Towards Dynamic KV-Cache Compression: Fine-Grained Evaluation of Key/Value Ranks in LLMs

Jian Chen[1,3*], **Zhuoran Wang**[2,4*], **Jiayu Qin**[2], **Ming Li**[5], **Meng Wang**[6]
Changyou Chen[1], **Yin Chen**[2], **Qizhen Weng**[2], **Yirui Liu**[2†]

[1]University at Buffalo [2]TeleAI [3]Dolby Laboratories
[4]Delft University of Technology [5]University of Maryland [6]ByteDance

Abstract

Large language models rely on KV-cache to avoid redundant computation during autoregressive decoding, but reading and writing the growing cache quickly overwhelms GPU memory bandwidth as context length increases. Recent studies therefore explore KV-cache compression, however existing work either overlook the data-dependent nature of key/value features or their layer level differences. In this work, we propose a method that directly computes the optimal data-dependent compression of key and value activations via singular value decomposition during inference. Our approach is gradient-free and incremental, enabling independent per-layer decomposition with batch computation and low memory cost. Using this method, we conduct a comprehensive analysis across multiple models and datasets spanning diverse domains and languages, uncovering fine-grained patterns of KV-cache compressibility. Our method serves as a valuable evaluation tool to reveal how LLMs allocate their representational capacity, offering actionable insights for designing dynamic and data-aware KV-cache compression strategies for deployment.

1 Introduction

Large language models (LLMs) adopt the Transformer architecture [1] and generate text autoregressively under a causal mask, which ensures that past key and value vectors can be cached (KV-cache) [2]. These caches are stored in high-bandwidth memory (HBM) on GPUs and repeatedly fetched into compute-unit registers during decoding, reducing computation but introducing a memory-bandwidth bottleneck as context length grows. This challenge has motivated both hardware innovation [3] and algorithmic approaches for KV-cache compression [4], with our work focusing on the latter.

A natural way to reduce KV-cache cost is to compress key and value representations into lower-dimensional spaces. Many methods use low-rank approximation of projection matrices [5, 6], but they ignore the data-dependent nature of key/value activations, whose intrinsic rank can be smaller in domain-specific tasks that exercise only part of the model's capacity [7]. Moreover, most approaches apply the same compression ratio across layers [8], overlooking distinct compressibility profiles. Methods for analyzing and comparing key/value rank across layers remain underexplored.

To address these limitations, we propose an incremental singular value decomposition (SVD) method that directly operates on key and value features computed over large datasets. Unlike approaches that approximate projection weights, our method is data-dependent and captures the intrinsic rank of the KV-cache induced by real inputs. It also allows independent per-layer decomposition without

[*]**Equal contribution**. Work done when Jian was at University at Buffalo and Zhuoran was at TeleAI.
[†]**Corresponding author**: liuyr17@chinatelecom.cn

NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle.

cross-layer coupling. The method is gradient-free, supports batch-wise computation with low memory overhead, and guarantees a globally optimal low-rank projection under the Frobenius norm. Unlike existing methods [9, 8] that only compute the optimal compression matrix, our method explicitly decomposes long sequences of key and value features to recover the singular value distributions of each layer on a given dataset, thereby enabling systematic evaluation of compressibility and providing an effective diagnostic tool for understanding representational capacity usage in LLMs.

We conduct extensive experiments across open-source LLMs of different sizes and architectures, spanning datasets in instruction following, code generation, medical QA, and multilingual tasks. We introduce normalized effective rank (NER) as a lightweight metric for per-layer compressibility and systematically compare key and value ranks across models and domains. Beyond static evaluation, end-to-end SVD-based truncation shows that NER correlates strongly with perplexity, validating it as a reliable proxy for compression sensitivity. These results reveal consistent layer-wise and data-dependent patterns in KV-cache compressibility, laying the groundwork for dynamic and adaptive strategies. Our contributions are as follows:

- We propose a novel algorithm for the efficient computation of the SVD of dataset-level KV-caches, which achieves the global optimum for low-rank compression under the Frobenius norm, enabling data-dependent rank analysis and compression.
- We conduct extensive experiments across models, datasets, and languages, and show that normalized effective rank correlates with perplexity, establishing it as a practical evaluation tool for guiding dynamic KV-cache compression.

2 Method

To evaluate and compress KV caches, we develop an efficient method that incrementally computes the SVD of keys and values over large datasets, enabling layer-wise and data-dependent evaluation of their compressibility in LLMs. At the same time, our method computes the optimal compression matrices, addressing challenges identified in prior work [9, 8, 10].

To illustrate our method and its advantages, we first introduce notations. As our method applies uniformly across layers and to both key and value spaces, we omit layer-specific notation in the rest of the section and focus on K as an example in the following discussion for simplicity. Consider a dataset containing l tokens, for a particular LLM, let $X = [\mathbf{x}_1; ...; \mathbf{x}_l] \in \mathbb{R}^{l \times d_e}$ be the sequence of activations for an attention layer, computed based on the dataset. Let W^K be the key projection weights of a layer. The corresponding key $K = [\mathbf{k}_1; ...; \mathbf{k}_l]$ features are computed as, XW^K. The data-dependent optimal low-rank compression of K is formulated as follows. For each key/value projection matrix W^K, we seek a low-rank matrix $\widetilde{W}^K$ with rank r that minimizes the compression error: $||XW^K - X\widetilde{W}^K||_F$. The solution $\widetilde{W}^K$ can be expressed as a pair of down- and up-projection matrices, which compress keys into dimension r and then reconstruct them for efficient autoregressive inference.

2.1 SVD-based KV-cache Analysis

Our key idea is to perform SVD on key and value spaces for each layer, allowing an analytical study of the layer-wise compressibility of KV caches given a LLM and a particular dataset. For example, the SVD on $K \in \mathbb{R}^{l \times m_h d_h}$ can be written as $K = \mathcal{U}\Sigma\mathcal{V}^T$, where $\mathcal{U} \in \mathbb{R}^{l \times l}$ and $\mathcal{V} \in \mathbb{R}^{m_h d_h \times m_h d_h}$ are left and right singular values, respectively, and $\Sigma \in \mathbb{R}^{l \times m_h d_h}$ denotes singular values. By the Eckart-Young-Mirsky theorem [11], the truncation of the largest singular values along with corresponding singular vectors $K_k = \mathcal{U}_k \Sigma_k \mathcal{V}_k^T$ forms the best k-rank approximation to K in the Frobenius norm sense. In other words, for any $k \in [1, r]$, the minimal $k-$rank approximation loss to K is a deterministic function of singular values. Thus, we use singular value–based metrics to evaluate the compressibility of key and value spaces in each attention layer, as detailed in the experimental section. Moreover, the right singular vectors $\mathcal{V}$ of K can be used to construct the dataset-dependent optimal k-rank approximation of W^K, given W^K, as $\widetilde{W}^K = W^K \mathcal{V}_k \mathcal{V}_k^T$. A detailed proof is provided in the appendix C.1.

2.2 Incremental SVD Algorithm for Dataset-level KV-cache

A key challenge is that the size of $K \in \mathbb{R}^{l \times m_h d_h}$ scales up with the number of tokens l in a Dataset, rendering direct SVD on K impractical due to excessive memory and computational demands. We propose a novel algorithm that mitigates memory and computation bottlenecks through batch computation without sacrificing accuracy. This method only requires holding and updating a $m_h d_h$-by-$m_h d_h$ covariance matrix, avoiding the direct SVD of K and still generating the mathematically equivalent singular values Σ and right singular vectors $\mathcal{V}$. Algorithm 1 shows the pseudo-code of our method. More details about the proof is included in appendix C.2.

Algorithm 1 An adaptive algorithm to compute Σ and $\mathcal{V}$ of K

Input: Dataset containing l tokens in total; LLM model weights W^K
Output: singular values Σ and right singular vectors $\mathcal{V}$ of K

$C \leftarrow m_h d_h \times m_h d_h$ zero matrix ▷ Initialize covariance matrix to zero
for $t = 1, \ldots, l$ **do**
  $\mathbf{k}_t \leftarrow \mathbf{x}_t W^K$
  $C \leftarrow C + \mathbf{k}_t^T \mathbf{k}_t$ ▷ Update covariance matrix in each step
end for
$\mathcal{V}, \Sigma^2, \mathcal{V}^T \leftarrow$ eigen-decomposition(C) ▷ perform eigen-decomposition on the final covariance matrix
return $\Sigma, \mathcal{V}$

3 Experiment

We evaluate our method on 5 open-source LLM series of varying sizes including Qwen3 (4B, 8B) [12], Mistral-7B [13], Gemma-1.1 (2B, 7B) [14], and Phi-3-mini-128k-instruct [15], where Gemma-1.1[3] is a recent update of the original instruction-tuned Gemma, incorporating a new RLHF method that improves overall performance. To study the data-dependence of KV-cache compression, we evaluate our method on 5 datasets including Alpaca [16], MedAlpaca [17], CodeAlpaca [18], WizardCoder [19], and FunctionCall[4], spanning multiple domains. We also evaluate rank patterns across different languages using the multilingual split of the VisR-Bench dataset [20], which comprises 15 languages. More experimental details are provided in Appendix D.

3.1 Evaluation Metric

We evaluate compression potential and downstream performance using two complementary metrics. The *Normalized Effective Rank* (NER) [21] quantifies the data-dependent, per-layer low-rank structure of the KV-cache. For a matrix M with singular values $\{\sigma_i\}$, the effective rank is $\exp(-\sum_i p_i \log p_i)$ where $p_i = \sigma_i / \sum_j \sigma_j$, i.e., the exponential of the Shannon entropy of the normalized spectrum; normalizing by the matrix dimension yields a score in $[0, 1]$, with lower values indicating stronger low-rankness. To assess the impact on language modeling quality, we report *perplexity* (PPL) [22], defined as the exponential of the empirical cross-entropy between the data and model distributions. Lower PPL means the model assigns higher probability to the observed data, while higher PPL indicates degraded predictive performance.

3.2 Benchmarking Experiment on KV-Cache Compressibility

3.2.1 Average NER across Models and Datasets

Table 1 reports the mean NER of keys (NER-K) and values (NER-V), averaged over all attention layers across 7 models on 5 datasets and 7 languages. Complete results are provided in Table E.1. We highlight four key findings: **1. Keys are more compressible than values:** Across all models and datasets, keys consistently exhibit lower NER, reflecting higher compressibility. **2. Language effects outweigh domain effects:** English datasets from different domains yield similar NER, whereas

[3] Gemma-1.1: https://huggingface.co/google/gemma-1.1-7b-it
[4] glaiveai/glaive-function-calling-v2: https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

Datasets	Qwen3-4B		Qwen3-8B		Gemma-2B		Gemma-7B		Mistral-7B		Phi-3		LLaMA-2-7B	
	K	V	K	V	K	V	K	V	K	V	K	V	K	V
Multi-domain Datasets														
Alpaca	0.428	0.724	0.452	0.753	0.612	0.900	0.359	0.469	0.449	0.773	0.409	0.616	0.023	0.464
MedAlpaca	0.429	0.723	0.452	0.752	0.594	0.889	0.344	0.455	0.441	0.771	0.403	0.604	0.176	0.310
CodeAlpaca	0.421	0.708	0.443	0.737	0.589	0.869	0.321	0.429	0.420	0.733	0.380	0.571	0.028	0.337
WizardCoder	0.425	0.726	0.447	0.753	0.597	0.889	0.329	0.445	0.420	0.750	0.385	0.587	0.265	0.304
FunctionCall	0.432	0.731	0.451	0.756	0.608	0.900	0.342	0.458	0.432	0.762	0.397	0.604	0.135	0.449
Multilingual Question in VisR-Bench Datasets														
English	0.424	0.717	0.446	0.745	0.597	0.884	0.337	0.448	0.430	0.753	0.393	0.593	0.088	0.141
German	0.383	0.640	0.400	0.666	0.536	0.802	0.305	0.400	0.392	0.676	0.345	0.523	0.092	0.138
Italian	0.377	0.632	0.392	0.655	0.529	0.793	0.299	0.393	0.387	0.669	0.339	0.515	0.084	0.135
Swedish	0.371	0.620	0.387	0.645	0.526	0.794	0.296	0.389	0.381	0.656	0.322	0.486	0.078	0.128
Spanish	0.367	0.629	0.384	0.654	0.523	0.790	0.299	0.396	0.381	0.665	0.334	0.513	0.064	0.095
Japanese	0.361	0.619	0.380	0.648	0.506	0.775	0.276	0.369	0.364	0.636	0.321	0.467	0.079	0.125
Arabic	0.337	0.582	0.354	0.609	0.503	0.769	0.270	0.361	0.338	0.594	0.288	0.413	0.052	0.173

Table 1: Average NER of keys and values across all layers of 7 models on diverse datasets

multilingual results show larger variation. **3. Earlier model is more compressible:** Models such as LLaMA-2-7B show dramatically lower NER than recent ones (e.g., Qwen3-4B), likely due to smaller key/value dimensions and weaker rank utilization. **4. Rank collapse in low-resource settings:** Languages with limited training data (e.g., Arabic) show unusually low NER, likely due to under-trained token embeddings collapsing into lower-rank spaces, suggesting NER as a potential diagnostic for coverage gaps.

3.2.2 Layer-wise NER Patterns

Figure 1 presents the layer-wise NER of Qwen3-4B on the 5 datasets and on 3 selected language subsets of VisR-Bench, with additional plots for other models included in the appendix E.2. The results show that NER is not uniform across layers. This heterogeneity suggests that future KV-cache compression should be layer-aware, as applying a uniform compression ratio may overly degrade high-rank layers while missing opportunities for stronger reduction in lower-rank ones.

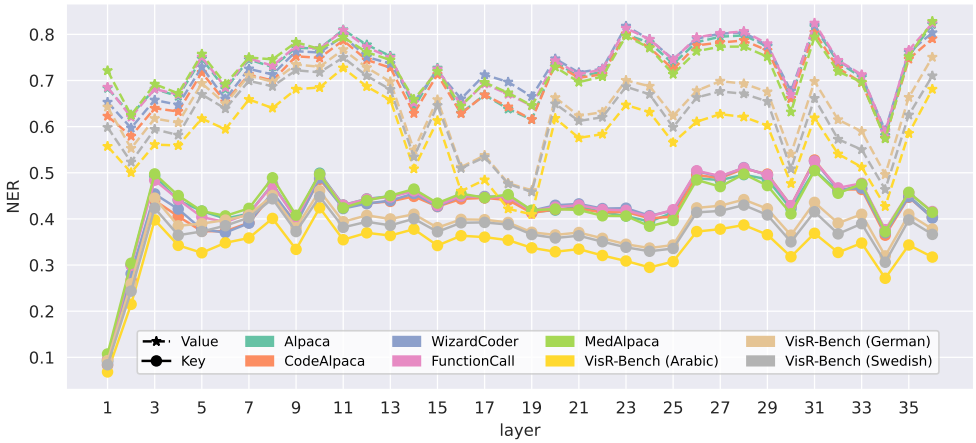

Figure 1: Layer-wise NER of key and value representations in Qwen3-4B, evaluated on 5 datasets and 3 languages from the VisR-Bench benchmark.

3.2.3 Effect of KV-cache Compression on PPL

We show the perplexity (PPL) of LLaMA-2-7B and Qwen3-4B across a grid of KV-cache compression ratios using heatmaps, included in Appendix E.3. Each heatmap reports PPL on one dataset. LLaMA-2-7B exhibits stable behavior, with only small PPL increases even at higher compression ratios, whereas Qwen3-4B is more sensitive, showing more drastic degradation. These results demonstrate that NER correlates positively with compressibility as reflected by PPL changes.

4 Conclusion

We presented an incremental SVD method for dataset-level KV-cache analysis that yields optimal low-rank projections with low memory cost, enabling evaluation of compressibility through singular value distributions. Experiments across models, domains, and languages reveal systematic patterns in key/value compressibility and show that normalized effective rank correlates with perplexity under compression. Our results provide a practical tool and guidance for designing future dynamic KV-cache compression methods.

References

[1] 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.

[2] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[3] Myunghyun Rhee, Joonseop Sim, Taeyoung Ahn, Seungyong Lee, Daegun Yoon, Euiseok Kim, Kyoung Park, Youngpyo Joo, and Hosik Kim. Hpu: High-bandwidth processing unit for scalable, cost-effective llm inference via gpu co-processing. *arXiv preprint arXiv:2504.16112*, 2025.

[4] Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

[5] Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms. *arXiv preprint arXiv:2502.14837*, 2025.

[6] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

[7] Hao Yu and Jianxin Wu. Compressing transformers: features are low-rank, but weights are not! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11007–11015, 2023.

[8] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*, 2024.

[9] Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Drone: Data-aware low-rank compression for large nlp models. *Advances in neural information processing systems*, 34:29321–29334, 2021.

[10] Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2024.

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

[12] Qwen Team. Qwen3 technical report, 2025.

[13] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[14] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[15] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[16] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[17] Tianyu Han, Lisa C Adams, Jens-Michalis Papaioannou, Paul Grundmann, Tom Oberhauser, Alexander Löser, Daniel Truhn, and Keno K Bressem. Medalpaca–an open-source collection of medical conversational ai models and training data. *arXiv preprint arXiv:2304.08247*, 2023.

[18] Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. `https://github.com/sahil280114/codealpaca`, 2023.

[19] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2025.

[20] Jian Chen, Ming Li, Jihyung Kil, Chenguang Wang, Tong Yu, Ryan Rossi, Tianyi Zhou, Changyou Chen, and Ruiyi Zhang. Visr-bench: An empirical study on visual retrieval-augmented generation for multilingual long document understanding. *arXiv preprint arXiv:2508.07493*, 2025.

[21] Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pages 606–610. IEEE, 2007.

[22] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[23] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

[24] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112*, 2022.

[25] Habib Hajimolahoseini, Walid Ahmed, Mehdi Rezagholizadeh, Vahid Partovinia, and Yang Liu. Strategies for applying low rank decomposition to transformer-based models. In *36th Conference on Neural Information Processing Systems (NeurIPS2022)*, volume 6, 2022.

[26] Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Losparse: Structured compression of large language models based on low-rank and sparse approximation. In *International Conference on Machine Learning*, pages 20336–20350. PMLR, 2023.

[27] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International conference on machine learning*, pages 2793–2803. PMLR, 2021.

[28] Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 35:27198–27211, 2022.

[29] Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. *arXiv preprint arXiv:2406.04112*, 2024.

[30] Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models. In *International conference on machine learning*, pages 864–873. PMLR, 2020.

[31] Enric Boix-Adsera, Etai Littwin, Emmanuel Abbe, Samy Bengio, and Joshua Susskind. Transformers learn through gradual rank increase. *Advances in Neural Information Processing Systems*, 36:24519–24551, 2023.

[32] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[33] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37:16692–16723, 2024.

[34] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[35] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[36] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[37] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[38] Carl D Meyer. *Matrix analysis and applied linear algebra*. SIAM, 2023.

Appendix

A Related Work

Rank Analysis in Language Models: Early work has investigated the relationship between the rank of transformer weights or representations and model performance, seeking either to leverage low-rank structure for efficiency [23, 24, 25, 26], to prevent rank collapse that limits expressivity [27, 28, 29], or to maximize rank utilization for enhanced modeling capacity [30, 31]. With the growing use of large language models (LLMs), research has turned to their inherent low-rank properties. LoRA [32] leverages this structure during fine-tuning, showing that many weight updates lie in low-dimensional subspaces. Loki [33] examined the key representations in attention layers and found that they often reside in lower-dimensional subspaces across models and datasets, which can be used for efficient sparse attention. These directions have also motivated growing efforts on KV-cache compression [4] to address the deployment bottleneck in reading and storing the KV cache [34]. Our work introduces a fine-grained method for evaluating the compressibility of KV caches in LLMs through effective rank analysis, uncovering layer-wise and data-dependent patterns that can inform the design of dynamic and adaptive compression strategies.

Low-Rank KV-cache Compression: DeepSeek [35] introduced Multi-head Latent Attention (MLA), which applies low-rank joint KV cache compression to enable scalable inference, unlike MQA [36] and GQA [37] which reduce parameters by merging heads in multi-head attention (MHA) [1]. MHA2MLA [5] and PALU [6] applies SVD to compress key and value projection weights, converting models based on MHA into the MLA structure for reduced KV cache size. However, this approach targets only the projection weights, while prior work [7] has shown that transformers weights typically have higher rank than the output features (keys/values), suggesting that data-dependent KV-cache compression is more effective. In this direction, DRONE [9] proposed a closed-form solution for data-aware low-rank compression of projected keys/values, and SVD-LLM [8] introduced an incremental optimization based on Cholesky decomposition [38] that achieves the same optimal compression loss with lower memory overhead. In comparison, our method achieves the same optimality with a much simpler formulation; moreover, it explicitly computes the SVD of key/value representations, whereas SVD-LLM only recovers the optimal compression matrix.

B Preliminary

B.1 MHA, MQA and GQA

We first introduce the standard MHA and two of its variants—MQA and GQA. Given an input embedding vector, MHA project it into key and value vectors for each attention head, causing the KV cache size to scale linearly with the number of heads. In contrast, MQA and GQA reduce the KV cache size by grouping heads and sharing the same key and value vectors within each group.

Focusing on GQA, the most general technique among the three, we define d_e, m_h, d_h and m_g as the embedding dimension, number of heads, dimension per head and number of groups, respectively. Given an input embedding vector $\mathbf{x}_t \in \mathbb{R}^{d_e}$ corresponding to the t-th token, GQA divides the m_h attention heads into m_g groups. Formally, this grouping can be described by a helper function g, which maps from head indices $\{1, ..., m_h\}$ to group indices $\{1, ..., m_g\}$ as,

$$g(i) = \left\lceil i \Big/ \frac{m_h}{m_g} \right\rceil, \quad \forall i \in \{1, ..., m_h\} \tag{B.1}$$

Then it projects $\mathbf{x}_t$ into query $\mathbf{q}_t \in \mathbb{R}^{1\times m_h d_h}$, key $\mathbf{k}_t \in \mathbb{R}^{1\times m_g d_h}$, and value vector $\mathbf{v}_t \in \mathbb{R}^{1\times m_g d_h}$ as follows:

$$[\mathbf{q}_{t,1}, ..., \mathbf{q}_{t,m_h}] = \mathbf{q}_t = \mathbf{x}_t W^Q \tag{B.2}$$

$$[\mathbf{k}_{t,1}, ..., \mathbf{k}_{t,m_g}] = \mathbf{k}_t = \mathbf{x}_t W^K \tag{B.3}$$

$$[\mathbf{v}_{t,1}, ..., \mathbf{v}_{t,m_g}] = \mathbf{v}_t = \mathbf{x}_t W^V \tag{B.4}$$

where $\mathbf{q}_{t,i}, \forall i \in \{1, ..., m_h\}$ denote the query vector for each attention head, and $\mathbf{k}_{t,i}, \mathbf{v}_{t,i}, \forall i \in \{1, ..., m_g\}$ represent the key and value vector for each group. The matrices $W^Q \in \mathbb{R}^{d_e \times m_h d_h}$ and

$W^K, W^V \in \mathbb{R}^{d_e \times m_g d_h}$ denote learnable model parameters. The attention of each head and the final projected output are computed as,

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \text{Softmax}_j \left(\frac{\mathbf{q}_{t,i} \mathbf{k}_{j,g(i)}^T}{\sqrt{d_h}} \right) \mathbf{v}_{j,g(i)}, \quad \forall i \in \{1, ..., m_h\} \tag{B.5}$$

$$\mathbf{y}_t = [\mathbf{o}_{t,1}, ..., \mathbf{o}_{t,m_h}] W^O \tag{B.6}$$

where $\mathbf{o}_{t,i} \in \mathbb{R}^{1 \times d_h}$, $W^O \in \mathbb{R}^{m_h d_h \times d_e}$ and $\mathbf{y}_t \in \mathbb{R}^{1 \times d_e}$ denote the attention output for i-th head, the output projection matrix and the projected output respectively.

Note that when $m_g = m_h$, GQA reduces to standard MHA, and when $m_g = 1$, it specializes MQA.

B.2 MLA

Unlike MHA and its variants, MLA projects the input embedding $\mathbf{x}_t \in \mathbb{R}^{d_e}$ of t-th token into two distinct spaces: a joint latent KV space and a decoupled key space designed to incorporate RoPE. Formally, this can be expressed as:

$$\mathbf{c}_t^{KV} = \mathbf{x}_t W^{DKV} \tag{B.7}$$

$$\mathbf{k}_t^R = \text{RoPE}(\mathbf{x}_t W^{KR}) \tag{B.8}$$

where $\mathbf{c}_t^{KV} \in \mathbb{R}^{1 \times d_c}$ and $\mathbf{k}_t^R \in \mathbb{R}^{1 \times d_R}$ denote the joint latent KV vector and the RoPE-encoded decoupled key vector, receptively. The projection matrices $W^{DKV} \in \mathbb{R}^{d_e \times d_c}$ and $W^{KR} \in \mathbb{R}^{d_e \times d_R}$ handle the corresponding down-projections. During attention calculation, $\mathbf{c}_t^{KV}$ is up-projected to get the key and value vectors, while the query vector is computed directly from the input embedding $\mathbf{x}_t$. These operations are described by following equations (B.9) and (B.10):

$$\begin{aligned} [\mathbf{k}_{t,1}^C, ..., \mathbf{k}_{t,n}^C] = \mathbf{k}_t^C &= \mathbf{c}_t^{KV} W^{UK} \\ \mathbf{k}_{t,i} &= [\mathbf{k}_{t,i}^C, \mathbf{k}_t^R] \\ [\mathbf{v}_{t,1}^C, ..., \mathbf{v}_{t,m_h}^C] = \mathbf{v}_t^C &= \mathbf{c}_t^{KV} W^{UV} \end{aligned} \tag{B.9}$$

$$\begin{aligned} \mathbf{c}_t^Q &= \mathbf{x}_t W^{DQ} \\ [\mathbf{q}_{t,1}^C, ..., \mathbf{q}_{t,m_h}^C] = \mathbf{q}_t^C &= \mathbf{c}_t^Q W^{UQ} \\ [\mathbf{q}_{t,1}^R, ..., \mathbf{q}_{t,m_h}^R] = \mathbf{q}_t^R &= \text{RoPE}(\mathbf{c}_t^Q W^{QR}) \\ \mathbf{q}_{t,i} &= [\mathbf{q}_{t,i}^C, \mathbf{q}_{t,i}^R] \end{aligned} \tag{B.10}$$

where $W^{UK}, W^{UV} \in \mathbb{R}^{d_c \times m_h d_h}$ are up-projection matrices for key and value vectors, respectively. The matrices $W^{DQ} \in \mathbb{R}^{d_e \times d_c'}$ and $W^{UQ} \in \mathbb{R}^{d_c' \times m_h d_h}$ serve as the down- and up-projection for queries, while $W^{QR} \in \mathbb{R}^{d_c' \times m_h d_h}$ is the up-projection matrix used to incorporate RoPE for the decoupled query vector. Note that both $\mathbf{k}_{t,i}$ and $\mathbf{q}_{t,i}$ are concatenations of their NoPE and RoPE components.

Using $\mathbf{q}_{t,i}$, $\mathbf{k}_{t,i}$ and $\mathbf{v}_{t,i}^C$, the attention of each head and the final projected output are computed as,

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \text{Softmax}_j \left(\frac{\mathbf{q}_{t,i} \mathbf{k}_{j,i}^T}{\sqrt{d_h + d_R}} \right) \mathbf{v}_{j,i}^C, \quad \forall i \in \{1, ..., m_h\} \tag{B.11}$$

$$\mathbf{y}_t = [\mathbf{o}_{t,1}, ..., \mathbf{o}_{t,m_h}] W^O \tag{B.12}$$

A key merit of MLA lies in its caching efficiency during token generation: only $\mathbf{c}_t^{KV}$ and $\mathbf{k}_t^R$ need to be cached, resulting in a cache size of $d_c + d_R$. Since both $d_c << m_h d_h$ and $d_R << m_h d_h$, MLA reduces KV cache size significantly compared to MHA, which requires caching $\mathbf{k}_t \in \mathbb{R}^{1 \times m_h d_h}$ and $\mathbf{v}_t \in \mathbb{R}^{1 \times m_h d_h}$ for t-th token.

C Theoretical Proof

C.1 Optimality of the Low-Rank Projection Matrix

Given singular values Σ and right singular vectors $\mathcal{V}$ of K, we can directly recover the dataset-dependent optimal k-rank approximation of W^K as,

$$\widetilde{W}^K = W^K \mathcal{V}_k \mathcal{V}_k^T \tag{C.1}$$

To see why $\widetilde{W}^K$ is optimal to minimize the error: $||X^{(i)}W^{K^{(i)}} - X^{(i)}\widetilde{W}^{K^{(i)}}||_F$, consider the following:

$$\begin{aligned} X\widetilde{W}^K = XW^K\mathcal{V}_k\mathcal{V}_k^T &\overset{T_1}{=} K\mathcal{V}_k\mathcal{V}_k^T \\ &\overset{T_2}{=} \mathcal{U}\Sigma(\mathcal{V}^T\mathcal{V}_k)\mathcal{V}_k^T \\ &\overset{T_3}{=} \mathcal{U}\left(\Sigma\begin{bmatrix} I_k \\ 0 \end{bmatrix}\right)\mathcal{V}_k^T \\ &= \left(\mathcal{U}\begin{bmatrix} \Sigma_k \\ 0 \end{bmatrix}\right)\mathcal{V}_k^T \\ &= \mathcal{U}_k\Sigma_k\mathcal{V}_k^T \end{aligned} \tag{C.2}$$

Where I_k is a k by k identity matrix. T_1 holds because by definition $K = XW^K$; T_2 holds because $\mathcal{U}\Sigma\mathcal{V}^T$ is the SVD of K; T_3 holds because singular vectors form an orthogonal basis, thus $\mathcal{V}_k^T\mathcal{V}_k$ yields a k by k identify matrix. Recall that by the Eckart-Young-Mirsky theorem [11], $\mathcal{U}_k\Sigma_k\mathcal{V}_k^T$ forms the best k-rank approximation of $K = XW^K$, we can conclude that $W^K\mathcal{V}_k\mathcal{V}_k^T$ is optimal to the optimization problem.

During LLM inference, a direct implementation is to replace W^K with a pair of down-projection $W^K\mathcal{V}_k$ and up-projection $\mathcal{V}_k^T$ matrices. This allows caching the low-dimensional key vector $\mathbf{x}_t W^K\mathcal{V}_k$ instead of the full-dimensional $\mathbf{x}_t W^K$ given t-th token, reducing both the memory and bandwidth consumption.

C.2 Correctness of the Incremental SVD Algorithm

For each token in a Dataset containing l tokens, $\mathbf{k}_t^T\mathbf{k}_t$ is computed and the covariance matrix C is updated. After l iterations, we will have the complete covariance matrix of K as $C = K^TK = \sum_{t=1}^{l}\mathbf{k}_t^T\mathbf{k}_t$. Finally we perform eigen-decomposition on C to obtain the singular values Σ and right singular vectors $\mathcal{V}$ of K.

To see why the final step of Algorithm 1 generates mathematically equivalent Σ and $\mathcal{V}$, consider the following proof,

$$K^TK = (\mathcal{U}\Sigma\mathcal{V}^T)^T\mathcal{U}\Sigma\mathcal{V}^T = \mathcal{V}\Sigma^T\mathcal{U}^T\mathcal{U}\Sigma\mathcal{V}^T = \mathcal{V}\Sigma^2\mathcal{V}^T \tag{C.3}$$

D Experiment Setup Details

D.1 Optimization Setup

All experiments are conducted on machines equipped with 8× NVIDIA A800 GPUs (80GB each), though all evaluations are executed on a single GPU without distributed computation. We use PyTorch 2.7.1 and Hugging Face Transformers 4.53.2 for model loading, compression, and inference. All evaluations are conducted in inference mode without gradient computation.

D.2 Datasets

To study the data-dependence of KV-cache compression, we evaluate our method on n datasets, spanning diverse English instruction-following tasks across multiple domains and multilingual QA. For English evaluation, the datasets cover general instruction following, code generation, medical QA, and function calling, including Alpaca [16], MedAlpaca [17], CodeAlpaca [18], WizardCoder [19], and FunctionCall[5]. For multilingual evaluation, we use the queries from the multilingual split of VisR-Bench [20], a question-driven, retrieval benchmark spanning English and 15 non-English languages (Spanish, Italian, German, French, Dutch, Arabic, Croatian, Japanese, Swedish, Vietnamese, Portuguese, Finnish, Czech, Slovenian, and Danish)—allowing us to assess performance across a linguistically diverse setting.

E Additional Experiment Results

E.1 Average NER Results

Datasets	Qwen3-4B		Qwen3-8B		Gemma-2B		Gemma-7B		Mistral-7B		Phi-3		LLaMA-2-7B	
	K	V	K	V	K	V	K	V	K	V	K	V	K	V
Multi-domain Datasets														
Alpaca	0.428	0.724	0.452	0.753	0.612	0.900	0.359	0.469	0.449	0.773	0.409	0.616	0.023	0.464
MedAlpaca	0.429	0.723	0.452	0.752	0.594	0.889	0.344	0.455	0.441	0.771	0.403	0.604	0.176	0.310
CodeAlpaca	0.421	0.708	0.443	0.737	0.589	0.869	0.321	0.429	0.420	0.733	0.380	0.571	0.028	0.337
WizardCoder	0.425	0.726	0.447	0.753	0.597	0.889	0.329	0.445	0.420	0.750	0.385	0.587	0.265	0.304
FunctionCall	0.432	0.731	0.451	0.756	0.608	0.900	0.342	0.458	0.432	0.762	0.397	0.604	0.135	0.449
Multilingual Question in VisR-Bench Datasets														
English	0.424	0.717	0.446	0.745	0.597	0.884	0.337	0.448	0.430	0.753	0.393	0.593	0.088	0.141
Czech	0.383	0.627	0.401	0.652	0.536	0.803	0.292	0.383	0.385	0.660	0.313	0.470	0.099	0.148
German	0.383	0.640	0.400	0.666	0.536	0.802	0.305	0.400	0.392	0.676	0.345	0.523	0.092	0.138
Italian	0.377	0.632	0.392	0.655	0.529	0.793	0.299	0.393	0.387	0.669	0.339	0.515	0.084	0.135
Dutch	0.376	0.628	0.395	0.657	0.534	0.802	0.299	0.394	0.385	0.664	0.331	0.499	0.116	0.108
Croatian	0.375	0.620	0.393	0.645	0.525	0.791	0.292	0.382	0.383	0.659	0.320	0.479	0.081	0.132
French	0.373	0.633	0.390	0.657	0.532	0.797	0.301	0.397	0.387	0.671	0.340	0.519	0.087	0.137
Vietnamese	0.373	0.625	0.392	0.653	0.542	0.814	0.291	0.384	0.367	0.630	0.305	0.442	0.071	0.119
Swedish	0.371	0.620	0.387	0.645	0.526	0.794	0.296	0.389	0.381	0.656	0.322	0.486	0.078	0.128
Spanish	0.367	0.629	0.384	0.654	0.523	0.790	0.299	0.396	0.381	0.665	0.334	0.513	0.064	0.095
Slovenian	0.366	0.603	0.383	0.628	0.518	0.785	0.281	0.369	0.372	0.642	0.306	0.458	0.075	0.121
Portuguese	0.362	0.614	0.378	0.639	0.521	0.785	0.284	0.375	0.379	0.656	0.326	0.498	0.083	0.133
Japanese	0.361	0.619	0.380	0.648	0.506	0.775	0.276	0.369	0.364	0.636	0.321	0.467	0.079	0.125
Finnish	0.360	0.597	0.378	0.625	0.502	0.769	0.280	0.369	0.353	0.616	0.305	0.455	0.073	0.118
Arabic	0.337	0.582	0.354	0.609	0.503	0.769	0.270	0.361	0.338	0.594	0.288	0.413	0.052	0.173
Average	0.387	0.653	0.407	0.679	0.548	0.822	0.306	0.405	0.395	0.684	0.345	0.520	0.083	0.134

Table E.1: Average NER of keys and values across all layers of 7 models on multilingual split of VisR-bench covering 15 languages

[5] glaiveai/glaive-function-calling-v2: https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

E.2 Layer-wise NER Results

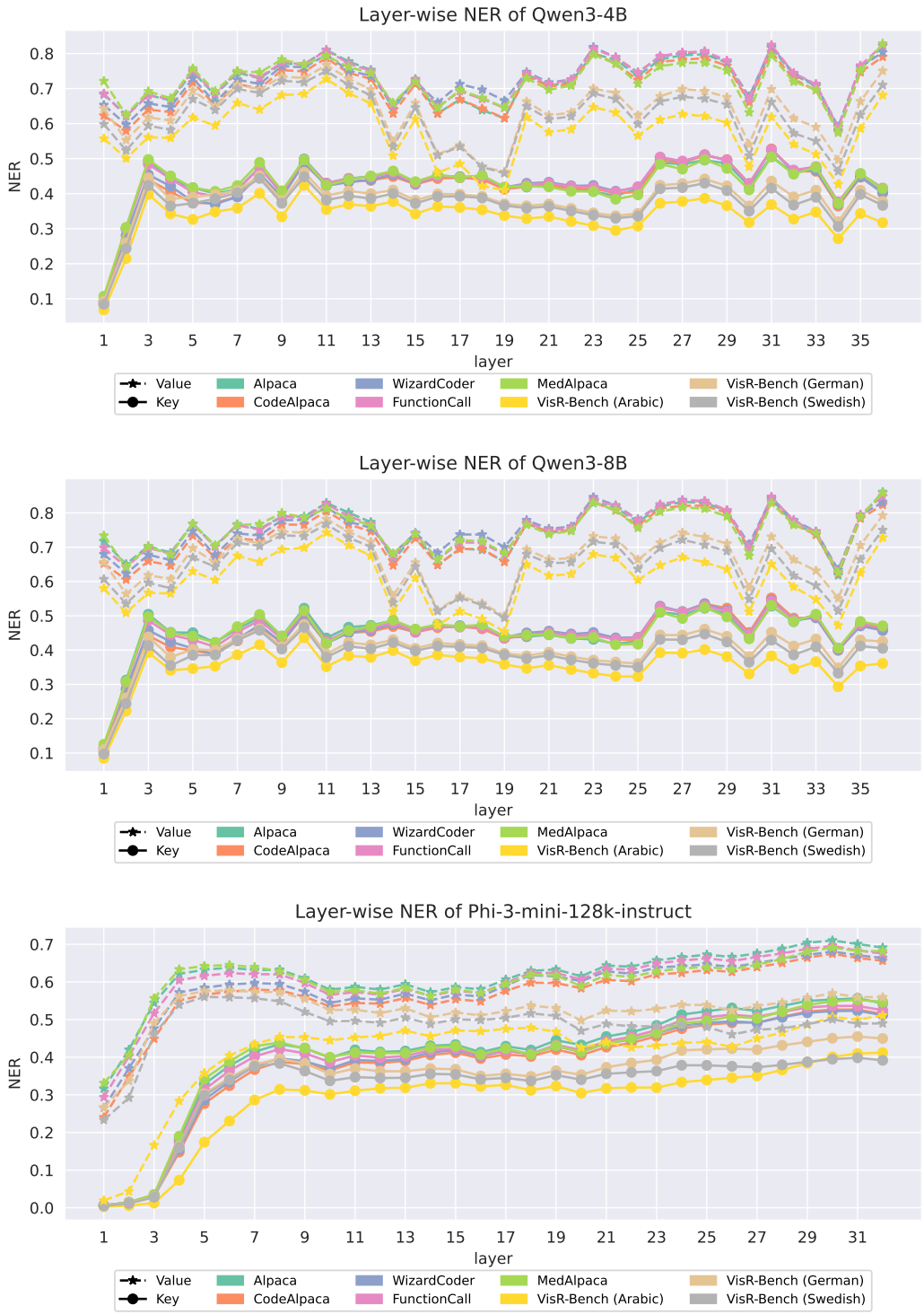

Figure E.1: Layer-wise NER of key and value representations in Qwen3-4B, Qwen3-8B, and Phi-3-mini evaluated on 5 datasets and 3 languages from the VisR-Bench benchmark.

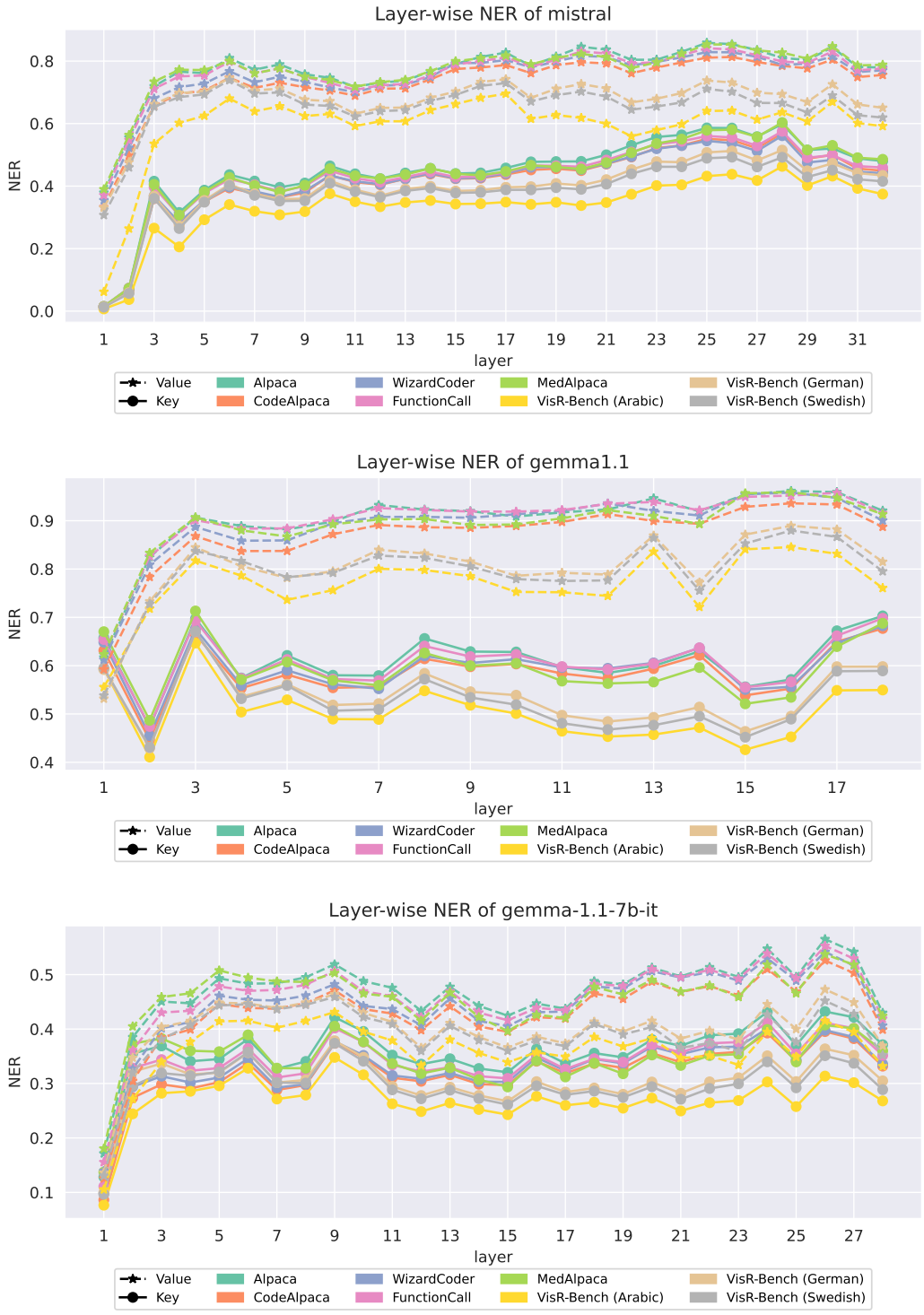

Figure E.2: Layer-wise NER of key and value representations in mistral, gemma1.1, and gemma-1.1-7b-it evaluated on 5 datasets and 3 languages from the VisR-Bench benchmark.

E.3 PPL Heatmap

Figure E.3: PPL heatmap of LLaMA-2-7B on 6 datasets.

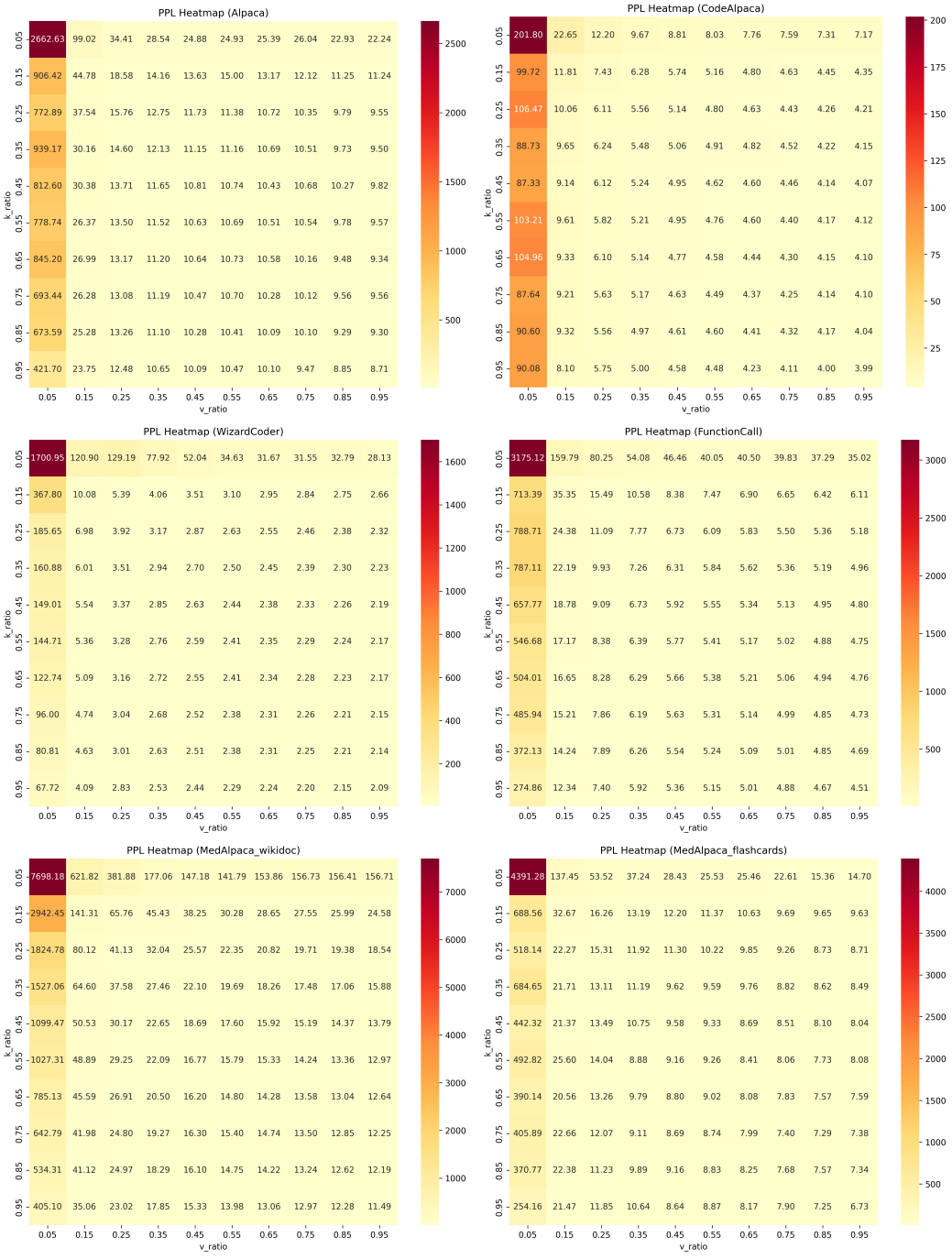

Figure E.4: PPL heatmap of Qwen3-4B on 6 datasets.