The Impossibility of Inverse Permutation Learning in Transformer Models

Rohan Alur[*]
Bridgewater AIA Labs
Massachusetts Institute of Technology
ralur@mit.edu

Chris Hays[*]
Massachusetts Institute of Technology
jhays@mit.edu

Manish Raghavan
Massachusetts Institute of Technology
mragh@mit.edu

Devavrat Shah
Massachusetts Institute of Technology
devavrat@mit.edu

Abstract

We study the problem of inverse permutation learning in decoder-only transformers. Given a permutation and a string to which that permutation has been applied, the model is tasked with producing the original ("canonical") string. We argue that this task models a natural robustness property across a variety of reasoning tasks, including long-context retrieval, multiple choice QA and in-context learning.

Our primary contribution is an impossibility result: under weak assumptions, we show that an arbitrary depth, decoder-only transformer cannot learn this task. This result concerns the expressive capacity of decoder-only transformer models and is agnostic to training dynamics or sample complexity.

We give a pair of alternative constructions under which inverse permutation learning is feasible. The first of these highlights the fundamental role of the causal attention mask, and suggests a gap between the expressivity of encoder-decoder transformers and the more popular decoder-only architecture. The latter result is more surprising: we show that simply duplicating the input yields a construction under which inverse permutation learning is possible. We conjecture that this result may suggest an alternative mechanism by which chain-of-thought prompting or, more generally, intermediate "thinking" tokens can enable reasoning in large language models.

1 Introduction

One of the most striking features of modern large language models (LLMs) is the emergence of general-purpose reasoning abilities at scale. Even relatively early LLMs were capable of in-context learning [4], multiple choice question answering [14], long-context reasoning [20], and deductive logical reasoning tasks [13]. It was not obvious, a priori, that a single model could be trained to handle such a diverse range of tasks, since each might appear to demand distinct—and potentially incompatible—architectures, training recipes or inductive biases.

In this work, we focus on a particular inductive bias: permutation invariance. Modern LLMs typically encode positional information through positional encodings [31]. This is, of course, a desirable property for a language model, as the order of words is inextricably linked to their meaning. However, this sensitivity to ordering can present challenges in other contexts. For example, consider in-context learning: given k labeled examples, a model is asked to predict the label for a final unlabeled input. It is typically desirable that these predictions are *invariant* to ordering of the examples. In multiple choice question answering (QA), we'd similarly like invariance to the order of answer options; in

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mechanistic Interpretability.

long-context reasoning, we might want invariance to the order of in-context facts; in deductive reasoning tasks a model should be invariant to the ordering of logical predicates; in generative verification tasks, the verifier should be invariant to the ordering of candidate solutions.

Unfortunately, at least as an empirical matter, modern LLMs fail to satisfy any of these desiderata [25, 5, 21, 27, 22, 19]. This sensitivity to ordering manifests as seemingly arbitrary failures, undermining reliability and, ultimately, trust in these models. This problem is well-studied, with proposed solutions including altering the training loss to encourage permutation invariance [6], optimizing input ordering [12, 3, 2], and modifications to the architecture [8, 33, 10] (we discuss additional related work in Appendix A). However, none of these solutions address the fundamental question: can a standard decoder-transformer *learn* to guarantee permutation invariance?

Inverse permutation learning. We observe that *inverse permutation learning* is a sufficient and natural manner in which to guarantee order invariance. For example, given a sequence of (key, value) pairs as in-context examples, it might first map these examples to a lexicographic ordering over keys, guaranteeing invariance to the order in which they are presented. We model this task as *inverse permutation learning*: given a permuted sequence and a description of the permutation which was applied, the model should learn to output a "canonical" version of the sequence.

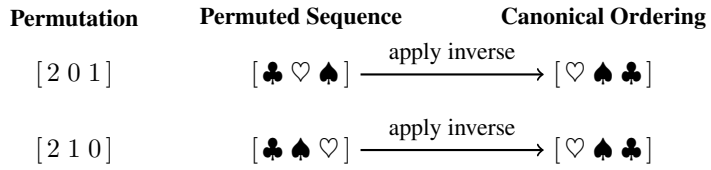


Figure 1: Visualizing inverse permutation learning. The model is given a permutation and a permuted sequence as input, and is tasked with inverting the permutation to recover the canonical ordering.

Note that this task is perhaps easier than our motivating examples, as the model is given both a description of the permutation to be "undone" as well as the permuted sequence. Nonetheless, we show that, under relatively weak assumptions, a transformer cannot learn to complete this task.

1.1 Contributions

In this work, we derive an impossibility result for inverse permutation learning with a decoder-only, attention-only transformer architecture. The key intuition underpinning this result is straightforward: all permutations (except the trivial identity permutation) require shifting at least one element from a later position in the sequence to an earlier position in the sequence. However, the structure of the causal attention mask prevents the necessary information transfer from later tokens to earlier ones.

We complement this impossibility result with a pair of existence proofs. The first shows, perhaps unsurprisingly, removing the causal structure to allow for general two-way attention mechanisms yields a construction which solves the inverse permutation learning problem. We show that this is borne out empirically in a simplified transformer model, which improves from the level of random guessing to nearly perfect accuracy if the causal attention mask is removed.[1] We also conduct a mechanistic analysis demonstrating that the learned weights correspond to a constructive proof. This result suggests that alternative architectures, particularly encoder-decoder architectures, may not have the same limitations as the decoder-only architecture which underpins the most popular modern LLMs.

Second, and more surprisingly, we show that *copying the input* — that is, providing (permutation, permuted sequence, permutation, permuted sequence) as input — also yields a construction for inverse permutation learning. Our construction illustrates that this transformation of the input provides the model with additional "scratch space" for computation during the forward pass. We conjecture that this mechanism may partially explain the success of "chain-of-thought" prompting [32], scratchpad prompting [24] or the use of intermediate reasoning tokens [7], which perform a similar function by allowing the model to perform additional computation in a single forward pass *whether or not the intermediate tokens encode meaningful information* (e.g., intermediate reasoning steps).

[1] Our code is available here: https://github.com/johnchrishays/icl.

2 Technical Results

We begin with the definition of the inverse permutation function.

Definition 1 (Inverse permutation function)**.** *For fixed n, an inverse permutation function takes any permutation matrix $P \in \{0,1\}^{n \times n}$ where the ith row represents the elementary unit vector $e_{\pi(i)}$ for some permutation $\pi(\cdot)$ and any n-row matrix Y_P and to produce Y, where $Y = P^{-1}Y_P$.*

We remark that one function satisfying Definition 1 is the trivial linear transformation $(P, Y_P) \mapsto P^\top Y_P$, since $P^{-1} = P^\top$. As mentioned in Section 1, this task is arguably simpler than the more realistic tasks which motivate it: the description of the permutation P is given explicitly, rather than being determined by some function of the permuted input Y_P (e.g., its lexicographic index). Thus, we have eliminated the difficulty of determining the order in which inputs are arranged or the choice of canonical ordering during internal model computations. Similarly, by presenting Y_P as a matrix, we have also reduced the difficulty of the problem by providing a fixed delimiting of the set of examples (rather than the model having to delimit free text into the set of choices).

The decoder-only transformer architecture underpins the most popular frontier LLMs, including the GPT series.[2] We focus on a simplified decoder-only, "disentangled" transformer architecture proposed by Nichani et al. [23]. We describe this architecture below.

A simplified transformer model. The input will consist of a matrix $X \in \mathbb{R}^{T \times d}$ where T is the length of the sequence and d is the input dimension. The input sequence is first passed through an embedding function, which consists of the concatenation of token and position embeddings. For simplicity, we will assume the token embeddings are just X itself and position embeddings are one-hot encodings of each of X's row positions, so that the encoding of an input is

$$h^{(0)} \stackrel{\text{def.}}{=} [X, I] \in \mathbb{R}^{T \times (d+T)}$$

where I is the $T \times T$ identity matrix. Subsequent layers of the transformer will consist of attention layers, which are each parameterized by a matrix A. Each attention layer consists of a softmax operation $\mathcal{S}(\cdot)$ composed with a causal attention mask operation $\mathrm{MASK}(\cdot)$:

$$\mathrm{attn}(h; A) \stackrel{\text{def.}}{=} \mathcal{S}(\mathrm{MASK}(hAh^\top))h$$

The softmax operation is applied row-wise to a matrix. For a given vector v, we let $\mathcal{S}(v)_i = \exp(v_i)/\sum_j \exp(v_j)$. For a matrix V, the causal attention mask just takes the lower-triangular entries of V so that $\mathrm{MASK}(V)_{ij} = V_{ij}$ if $i \geq j$ and $-\infty$ above the diagonal. We will denote the weight matrix at the ith layer as $A^{(i)}$.

Typically, the outputs of a layer are added to its inputs; i.e., the residual stream out of layer $i+1$ is $h^{(i)} + \mathrm{attn}(h^{(i)}; A^{(i+1)})$. We will instead analyze a *disentangled transformer*, proposed by Nichani et al. [23], which concatenates the outputs of a layer with its inputs to form the inputs to the next layer:

$$h^{(i+1)} \stackrel{\text{def.}}{=} [h^{(i)}, \mathrm{attn}(h^{(i)}; A^{(i+1)})].$$

The disentangled transformer is exactly as expressive as a vanilla (layer-sum) transformer (see, Nichani et al. [23], Theorem 3). It is analytically useful because it clarifies the structure of the *residual stream* [9], which describes how transformers may use orthogonal subspaces of layer outputs as communication channels. In particular, while standard transformers store information in orthogonal subspaces of the residual stream, disentangled transformers store the outputs of each layer in separate matrix blocks. We will refer to $h^{(i)}$ as the residual stream at layer i. Throughout this paper, we will analyze disentangled transformers. For simplicity, we will consider transformers with a single head. All of our results hold for multi-head transformers.

For expressibility of inverse permutation functions (Definition 1), the input to the transformer is the concatenation of the task inputs, represented as $X = [P; Y_P]$. We note that Y_P is represented with d rows since this is the dimension of the permutation matrix. Thus, $T = 2d$.

[2]The most popular open source models (e.g., the Llama series, Qwen series and Mixtral series) are built on decoder-only architectures, as was OpenAI's GPT-3 model. While the architecture for newer closed-source models like GPT-5 or the Gemini, Claude or Grok series has not been publicly disclosed, it is folklore in the AI research community that these models are also underpinned by variants of a decoder-only architecture.

2.1 The impossibility of inverse permutation learning.

Our main theoretical result states that, for any nontrivial permutation P, no decoder-only transformer of any depth can output Y to a block of the residual stream. We state this result below.

Theorem 1. *For all k and parameter matrices $\{A^{(i)}\}_{i\in[k]}$, and all permutation matrices P other than the identity permutation, there exists a target matrix Y such that a decoder-only, attention-only transformer parameterized by $\{A^{(i)}\}_{i\in[k]}$ given $[P; Y_P]$ as input does not output Y to any block of the residual stream.*

As described in Section 1, the intuition for this result is straightforward: for any nontrivial permutation, at least one element which must be moved from a later position to an earlier position. However, the causal attention mask precludes this: any row in the residual stream corresponding to position i must be invariant to changes in rows corresponding to positions $j > i$. This interpretation is supported by Theorem 3, which we state in in Appendix B. The proof of Theorem 3 gives a construction for inverse permutation learning if the causal attention mask is removed.

We provide complementary empirical results in Appendix C, which demonstrate that performance improves from the level of random guessing to near perfect accuracy if we remove the causal attention mask. We also conduct a mechanistic analysis of the trained model which demonstrates that the learned weights can constitute a constructive proof of Theorem 3. Proofs of both Theorem 1 and Theorem 3 are provided in Appendix D.

2.2 The possibility of inverse permutation learning under input copying.

We now turn to a more surprising result — simply copying the input is also sufficient to demonstrate a construction that solves the inverse permutation learning task. Let $s \in \mathbb{R}^d$ be the embedding associated with a special scratch token. Let $S = \mathbf{1}s^\top \in \mathbb{R}^{d\times d}$ be the matrix with d rows where each row is s. The content of the representation does not matter for our purposes, so it may overlap with a regular token embedding or be an embedding not corresponding to any regular token. We state this theorem below.

Theorem 2. *There exist parameter matrices $A^{(1)}$ and $A^{(2)}$ such that, for any permutation matrix P and target matrix Y, a two-layer decoder-only attention-only transformer parameterized by $A^{(1)}, A^{(2)}$ given $[P; Y_P; S]$ as input outputs Y to a block of the residual stream.*

We provide a proof in Appendix D. The proof suggests that the specific content of the "padding" tokens (here, the second copy of P and Y_P) are unimportant. Instead, as discussed in Section 1, these padding tokens provide the model with additional "scratch space" with which to perform the necessary matrix operations. Our proof also requires that the first token of the input be a `BOS` token with both the token and positional embedding set to all zeros. (Equivalently, we just need that the context length be at least one token longer than the input string.) Unlike Theorem 3, this result provides a recipe for performing inverse permutation learning without modifying the decoder-only architecture which underpins most modern LLMs. We provide a conjecture regarding the broader implications of this finding below.

3 Discussion and Future Work

Our results identify a sharp limitation on the expressivity of decoder-only transformers: under mild assumptions, they cannot learn the inverse permutation function for any nontrivial permutation, even with unbounded depth. We give two alternative constructions under which feasibility is restored: removing the causal attention mask or, more surprisingly, simply duplicating the input.

A mechanism for multi-step reasoning. As discussed in Section 2.2, this latter construction works by providing the model with "scratch space" in the residual stream that can be used to perform additional computation. We conjecture that this may have implications well beyond permutation invariance. If there is indeed a broader class of reasoning problems which can only be solved by duplicating or otherwise padding inputs, this suggests a concrete and, to our knowledge, thus far unexplored mechanism by which popular reasoning strategies — including chain of thought prompting [32], scratchpad prompting [24] and the generation of thinking tokens [7] — might enable problem solving in transformer models. In particular, these intermediate output tokens may enable reasoning *even if* they encode no useful semantic information about the problem.

This interpretation avoids anthropomorphizing models as articulating "thoughts" or intermediate computation via output tokens; as Hubinger et al. [16] and Baker et al. [1] argue, this can also be dangerous, as models may misrepresent their own behavior or objectives. Instead, our work suggests a different path toward a mechanistic understanding of reasoning in LLMs.

Limitations. We study the stylized transformer first proposed by Nichani et al. [23]. Our model preserves the most important features of the architecture, but omits MLP blocks, multi-head attention, and nontrivial positional embeddings, among other complexities. Our results also assume a natural but particular input representation. Furthermore, our constructive results concern the expressive capacity of transformers, but do not examine training dynamics or sample complexity required to achieve these configurations. We look forward to addressing these complexities in future work.

References

[1] Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025.

[2] Rahul Atul Bhope, Praveen Venkateswaran, K. R. Jayaram, Vatche Isahagian, Vinod Muthusamy, and Nalini Venkatasubramanian. Optiseq: Ordering examples on-the-fly for in-context learning, 2025.

[3] Rahul Atul Bhope, Praveen Venkateswaran, K. R. Jayaram, Vatche Isahagian, Vinod Muthusamy, Nalini Venkatasubramanian, Taylor Shin, Yasaman Razeghi, Robert L Logan, Eric Wallace, Sameer Singh. 2020, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y Ng, Christopher Potts, Recursive, Taylor Sorensen, Joshua Robinson, Christopher Rytting, Alexander Glenn Shaw, Kyle Jeffrey Rogers, Alexia Pauline Delorey, Mahmoud Khalil, Nancy Fulda, David Wingate 2022, An, Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, and Eric Hambro. Optiseq: Ordering examples on-the-fly for in-context learning. 2025. URL `https://api.semanticscholar.org/CorpusID:275921280`.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[5] Xinyun Chen, Ryan A Chi, Xuezhi Wang, and Denny Zhou. Premise order matters in reasoning with large language models. February 2024.

[6] Edo Cohen-Karlik, Avichai Ben David, and Amir Globerson. Regularizing towards permutation invariance in recurrent models, 2020.

[7] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su,

Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[8] Beni Egressy and Jan Stühmer. Set-LLM: A Permutation-Invariant LLM. May 2025.

[9] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021.

[10] Josh Gardner, Juan C. Perdomo, and Ludwig Schmidt. Large scale transfer learning for tabular data via language modeling, 2024.

[11] George Giapitzakis, Artur Back de Luca, and Kimon Fountoulakis. Learning to add, multiply, and execute algorithmic instructions exactly with neural networks. *arXiv preprint arXiv:2502.16763*, 2025.

[12] Qi Guo, Leiyu Wang, Yidong Wang, Wei Ye, and Shikun Zhang. What makes a good order of examples in in-context learning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14892–14904, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.884. URL https://aclanthology.org/2024.findings-acl.884/.

[13] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander R Fabbri, Wojciech Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: Natural language reasoning with first-order logic. September 2022.

[14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. September 2020.

[15] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second, 2022.

[16] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper agents: Training deceptive llms that persist through safety training, 2024.

[17] Belinda Z Li, Zifan Carl Guo, and Jacob Andreas. (how) do language models track state? *arXiv preprint arXiv:2503.02854*, 2025.

[18] Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. arxiv. *arXiv preprint arXiv:2210.13382*, 2022.

[19] Xiaonan Li and Xipeng Qiu. Finding supporting examples for in-context learning. February 2023.

[20] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. July 2023.

[21] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. July 2023.

[22] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. April 2021.

[23] Eshaan Nichani, Alex Damian, and Jason D. Lee. How Transformers Learn Causal Structure with Gradient Descent, August 2024. URL `http://arxiv.org/abs/2402.14735`. arXiv:2402.14735 [cs].

[24] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

[25] Pouya Pezeshkpour and Estevam Hruschka. Large language models sensitivity to the order of options in multiple-choice questions. August 2023.

[26] Jonathan Richens, David Abel, Alexis Bellot, and Tom Everitt. General agents need world models. *arXiv preprint arXiv:2506.01622*, 2025.

[27] Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in LLM-as-a-Judge. June 2024.

[28] Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. Chess as a testbed for language model state tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11385–11393, 2022.

[29] Keyon Vafa, Justin Y. Chen, Jon Kleinberg, Sendhil Mullainathan, and Ashesh Rambachan. Evaluating the World Model Implicit in a Generative Model, June 2024. URL `http://arxiv.org/abs/2406.03689`. arXiv:2406.03689 [cs].

[30] Keyon Vafa, Peter G Chang, Ashesh Rambachan, and Sendhil Mullainathan. What has a foundation model found? using inductive bias to probe for world models. *arXiv preprint arXiv:2507.06952*, 2025.

[31] 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.

[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022.

[33] Renjun Xu, Kaifan Yang, Ke Liu, and Fengxiang He. $E(2)$-Equivariant vision transformer. June 2023.

A Additional related work

Our work builds on a rich and rapidly growing literature on reasoning, world modeling and state tracking modern large language models. It is perhaps most closely related to Li et al. [17], which examines whether and how LLMs track state in a permutation composition task. Unlike our work, they explicitly train models to output intermediate states. Another related line of work interrogates the coherence of internal "world models" [29, 30, 18, 28]; these internal models are desirable precisely because they guarantee, for example, that the behavior of the LLM is invariant to perturbations of inputs which correspond to the same logical state. Recent work argues that such a world model is *necessary* for general purpose agents [26]. Our work is also closely related to Giapitzakis et al. [11], which studies the task of learning permutations (given an input, apply a given, fixed permutation to it) in two-layer feedforward neural networks. In contrast, the task we study involves the "in context" inversion of arbitrary permutations given as inputs. Our technical results build on the "disentangled" transformer model, first proposed by Nichani et al. [23] to study the emergence of causal reasoning in transformer models.

More generally, we build on work studying transformers [31] and their capabilities. In particular, the problem of reconstructing "canonical" permutations can be viewed as a building block towards

robust in-context learning [4] and tabular foundation models models [15, 10], wherein performance should be invariant to permutations of in-context examples or columns.

B Inverse permutation learning with an unrestricted attention mask.

Our next result serves to reinforce the key intuition from Section 2.1; namely, that the causal attention mask precludes the transfer of information which is necessary to perform inverse permutation learning. First, we define the causal mask-free attention operation.

Definition 2. *We will say an attention layer is causal mask-free (CMF) if it computes*

$$\mathrm{attn}_{\mathrm{CMF}}(h; A) \stackrel{\mathrm{def.}}{=} \mathcal{S}(hAh^\top)h.$$

We say a decoder-only, attention-only transformer is CFM if all of its attention layers are CFM.

That is, $\mathrm{attn}_{\mathrm{CMF}}$ is defined identically to attn except that the MASK operation is removed. In that sense, the causal mask-free attention operation is no longer "causal", and instead allows for unrestricted information flow between tokens.

Our next result shows that this change is sufficient to recover a construction which solves the inverse permutation learning task.

Theorem 3. *There exists parameter matrices $A^{(1)}$ and $A^{(2)}$ such that, for any permutation matrix P and target matrix Y, a two-layer, decoder-only, attention-only, causal mask-free, transformer parameterized by $A^{(1)}, A^{(2)}$ given $[P; Y_P]$ as input outputs Y to a block of the residual stream.*

We provide a constructive proof of this result in Appendix D. As discussed in Section 1, this result is perhaps unsurprising: the intuition underpinning Theorem 1 was that causal attention restricts the necessary flow of information; this result reinforces that interpretation by showing that removing the causal attention mask suffices to circumvent this impossibility result.

We validate this result by training a disentangled transformer using stochastic gradient descent. We find the performance of the model after training is near perfect, and inspection of the weights shows that the model has indeed learned to invert permutations (although the construction it learns is different than the one we use in our proof). Full details of our setup, data generating process and approach are provided in Appendix C below.

C Empirical validation

Here we provide details on our empirical validation of our theoretical results. All of our code was forked from the code provided in Nichani et al. [23] (the model training, plotting and logging are all nearly identical to their code; the data generating process and model itself are modified to fit our question and setting). Our code is available at `https://github.com/johnchrishays/icl`.

C.1 Setup.

We train a disentangled transformer using the architecture described in Section 2. The only difference is that, in the paper, our results are about the (possibility or impossibility) of copying Y to the residual stream and in our empirical validation we produce an *output*. This is to enable taking gradients and applying the usual model training pipeline. To generate outputs, we define output weights W and multiply the residual stream output in the last layer with W: $h^{(2)}W^\top$.

We train disentangled transformers with a causal mask and without a causal mask, as defined in Section 2.1 and Appendix B, respectively. For each of the experiments, the training parameters, loss function and all other details of the implementation and training are exactly the same (including the random seed).

We use squared loss to optimize the model, 2^{16} training steps and batches of size 1024. Model training takes less than 5 minutes on one Nvidia A100 GPU. We set the dimension $d = 10$.

To generate the inputs, we generated P by sampling uniformly at random from the set of permutations on d elements. To generate Y, we sampled each entry of a $d \times d$ matrix uniformly at random from $\{0, 1\}$.

C.2 Results.

Transformers with causal masks. After training the model with causal mask, mean squared error of the model is approximately 2.5, which is the MSE corresponding to random guessing.

Transformers without causal masks. After training the model without the causal mask, the MSE of the model is about 0.00015. We reproduce the weights corresponding to the trained model in Figure 2. Panel (a) shows the weights for the first attention layer $A^{(1)}$, panel (b) is for the second attention layer $A^{(2)}$ and (c) is for the output layer W. The construction recovered by the model training process is different than the construction provided in our proof of Theorem 3; both are valid. We visualize weights with a heatmap: each small square in each figure represents a single weight parameter. Thus, $A^{(1)}$ is a matrix of size $3d \times 3d = 30 \times 30$. In (a), we label weights corresponding to token and position embeddings above and to the side of the matrix. Weights that are larger are closer to yellow and weights that are close to zero are dark blue. Thus, the weights matrix $A^{(1)}$ is approximately equal to

$$A^{(1)} = \beta \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}$$

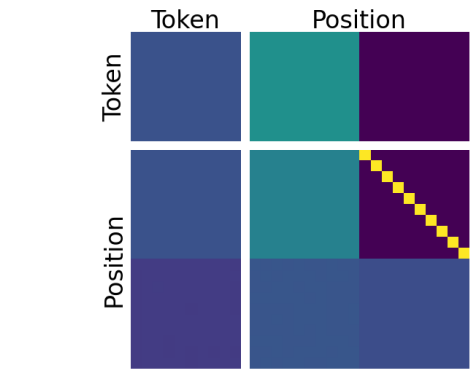
for some $\beta \gg 0$, where each entry corresponds to a $d \times d$ block.

In (b), we label weights corresponding to the different blocks of the residual stream: the top/left blocks correspond to the input sequence and the bottom/right blocks correspond to the layer-1 outputs. Thus, the weights matrix $A^{(2)}$ is approximately equal to

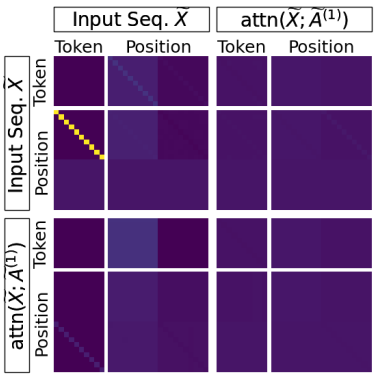
$$A^{(2)} = \beta \left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

for some $\beta \gg 0$.

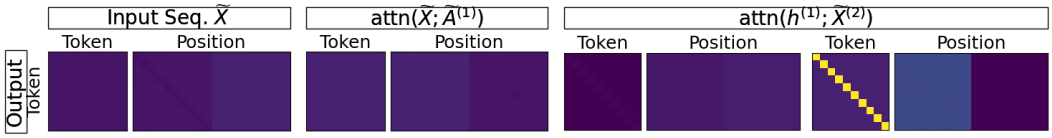
In (c), the output weights W can be seen to be approximately equal to the all-zeros matrix except for the 10th block, which is the identity matrix.



(a) Weights for first attention layer $A^{(1)}$



(b) Weights for second attention layer $A^{(2)}$



(c) Weights for output layer W

Figure 2: Weights for decoder-only disentangled transformer trained to inverse permutations.

D Proofs

We begin by stating a lemma we will use in the proof of our main theorem. It formalizes the notion that the causal mask induces an invariance of each row to any subsequent row in the residual stream.

Lemma 1. *Let h and h' be any two inputs to the ℓth layer such that their first i rows are the same: $h_{1:i} = h'_{1:i}$. Then for any weights A, $\mathrm{attn}(h; A)_i = \mathrm{attn}(h'; A)_i$.*

Proof. By inspecting the activation function, we observe the ith row of the output of a given layer with weights A are:

$$(\mathcal{S}(\mathrm{MASK}(hAh^\top))h)_i = \frac{1}{\sum_{j\le i}\exp(h_i A h_j^\top)}\sum_{j\le i}\exp(h_i A h_j^\top)h_j$$

Since this function does not depend on any row h_j for $j > i$, $\mathrm{attn}(h; A)_\ell = \mathrm{attn}(h'; A)_\ell$. □

D.1 Proof of Theorem 1

For ease of reference, we restate each result before its proof.

Theorem 1. *For all k and parameter matrices $\{A^{(i)}\}_{i\in[k]}$, and all permutation matrices P other than the identity permutation, there exists a target matrix Y such that a decoder-only, attention-only transformer parameterized by $\{A^{(i)}\}_{i\in[k]}$ given $[P; Y_P]$ as input does not output Y to any block of the residual stream.*

As discussed above, the intuition for this result is straightforward: for any permutation other than the identity permutation, there exists at least one element which must be moved from a later position to an earlier position in the sequence. We'll argue, by Lemma 1, that the causal attention mask precludes this.

Proof. Consider some P that is not the identity matrix. Recall that rows and columns of P must sum to one. Let P_i denote the ith-row of P. We first prove that P must have at least one nonzero below-diagonal entry. That is, we claim that there exists i such that $P_i = e_j$ for $j < i$. To see this, let i be the index of the last row which is not equal to its corresponding elementary basis vector e_i. (By the fact that P is not the identity, there must some such row index.) Since for all $j > i$, the jth row is equal to e_j, the last $d - i$ entries of P_i must be zero. Also, since $P_i \neq e_i$, the diagonal entry must be zero. Therefore, the non-zero entry of P_i must be at some entry j for $j < i$. This is a non-zero below-diagonal entry of P_i. Put another way: any non-identity permutation has at least one "cycle" $(i_1 \to i_2 \to \cdots \to i_1)$, which implies there exists a nonzero below-diagonal entry.

Now, let i, j be any indices such that $P_i = e_j$ for $j < i$. In order to store Y in some block of the residual stream, the jth row of Y cannot be output into any row of the residual stream more than $d-j$ rows from the bottom. Intuitively, if Y is to "fit" into the residual stream, row j cannot be output at too low of a row in the residual stream: there must be space for at least $d - j$ more rows. Formally, the jth row of Y must be output somewhere in the ℓ-th row of the residual stream for $\ell \le d + j + 1$. However, since $Y_P = PY$ and $j < i$, Y_j is located in the ith row of Y_P, which is in row $d + i + 1$ of the input. Since $d + i + 1 > d + j + 1 \ge \ell$, by Lemma 1, row ℓ of the residual stream must be invariant to row $d + i + 1$. Thus, for any fixed weight matrices $\{A^{(m)}\}_{m\in[k]}$, there exists some Y such that it is not output to the residual stream. □

Theorem 3. *There exists parameter matrices $A^{(1)}$ and $A^{(2)}$ such that, for any permutation matrix P and target matrix Y, a two-layer, decoder-only, attention-only, causal mask-free, transformer parameterized by $A^{(1)}, A^{(2)}$ given $[P; Y_P]$ as input outputs Y to a block of the residual stream.*

Proof of Theorem 3. We proceed by construction.

The first layer copies Y_P into the top block-row of the residual stream. For constant β_1, we will have attention weights represented by the following block matrix

$$A^{(1)} = \beta_1 \begin{bmatrix} 0_d & 0_d & 0_d \\ 0_d & 0_d & 0_d \\ 0_d & I_d & 0_d \end{bmatrix}$$

where each 0_d is the $d \times d$ all-zeros matrix and $I_d \in \mathbb{R}^{d \times d}$ is the $d \times d$ identity matrix. This implies pre-activations

$$XA^{(1)}X^\top = \beta_1 \begin{bmatrix} 0_d & 0_d \\ I_d & 0_d \end{bmatrix} \in \mathbb{R}^{T \times T}$$

This implies activations

$$\mathcal{S}(XA^{(1)}X^\top) = \begin{bmatrix} 1/(2d)\mathbf{1}\mathbf{1}^\top & 1/(2d)\mathbf{1}_d\mathbf{1}_d^\top \\ I_d & 0_d \end{bmatrix}$$

where $1/(2d)\mathbf{1}\mathbf{1}^\top$ is the $d \times d$ constant matrix with entries $1/(2d)$. Finally, we have layer-1 outputs

$$\mathcal{S}(XA^{(1)}X^\top)X = \begin{bmatrix} 1/(2d)\mathbf{1}\mathbf{1}^\top(P + Y_P) & \dots & \dots \\ P & \dots & \dots \end{bmatrix}$$

where we write $\dots$ for block matrices that will not factor into later computations and therefore do not matter.

The second attention layer computes Y. Recall that the input to the second attention is

$$\begin{aligned} h^{(1)} &= \begin{bmatrix} X & \mathcal{S}(XA^{(1)}X^\top)X \end{bmatrix} \\ &= \begin{bmatrix} P & I_d & 0 & 1/(2d)\mathbf{1}\mathbf{1}^\top(P + Y_P) & \dots & \dots \\ Y_P & 0 & I_d & P & \dots & \dots \end{bmatrix} \end{aligned}$$

Define a constant β_2. We will have attention weights

$$A^{(2)} = \beta_2 \left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

This implies pre-activations

$$\begin{aligned} h^{(1)}A^{(2)}h^{(1)\top} &= \beta_2 \begin{bmatrix} I_d \\ 0_d \end{bmatrix} \begin{bmatrix} 1/(2d)\mathbf{1}\mathbf{1}^\top(P + Y_P)^\top & P^\top \end{bmatrix} \\ &= \beta_2 \begin{bmatrix} 1/(2d)\mathbf{1}\mathbf{1}^\top(P + Y_P)^\top & P^\top \\ 0 & 0 \end{bmatrix} \end{aligned}$$

Note that $1/(2d)\mathbf{1}\mathbf{1}^\top(P + Y_P)^\top < 1$ so, as $\beta_2 \to \infty$, we have activations

$$\mathcal{S}(h^{(1)}A^{(2)}h^{(1)\top}) = \begin{bmatrix} 0 & P^\top \\ 1/(2d)\mathbf{1}\mathbf{1}^\top & 1/(2d)\mathbf{1}\mathbf{1}^\top \end{bmatrix}$$

This yields layer-2 outputs

$$\mathcal{S}(h^{(1)}A^{(2)}h^{(1)\top})h^{(1)} = \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ P^\top Y_P & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Finally, notice $P^\top Y_P = Y$ so that block recovers the desired canonical order. □

Theorem 2. *There exist parameter matrices $A^{(1)}$ and $A^{(2)}$ such that, for any permutation matrix P and target matrix Y, a two-layer decoder-only attention-only transformer parameterized by $A^{(1)}, A^{(2)}$ given $[P; Y_P; S]$ as input outputs Y to a block of the residual stream.*

Proof of Theorem 2. Observe, since we pad the beginning of the input with a `BOS` token so to be all zeros, that

$$\begin{aligned} X &= \texttt{embed}(s_{1:T}) \\ &= \begin{bmatrix} 0 & 0_{1 \times T} \\ P & \\ Y_P & I_T \\ S & \end{bmatrix} \\ &= \begin{bmatrix} 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} \\ P & I_d & 0_d & 0_d \\ Y_P & 0_d & I_d & 0_d \\ S & 0_d & 0_d & I_d \end{bmatrix} \in \mathbb{R}^{(T+1)\times(d+T)} \end{aligned}$$

The first layer copies P into the second block-row of the residual stream. For constant β_1, we will have attention weights represented by the following block matrix

$$A^{(1)} = \beta_1 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & I_d & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where each 0 is the $d \times d$ all-zeros matrix. This implies pre-activations

$$XA^{(1)}X^\top = \beta_1 \begin{bmatrix} 0 & 0 & 0 \\ I_d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{T \times T}$$

Let $\Delta^{(1)}$ be the lower triangular matrix with $\Delta_{ij} = 1/i$ for $j \leq i$. Let $\Delta^{(1)}$ be the $d+1 \times d+1$ matrix with $\Delta_{ij} = 1/(2d+i+1)$. And as $\beta_1 \to \infty$,

$$\mathcal{S}(\text{MASK}(XA^{(1)}X^\top)) = \begin{bmatrix} \Delta^{(1)} & & 0 & 0 \\ 0_{d\times 1} & I_d & 0 & 0 \\ 0_{d\times 1} & \Delta^{(2)} & \cdots & \cdots \end{bmatrix}$$

We will write dots when the values in a block do not matter for subsequent computations. This yields layer 1 outputs:

$$\mathcal{S}(\text{MASK}(XA^{(1)}X^\top))X = \begin{bmatrix} \Delta^{(1)}[0_{1\times d}; P] & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} \\ & \cdots & \cdots & \cdots \\ P & 0_{d\times d} & 0_{d\times d} & 0_{d\times d} \\ 0_d & \cdots & \cdots & \cdots \end{bmatrix} \in \mathbb{R}^{T\times(d+T)}$$

The second layer will produce $P^\top$ in the activations so that Y is written to the residual stream. Recall:

$$h^{(1)} = \begin{bmatrix} 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & \Delta^{(1)}[0_{1\times d}; P] & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} \\ P & I_d & 0_d & 0_d & & \cdots & \cdots & \cdots \\ Y_P & 0_d & I_d & 0_d & P & 0_{d\times d} & 0_{d\times d} & 0_{d\times d} \\ 0_d & 0_d & 0_d & I_d & 0_d & \cdots & \cdots & \cdots \end{bmatrix}$$

For constant β_2, we will have attention weights represented by the following block matrix

$$A^{(2)} = \beta_2 \left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_d & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

This implies pre-activations

$$\begin{aligned} h^{(1)}A^{(2)}h^{(1)\top} &= \beta_2 \begin{bmatrix} 0 \\ 0 \\ I_d \end{bmatrix} \begin{bmatrix} \Delta^{(1)}[0_{1\times d}; P] & P & 0_d \end{bmatrix} \\ &= \beta_2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \Delta^{(1)}[0_{1\times d}; P] & P^\top & 0 \end{bmatrix}. \end{aligned}$$

Now, notice that all entries of $\Delta^{(1)}[0_{1\times d}; P]$ are less than 1, since all non-zero entries of P are 1 and all entries below the first row of $\Delta^{(1)}$ are less than 1. Thus, as $\beta_2 \to \infty$, we have activations

$$\mathcal{S}(\text{MASK}(h^{(1)}A^{(2)}h^{(1)\top})) = \begin{bmatrix} \cdots & 0 & 0 \\ \cdots & \cdots & 0 \\ 0 & P^\top & 0 \end{bmatrix}.$$

This implies layer-2 outputs:

$$\mathcal{S}(\mathrm{MASK}(h^{(1)}A^{(2)}h^{(1)\top}))h^{(1)} = \begin{bmatrix} 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} & 0_{1\times d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ Y & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

which recovers Y as desired. □