STOCHASTIC KV ROUTING: ENABLING ADAPTIVE DEPTH-WISE CACHE SHARING

Anastasiia Filippova, David Grangier, Marco Cuturi, João Monteiro
Apple MLR

ABSTRACT

Serving transformer language models efficiently is constrained by the significant memory footprint of the Key-Value (KV) cache. While recent optimizations focus on compressing the cache along the temporal axis, we argue that the *depth* dimension offers a robust, orthogonal avenue for improving cache efficiency. Existing cross-layer sharing methods often suffer from throughput or latency overhead. In this work, we introduce Random Cross-Layer Attention (R-CLA), a training scheme where layers stochastically attend to either their own KV states or those of a preceding layer. This simple approach decouples layers from specific features, enabling flexible depth-wise cache sharing at inference time. We demonstrate that R-CLA allows for significant memory savings and acts as a regularizer that improves generalization in larger models.

1 INTRODUCTION

The deployment of Large Language Models (LLMs) requires handling the trade-offs between serving costs and generation throughput or latency. Often used as a means to improve throughput, KV caching introduces memory and communication bottlenecks. The cache scales linearly with depth and sequence length, often outsizing the model parameters themselves, as illustrated in Figure 1; for example, the KV cache size for a 7B parameters model represents a data expansion of over **100,000**$\times$ relative to the input tokens. While recent methods have attempted to mitigate this via temporal eviction (dropping unimportant tokens over time), the *depth dimension* remains underexplored despite high inter-layer redundancy. Current approaches to exploit this redundancy, such as sharing caches across layers, often require complex auxiliary encoders or incur latency penalties that negate the memory benefits.

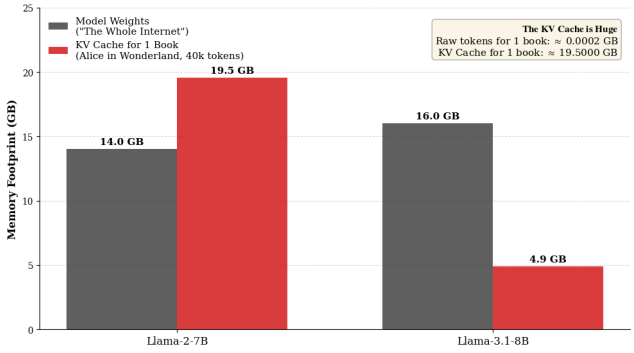

Figure 1: LLMs are often described as efficient databases. The KV cache however doesn't reflect that, and a single book may require as much space as the model itself. Memory footprint per token in Bytes can be obtained as follows for transformer decoders: $2\times$`#layers`$\times$`#kv_heads`$\times$`#head_dim`$\times 2$, where the leftmost factor of 2 is due to storing both K and V representations, while the rightmost factor of 2 is due to each floating point value requiring two bytes, assuming FP16 precision.

To bridge this gap, we propose Random Cross-Layer Attention (R-CLA), a training scheme that breaks the rigid dependency between a layer and its specific KV states. During training, layers randomly select whether to attend to their own states or those of a preceding layer, simulating structural cache faults. This stochastic process forces the model to learn robust representations that allow for flexible *cache sharing strategies* at deployment time. Consequently, a single model can adapt to diverse hardware constraints by retaining only a fraction of the layer caches. Our contributions are as follows:

(i) We identify that full layer-wise caching is redundant and that the depth dimension is a viable target for cache optimization.

(ii) We introduce R-CLA, a randomized training method that enables robust depth-wise cache sharing.

(iii) We demonstrate that R-CLA acts as a regularizer for larger models, improving downstream performance by preventing over-fitting to layer-specific features.

2 Enabling Cross-Layer Attention

Standard autoregressive inference consists of two phases: a compute-bound *pre-fill* phase that processes the prompt and builds the Key-Value (KV) cache, and a memory-bound *sampling* phase that generates tokens sequentially. During sampling, the KV cache grows linearly with sequence length and depth. For a model with L layers and sequence length T, the memory footprint is $M \propto L \times T$. As T increases, the bandwidth required to load KV tensors becomes the primary bottleneck (see Algorithm 1 in Appendix B). To mitigate this, we explore optimizing the depth dimension L. We define a *Cache Sharing Strategy* $\mathcal{S} \subseteq \{1, \ldots, L\}$ as the subset of layers authorized to maintain a cache. Layers $l \notin \mathcal{S}$ do not store their own states; instead, they perform Cross-Layer Attention (CLA) by attending to the cached states of the nearest preceding layer $\mu(l) = \max\{j \in \mathcal{S} \mid j < l\}$. This allows the cache to be loaded once and reused across multiple layers, significantly reducing memory I/O (detailed in Algorithm 2, Appendix B).

However, applying this strategy to standard pre-trained models degrades performance, as layers rely on specific feature alignments with their own KV pairs. To address this, we propose Random Cross-Layer Attention (R-CLA). Rather than training for a fixed strategy $\mathcal{S}$, we introduce stochasticity to decouple query projections from specific layer features.

During training, for every layer l in a forward pass, we sample a decision $d \sim \text{Bernoulli}(p)$:

• If $d = 1$: The layer performs standard self-attention using its own states (K_l, V_l).

• If $d = 0$: The layer performs CLA using $(K_{l'}, V_{l'})$, where l' is sampled uniformly from preceding layers $\{1, \ldots, l-1\}$.

This stochastic process simulates structural cache faults, as illustrated in Figure 5, forcing the model to learn robust representations invariant to the specific source of keys and values. Consequently, at test time, a single R-CLA model can adapt to various hardware constraints by employing different deterministic sharing strategies $\mathcal{S}$ without retraining. Furthermore, we find this randomness acts as a regularizer in data-constrained settings, preventing overfitting to layer-specific features.

3 Evaluation

We evaluate R-CLA in two distinct settings: compute-constrained pre-training (Section 3.1) to assess stability and architecture trade-offs, and data-constrained fine-tuning on Question-Answer (QA) tasks (Section 3.2) to measure information retention and regularization effects.

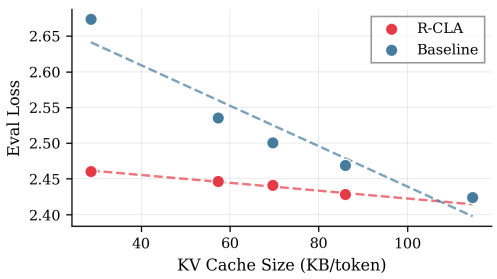

Figure 2: Cache size vs. eval loss for Qwen3-1.7B. R-CLA models (red) outperform shallower baselines (blue) at equivalent memory budgets.

3.1 Pre-training

We pre-train `Qwen-1.7B`-style decoder-only Transformers from scratch on 34B tokens (Chinchilla-optimal) from the OpenWeb corpus (Gokaslan et al., 2019). To verify that our scheme does not destabilize training, we compare R-CLA models with varying sharing probabilities p both against matching size baseline models and shallower models of equivalent effective cache size.

Table 1: Pre-training evaluation loss for Qwen3-1.7B trained with R-CLA at varying cross-layer attention probability p. Training remains stable across all values, with loss increasing by less than 2% even at $p = 0.75$.

p	0.00	0.25	0.50	0.60	0.75
Eval Loss	2.424	2.428	2.441	2.446	2.461

As shown in Table 1, even when redirecting 75% of attention operations to preceding layers ($p = 0.75$), the evaluation loss degrades by less than 2% compared to standard training of a full-size self-attention model. Stable training trajectories are also shown in Figure 4 in the appendix.

Crucially, R-CLA offers a superior trade-off between performance and memory. Figure 2 illustrates that R-CLA models (using 28 layers with sharing) consistently achieve lower evaluation loss than baseline models with fewer layers but full caching. This demonstrates that maintaining model depth with shared caches is preferable to reducing depth to meet memory constraints.

3.2 Fine-tuning

We fine-tune pre-trained models (`Qwen3-8B`, `Mistral-7B`, `Llama-3.1-8B`) to evaluate whether R-CLA preserves information retrieval capabilities under cache sharing, for various sharing strategies. To do so, we curated a diverse dataset from five sources: HotpotQA (Yang et al., 2018) for multi-hop reasoning; SQuAD v2 (Rajpurkar et al., 2018), MSMarco (Bajaj et al., 2016), and TriviaQA (Joshi et al., 2017) for reading comprehension. Additionally, we included RepLiQA (Monteiro et al., 2024b), a dataset consisting of fictional content. This inclusion is critical as it ensures our evaluation strictly measures the model's ability to find answers within the context, rather than relying on parametric knowledge acquired during pre-training.

To improve robustness and prevent input formatting biases, we applied specific data augmentation strategies. For all datasets, we randomized the input structure, swapping the standard `context` $\rightarrow$ `question` order with `question` $\rightarrow$ `context` with probability 0.5. Furthermore, for HotpotQA, we generated variations with permuted context passages to prevent the model from relying on the position of relevant information versus confounders. All models were fine-tuned for 50,000 steps with identical data.

The results, summarized in Table 2 and fully covered in Table 3, for models using R-CLA at $p = 0.6$ highlight three key findings. First, R-CLA enables efficient depth-wise cache sharing; where base models suffer catastrophic collapse at low retention rates (e.g., 25%), R-CLA models preserve substantial capability. Second, R-CLA allows us to drop significant portions of the cache (up to 50%) with negligible performance impact in various tasks. Finally, we observe a distinct regularization

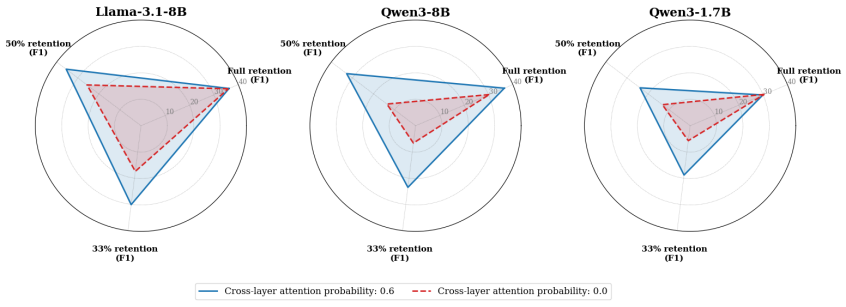

Figure 3: Performance comparison under varying cache retention rates for base models and their counterparts trained with random cross-layer attention. We report F1 scores for Question-Answer evaluations under different levels of cache retention for standard and R-CLA fine-tuned models. R-CLA avoids much of the degradation base models incur upon cache sharing, and preserves or improves performance at full cache.

effect: R-CLA models frequently outperform their standard counterparts even in the full-retention setting (e.g., +50.7% improvement on HotpotQA for Llama-3.1). This suggests that the stochastic training objective prevents overfitting to specific layer-wise features, improving generalization in data-constrained environments. Ablations are reported in Appendix C.1 where we employ deterministic cache sharing schemes during training that match those of testing.

Table 2: F1 scores for models fine-tuned on QA tasks. We compare standard Base models vs. R-CLA (ours) across three cache retention levels ($100\%, 50\%, 25\%$). R-CLA consistently maintains higher performance as cache depth decreases. Training with R-CLA is done at $p = 0.6$ for all models

	Base Model (Standard)			**R-CLA (Ours)**		
Dataset / Model	**100%**	**50%**	**25%**	**100%**	**50%**	**25%**
HotpotQA						
Llama-3.1-8B	0.203	0.171	0.080	**0.306**	**0.305**	**0.237**
Mistral-7B	0.215	0.162	0.031	**0.242**	**0.211**	**0.162**
Qwen3-8B	0.233	0.085	0.011	**0.357**	**0.314**	**0.098**
MSMarco						
Llama-3.1-8B	0.301	0.226	0.047	**0.318**	**0.324**	**0.217**
Mistral-7B	0.310	0.236	0.069	**0.316**	**0.300**	**0.180**
Qwen3-8B	0.291	0.112	0.027	**0.329**	**0.227**	**0.058**
RepLiQA						
Llama-3.1-8B	**0.655**	0.314	0.073	0.649	**0.597**	**0.307**
Mistral-7B	**0.649**	0.291	0.084	0.648	**0.547**	**0.214**
Qwen3-8B	0.441	0.154	0.029	**0.478**	**0.349**	**0.065**
SQuAD v2						
Llama-3.1-8B	0.583	0.427	0.257	**0.758**	**0.740**	**0.628**
Mistral-7B	0.573	0.388	0.078	**0.750**	**0.654**	**0.500**
Qwen3-8B	0.705	0.269	0.042	**0.728**	**0.627**	**0.284**
TriviaQA						
Llama-3.1-8B	0.328	0.290	0.137	**0.360**	**0.351**	**0.291**
Mistral-7B	0.205	0.138	0.032	**0.216**	**0.158**	**0.121**
Qwen3-8B	0.146	0.032	0.005	**0.295**	**0.248**	**0.131**

The effect of R-CLA can also be observed in Figure 3 for different fine-tuned models. Upon different levels of cache retention, obtained through varied test caching strategies that keep only every k-th layer, models trained under R-CLA do withstand cache sharing and preserve base full cache performance to a greater extent than base models. Perhaps also relevant, performance at full cache is preserved or improved, which highlights that the random nature of R-CLA training offers a regularization effect.

4 Conclusion

We addressed the KV cache memory bottleneck in language model inference. In line with previous work, we showed that the standard practice of maintaining a full cache for every layer is redundant and that models can effectively operate with significantly leaner caches by exploiting depth-wise correlations. By introducing random cross-layer attention, a simple yet effective randomized training strategy, we showed that pre-trained models can be adapted to be robust against various depth-wise cache sharing strategies, and using R-CLA early, during pre-training, can be done effectively. Our evaluations confirm that this approach not only reduces memory requirements, enabling longer contexts and larger batch sizes, but also acts as a regularizer for larger models under data-constrained settings. By breaking rigid layer-wise dependencies, R-CLA maintains high prediction quality even under severe retention limits, and frequently outperforms base models in full-cache regimes.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, et al. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model. *arXiv preprint arXiv:2508.14444*, 2025.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. Reducing transformer key-value cache size with cross-layer attention, 2024. *URL https://arxiv. org/abs/2405.12981*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

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

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Coleman Hooper, Sebastian Zhao, Luca Manolache, Sehoon Kim, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Multipole attention for efficient long context reasoning. *arXiv preprint arXiv:2506.13059*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

Joao Monteiro, Étienne Marcotte, Pierre-Andre Noel, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. Xc-cache: Cross-attending to cached context for efficient llm inference. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15284–15302, 2024a.

Joao Monteiro, Pierre-Andre Noel, Etienne Marcotte, Sai Rajeswar, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. Repliqa: A question-answering dataset for benchmarking llms on unseen reference content. *Advances in Neural Information Processing Systems*, 37:24242–24276, 2024b.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. In *The Thirteenth International Conference on Learning Representations*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Yiqun Shen12, Song Yuan, Zhengze Zhang12, Xiaoliang Wang, Daxin Jiang, and Cam-Tu Nguyen12. Lava: Layer-wise kv cache eviction with dynamic budget allocation. 2025.

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.

Bailin Wang, Chang Lan, Chong Wang, and Ruoming Pang. Rattention: Towards the minimal sliding window size in local-global attention models. *arXiv preprint arXiv:2506.15545*, 2025.

Haoyi Wu and Kewei Tu. Layer-condensed kv cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11175–11188, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=r8H7xhYPwz.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Yuanbing Zhu, Zhenheng Tang, Xiang Liu, Ang Li, Bo Li, Xiaowen Chu, and Bo Han. OracleKV: Oracle guidance for question-independent KV cache compression. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025. URL https://openreview.net/forum?id=KHM2YOGgX9.

A TRAINING CURVES

In Figure 4, we show training curves for experiments reported in Section 3.1. We pre-train a `Qwen-1.7B`–style decoder from scratch on a subset of the OpenWeb corpus (Gokaslan et al., 2019). We train different models using different levels of cache sharing od dropping ratio. That is, we use R-CLA with $p \in \{0.25, 0.5, 0.6, 0.75\}$. All models are trained for an identical token budget of 34B tokens. Training curves show that R-CLA incurs no training instability evan under aggressive p, and hyperparamters transfer reasonably well from standard self-attention models to R-CLA ones. R-CLA can be introduced at pre-training time if cache sharing is desirable downstream.

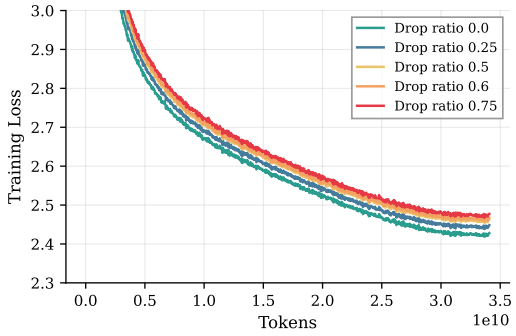

Figure 4: Training loss as a function of tokens processed for Qwen3-1.7B with varying KV cache sharing ratios.

B Additional Details on R-CLA

B.1 Inference Algorithms

We provide the pseudocode for standard inference versus our proposed depth-wise cache sharing inference below. In standard inference (Algorithm 1), memory I/O is incurred at every layer. In our approach (Algorithm 2), memory I/O occurs only for layers in the strategy set $\mathcal{S}$.

Algorithm 1 Standard Inference (Memory Bound)

for $l = 1$ **to** L **do**
  Compute Query Q_l
  Load K_l, V_l
  Compute Attn(Q_l, K_l, V_l)
  Update K_l, V_l
end for

Algorithm 2 Inference with Depth-Wise Sharing

for $l = 1$ **to** L **do**
  Compute Query Q_l
  if $l \in \mathcal{S}$ **then**
    Load K_l, V_l
    Cache $\leftarrow K_l, V_l$
  end if
  $K_{l'}, V_{l'} \leftarrow$ Cache
  Compute Attn$(Q_l, K_{l'}, V_{l'})$
  if $l \in \mathcal{S}$ **then**
    Update K_l, V_l
  end if
end for

B.2 R-CLA Illustration

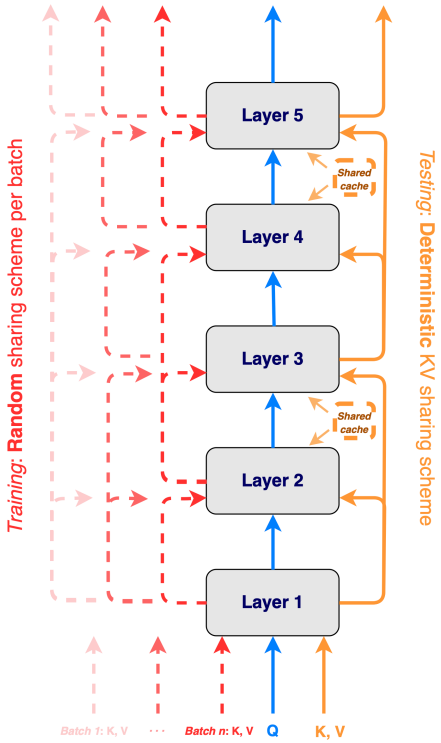

Figure 5: Behavior of a model implementing R-CLA. During *training*, layers randomly attend to their own states or a random previous layer. During *testing*, a fixed deterministic cache sharing scheme is employed (e.g., every 3rd layer cached).

C ADDITIONAL FINE-TUNING RESULTS

Table 3: We report Base, R-CLA ($p = 0.6$), and relative improvement ($\Delta\%$) for F1, Exact Match (EM), and ROUGE-L across three cache retention levels.

Dataset	Model	Retention	F1			Exact Match (EM)			ROUGE-L		
			Base	R-CLA	$\Delta\%$	Base	R-CLA	$\Delta\%$	Base	R-CLA	$\Delta\%$
HotpotQA	Llama-3.1-8B	100%	0.203	**0.306**	+51.1%	0.128	**0.221**	+72.3%	0.206	**0.307**	+49.0%
		50%	0.171	**0.305**	+78.7%	0.102	**0.216**	+112.6%	0.173	**0.305**	+75.8%
		25%	0.080	**0.237**	+196.2%	0.035	**0.153**	+339.6%	0.088	**0.240**	+171.6%
	Mistral-7B	100%	0.215	**0.242**	+12.4%	0.122	**0.141**	+15.8%	0.216	**0.243**	+12.6%
		50%	0.162	**0.211**	+30.0%	0.081	**0.106**	+30.5%	0.165	**0.212**	+28.6%
		25%	0.031	**0.162**	+421.3%	0.014	**0.078**	+467.3%	0.037	**0.168**	+352.3%
	Qwen3-8B	100%	0.233	**0.357**	+53.1%	0.157	**0.273**	+73.3%	0.235	**0.359**	+52.8%
		50%	0.085	**0.314**	+267.8%	0.024	**0.227**	+863.8%	0.085	**0.316**	+270.1%
		25%	0.011	**0.098**	+826.6%	0.000	**0.041**	-	0.014	**0.108**	+692.5%
MSMarco	Llama-3.1-8B	100%	0.301	**0.318**	+5.6%	0.047	**0.056**	+20.8%	0.290	**0.306**	+5.7%
		50%	0.226	**0.324**	+43.2%	0.021	**0.054**	+160.0%	0.222	**0.310**	+39.5%
		25%	0.047	**0.217**	+358.2%	0.013	**0.039**	+214.8%	0.059	**0.225**	+284.3%
	Mistral-7B	100%	0.310	**0.316**	+1.7%	0.047	**0.048**	+1.0%	0.296	**0.301**	+1.6%
		50%	0.236	**0.300**	+26.9%	0.013	**0.043**	+244.4%	0.229	**0.286**	+25.0%
		25%	0.069	**0.180**	+161.8%	0.002	**0.019**	+900.0%	0.084	**0.181**	+116.0%
	Qwen3-8B	100%	0.291	**0.329**	+13.0%	0.043	**0.064**	+49.5%	0.281	**0.321**	+14.1%
		50%	0.112	**0.227**	+102.4%	0.002	**0.038**	+1540.0%	0.113	**0.228**	+102.1%
		25%	0.027	**0.058**	+110.6%	0.000	**0.009**	-	0.043	**0.067**	+55.3%
RepLiQA	Llama-3.1-8B	100%	**0.655**	0.649	-0.9%	0.261	**0.265**	+1.6%	**0.632**	0.629	-0.6%
		50%	0.314	**0.597**	+90.0%	0.022	**0.252**	+1033.7%	0.316	**0.578**	+82.8%
		25%	0.073	**0.307**	+322.3%	0.008	**0.090**	+1067.7%	0.124	**0.312**	+152.3%
	Mistral-7B	100%	**0.649**	0.648	-0.2%	**0.265**	0.248	-6.4%	0.628	**0.628**	+0.1%
		50%	0.291	**0.547**	+88.2%	0.023	**0.207**	+790.3%	0.302	**0.532**	+76.0%
		25%	0.084	**0.214**	+154.7%	0.001	**0.030**	+3966.7%	0.121	**0.224**	+85.8%
	Qwen3-8B	100%	0.441	**0.478**	+8.4%	0.072	**0.100**	+39.0%	0.420	**0.458**	+8.9%
		50%	0.154	**0.349**	+126.8%	0.009	**0.070**	+705.7%	0.150	**0.339**	+126.5%
		25%	0.029	**0.065**	+126.4%	0.000	**0.002**	-	0.049	**0.081**	+66.3%
SQuAD v2	Llama-3.1-8B	100%	0.583	**0.758**	+30.0%	0.467	**0.658**	+40.8%	0.586	**0.759**	+29.6%
		50%	0.427	**0.740**	+73.2%	0.287	**0.638**	+122.2%	0.433	**0.740**	+71.0%
		25%	0.257	**0.628**	+144.6%	0.147	**0.483**	+229.2%	0.286	**0.637**	+122.3%
	Mistral-7B	100%	0.573	**0.750**	+30.8%	0.466	**0.642**	+37.7%	0.576	**0.750**	+30.3%
		50%	0.388	**0.654**	+68.4%	0.259	**0.537**	+107.1%	0.398	**0.656**	+64.8%
		25%	0.078	**0.500**	+541.7%	0.027	**0.347**	+1192.6%	0.097	**0.520**	+436.2%
	Qwen3-8B	100%	0.705	**0.728**	+3.3%	0.611	**0.648**	+5.9%	0.705	**0.729**	+3.4%
		50%	0.269	**0.627**	+132.9%	0.117	**0.542**	+363.0%	0.265	**0.632**	+138.2%
		25%	0.042	**0.284**	+575.0%	0.002	**0.165**	+8175.0%	0.052	**0.310**	+492.3%
TriviaQA	Llama-3.1-8B	100%	0.328	**0.360**	+9.8%	0.266	**0.301**	+13.3%	0.327	**0.359**	+9.8%
		50%	0.290	**0.351**	+20.8%	0.249	**0.298**	+19.5%	0.290	**0.349**	+20.5%
		25%	0.137	**0.291**	+112.2%	0.098	**0.243**	+149.2%	0.138	**0.290**	+110.4%
	Mistral-7B	100%	0.205	**0.216**	+5.6%	**0.119**	0.110	-7.7%	0.206	**0.216**	+4.5%
		50%	0.138	**0.158**	+14.6%	0.066	**0.088**	+33.6%	0.139	**0.157**	+13.6%
		25%	0.032	**0.121**	+273.9%	0.016	**0.070**	+342.9%	0.034	**0.123**	+257.3%
	Qwen3-8B	100%	0.146	**0.295**	+101.7%	0.079	**0.235**	+196.8%	0.145	**0.295**	+102.6%
		50%	0.032	**0.248**	+671.9%	0.005	**0.200**	+3709.5%	0.028	**0.248**	+776.5%
		25%	0.005	**0.131**	+2287.3%	0.000	**0.091**	-	0.005	**0.131**	+2279.9%

C.1 ABLATION

We conducted an ablation study using `Llama-3.1-8B` to assess the specific contribution of randomness in our approach versus the general benefits of sharing KV states. We compare our fully random method (R-CLA) against two baseline categories. First, CLA@k represents a deterministic fixed sharing scheme where groups of k layers share KV states (*e.g.*, layers 0 to $k-1$ share one KV). Second, to disentangle the effect of stochastic training, we evaluate RD-CLA@k (Random-Deterministic CLA). This method uses the same fixed sharing scheme as CLA@k, but applies cross-layer attention stochastically: at each forward step at each layer, the model flips a coin to decide whether to attend to the fixed shared layer or its local KV.

The results in Table 4 highlight distinct roles for sharing and stochasticity. In the no-sharing (100% retention) regime, fixed CLA@k schemes frequently match or slightly exceed the performance of R-CLA. This suggests that the regularization benefits observed in previous sections stem primarily

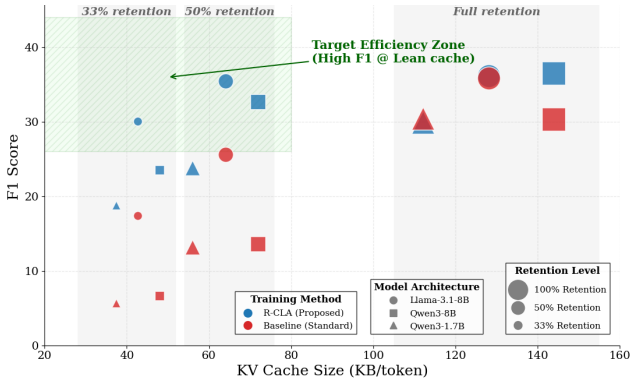

Figure 6: Cache size and prediction performance trade-offs for different models.

from the information bottleneck created by KV sharing, rather than randomness itself. However, under missing KV states (50% and 25% retention), the role of randomness becomes apparent, and adding stochasticity to fixed schemes (RD-CLA) provides improvements over deterministic baselines. Ultimately, the unstructured randomness of R-CLA consistently outperforms all other variations under missing states, indicating that exposing the model to a diverse set of sharing patterns during training helps learn representations that are resilient to missing cache.

Table 4: Ablations on top of Llama-3.1-8B. We compare R-CLA ($p = 0.6$) against deterministic CLA fixed at every 2^{nd} or 4^{th} layer, and their randomly applied counterparts (RD-CLA, $p = 0.6$).

Dataset	Method	Retention		
		100%	**50%**	**25%**
HotpotQA	R-CLA@0.6 (Ours)	0.306	**0.305**	**0.237**
	CLA@2	0.192	0.149	0.089
	CLA@4	**0.331**	0.284	0.172
	RD-CLA@2	0.223	0.179	0.100
	RD-CLA@4	0.167	0.134	0.071
MSMarco	R-CLA@0.6 (Ours)	0.318	**0.324**	**0.217**
	CLA@2	0.282	0.184	0.065
	CLA@4	0.303	0.159	0.085
	RD-CLA@2	**0.323**	0.247	0.062
	RD-CLA@4	0.300	0.211	0.076
RepLiQA	R-CLA@0.6 (Ours)	0.649	**0.597**	**0.307**
	CLA@2	0.520	0.237	0.087
	CLA@4	0.476	0.162	0.085
	RD-CLA@2	**0.652**	0.298	0.075
	RD-CLA@4	0.621	0.270	0.076
SQuAD v2	R-CLA@0.6 (Ours)	0.758	**0.740**	**0.628**
	CLA@2	0.633	0.387	0.290
	CLA@4	**0.768**	0.456	0.397
	RD-CLA@2	0.660	0.521	0.314
	RD-CLA@4	0.630	0.521	0.313
TriviaQA	R-CLA@0.6 (Ours)	0.360	**0.351**	**0.291**
	CLA@2	0.332	0.280	0.145
	CLA@4	**0.376**	0.321	0.165
	RD-CLA@2	0.357	0.326	0.170
	RD-CLA@4	0.357	0.315	0.158

D RELATED WORK

The potential impact of the size of the KV cache in scaling inference has sparked a vast body of literature, primarily focused on the temporal dimension, architectural modifications, and, more recently, depth-wise redundancy.

Temporal Eviction and Compression. The majority of research to date has focused on *temporal eviction* or compression, trying to reduce the cache size by dropping or compressing tokens along the time axis. Early efforts, like StreamLM (Xiao et al., 2024), identified a recency bias in attention patterns, proposing to keep only a small sliding window of recent tokens alongside crucial "attention sinks" (initial tokens). However, this often discards relevant semantic information stored in the middle of a sequence. To address this, methods like SnapKV (Li et al., 2024) use attention scores to identify and preserve important chunks of the cache. This idea was extended by methods like PyramidKV (Cai et al., 2024), CAKE (Qin et al., 2025), and LAVa (Shen12 et al., 2025), which vary the cache budget across different layers and attention heads. Despite their prevalence, these methods are inherently *query-dependent* (Zhu et al., 2025); because the importance of a token changes based on the query, eviction must be re-computed dynamically. While some recent attempts like OracleKV (Zhu et al., 2025) and KVZip (Kim et al., 2025) try to mitigate this by predicting future token relevance using past queries, or by using streaming clustering for sub-linear query scoring (Hooper et al., 2025), they either risk information loss or maintain a full index in memory that offsets the communication gains. Furthermore, methods like H2O (Zhang et al., 2023) and FastGen (Ge et al., 2023) focus on cache updates during generation, yet they do not reduce the peak memory footprint during the pre-filling phase for long contexts.

Architectural Improvements and Efficiency. One could argue that the most successful reductions in cache footprint have come from architectural shifts rather than post-hoc eviction or compression approaches. By changing how the model handles attention, past work has meaningfully reduced the number of K and V vectors that need to be stored. Multi-Query Attention (MQA) (Raffel et al., 2020) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) achieve this by sharing KV states across multiple query heads within a single layer. The impact of these architectural choices is significant: as noted previously, `Llama-2-7B` requires 512KB/token, whereas the more modern `Qwen3-8B` architecture (Yang et al., 2025a) reduces the cache memory footprint to 144KB/token primarily through the use of GQA. Similarly, cross-layer attention (CLA) (Brandon et al.) makes it so two neighbor layers share a single set of KV states. Other structural innovations include State Space Models (SSMs) (Dao & Gu, 2024; Yang et al., 2025b) and hybrid architectures like `Kimi-k1.5` (Team et al., 2025) and `Nemotron-Nano-9B-v2` (Basant et al., 2025), which interleave attention with SSM layers and require no cache in non-attention blocks. Similarly, local/global self-attention architectures have been defined where local layers attend only to a small neighborhood close to the query, requiring a smaller cache (Beltagy et al., 2020; Wang et al., 2025). Additionally, KV quantization (Hooper et al., 2024) has proven effective at reducing the numerical precision of the cache. Notably, our proposed depth eviction is orthogonal to these architectural improvements and can be combined with GQA or quantization for compounded gains.

Depth-Wise Cache Sharing. While less explored than the temporal axis, one may exploit the redundancy of representations across the model's layers. XC-Cache (Monteiro et al., 2024a) demonstrated that a shared cache across layers is feasible but requires expensive updates due to calls to an external bi-directional encoder. Similarly, Layer-Condensed KV Cache (Wu & Tu, 2024) proposes a shared single-layer cache but suffers from a significant increase in time-to-first-token due to the need for sequential pre-filling or multiple forward passes. MiniCache (Liu et al., 2024) takes a post-hoc approach by merging pairs of late layers, but its gains are modest since the model is not trained to handle this type of weight sharing. These works provide strong evidence for the hypothesis that Transformer decoders can function with a shared set of KV states, much like the cross-attention mechanism in encoder-decoder models. Our work builds on this intuition by proposing a practical and efficient training scheme. Just as GQA shares KV states across *heads*, we follow a similar approach to CLA and share them across *layers*, though we use a randomized training scheme to ensure models remain robust to various cache sharing strategies.