Identifying latent algorithms in Transformers via RASP

Rimon Melamed [1 2] **H. Howie Huang** [1]

Abstract

Transformers trained on certain algorithmic tasks have been found to generalize on out-of-distribution (OOD) examples. In this work, we identify several causal mechanisms ("latent algorithms") that are responsible for OOD generalization. Specifically, given an algorithmic task, we use intermediate computations from a RASP-L program that implements the task to probe the alignment of the model's learned representations with the RASP-L program. For several algorithms, the intermediate computations predicted by the RASP-L program are linearly decodable from the model's activations. Causal intervention analysis reveals that the probe subspaces are crucial for high task accuracy. Overall, we take a new perspective on understanding the hidden computations and OOD generalization of Transformer language models. [1]

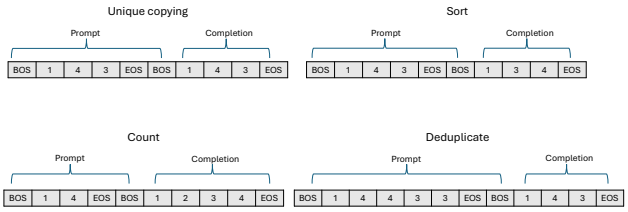

Figure 1. Overview of tasks. We consider four simple algorithmic tasks on which Transformers have been found to length-generalize.

1. Introduction

Transformers (Vaswani et al., 2017) have become the standard deep learning model for many domains, including Natural Language Processing (NLP). Large language models (LMs) have advanced significantly, and are able to perform complex tasks such as multi-step reasoning and program synthesis (OpenAI, 2024; Shao et al., 2024; Hassabis & Kavukcuoglu, 2025; Anthropic, 2026; Kwa et al., 2026). Despite these advances, these models continue to struggle with simple tasks (Song et al., 2026), demonstrating a lack of robustness. Given this "jagged" view of model capabilities (Dell'Acqua et al., 2023), it remains unclear what mechanisms these models have learned to solve specific tasks – do they implement generalizable algorithms, or merely a set of heuristics and shortcut solutions?

Algorithmic generalization of Transformers has been studied extensively through the lens of length generalization – training on shorter sequences than seen during test time (Anil et al., 2022; Jelassi et al., 2023; Kazemnejad et al., 2023; Deletang et al., 2023; Abbe et al., 2024; Zhou et al., 2024; Chang & Bisk, 2025; Izzo et al., 2025; Huang et al., 2025). Length-generalization is crucial to understand under which circumstances Transformers can "reason" because it eliminates the possibility of training data memorization. Separately, several works attempt to model Transformer computations/layers as sequential parallel operations on a sequence in a programming language called RASP (Weiss et al., 2021; Lindner et al., 2023; Yang et al., 2024; 2025). RASP provides core functions which simulate the Transformer attention and MLP layers; see Section 2.1.

Recent work introduces the RASP-L conjecture: Transformers are able to length-generalize on an algorithmic task if the task is "simple" to represent in RASP-L, a restricted variant of RASP designed specifically for decoder-only Transformers (Zhou et al., 2024). In this work, we investigate the RASP-L Conjecture by dissecting the learned representations of decoder Transformers. We show that models trained on algorithmic tasks that can be represented by short RASP-L programs exhibit internal structure that corresponds to the predicted intermediate computations from RASP-L.

1.1. Our contributions

Probing Transformers for RASP-L intermediate computations Given Transformers trained on algorithmic tasks, we construct labels for intermediate RASP-L computations, and train linear probes to classify these labels using model activations. For several tasks, the linear probes achieve high accuracy on RASP-L label classification; see Section 4.

[1]George Washington University [2]Prompt Inversion LLC. Correspondence to: Rimon Melamed <rmelamed@gwu.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

[1]We will release all code and data upon publication.

Causal verification of RASP-L representations We seek to validate that the probes have learned meaningful representations that are causally relevant to model performance on algorithmic tasks. To validate probe causality, we perform ablation experiments on the activation directions identified by the probes. For several tasks and representations, the ablations cause model accuracy to drop drastically, confirming the causal importance of the intermediate RASP-L computations identified by the probes; see Section 5.

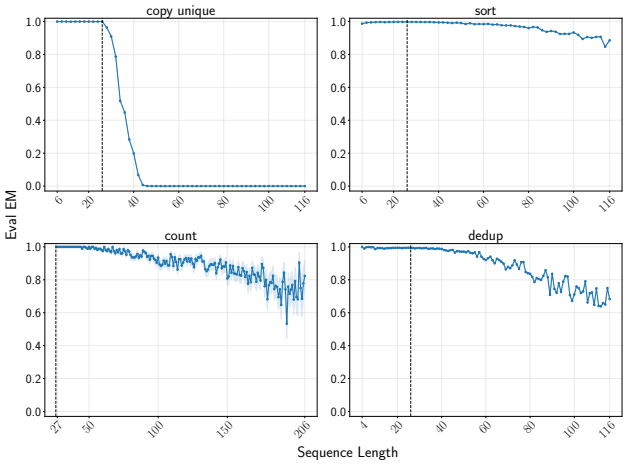

Figure 2. Out-of-distribution exact match (EM) accuracy for models across four algorithmic tasks. Each task has a corresponding short RASP-L program, and length generalizes successfully. The dashed vertical line indicates the maximum sequence length seen during train time. Error bars are per-length Bernoulli standard errors for the Eval EM.

2. Background

2.1. Restricted Access Sequence Processing (RASP)

RASP (Weiss et al., 2021) defines a computational model to represent transformers. At its core, a RASP program takes in a sequence of length n, and outputs a transformed sequence of the same length. Input to a RASP program consists of tokens and indices. The goal of RASP is to define functions which are "easy" for transformers to represent. RASP programs cannot contain branching, control flow, or loops; they can only perform sequence-to-sequence operations. The allowed operations include arbitrary sequence-to-sequence element-wise mappings $\mathbb{R}^n \to \mathbb{R}^n$, and a `kqv` operation that applies a square boolean matrix to the input sequence based on two length-n sequences (keys and queries) and a binary predicate (e.g., `equals`, which performs an element-wise equality comparison of the keys and queries).

RASP-L (Zhou et al., 2024) is a restricted version of RASP designed to represent autoregressive decoder-only transformers. All operations must be performed in a causal manner, and the last token of the output sequence represents the next token prediction. RASP-L contains several restrictions: all variables are bounded integers, and arbitrary arithmetic within token positions is prohibited. This construction allows us to represent discrete labels for intermediate representations in the RASP-L programs. In order to simulate multiple steps of generation, the algorithms runs via an "autoregressive" outer-loop, meaning that for each iteration of the loop, the last token is selected as the output, and concatenated onto the input sequence for the next iteration (Zhou et al., 2024).

2.2. Linear representation hypothesis

There is increasing empirical and theoretical evidence that trained neural networks contain high-level features that can be represented as linear functions of the network's activations (Mikolov et al., 2013; Bolukbasi et al., 2016; Conneau et al., 2018; Vargas & Cotterell, 2020; Li et al., 2021; Elhage et al., 2022; Tigges et al., 2023; Park et al., 2023; Li et al., 2023; Nanda et al., 2023; Marks & Tegmark, 2024a; Park et al., 2024; Boix-Adsera, 2024; Hernandez et al., 2024; Allen-Zhu & Li, 2025). This is known as the Linear Representation Hypothesis (LRH). These features range from discerning true and false statements (Marks & Tegmark, 2024a) to encoding states about the board in Othello (Nanda et al., 2023). Due to the pervasive evidence of linearity in LM features, we hypothesize that Transformer models trained on algorithmic tasks exhibit linear representations of intermediate RASP-L computations.

2.3. Language model interpretability

Many resources have been devoted to demystifying the opaque nature of neural language models. Mechanistic interpretability is one approach which aims to describe individual features and circuits within models that have causal implications for model output. Elhage et al. (2022) view a model's output at each layer as a "residual stream" which the model can read/write to at each individual layer. Additionally, several works have identified specific attention heads and circuits which correspond to tracking previously seen repeated tokens (Olsson et al., 2022; Wang et al., 2023).

Probing classifiers have become a popular technique to understand model structure by training classifiers on model activations to test some external property. (Alain & Bengio, 2017; Belinkov, 2022; Elazar et al., 2021). These are typically binary classifiers, where the classifier accuracy corresponds with the presence of a specific feature (Azaria & Mitchell, 2023; Marks & Tegmark, 2024b).

Several works examine the connection between Transformers and RASP. Tracr (Lindner et al., 2023) introduces a library to compile RASP programs directly into Transformer weights by translating the RASP computation graph. On the other hand, Friedman et al. (2023) explore the opposite scenario – they design a simplified Transformer which can be

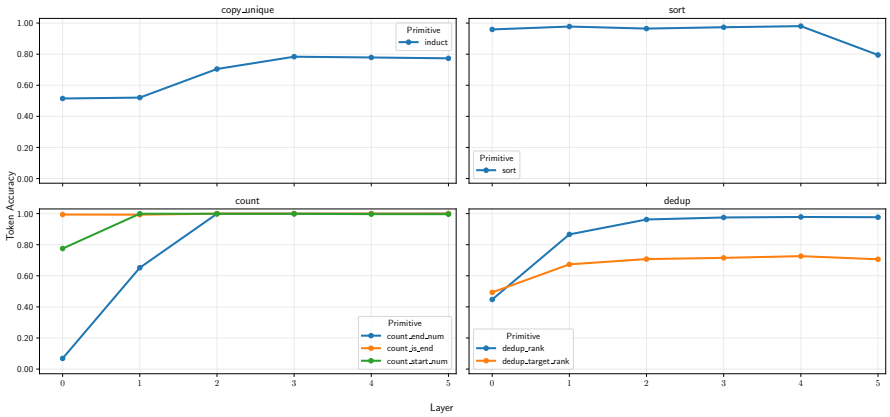

Figure 3. RASP-L label probe accuracy. For each task, we select several RASP-L intermediate computations, generate discrete labels, and train linear probes at each layer output. Reported accuracy is at the token level for the completion tokens. We find that the probes achieve high accuracy for several RASP-L intermediate computations; see Section 4.

automatically converted into a RASP program. Additionally, Vergara Browne & Soto (2025) attempt to improve trained model performance by "distilling" RASP representations directly into the model.

3. Experimental methodology

The models that we use for training are decoder Transformers with learned positional embeddings. For each task, we train with a maximum sequence length of 30 tokens, and evaluate the OOD exact match (EM) accuracy of the entire sequence on longer lengths (up to 256 for `count`). In order to train the model at all positions, we add a random offset to the position IDs for each training sample following prior work (Zhou et al., 2024; Chang & Bisk, 2025). We use the AdamW optimizer (Loshchilov & Hutter, 2019), cosine annealing learning rate scheduler with linear warmup, and a randomly sampled unique dataset for each task; see Appendix A for details on each task.

3.1. Tasks and selected intermediate computations

We construct the short RASP-L programs corresponding to each task. We briefly discuss the core components of each program for each task below, for a more thorough treatment we refer the reader to Appendix C and the full RASP-L specification (Zhou et al., 2024).

Unique copy This task is solved by a one line program, using the `induct` primitive. `Induct` consists of two core operations: selecting the next index to copy by shifting the sequence to the right, and then selecting the value of the index to copy; see Appendix C, Listing 2 for the full program. For example, in the sequence from Figure 1, running `induct` on the input BOS | 1 | 4 | 3 | EOS | BOS would produce -1 | -1 | -1 | -1 | -1 | 1, where the final position now stores the next value to be copied. The next value is continually concatenated to the sequence via the autoregressive outer-loop, and the final sequence becomes BOS | 1 | 4 | 3 | EOS | BOS | 1 | 4 | 3 | EOS | BOS with corresponding `induct` RASP-L labels -1 | -1 | -1 | -1 | -1 | 1 | 4 | 3 | EOS.

Sort The sorting task is also implemented by one line program, using the `kqv` operation; see Appendix C, Listing 3. Specifically, this is done by first constructing the square boolean matrix, where the keys and queries are the input sequence, and the predicate is `greater_than`. Then, the boolean mask is applied to the sequence with the `max` aggregation, which essentially finds the smallest next value in the sequence (because the next smallest value will have the greatest number of True values in the boolean attention matrix).

Count Counting requires several lines of RASP-L library code, and is detailed in Appendix C, Listing 4. At a high level, the program first determines the position of the start/end number in the input sequence by using the position of the first BOS token. Then, the program checks for positions where the BOS token appears (to check if we need to emit the start number), and where the last number appears (to check if we are at the end and need to emit EOS). At every other position, the program adds one to current sequence, and emits the next $x + 1$ number.

Deduplicate The deduplication task also requires several lines of RASP-L code; see Appendix C, Listing 5. The program starts by indexing the first occurrence of each token in the sequence. Next, it assigns each first occurrence a rank (e.g., first unique token is rank 1, second is rank 2, etc.). Every token then uses the rank of its first occurrence to find the token with the next rank, which is then emitted as the next token in the sequence based on the running rank.

Table 1. Completion exact-match (EM) accuracy before and after probe-guided erasure at the layer with the strongest ablation effect for each RASP-L intermediate. The baseline is computed over all possible test lengths; see Figure 2 for specific model OOD performance. For each task, we find that ablating the direction in activation space identified by the probes results in a decrease in the model's performance, which is more significant than a random baseline. Random-erasure error bars report SEM over 10 randomly selected subspaces. Appendix B shows detailed results for all sequence lengths.

Task	RASP-L intermediate	Layer	Baseline EM	Subspace Erasure EM ↓	Random Erasure EM
Copy unique	`induct`	0	0.265	**0.004**	0.198 ± 0.004
Sort	`sort`	1	0.968	**0.000**	0.824 ± 0.017
Count	`count_start_num`	0	0.936	0.888	$\mathbf{0.814} \pm 0.012$
	`count_end_num`	2	0.936	**0.705**	0.847 ± 0.010
	`count_is_end`	3	0.936	**0.916**	0.936 ± 0.000
Dedup	`dedup_rank`	3	0.889	**0.821**	0.891 ± 0.001
	`dedup_target_rank`	0	0.889	**0.751**	0.883 ± 0.001

4. Probing for RASP-L computations

After we generate discrete labels for the intermediate RASP-L computations for each task, we train probing classifiers on the outputs of each layer. Specifically, given samples drawn from the validation dataset $\boldsymbol{x} \sim \mathcal{D}$, let $h_t^{(\ell)}(\boldsymbol{x}) \in \mathbb{R}^d$ denote the model's activation vector at layer ℓ for the t-th token, and let $y_t(\boldsymbol{x}) \in \{1, \ldots, C\}$ be the corresponding intermediate RASP-L class label. We train linear probes $p_t^{(\ell)} : \mathbb{R}^d \to \mathbb{R}^{|C|}$ via cross-entropy loss to predict the RASP-L class label at position t.

For each task, we find that the intermediate probes achieve high accuracy; see Figure 3. Specifically, for the copying and sorting task, the probes corresponding to labels for the primitives (`induct` and `sort`) achieve high accuracy. For the counting task, we probe for several primitives including `start_num` (determining position of the starting number), `end_num` (determining position of the ending number), and `count_is_end` (checking if the emitted sequence has reached the end). Finally for the deduplication task, we probe for `dedup_rank` (computing the rank of each first occurrence), and `dedup_target_rank` (looking up the next token to emit based on rank).

5. Causal intervention analysis

Given the high probe accuracy, we investigate the causal impact of each RASP-L component for each task by projecting out the feature space captured by the probe. Specifically, given each layer output $h_t^{(\ell)}(\boldsymbol{x}) \in \mathbb{R}^d$, let $\boldsymbol{V} \in \mathbb{R}^{r \times d}$ be the orthonormal basis of the feature subspace identified by the linear probe, where r is the rank of the subspace.

To test whether the identified subspace is necessary for task performance, we ablate the probed feature direction

$$\boldsymbol{P}_{\text{erase}} = \boldsymbol{I} - \boldsymbol{V}^\top \boldsymbol{V}.$$

During the forward pass, we apply a causal intervention at the probed layer ℓ by replacing the original activation with its ablated counterpart: $\tilde{h}_t^{(\ell)}(\boldsymbol{x}) = \boldsymbol{P}_{\text{erase}} h_t^{(\ell)}(\boldsymbol{x})$. If the model's OOD EM accuracy drops significantly upon this ablation, it gives evidence that the removed subspace is causally necessary for algorithmic generalization (Elazar et al., 2021; Geiger et al., 2020). We find that for all tasks, ablating many of the RASP-L intermediate representations results in a drop in final exact match accuracy; see Table 1.

Interestingly, different RASP-L representations in different tasks have widely varying effects on the EM accuracy. For unique copying and sorting, the ablations almost fully delete the model's ability to perform the task. On the other hand, for counting, the `count_end_num` component is far more causally relevant than `count_is_end` or `count_start_num`. Intuitively, this is because the `count_is_end` and `count_start_num` computations are only sparsely used to determine if we have reached the end position or are at the start position, while `count_end_num` is important at each iteration, as it keeps track of the current bound of the loop, which affects all following autoregressive generations. Similarly, for the deduplication task, `dedup_rank` builds the index of first occurrences, while `dedup_target_rank` computes the rank of the next unique token to emit.

6. Discussion

Our work investigates the generalization abilities of Transformers by studying their internal representations of algorithmic tasks, and understanding the internal alignment to RASP-L programs. Internal alignment is measured via linear probes, where the labels are the outputs of intermediate components of the RASP-L programs. We find that for several tasks, there is strong alignment between the predicted intermediate computations via RASP-L, and the actual learned representations in the model. We verify this through causal ablation experiments, which show that the task accuracy drops significantly when projecting out the RASP-L representation direction.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbe, E., Bengio, S., Lotfi, A., and Rizk, K. Generalization on the unseen, logic reasoning and degree curriculum. *Journal of Machine Learning Research*, 25(331):1–58, 2024.

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes, 2017. URL https://openreview.net/forum?id=ryF7rTqgl.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 1, learning hierarchical language structures, 2025. URL https://arxiv.org/abs/2305.13673.

Anil, C., Wu, Y., Andreassen, A., Lewkowycz, A., Misra, V., Ramasesh, V., Slone, A., Gur-Ari, G., Dyer, E., and Neyshabur, B. Exploring length generalization in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 38546–38556. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/fb7451e43f9c1c35b774bcfad7a5714b-Paper-Conference.pdf.

Anthropic. Introducing Claude Opus 4.6, February 2026. URL https://www.anthropic.com/news/claude-opus-4-6. Blog post.

Azaria, A. and Mitchell, T. The internal state of an LLM knows when it's lying. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 967–976, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.68. URL https://aclanthology.org/2023.findings-emnlp.68/.

Belinkov, Y. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7/.

Boix-Adsera, E. Towards a theory of model distillation. *arXiv preprint arXiv:2403.09053*, 2024.

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf.

Chang, Y. and Bisk, Y. Language models need inductive biases to count inductively. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s3IBHTTDYl.

Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2126–2136, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1198. URL https://aclanthology.org/P18-1198/.

Deletang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Cundy, C., Hutter, M., Legg, S., Veness, J., and Ortega, P. A. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WbxHAzkeQcn.

Dell'Acqua, F., McFowland III, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., and Lakhani, K. R. Navigating the jagged technological frontier: Field experimental evidence of the effects of ai on knowledge worker productivity and quality. *Harvard business school technology & operations mgt. Unit working paper*, (24-013), 2023.

Elazar, Y., Ravfogel, S., Jacovi, A., and Goldberg, Y. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021. doi: 10.1162/tacl_a_00359. URL https://aclanthology.org/2021.tacl-1.10/.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. Toy models of superposition. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/toy_model/index.html.

Friedman, D., Wettig, A., and Chen, D. Learning transformer programs. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in*

Neural Information Processing Systems, volume 36, pp. 49044–49067. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/995f693b73050f90977ed2828202645c-Paper-Conference.pdf.

Geiger, A., Richardson, K., and Potts, C. Neural natural language inference models partially embed theories of lexical entailment and negation. In Alishahi, A., Belinkov, Y., Chrupala, G., Hupkes, D., Pinter, Y., and Sajjad, H. (eds.), *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 163–173, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.blackboxnlp-1.16. URL https://aclanthology.org/2020.blackboxnlp-1.16/.

Hassabis, D. and Kavukcuoglu, K. A new era of intelligence with Gemini 3, November 2025. URL https://blog.google/products/gemini/gemini-3/. Google DeepMind blog post.

Hernandez, E., Sharma, A. S., Haklay, T., Meng, K., Wattenberg, M., Andreas, J., Belinkov, Y., and Bau, D. Linearity of relation decoding in transformer language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=w7LU2s14kE.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., Nakkiran, P., and Hahn, M. A formal framework for understanding length generalization in transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=U49N5V51rU.

Izzo, Z., Nichani, E., and Lee, J. D. Quantitative bounds for length generalization in transformers. *arXiv preprint arXiv:2510.27015*, 2025.

Jelassi, S., d'Ascoli, S., Domingo-Enrich, C., Wu, Y., Li, Y., and Charton, F. Length generalization in arithmetic transformers, 2023. URL https://arxiv.org/abs/2306.15400.

Kazemnejad, A., Padhi, I., Natesan Ramamurthy, K., Das, P., and Reddy, S. The impact of positional encoding on length generalization in transformers. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 24892–24928. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4e85362c02172c0c6567ce593122d31c-Paper-Conference.pdf.

Kwa, T., West, B., Becker, J., Deng, A., Garcia, K., Hasin, M., Jawhar, S., Kinniment, M., Rush, N., Arx, S. V., Bloom, R., Broadley, T., Du, H., Goodrich, B., Jurkovic, N., Miles, L. H., Nix, S., Lin, T., Parikh, N., Rein, D., Sato, L. J. K., Wijk, H., Ziegler, D. M., Barnes, E., and Chan, L. Measuring ai ability to complete long software tasks, 2026. URL https://arxiv.org/abs/2503.14499.

Li, B. Z., Nye, M., and Andreas, J. Implicit representations of meaning in neural language models. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1813–1827, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.143. URL https://aclanthology.org/2021.acl-long.143/.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Lindner, D., Kramar, J., Farquhar, S., Rahtz, M., McGrath, T., and Mikulik, V. Tracr: Compiled transformers as a laboratory for interpretability. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=tbbId8u7nP.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Marks, S. and Tegmark, M. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets, 2024a. URL https://arxiv.org/abs/2310.06824.

Marks, S. and Tegmark, M. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *First Conference on Language Modeling*, 2024b. URL https://openreview.net/forum?id=aajyHYjjsk.

Mikolov, T., Yih, W.-t., and Zweig, G. Linguistic regularities in continuous space word representations. In Vanderwende, L., Daumé III, H., and Kirchhoff, K. (eds.), *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 746–751, Atlanta,

Georgia, June 2013. Association for Computational Linguistics. URL https://aclanthology.org/N13-1090/.

Nanda, N., Lee, A., and Wattenberg, M. Emergent linear representations in world models of self-supervised sequence models. In Belinkov, Y., Hao, S., Jumelet, J., Kim, N., McCarthy, A., and Mohebbi, H. (eds.), *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.2. URL https://aclanthology.org/2023.blackboxnlp-1.2/.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI. Learning to reason with llms, September 2024. URL https://openai.com/index/learning-to-reason-with-llms/. Accessed: 2025-05-27.

Park, K., Choe, Y. J., and Veitch, V. The linear representation hypothesis and the geometry of large language models. In *Causal Representation Learning Workshop at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=T0PoOJg8cK.

Park, K., Choe, Y. J., Jiang, Y., and Veitch, V. The geometry of categorical and hierarchical concepts in large language models. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL https://openreview.net/forum?id=KXuYjuBzKo.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Song, P., Han, P., and Goodman, N. Large language model reasoning failures. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=vnX1WHMNmz. Survey Certification.

Tigges, C., Hollinsworth, O. J., Geiger, A., and Nanda, N. Linear representations of sentiment in large language models, 2023. URL https://arxiv.org/abs/2310.15154.

Vargas, F. and Cotterell, R. Exploring the linear subspace hypothesis in gender bias mitigation. In Webber, B., Cohn, T., He, Y., and Liu, Y. (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2902–2913, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.232. URL https://aclanthology.org/2020.emnlp-main.232/.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vergara Browne, T. and Soto, A. Tracr-injection: Distilling algorithms into pre-trained language models. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2831–2843, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.146. URL https://aclanthology.org/2025.findings-acl.146/.

Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Yang, A., Chiang, D., and Angluin, D. Masked hard-attention transformers recognize exactly the star-free languages. *Advances in Neural Information Processing Systems*, 37:10202–10235, 2024.

Yang, A., Cadilhac, M., and Chiang, D. Knee-deep in c-RASP: A transformer depth hierarchy. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=jPduiyxyfw.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AssIuHnmHX.

A. Additional experimental details

All models are decoder-only Transformers utilizing absolute positional embeddings (APE). We train each model for a single epoch using the AdamW optimizer. The learning rate follows a cosine annealing schedule with a 5% linear warmup. We apply a 10% holdout for testing across all datasets. During training, sequences are restricted to a maximum length of 30 tokens. To evaluate out-of-distribution (OOD) length generalization, we test on sequence lengths up to 120 tokens for the copy unique, sort, and dedup tasks, and up to 210 tokens for the count task. The core architectural constraints, dataset sizes, and remaining optimization hyperparameters are detailed in Table 2. All experiments are performed on a single NVIDIA RTX 4500 Blackwell with 32GiB of memory.

Hyperparameter	**Copy Unique**	**Sort**	**Count**	**Dedup**
Layers	6	6	6	6
Heads	8	8	8	8
Embedding dimension (d)	256	768	768	768
Maximum sequence length	128	128	256	128
Vocabulary size ($\mathcal{V}$)	100	100	220	100
Dataset size	10,000,000	10,000,000	1,000,000	5,000,000
Batch size	1024	1024	256	1024
Initial learning rate	10^{-4}	10^{-3}	10^{-3}	10^{-3}
Final learning rate	10^{-6}	10^{-6}	10^{-5}	10^{-6}
Weight decay	0.1	0.1	0.1	0.1

Table 2. Task-specific dataset sizes, model architecture, and optimization hyperparameters.

B. Additional probe erasure results

We provide the full causal ablation results for each task; see Figure 4 for the copy task, Figure 5 for the counting task, Figure 6 for the deduplicate task, and Figure 7 for the sorting task.

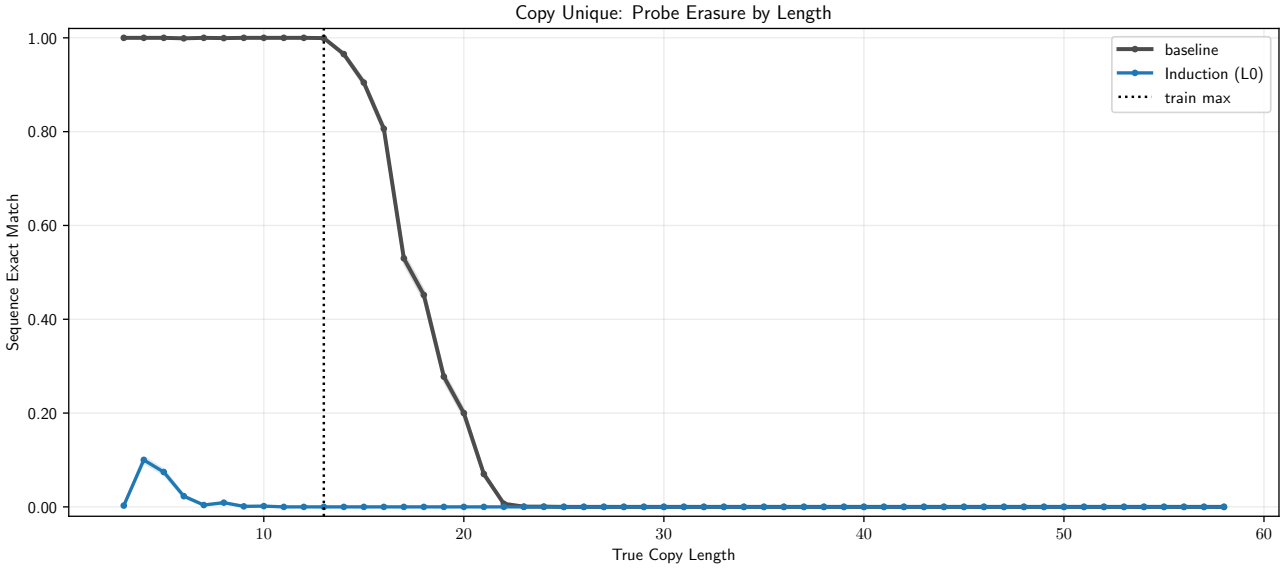

Figure 4. Unique copying test accuracy at varying lengths after causal ablation.

C. RASP-L listings

We detail the RASP-L programs for each of the studied tasks. For a full reference of the RASP-L library, we refer the reader to Zhou et al. (2024).

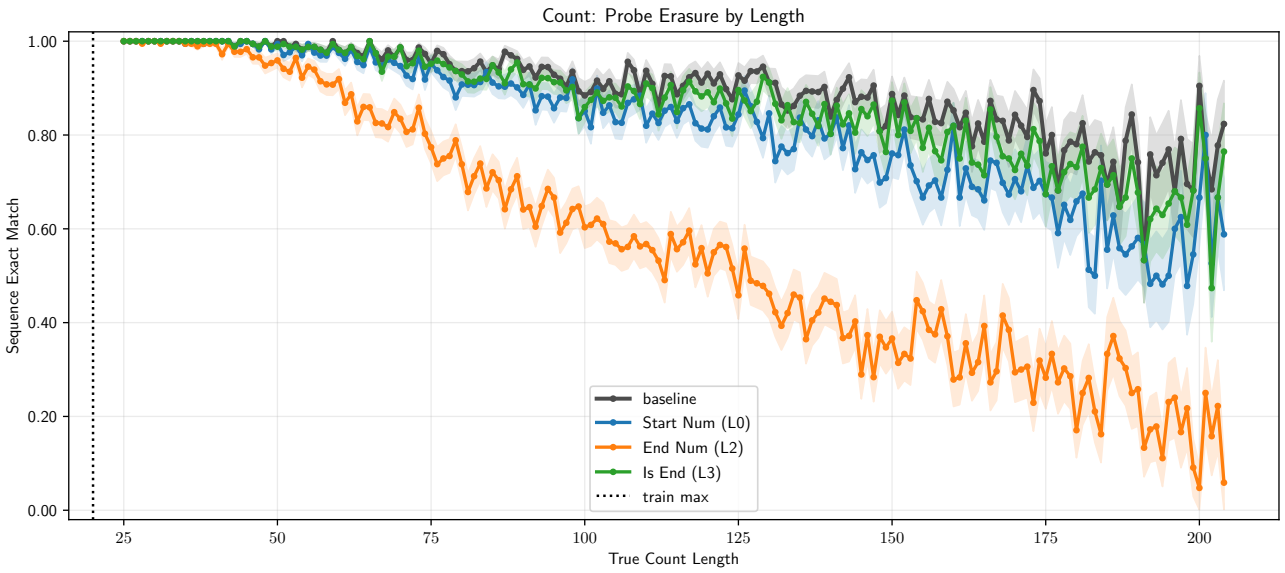

Figure 5. Counting test accuracy at varying lengths after causal ablation.

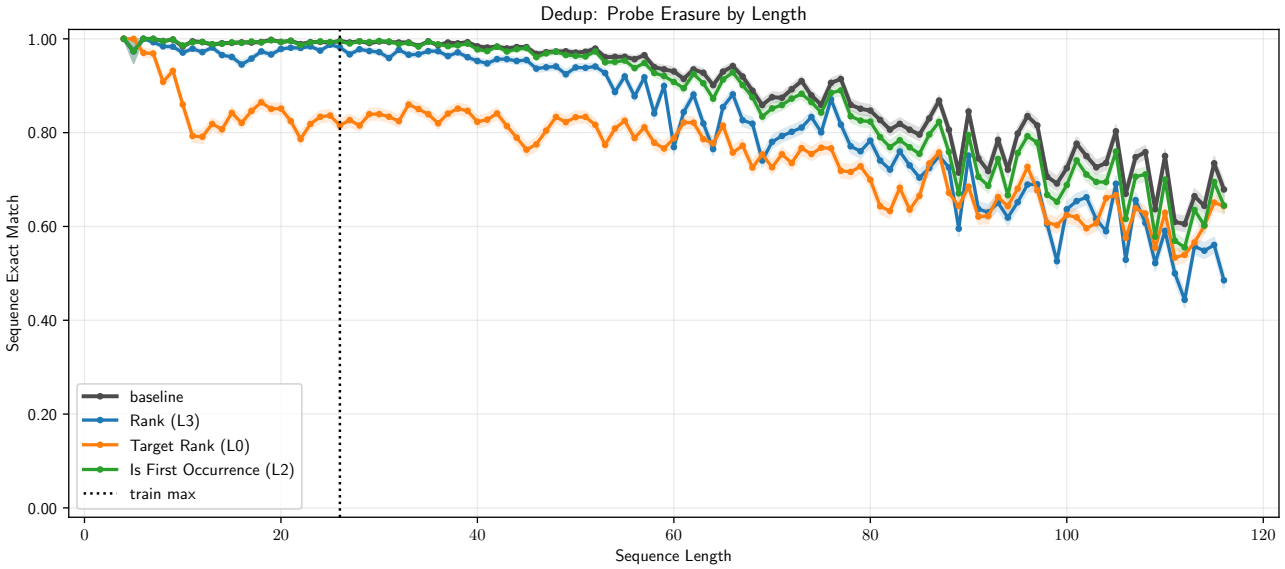

Figure 6. Deduplication test accuracy at varying lengths after causal ablation.

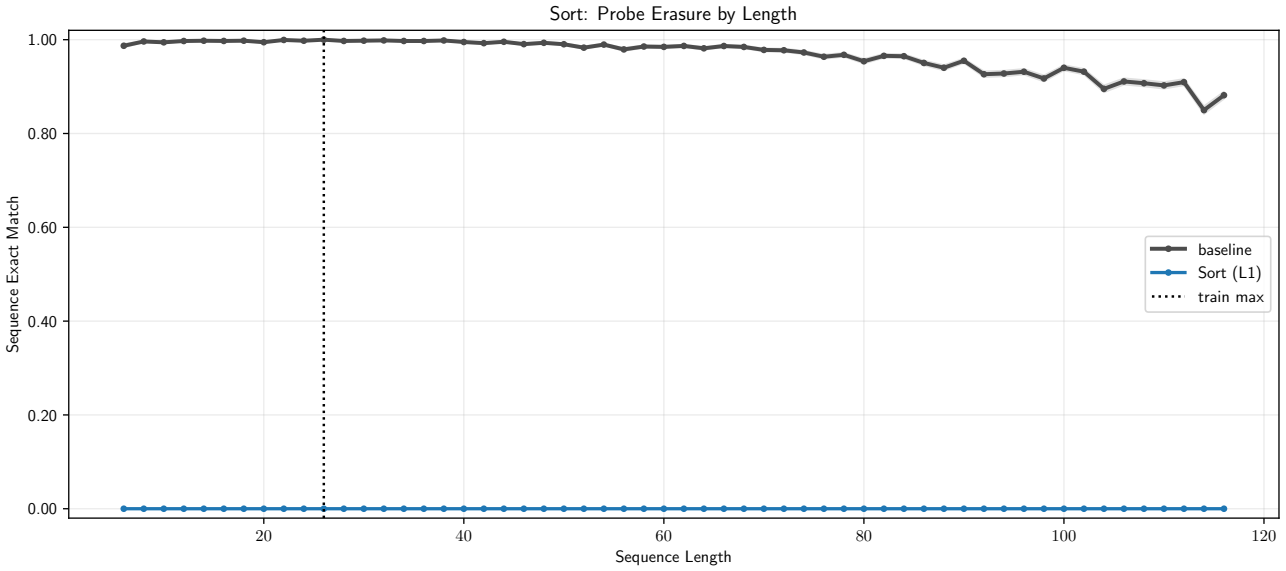

Figure 7. Sorting test accuracy at varying lengths after causal ablation.

```python
def sample_ar(x, func, max_len=-1):
  """
  simulate autoregressive decoding until EOS is reached
  """
  prompt = np.concatenate(([BOS_TOKEN_ID_EX], x, [EOS_TOKEN_ID_EX], [BOS_TOKEN_ID_EX]))
  seq = prompt.copy()
  cur_len = len(seq)

  while seq[-1] != EOS_TOKEN_ID_EX:
    next_tok = func(seq)[-1]
    seq = np.concatenate((seq, [next_tok]))

    cur_len += 1
    if max_len > 0 and cur_len >= max_len:
      break

  return seq
```

Listing 1. The autoregressive sampling method. Given a sequence $\boldsymbol{x}$, a RASP-L function `func`, and a number of sampling steps, the method applies `func` to $\boldsymbol{x}$ and concatenates the final predicted token for the given sampling steps.

```
def induct_kqv(k, q, v, offset, default=0, null_val=-999):
  # (library function) shift all values in k to the right by one by applying kqv where k
    is shifted by 1, and the predicate is equals
  shifted = shift_right(k, offset, default=null_val)

  # finds index of next first occurrence of non-special/null token (via another kqv
    operation)
  indices_to_copy = firsts(shifted, q, default=null_val)

  # select from the sequence where the indices match the indices_to_copy values (i.e., the
     last positions where we found non-special tokens)
  copied_values = index_select(v, indices_to_copy, default=default)

  # the entire sequence is returned, and then the 'sample_ar' method is used to extract
    the next token via the value at the last position
  return copied_values
```

Listing 2. RASP-L core functions used for the unique copy task.

```
def next_tok_sort(x):
  # At each autoregressive position, treat the current token as the query and select
    previous tokens whose value is greater. Take the minimum selected value as the next
    token.
  return kqv(x, x, x, gt, reduction="min", default=EOS_TOKEN_ID_EX)
```

Listing 3. RASP-L program for the sorting task.

```
def count(x):
  # Read start/end numbers via FIRST BOS
  first_bos = firsts(x, full(x, BOS_TOKEN_ID_EX))
  start_nums = index_select(x, first_bos + 1)
  end_nums = index_select(x, first_bos + 2)

  # Detect BOS positions
  is_bos = seq_map(x, full(x, BOS_TOKEN_ID_EX), equals)
  is_end = (~is_bos) & seq_map(x, end_nums, equals)

  next_tok = where(
    is_bos,
    start_nums,
    where(
      is_end,
      full(x, EOS_TOKEN_ID_EX),
      x + 1,
    ),
  )
  return next_tok
```

Listing 4. RASP-L program for the counting task.

```
def dedup(x):
  # Mark first occurrences by checking whether the first match is the current index.
  my_first_occ = firsts(x, x)
  is_first = seq_map(my_first_occ, indices(x), equals)

  # Cumulative count of first occurrences = dedup rank
  dedup_rank = cumsum(is_first)

  # Mask rank to first-occurrence positions
  first_dedup_rank = where(is_first, dedup_rank, full(x, NULL_VAL))

  # Find current token's first occurrence and look up its dedup rank
  my_rank = index_select(dedup_rank, my_first_occ)

  # Target = next dedup rank
  target_rank = my_rank + 1

  # Retrieve the token at the target rank (EOS if we've exhausted all unique tokens)
  next_tok = kqv(first_dedup_rank, target_rank, x, equals, default=EOS_TOKEN_ID_EX)

  return next_tok
```

Listing 5. RASP-L program for the deduplication task.