form. The cuSOLVER-based SVD implementation also requires runtime-dependent iterations, host synchronization, and dynamic workspace management, which are incompatible with the CUDA-graph-captured decoding path used by vLLM. We therefore use truncated SVD only as a quality reference in the ablation study rather than for online recompression.