MatShrink: Lossless weight compression for transformers

Nils Graef[*], **Siddharth Mohan**
OpenMachine

Abstract

MatShrink reduces the number of weights for back-to-back matrices in general, and for transformer models in particular. Unlike standard weight compression and pruning techniques, MatShrink uses matrix inversion to eliminate weights in a mathematically equivalent way and thus without compromising model accuracy. MatShrink is applicable to both inference and training: Existing models can be retrofitted with MatShrink in a mathematically equivalent way, and future models can use MatShrink for both training and inference. We also propose a simplified MLA (multi-head latent attention) scheme. See [1] for code and more transformer tricks.

For two back-to-back weight matrices W_A and W_B, Fig. 1 illustrates how MatShrink reduces the size of W_B in a mathematically equivalent way by using matrix inversion.

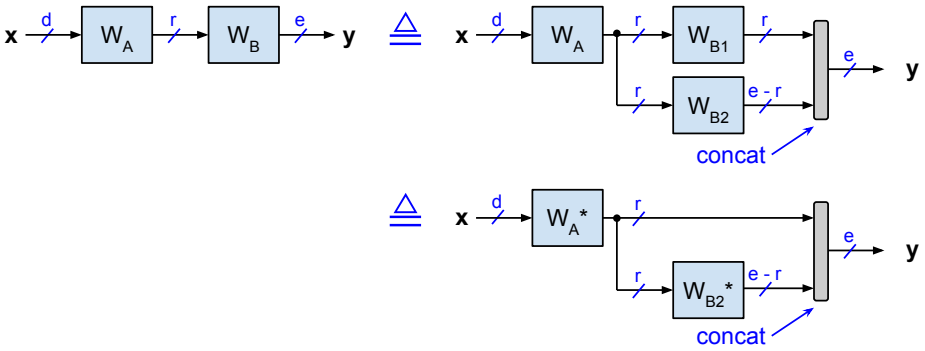

Figure 1: Mathematically equivalent implementations of two back-to-back weight matrices W_A and W_B with rank r, where $d > r$ and $e > r$, which reduces the total number of weights by r^2.

Specifically, W_A is a $d \times r$ matrix, W_B is an $r \times e$ matrix with rank r, where $d > r$ and $e > r$. We can split W_B into two submatrices $W_{B1} \in \mathbb{R}^{r \times r}$ and W_{B2} such that $W_B = [W_{B1}, W_{B2}]$. We can then eliminate W_{B1} by merging it into W_A as $W_A^* = W_A W_{B1}$ and by changing W_{B2} to $W_{B2}^* = W_{B1}^{-1} W_{B2}$. This saves r^2 weights and r^2 multiply operations per token x. The following equation shows the mathematical identity of the modified back-to-back matrices, where I is the $r \times r$ identity matrix:

$$W = W_A \cdot W_B = W_A \cdot [W_{B1}, W_{B2}] = W_A W_{B1} \cdot \left[I, W_{B1}^{-1} W_{B2}\right] = W_A^* \cdot [I, W_{B2}^*]$$

Inverting the submatrix W_{B1} requires that this submatrix is invertible, which is often the case because it is extremely rare for large matrices to be non-invertible. In the rare case that W_{B1} is non-invertible, we can first permute the columns of the original matrix W_B such that the first r columns of the permuted matrix form an invertible submatrix W_{B1}.

[*] info@openmachine.ai

For completeness, if $e = r$, then the entire matrix W_{B2} is eliminated. In general, if $e \leq r$, then the two matrices W_A and W_B are fused into a single matrix $W^* = W_A \cdot W_B$ with only de weights (instead of $dr + re$ weights for the original matrices W_A and W_B). This type of weight fusion is utilized in [2, 3].

Alternative way. Alternatively, we can split matrix W_A into two submatrices $W_{A1} \in \mathbb{R}^{r \times r}$ and W_{A2} such that $W_A = [W_{A1}; W_{A2}]$. We can then eliminate W_{A1} as $W = [W_{A1}; W_{A2}]\, W_B = [I; W^*_{A2}]\, W^*_B$ with identity matrix $I \in \mathbb{R}^{r \times r}$ and where $W^*_B = W_{A1} W_B$ and $W^*_{A2} = W_{A2} W^{-1}_{A1}$, see Fig. 2. This also saves r^2 weights and r^2 multiply operations per token x.

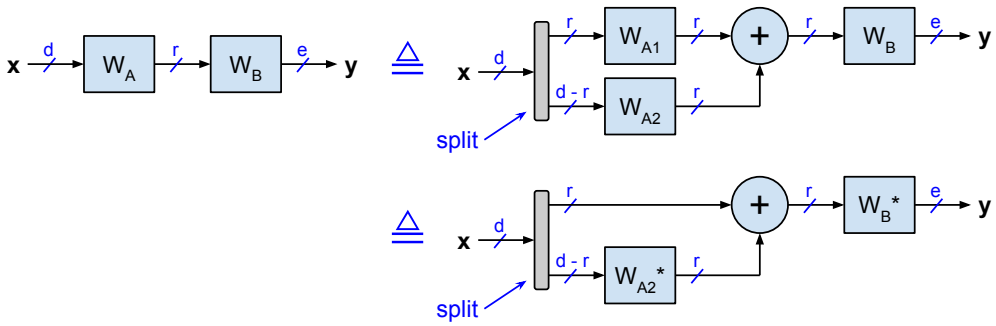

Figure 2: Alternative way of shrinking W_A instead of W_B

MatShrink for transformers. MatShrink reduces the number of weights for the following back-to-back weight matrices in transformer models:

- The V (value) and O (output) projections for each attention-head, see Fig. 3 and 4
- The Q (query) and K (key) projections for each attention-head (without the RoPE portion), see Fig. 5 and 6
- The latent projections of MLA (multi-head latent attention), see Fig. 7(b)

Related work. Many weight compression schemes for transformers have been proposed such as [4] and [5]. However, these schemes approximate the original weight matrices by using SVD (singular value decomposition) or other approximations. MatShrink on the other hand is not an approximation but an exact, mathematically equivalent optimization for back-to-back matrices. Furthermore, MatShrink is similar to slim attention [6] in its use of matrix inversion to compute projections from each other.

1 MatShrink for MHA transformers

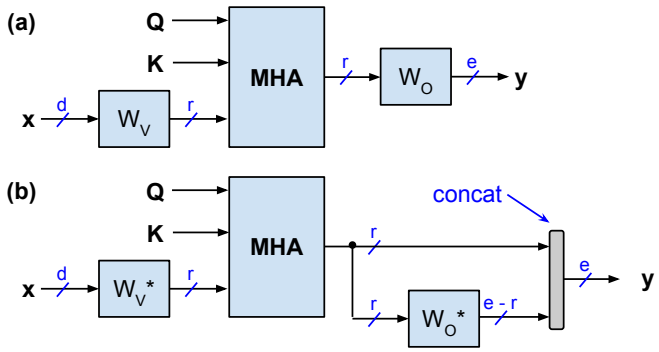

Figure 3: MatShrink for V and O projections of a single attention head: (a) original V and O projections; (b) equivalent implementation using MatShrink, which eliminates r^2 weights from W_O.

MatShrink for V and O projections. Note that the value (V) and output (O) projections for each head i of multi-head attention (MHA) are two back-to-back weight matrices $W_{V,i}$ and $W_{O,i}$ as

illustrated in Fig. 3 for a single attention head. Fig. 3 shows how MatShrink eliminates r^2 weights from the original O projection weight matrix W_O. Alternatively, Fig. 4 illustrates how the alternative MatShrink scheme from Fig. 2 can eliminate r^2 weights from the original V projection (instead of the O projection).

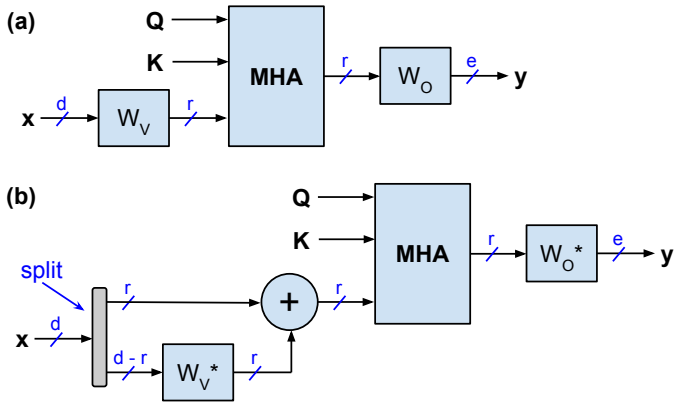

Figure 4: Alternative MatShrink implementation for V and O projections of a single attention head: (a) original V and O projections; (b) equivalent implementation using the alternative MatShrink scheme from Fig. 2, which eliminates r^2 weights from W_V.

For MHA, we can apply the MatShrink scheme to each head. Specifically:

- For the vanilla MHA with h heads, each head has dimension $d_k = d/h$, and $d = d_{\text{model}}$.
- So for the dimensions r and e of Fig. 1, we have $r = d/h$ and $e = d$.
- This saves $r^2 = d^2/h^2$ weights for each head, so d^2/h weights in total.
- Note that for single-head attention (where $h = 1$), we can save $2d^2$ weights: Specifically, we can merge the V and O weight matrices into a single $d \times d$ matrix; and the Q and K weight matrices into a single $d \times d$ matrix (if there is no RoPE).

Table 1 lists the configurations of various MHA transformer models, the number of weights for their attention projections, and the weight savings provided by MatShrink.

Table 1: Weight savings for MHA transformer models using MatShrink

Model	d	d_k	h	weights $d \times (d_k h)$	savings $d_k^2 h$	savings %
Whisper-tiny [7]	384	64	6	147K	25K	17%
CodeGemma-7B [8]	3,072	256	16	12.6M	1.0M	8%
T5-3B [9]	1,024	128	32	4.2M	0.5M	12%
T5-11B [9]	1,024	128	128	16.8M	2.1M	13%

MatShrink for Q and K projections. For models that don't use RoPE (such as Whisper [7] and T5 models [9]), the query (Q) and key (K) projections for each head i of MHA are two back-to-back weight matrices $W_{Q,i}$ and $W_{K,i}$ as illustrated in Fig. 5 for a single attention head.

Fig. 5 shows how MatShrink eliminates r^2 weights from the original Q weight matrix W_Q. Note that the queries Q^* and keys K^* generated by the modified linear layers W_Q^* and W_K^* are not identical to the original queries Q and keys K, but their dot-products p are identical, i.e. $p = Q \cdot K = Q^* \cdot K^*$.

For many models that use RoPE, we can also apply this trick as follows: Many modern transformer models use partial RoPE, which applies RoPE to only a portion of the head-dimension $d_k = d/h$, usually only to one half of d_k. So in this case $r = d_k/2 = d/(2h)$, which saves only $r^2 = d^2/(4h^2)$ weights for each head, so $d^2/(4h)$ weights in total.

For completeness, Fig. 6 illustrates an alternative implementation of MatShrink that removes r^2 weights from W_K instead of W_Q.

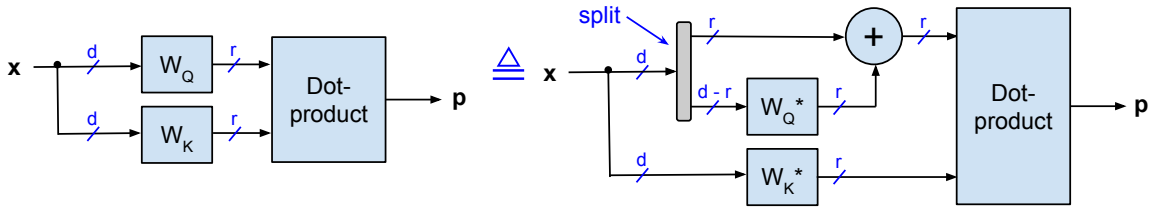

Figure 5: MatShrink for Q and K projections of a single attention head. Left: original Q and K projections and their dot-product p. Right: equivalent implementation using MatShrink, which eliminates r^2 weights from W_Q.

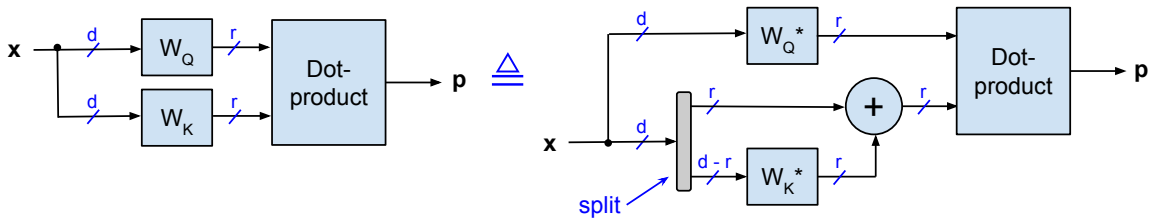

Figure 6: Alternative MatShrink implementation for Q and K projections of a single attention head. Left: original Q and K projections and their dot-product p. Right: equivalent implementation using MatShrink, which eliminates r^2 weights from W_K (instead of W_Q).

2 MatShrink for MLA transformers

Table 2 shows the configurations of various transformer models with MLA. We are using the following parameter names similar to [10]:

- For Q (query):
  - r_{Q}: rank of Q-latent projection
  - W_{DQ}: down-projection for Q
  - W_{UQ}: up-projection for Q-part without RoPE (aka NoPE)
  - W_{QR}: up-projection for Q-part with RoPE
- For KV (key-value):
  - r_{KV}: rank of KV-latent projection
  - W_{KR}: projection for K-part with RoPE (has its own cache, used for all queries as MQA)
  - W_{DKV}: down-projection for KV
  - W_{UK}: up-projection for K-part without RoPE (aka NoPE)
  - W_{UV}: up-projection for V

Table 2: Configurations of various MLA models

Model	Params	d	r_{Q}	r_{KV}	h	d_{NOPE}	d_{ROPE}
Perplexity R1-1776, DeepSeek-R1, V3	685B	7,168	1,536	512	128	128	64
Mistral Large 3 675B	675B	7,168	1,536	512	128	128	64
DeepSeek-V2.5	236B	5,120	1,536	512	128	128	64
DeepSeek-V2-lite, VL2-small [10]	16B	2,048	N/A	512	16	128	64
OpenBMB MiniCPM3-4B [11]	4B	2,560	768	256	40	64	32

DeepSeek's MLA (multi-head latent attention) scheme [10] has two latent projections, one for Q (queries) and one for KV (keys and values). We can apply MatShrink to each of them:

- The Q-latent projection and query (Q) projections are two back-to-back weight matrices W_{DQ} and W_{UQ}.
- The KV-latent projection and key/value (KV) projections are two back-to-back weight matrices W_{DKV} and the union of W_{UK} and W_{UV}.

We can also apply MatShrink to each V-O head and the non-RoPE portion of the Q-K heads. Specifically, we can apply the MatShrink to the MLA weight matrices in the following order:

1. Apply MatShrink to the V-O weight matrices.
2. Apply MatShrink to the NoPE portion (i.e. the non-RoPE portion) of the Q-K weight matrices.
3. Apply MatShrink to the Q-latent projections. This step must be done after applying MatShrink to the Q-K weights.
4. Apply MatShrink to the KV-latent projections. This step must be done after applying MatShrink to the V-O weights.

Applying MatShrink to the KV-latent projections not only reduces weight matrices and corresponding compute, it can also reduce the compute complexity as follows, where r_{KV} is the rank of the KV-latent projections.

- Option 1: Use the r_{KV} neurons that don't require a weight matrix as keys. The number of those keys is $r_{\text{KV}}/d_{\text{NOPE}}$. Then these keys can be directly used for the softmax arguments, which saves some computation complexity.
- Option 2: Use the r_{KV} neurons as values (instead of keys). Then these values can be directly multiplied with the softmax scores, which saves some compute complexity.

TODO: details the savings, perhaps with a new table

3 Simplified MLA

In this section we propose a simplification for DeepSeek's MLA (multi-head latent attention).

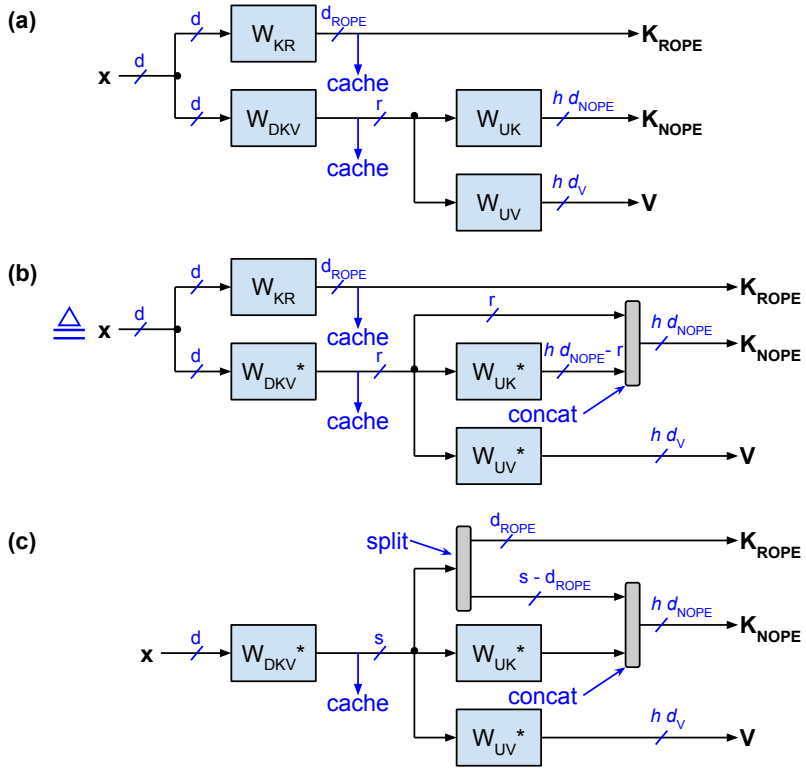

Figure 7: K and V projections for MLA. (a) original version; (b) equivalent version optimized by MatShrink; (c) proposed simplification

Fig. 7 shows the K and V projections of MLA and the proposed simplification:

- Fig. 7(a) shows the MLA projections for K (keys) and V (values). Note that a single d_{ROPE} head is shared among all query-heads, where $d_{\text{ROPE}} = 64$ or 32 usually.

- Fig. 7(b) shows the mathematically equivalent version with MatShrink applied to the weight matrices W_{DKV} and W_{UK}.
- Fig. 7(c) shows the proposed simplified MLA scheme where the d_{ROPE} units (or channels) are sourced directly from the latent cache, instead of having a separate cache and W_{KR} for the d_{ROPE} units:
  - Note that this simplified scheme is not mathematically identical to the standard MLA scheme shown in Fig. 7(a).
  - The rank s of the simplified scheme could be larger than r (e.g. $s = r + d_{\text{ROPE}}$) or slightly lower than this (e.g. $s = r$).
  - Advantages include: If $s > r$, then there is more usable rank for the keys and values. So the cached latent space is better utilized. And if $s < r + d_{\text{ROPE}}$ then the total cache size is reduced.

This simplification enhances MLA by directly leveraging the latent cache for RoPE components, potentially increasing effective rank and optimizing cache utilization. In DeepSeek-V2, standard MLA already reduces KV cache by compressing into latent vectors, but our proposal further streamlines this by eliminating separate RoPE projections, leading to additional memory savings especially for long-sequence inference.

4 MatShrink for GQA and MQA

MatShrink is not limited to MHA and MLA only. It's also applicable to GQA (grouped query attention) and MQA (multi-query attention). However, the savings are smaller than for MHA and MLA. Specifically, the savings are reduced by a factor g, where g is the number of queries that are shared among a single KV-pair, or in other words $g = n_{\text{heads}}/n_{\text{KV-heads}}$ (where n_{heads} is the number of query-heads, and $n_{\text{KV-heads}}$ is the number of KV-heads).

5 MatShrink for SVD

Approximate weight compression schemes such as [5] and [4] use SVD (singular value decomposition) to reduce the ranks of weight matrices, and thus reduce weights. This is applicable for example for the large weight matrices of the transformer's FFN (feedforward networks). The SVD decomposition factorizes the original matrix $W \in \mathbb{R}^{d \times e}$ into two matrices W_A and W_B where r is the compressed rank. After performing SVD and compressing the rank by a certain percentage, we can then eliminate r^2 weights using our MatShrink scheme. Note that reducing the rank by a certain percentage is not an exact implementation of the original matrix W but an approximation.

6 Conclusion

TODO, also add experimental results

References

[1] OpenMachine. Transformer tricks. 2024. URL https://github.com/OpenMachine-ai/transformer-tricks.

[2] Nils Graef. KV-weights are all you need for skipless transformers. April 2024. *arXiv:2404.12362*.

[3] Marko Karbevski and Antonij Mijoski. Key and Value weights are probably all you need: On the necessity of the Query, Key, Value weight triplet in encoder-only and decoder-only transformers. January 2026. *arXiv:2510.23912*.

[4] Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. MoDeGPT: Modular Decomposition for Large Language Model Compression. 2024. *arXiv:2408.09632*.

[5] Pratyusha Sharma, Jordan T Ash, and Dipendra Misra. The truth is in there: Improving reasoning in language models with LAyer-SElective Rank reduction. December 2023. *arXiv:2312.13558*.

[6] Nils Graef and Andrew Wasielewski. Slim attention: cut your context memory in half without loss – K-cache is all you need for MHA. March 2025. *arXiv:2503.05840*.

[7] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. December 2022. *arXiv:2212.04356*.

[8] CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Zhitao Gong, Jane Fine, Tris Warkentin, Ale Jakse Hartman, Bin Ni, Kathy Korevec, Kelly Schaefer, and Scott Huffman. CodeGemma: Open Code Models Based on Gemma. June 2024. *arXiv:2406.11409*.

[9] 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. October 2019. *arXiv:1910.10683*.

[10] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, et al. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. 2024. *arXiv:2405.04434*.

[11] Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Huarong Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM-V: A GPT-4V Level MLLM on Your Phone. 2024. *arXiv:2408.01800*.