Different simultaneous mechanisms for in-context recall have distinct learning dynamics

Sultan Daniels SULTAN_DANIELS@BERKELEY.EDU
University of California, Berkeley

Dylan Davis DYLANJD@BERKELEY.EDU
University of California, Berkeley

Dhruv Gautam DHRUVGAUTAM@BERKELEY.EDU
University of California, Berkeley

Wentinn Liao WENLIAO@SEAS.UPENN.EDU
University of Pennsylvania

Gireeja Ranade GIREEJA@BERKELEY.EDU
University of California, Berkeley

Anant Sahai ASAHAI@BERKELEY.EDU
University of California, Berkeley

Abstract

We introduce a new family of toy problems that combine features of linear-regression style continuous in-context learning (ICL) with discrete associative recall. We pretrain transformer models on sample traces from this toy, specifically symbolically labeled interleaved observations from randomly drawn linear deterministic dynamical systems, and study if these transformer models can recall the state of a process previously seen in its context when prompted to do so with its in-context label. Training dynamics reveal the emergence of classic recall ability well into training, but surprisingly, well before this recall ability has emerged, a closely related task — predicting the second token in a recalled sequence given the first — shows clear evidence of seemingly recall-related behavior.

Through out-of-distribution experiments, and a mechanistic analysis on model weights via edge pruning, we find that next-token prediction for this toy problem involves two separate mechanisms. One mechanism uses the discrete labels to do the associative recall required to predict the start of a resumption of a previously seen sequence, and the second mechanism, which is largely agnostic to the discrete labels, performs a Bayesian-style prediction based on the previous token and the context. These two mechanisms have different learning dynamics.

To confirm that this two-mechanism (manifesting as separate emergence) phenomenon is not just an artifact of our toy setting, we used OLMo training checkpoints on an ICL translation task to see a similar phenomenon: a decisive gap in the emergence of good first-task-token performance vs second-task-token performance.

1. Introduction

The release of GPT-3 [6] demonstrated the power of Large Language Models' (LLMs) ability to do in-context learning (ICL). Since then, there has been significant progress in understanding ICL for language models themselves [1, 23, 32, 38, 39, 55–58]. There has also been work that focuses on understanding ICL for simpler toy problems [12, 14, 16, 36, 45, 46, 50]. Toys (e.g., linear

regression [16, 19, 46]) allow us to study the learned ICL behavior of deep neural networks in settings where optimal strategies are known, allowing complex prediction mechanisms to be disentangled. In this paper, we build on previous work to create a new toy problem involving interleaved vector-valued time-series. A more comprehensive presentation of this work is available in [10].

We start with underlying time-series that come from the evolution of random deterministic linear systems and thus play the role of noise-free least-squares problems in [16] — each consecutive time-series observation is defined by its underlying deterministic linear system (defined by an unknown matrix — just as in linear regression). This continuous-state problem has a naturally continuous error metric: mean-squared-error. We restrict attention to noiseless time-series defined by orthogonal matrices — consequently, once we have seen enough information in the observation sequence segments for a specific time-series, in principle, perfect prediction accuracy (i.e. 0 MSE) is possible.

Segments from different time-series are interleaved with "symbolic punctuation labels," tokens [54] that unambiguously demarcate different segments as belonging to different time-series. These discrete symbolic labels and the fact that they can occur repeatedly introduces a dimension of MQAR-style associative recall [2]. However, successful use of this recall is not simply a matter of copying a particular surface-level value from the context. Instead, the corresponding task (predicting the next observation in this particular sequence) must be done.

We find clear evidence during training of the emergence of associative recall in our toy problem. We explore two natural hypotheses for recall mechanisms:

H1: Label-based recall. The model uses in-context learning of the association of symbolic labels to time-series, and then performs inference based on recalling the referenced time-series and continuing its evolution.

H2: Observation-based Bayesian recall. The model ignores the symbolic labels. The noise-free nature of the toy problem means that once we see an observation, we can figure out which prior time-series it could have come from. Then, we can do Bayesian prediction [28, 34, 57] based on previous observations for future predictions.

However, **we find that H1 and H2 are both false as complete explanations.** Instead, both are true simultaneously! H1 is used for predicting the first token after a particular time-series is being resumed. But for the second token and beyond in a resumed sequence, the information in the observation allows a variant of H2 to work.

We observe further that there is a difference between the emergence of the ability to learn to predict the first versus second token after a symbolic label, even though information-theoretically, they both require recalling the in-context-learned nature of that specific time-series. And somewhat surprisingly, the successful use of the symbolic label (which feels conceptually easier for a human) occurs *after* the model has clearly learned to do some approximate version of the more Bayesian H2 for those tokens on which it is a viable strategy. (We verify this using out-of-distribution experiments.)

This leads to the following conjecture.

C3: Transformers use multiple mechanisms for a single multi-token task. Distinct mechanisms, with different training dynamics, are used to initiate a new episode of an ICL-specified task (i.e., predict the first token), versus continuing that task (i.e., predict the second token).

By modifying a classic LLM emergence experiment [54, 55] and using OLMo checkpoints [37], we confirm that this conjecture holds for an NLP task — i.e. even before the emergence of successful initiation of an ICL-specified task, models can successfully continue that task.

2. Setup and Key Results

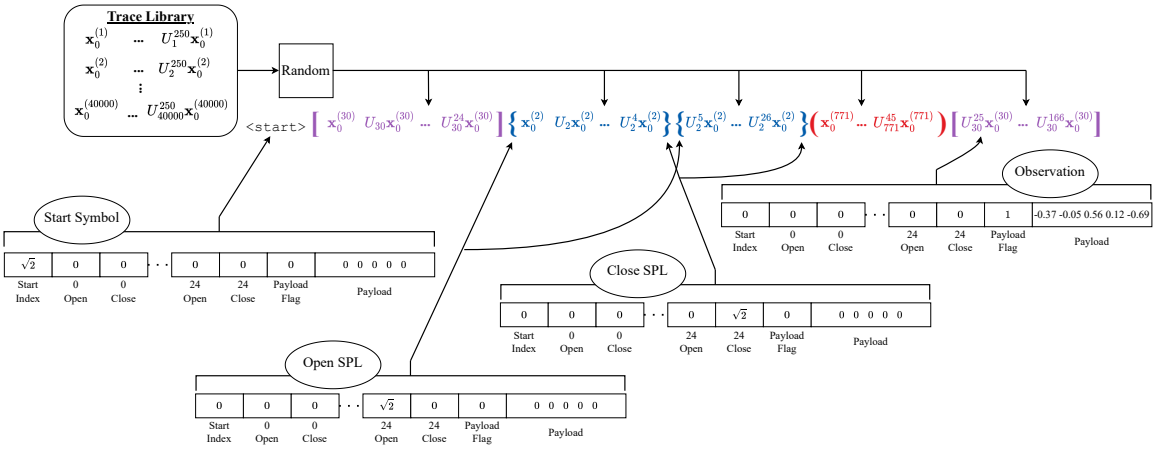


Figure 1: Generating a training example — Notice in this example the continuation from the first segment to the last (system U_{30}), and from the 2nd segment to the 3rd (system U_2). The "parentheses" (symbolic punctuating labels) are encoded as special tokens as shown.

For a more detailed explanation of the setup see Appendix A. Consider predicting the continuous-state of an *unknown* linear dynamical system from the orthogonally evolved family [47], where the system $U \in \mathbb{R}^{5\times5}$ is a uniformly drawn-at-random [30] orthogonal matrix. The initial state is $\mathbf{x}_0 \sim \mathcal{N}\left(0, \frac{1}{5}I\right)$, with state updates: $\mathbf{x}_{i+1} = U\mathbf{x}_i = U^{i+1}\mathbf{x}_0$. The system state is eventually perfectly predictable, but only after six positions in the sequence are observed by solving for $U = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \mathbf{x}_5\end{bmatrix}\begin{bmatrix}\mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4\end{bmatrix}^{-1}$. As described in further detail in Appendix A.2 and illustrated in Fig. 1, sequences drawn from different such systems are cut and their segments braided together into one long context — with each segment clearly delimited on both sides by symbolic label tokens identifying unique systems. Traditional loss curves during training are in Appendix D.

Before testing recall, we first confirmed that our trained model is able to learn to predict long sequences from unseen systems in-context. Details on this are available in Appendix D. From a training dynamics point of view, this ability seems to develop steadily during training — no evidence of "emergence."

To study associative recall, we use structured test traces as depicted in Fig. 2: the initial part of the context has N individually punctuated (with distinct open and close symbols) ten-entry-long segments from N distinct systems. We follow with a query for predictions from exactly one of the N systems, by using the corresponding open token followed by the continuation of that particular system observation sequence as the test segment where prediction performance can be evaluated using squared error. See Appendix B for more details on the test setup. The key results can be

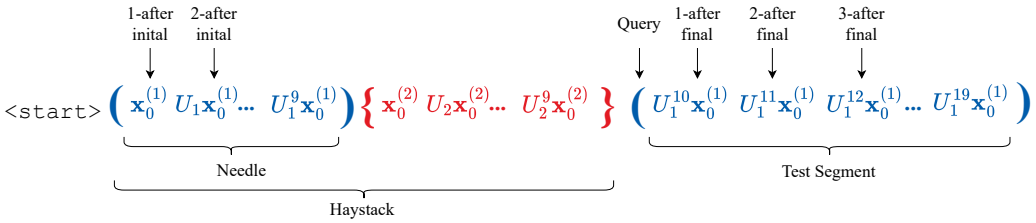
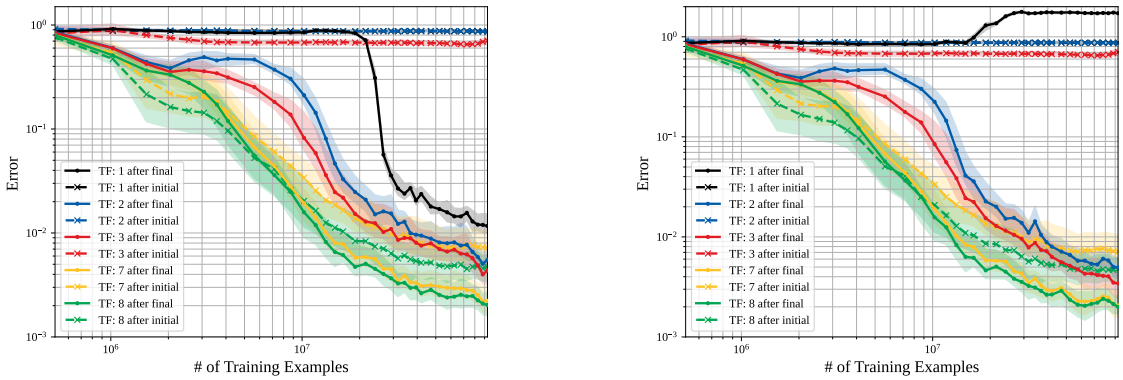


Figure 2: Test format with a two-system haystack and a query to resume the first system.

seen with $N = 2$ in Fig. 3. (Results for some other N are in Fig. 13.) Notice the black curve for performance at initiating the recalled segment. With a correct query (Plot (a) to the left), good performance emerges suddenly shortly before 2×10^7 training examples. Before that, it appears that no recall is happening. With a misdirected query (i.e. giving the open symbol associated with the other system in the context – as depicted in Plot (b) to the right), errors jump upward at the same point in training — which makes sense since the model is resuming the wrong sequence as it was told to do.



(*a*) Normal recall.

(*b*) Misdirected recall.

Figure 3: For two systems in the haystack, we show the training dynamics in terms of performance on recall tasks. The above curves are the quartiles of the median-squared error of the transformer model's predictions versus the number of training examples it has seen for indices 1, 2, 3, 7, and 8 steps after the initial and final open symbols. To the left, we see what happens with normal prompting. To the right, with misdirected prompting that asks to recall the wrong sequence.

However, notice also the striking similarity in all other curves in Fig. 3 with or without the misdirection — the performance on continuing the test segment is essentially unaffected by the misdirection! The mechanism here is clearly ignoring the content of the symbolic query. Notice also the interesting learning dynamics — while the black curve shows a classic "emergence-type" behavior, the blue curve for the performance on the second index into the test segment is very different: pointing to different learning dynamics. It is clearly showing recall much earlier with a sharp improvement starting before 1×10^7 training examples — well before the black curve does.

Further edge-pruning based investigations (See Appendix H.1.4.) show that the circuits for predicting the first and second tokens are completely distinct.

3. Do Pretrained LLMs Also Display Multi-Mechanism Tendencies? Yes!

To see whether our conjecture C3 (multiple mechanisms for a single multi-token task) holds for natural language problems solvable by prompting LLMs, we leverage OLMo-2 7B checkpoints [37] and a basic English to Spanish translation task that is inspired[1] by the IPA translation task used in previous works benchmarking and studying emergent behaviors [4, 53]. In Fig. 4 (right), we see a similar phase transition in the first token prediction task, a parallel of the 1-after dynamics of the associative recall setup (Appendix F). Meanwhile, the second-token performance is both better and more gradual in its improvement across training. This matches what we saw in our toy problem in Fig. 3 (a).

One natural question is whether what we are seeing in Fig. 4 (right) is the emergence of true ICL recall or just the underlying ability to start a translation itself. This can be probed by replacing the purely symbolic task labels "X:" and "Y:" in the few-shot examples with semantically informative "Spanish:" and "English:" labels. This replacement switches the problem from pure ICL for task recognition (*in-context associative recall*) to leveraging a learnt label (*in-weights associative recall*). The resulting performance is seen in Fig. 4 (left). Notice the marked improvement in the first-token performance that erases the entire gap to the second-token performance. This establishes that the model knows how to start a translation, it just can't in-context-learn well enough to know that is what it is supposed to do before the phase transition around 50k training steps.

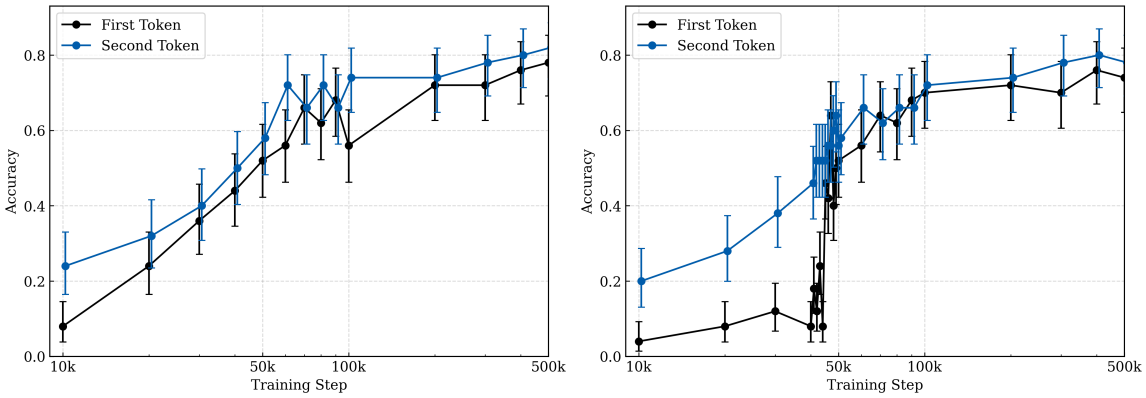


Figure 4: Comparative example of in-weights associative recall (left) and in-context associative recall (right) in a 2-shot prompting setting. Each point is a separate OLMo-2 7B model at different training steps. We report 95% credible intervals using Jeffreys prior ($Beta(0.5, 0.5)$) based on 100 samples per evaluation point.

1. We use Spanish instead of the International Phonetic Language (IPA) as IPA has tokens that are not compatible with the OLMo 2 tokenizer. Examples of the English to Spanish task are shown in Section 3. We also change the in-context labels to have no semantic meaning in light of [54].

4. Discussion

Benchmark performance of large language models (LLMs) has been observed to improve abruptly at certain scales in a seemingly unpredictable manner [15, 53], leading to the conclusion that LLMs exhibit *emergent* abilities, where different abilities may emerge at different scales or points during training. At this point, the reality of phase-transitions in abilities during training is well established, with arguably the strongest evidence of this coming from the "grokking" literature [11, 18, 20, 25, 29, 33, 35, 42–44, 51, 59]. What exactly drives emergence is an ongoing topic of investigation. While earlier work talked about model sizes and total compute, the story now is more nuanced. Powerful evidence connects the emergence of abilities to the pretraining losses attained [13], other information-oriented metrics [9], and the idea that more complex or specialized abilities can only emerge after a model acquires prerequisite abilities during training [8, 27].

Currently, the community also understands that ICL is rich and nuanced [21, 23, 32, 40, 52]. We contribute a new dimension of nuance by empirically pointing out that *a single ICL-driven task can be performed, on different tokens, using multiple mechanisms that emerge separately with their own training dynamics*. This prediction was directly obtained from seeing the behavior of our toy model — nobody had any reason to suspect it otherwise. And because the toy is simple, we have hope that it will help the community more deeply understand why and how this phenomenon occurs as well.

Of course, once we have actually seen and confirmed this behavior, we can speculate on why it occurs. When tasks have tight local coherency, there can often be approximate local underspecification — there are multiple ways of knowing what the model is supposed to be doing here. The intrinsic Bayesian orientation of autoregressive next-token prediction [57] means that this local underspecification can get picked up on, while the average-loss-over-tokens driving the training gradients means that an approximately correct mechanism that works most of the time will be rewarded and improved even if it can't solve the task completely. The very success of this approximately correct mechanism during training will further reduce the overall gradient pressure [41] for alternative and potentially better mechanisms, potentially forcing them to develop more slowly [49]. However, structurally, there are certain aspects of tasks that are likely to require the use of the better mechanisms — and it seems that starting an episode of a task might be one of them. These better mechanisms therefore can emerge later in training — but their emergence does not mean that the better information they are acting on will automatically be incorporated by the mechanisms already favored for other parts of the task. This contrasts with situations where the earlier emerging mechanism foregrounds information that can be used by the later mechanism [50].

References

[1] Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms. *arXiv preprint arXiv:2401.12973*, 2024.

[2] Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

[3] Per Bak. *How nature works : the science of self-organized criticality*. Copernicus, New York, NY, USA, 1996. ISBN 9780387987385.

[4] BIG bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

[5] Adithya Bhaskar, Alexander Wettig, Dan Friedman, and Danqi Chen. Finding transformer circuits with edge pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] 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. URL https://arxiv.org/abs/2005.14165.

[8] Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in MLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=MO5PiKHELW.

[9] Hang Chen, Xinyu Yang, Jiaying Zhu, and Wenya Wang. Quantifying semantic emergence in language models, 2024. URL https://arxiv.org/abs/2405.12617.

[10] Sultan Daniels, Dylan Davis, Dhruv Gautam, Wentinn Liao, Gireeja Ranade, and Anant Sahai. Decomposing prediction mechanisms for in-context recall, 2025. URL https://arxiv.org/abs/2507.01414.

[11] Xander Davies, Lauro Langosco, and David Krueger. Unifying grokking and double descent. *arXiv preprint arXiv:2303.06173*, 2023.

[12] Zhe Du, Haldun Balim, Samet Oymak, and Necmiye Ozay. Can transformers learn optimal filtering for unknown systems? *IEEE Control Systems Letters*, 7:3525–3530, 2023. doi: 10.1109/LCSYS.2023.3335318.

[13] Zhengxiao Du, Aohan Zeng, Yuxiao Dong, and Jie Tang. Understanding emergent abilities of language models from the loss perspective, 2025. URL https://arxiv.org/abs/2403.15796.

[14] Benjamin L Edelman, Ezra Edelman, Surbhi Goel, Eran Malach, and Nikolaos Tsilivis. The evolution of statistical induction heads: In-context learning markov chains. *arXiv preprint arXiv:2402.11004*, 2024.

[15] Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova Dassarma, Dawn Drain, Nelson Elhage, et al. Predictability and surprise in large

generative models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1747–1764, 2022.

[16] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

[17] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[18] Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

[19] Ruomin Huang and Rong Ge. Task descriptors help transformers learn linear models in-context. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=lZNb1CVm5O.

[20] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep networks always grok and here is why. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=zMue490KMr.

[21] Andrew Kyle Lampinen, Stephanie C. Y. Chan, Aaditya K. Singh, and Murray Shanahan. The broader spectrum of in-context learning, 2024. URL https://arxiv.org/abs/2412.03782.

[22] Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, et al. Surge phenomenon in optimal learning rate and batch size scaling. *arXiv preprint arXiv:2405.14578*, 2024.

[23] Ziqian Lin and Kangwook Lee. Dual operating modes of in-context learning, 2024. URL https://arxiv.org/abs/2402.18819.

[24] Jerry Weihong Liu, Jessica Grogan, Owen M Dugan, Simran Arora, Atri Rudra, and Christopher Re. Can transformers solve least squares to high precision? In *ICML 2024 Workshop on In-Context Learning*, 2024.

[25] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2022.

[26] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[27] Ekdeep Singh Lubana, Kyogo Kawaguchi, Robert P. Dick, and Hidenori Tanaka. A percolation model of emergence: Analyzing transformers trained on a formal language, 2024. URL https://arxiv.org/abs/2408.12578.

[28] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[29] Neil Rohit Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: grokking modular arithmetic via average gradient outer product. In *NeurIPS 2024 Workshop on Mathematics of Modern Machine Learning*, 2024.

[30] Francesco Mezzadri. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006.

[31] Eric Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. The quantization model of neural scaling. *Advances in Neural Information Processing Systems*, 36, 2023.

[32] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022. URL https://arxiv.org/abs/2202.12837.

[33] Mohamad Amin Mohamadi, Zhiyuan Li, Lei Wu, and Danica Sutherland. Grokking modular arithmetic can be explained by margin maximization. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023. URL https://openreview.net/forum?id=QPMfCLnIqf.

[34] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference, 2024. URL https://arxiv.org/abs/2112.10510.

[35] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

[36] Eshaan Nichani, Jason D. Lee, and Alberto Bietti. Understanding factual recall in transformers via associative memories, 2024. URL https://arxiv.org/abs/2412.06538.

[37] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

[38] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[39] Jane Pan, Tianyu Gao, Howard Chen, and Danqi Chen. What in-context learning "learns" in-context: Disentangling task recognition and task learning, 2023. URL https://arxiv.org/abs/2305.09731.

[40] Core Francisco Park, Ekdeep Singh Lubana, Itamar Pres, and Hidenori Tanaka. Competition dynamics shape algorithmic phases of in-context learning, 2025. URL https://arxiv.org/abs/2412.01003.

[41] Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron C Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. *Advances in Neural Information Processing Systems*, 34:1256–1272, 2021.

[42] Mohammad Pezeshki, Amartya Mitra, Yoshua Bengio, and Guillaume Lajoie. Multi-scale feature learning dynamics: Insights for double descent. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17669–17690. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/pezeshki22a.html.

[43] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[44] Lucas Prieto, Melih Barsbey, Pedro AM Mediano, and Tolga Birdal. Grokking at the edge of numerical stability. *arXiv preprint arXiv:2501.04697*, 2025.

[45] Nived Rajaraman, Marco Bondaschi, Ashok Vardhan Makkuva, Kannan Ramchandran, and Michael Gastpar. Transformers on markov data: Constant depth suffices. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=5uG9tp3v2q.

[46] Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression. *Advances in Neural Information Processing Systems*, 36, 2024.

[47] Michael Eli Sander, Raja Giryes, Taiji Suzuki, Mathieu Blondel, and Gabriel Peyré. How do transformers perform in-context autoregressive learning ? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 43235–43254. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sander24a.html.

[48] Hinrich Schutze and Christopher Manning. I preliminaries. In *Foundations of Statistical Natural Language Processing*. MIT Press, United States, 1999. ISBN 9780262133609.

[49] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.

[50] Aaditya K. Singh, Ted Moskovitz, Sara Dragutinovic, Felix Hill, Stephanie C. Y. Chan, and Andrew M. Saxe. Strategy coopetition explains the emergence and transience of in-context learning, 2025. URL https://arxiv.org/abs/2503.05631.

[51] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19: 1–57, 2018.

[52] Xiaolei Wang, Xinyu Tang, Wayne Xin Zhao, and Ji-Rong Wen. Investigating the pre-training dynamics of in-context learning: Task recognition vs. task learning, 2024. URL https://arxiv.org/abs/2406.14022.

[53] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

[54] Jerry Wei, Le Hou, Andrew Lampinen, Xiangning Chen, Da Huang, Yi Tay, Xinyun Chen, Yifeng Lu, Denny Zhou, Tengyu Ma, et al. Symbol tuning improves in-context learning in language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 968–979, 2023.

[55] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently, 2023. URL https://arxiv.org/abs/2303.03846.

[56] Noam Wies, Yoav Levine, and Amnon Shashua. The learnability of in-context learning, 2023. URL https://arxiv.org/abs/2303.07895.

[57] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

[58] Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning?, 2025. URL https://arxiv.org/abs/2502.14010.

[59] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Appendix A. Training Details

Consider predicting the continuous-state of an *unknown* linear dynamical system. We focus on the orthogonally evolved system family [47], where the system is defined by $U \in \mathbb{R}^{5\times5}$, a random orthogonal matrix. Each U is generated by the algorithm presented in [30], which ensures a uniform sampling over all $\mathbb{R}^{5\times5}$ orthogonal matrices. The initial state is $\mathbf{x}_0 \sim \mathcal{N}\left(0, \frac{1}{5}I\right)$, with state updates:

$$\mathbf{x}_{i+1} = U\mathbf{x}_i = U^{i+1}\mathbf{x}_0. \tag{1}$$

The system state is in-principle perfectly predictable, but only after six positions in the sequence are observed by solving for

$$U = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \mathbf{x}_5\end{bmatrix}\begin{bmatrix}\mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4\end{bmatrix}^{-1}. \tag{2}$$

A.1. Optimal Pseudoinverse Predictor

Following from (2), given the state observations $\{\mathbf{x}_0, \ldots, \mathbf{x}_i\}$, an optimal predictor for this problem computes $\widehat{\mathbf{x}}_{i+1} = \widehat{U}\mathbf{x}_i$, where

$$\widehat{U} = \begin{bmatrix}\mathbf{x}_1 & \ldots & \mathbf{x}_i\end{bmatrix}\begin{bmatrix}\mathbf{x}_0 & \ldots & \mathbf{x}_{i-1}\end{bmatrix}^{\dagger}, \tag{3}$$

and $X^{\dagger}$ denotes the Moore-Penrose pseudoinverse of X. Essentially, this baseline only makes non-zero errors on the first, second, third, fourth, fifth, and sixth entry in any sequence — it gets everything else perfectly correct.

A.2. Data generation and training

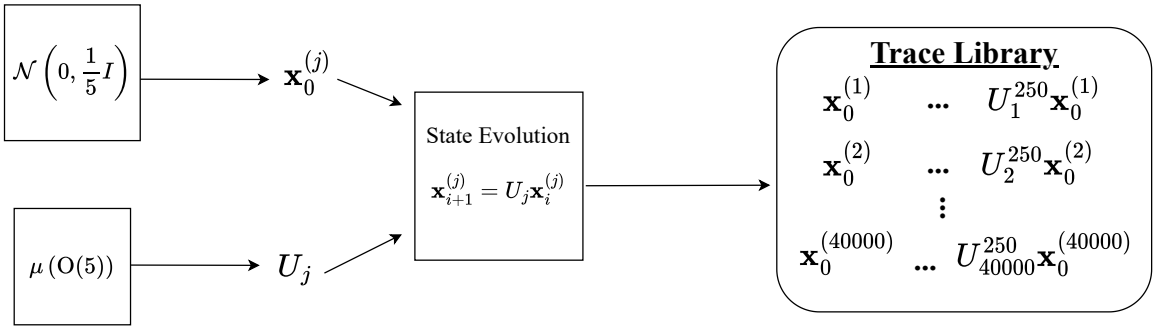


Figure 5: The generation of a train or test library of sequences.

Generating a library of training sequences: Depicted visually in Fig. 5, we first compile a training library by the following method:

1. Generate 40000 orthogonal matrices iid uniformly over all orthogonal matrices $U_1, \ldots, U_{40000} \overset{iid}{\sim} \mu\left(\mathrm{O}(5)\right)$, where $\mu\left(\mathrm{O}(5)\right)$ is the Haar measure over orthogonal matrices in $\mathbb{R}^{5\times5}$. [30]
2. Generate 40000 iid initial states that will correspond to each training system $\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(40000)} \overset{iid}{\sim} \mathcal{N}\left(0, \frac{1}{5}I\right)$.
3. Roll out the states to get observation sequences that are each 251 entries long, and compile the sequences as our training library.

Cutting and interleaving training sequences To form a training trace, we interleave segments of observation sequences from the library into a context window of length 251, by this process:

1. Insert the start symbol at index 0.
2. Sample the maximum number of systems in the trace N from a $\mathrm{Zipf}(1.5, 25)$ distribution depicted graphically[2] in Fig. 7(a). This means that no more than 25 systems will ever appear in a training trace.
3. Choose N of the 40,000 systems in the training library uniformly at random without replacement.
4. Randomly assign to each of the N systems a pair of symbolic open and close labels for this training example.
5. Sample the number of cuts $C \sim \mathrm{Poisson}(2N)$ to be made in the trace. This means that there will be $C + 1$ trace segments in the trace.[3]
6. Place the C cuts uniformly at random with replacement within the context window.
7. For each segment created by the cuts, in order, uniformly at random choose one of the N systems with replacement.
8. At the cut at the beginning of the segment, the open label for this segment's system is inserted.[4]
9. For the system chosen, check if it has appeared in a previous segment of the trace. If not, insert the system's segment from the training trace library starting at index 0. If this system has appeared in a previous segment, insert the system's segment from the training trace library starting at the index that corresponds to the continuation of the previous segment for this system.
10. At one index before the next cut, insert the close label for this segment's system.

Figure 6: Example of a 251-element-long interleaved training example.

2. The Zipf distribution was chosen for its ubiquity in nature [3] and natural language [48], along with recent work pointing to its importance in modern neural networks [31].
3. On average, each training trace has $2N + 1$ segments to ensure that the trained model has seen ample interruptions and continuations of systems.
4. Since the open and close labels occupy two indices in the context window, there are three special cases that can occur: (1) If two cuts are sampled to be on top of each other, then the first of the two cuts that were sampled is ignored; (2) If the two cuts are sampled to occupy adjacent indices, then only the close label for the system corresponding to first of the two cuts is inserted, effectively making that index meaningless as close labels are masked; (3) If the two cuts are sampled so that there is only one index between them, then the open label for the system corresponding to the first of the two cuts is inserted and is immediately followed by the close label for that system, effectively making both indices meaningless due to the masking of the labels. Note that the distributions shown in Fig. 7 do not account for these rare special cases.

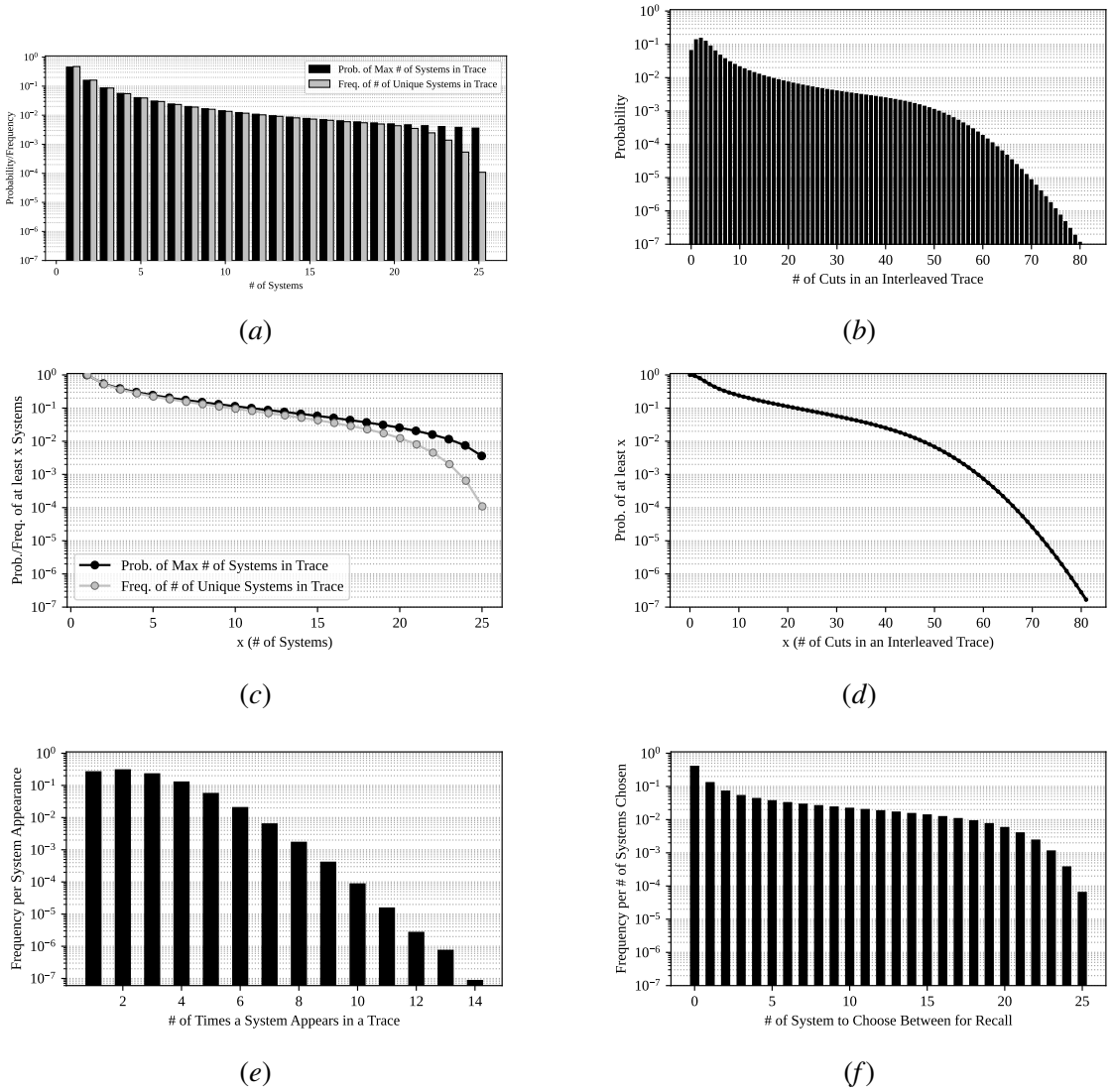


Figure 7: Distributions and complementary cumulative distribution functions (CCDFs) used in data generation — Fig. 7(a) is the $\text{Zipf}(1.5, 25)$ distribution for the maximum number of systems per trace in black and the number of unique systems per trace for 1×10^7 traces in silver. Fig. 7(c) shows the CCDFs for these distributions. Fig. 7(b) shows the PMF and Fig. 7(d) shows the CCDF of the number of cuts per trace. Fig. 7(e) shows the frequency of the number of times a system will appear in the same trace per system appearance. Lastly, Fig. 7(f) shows the frequency of the number of previously seen systems a predictor must choose between to recall in a trace per system appearance. For example, if system 0 is chosen then system 1, then system 0, then the model must choose between two systems to recall.

Note that within a single training example, segments of a particular system always start with the same open token and always end with its corresponding close token. These random assignments are redrawn at the beginning of the interleaving process for each training example; therefore, *the same system can have different symbolic open and close labels when it appears in different training examples*. See Fig. 6 for a diagram of an interleaved training example.

Given this randomized procedure for generating interleaved training examples, we can analyze the training distribution to better understand how frequently a model must recall a system, or sees many systems in a trace.

In Fig. 7, we show relevant distributions that are derived from the randomized interleaving procedure in this section. Figs. 7(a) and 7(c) show the $\mathrm{Zipf}(1.5, 25)$ distribution for the maximum number of systems per trace in black and the number of unique systems per trace in silver. The frequency of number of unique systems per trace follows closely the $\mathrm{Zipf}(1.5, 25)$ distribution for the smaller quantities of systems, but diminishes quicker for larger numbers of systems, due to the coupon-collecting phenomenon of picking the same system multiple times in a trace. The PMF for the number of cuts made in an interleaved trace is shown in Fig. 7(b), while the CCDF for this quantity is given in Fig. 7(d). Fig. 7(e) shows the frequency of how many times a system appears in a training trace. If the same system appears more than once, than the model must perform recall. Therefore, Fig. 7(e) gives an idea of how often the model must recall a system during training. Finally, Fig. 7(f) provides the frequency of the number of previously seen systems in the training trace that are candidates to be continued when a predictor is tasked to recall a system. The value zero on the x-axis of this figure means that the model is seeing a system for the first time in a training example and has no need to recall. Later, in Appendix B.2, we construct tests for the associative recall ability of the trained model for different numbers of candidate systems to be continued in the trace. Fig. 7(f) shows that for 19 candidate systems, the largest number of candidate systems that we tested on, the model has been presented with this situation less than 1% of the time during training.

Input structure and embedding The input dimension of our models is 57. There are 50 dimensions for encoding paired symbolic open and close labels, a dimension for the start symbol, a dimension for the payload flag and 5 dimensions to hold the 5-dimensional observation vectors. The special symbols are one-hot encoded vectors; see Fig. 8 for an example of the open symbol. For the observation sequence between the SPLs, the 5-dimensional state vectors are inserted into the payload portion of the input vector, the payload flag is set to 1, and the rest of the input vector dimensions are zeroed out.

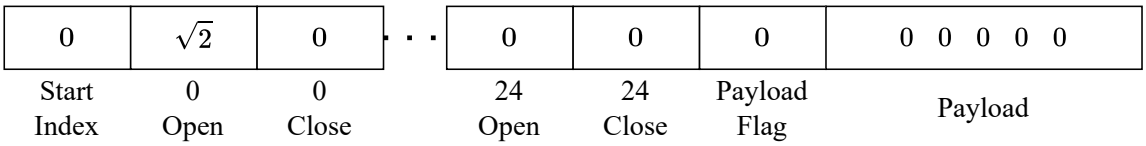


Figure 8: The one-hot encoding of an open symbolic label. In this example, the system corresponding to this label is assigned to be "system 0."

The entire randomized procedure of generating interleaved training traces is depicted in Fig. 1 along with the structure of the inputs into the model.

Model and embedding Building off of the codebase in [12], which was influenced by [16], we train a 2.42M parameter GPT-2 style transformer to perform this task. Our model[5] has hidden dimension 128, 12 layers, and 8 heads. Our model's input embedding is 128×57 dimensional. The model's output layer is 5×128 to ensure that the model makes 5-dimensional predictions. The input and output layers are untied [12].

Training and Hyperparameters New interleaved training examples are generated for each training iteration and our GPT-2-style model was trained for next token prediction on these training traces.[6] The loss for all SPLs on the output were zeroed out. A model that successfully recalls the state of a system seen previously in its context should make predictions after the open token that perform as it would have if the relevant sequence had continued on without interruption.

Following the choice made in [12], we trained our model with a weight decay of 1×10^{-2}. We used a batch size of 512, a learning rate of $\approx 1.58 \times 10^{-5}$, and trained on a single NVIDIA L40S GPU with 45GB of RAM. A single training run takes around 5 days. We used the AdamW Optimizer [26] and trained using mean-squared error loss.

Appendix B. Test Setup

B.1. Uninterleaved sequence test

To test the model's ICL ability for the first system that is seen (Appendix D), we generate 100 held-out systems and 1000 different held-out initial states[7] by the same method described in Appendix A.2 to form our testing library. We then evaluate the model on the uninterleaved traces from this testing library.

B.2. Needle-in-a-haystack test

To evaluate the model's ability to restart ICL on a new system (Appendix G) and recall a previously seen system (Appendix F), we generate a series of structured "needle-in-a-haystack" test traces through interleaving the traces in the testing library generated in Appendix B.1. A single "needle-in-a-haystack" trace is generated by the following procedure:

1. Choose $N \in [1, 19]$ to be the number of distinct systems in a test trace.
2. For each of the N systems, insert a segment of 10 state observations starting from index 0 from the testing library into the "needle-in-a-haystack" test trace. Each of these segments are individually punctuated with a unique open and close symbol pair. We call this portion of the trace the "haystack".
3. Append a query open symbol to the test trace that signifies which system in the haystack will be be continued. The segment that will be continued is called the "needle".

5. These parameter counts and model dimensions are for our "medium" model. Three other models of different sizes were also trained, and their model dimensions are given Table 1 in Appendix C.
6. The model sees newly interleaved training examples at each iteration, but the training traces that are interleaved into the training example are drawn from the fixed training library of 40,000 sequences. Therefore, the model undergoes single-epoch training where the information within a training example might have appeared in many other training examples.
7. We generate 1000 initial states for each system to narrow down the quartiles in the squared-error curves.

4. Append 10 state observations from the continuation of the system in the haystack corresponding to the query open symbol. This portion is called the "test segment".

See Fig. 2 for a diagram of a test trace for $N = 2$ systems in the haystack and system U_1 as the needle.

For the full "needle-in-a-haystack" test dataset, we would like to ensure that we test on the same systems for the different values of N, while having diverse systems in our dataset so that our results are statistically meaningful. To achieve this, we test on 50 "needle-in-a-haystack" trace configurations. A trace configuration is a specific ordering of systems from the testing library in the positions of the haystack. For the first needle-in-a-haystack trace configuration that we generate, we place a segment from "system 0" from the testing library into the first position in the haystack, and fill the rest of the haystack positions consecutively until the N-th position is filled with a segment from "system $N-1$". For the next trace configuration, the first position in the haystack is filled with a segment from "system 1" and the rest of the haystack positions are filled consecutively until the N-th position is filled with a segment from "system N". This pattern continues until the last trace configuration. In our case, we tested on 50 trace configurations, meaning the haystack of the last trace configuration started with "system 49" and ended with "system $48+N$". Each trace configuration is populated with 1000 different initial states for each system. For the results in the main paper, the test segment is a continuation of the segment in the first position of the haystack. For results where segments in other haystack positions are continued in the test segment see [10].

$$\begin{matrix} \{0 & \dots & N-1\} \\ \{1 & \dots & N\} \\ \vdots & \vdots & \vdots \\ \{49 & \dots & 48+N\} \end{matrix}$$

Figure 9: When testing on 50 needle-in-a-haystack trace configurations, the order of system indices from the testing library in a haystack of size N for each needle-in-a-haystack test trace is given above. For each system, 1000 sequences are interleaved to build a testing dataset of shape $50 \times 1000 \times (12N+1) \times 5$. The shape of the second to last axis is due to the start token and haystack segments being 10 context indices long plus two indices for the symbolic open and close labels. The last axis is 5-dimensional since every system has 5-dimensional observations.

Appendix C. The Effects of Model Size

In order to test the effect of model size on our emergence results, we trained models across 4 different model sizes as shown in Table 1. We originally tuned the learning rate for the medium model with a batch size of 512 on a single GPU. Following the model scaling that was done[8] in [7], when decreasing the size of our model from "medium" to "small" we halved the number of layers, multiplied the model dimension by 0.75, maintained the same head dimension, and doubled

8. Alternatively, one could follow the model scaling done in [17]. There, they also halve the number of layers when decreasing the model size, but decide to halve the model dimension and number of attention heads as well.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	**Learning Rate**	**Batch Size**
Orthogonal Tiny	212K	3	72	6	12	1.7×10^{-4}	2048
Orthogonal Small	701K	6	96	6	16	4.5×10^{-5}	1024
Orthogonal Medium	2.42M	12	128	8	16	1.6×10^{-5}	512
Orthogonal Big	10.7M	24	192	12	16	1.5×10^{-5}	640

Table 1: Model size and training hyperparameters.

the learning rate. To go from "small" to "tiny", we used the same process except we chose the head dimension to be 12 to maintain an integer value for the number of heads. If we would have maintained the head dimension of 16, the tiny model would have had 4.5 heads. For this reason, 12 was chosen as the head dimension since it is the largest integer less than 16 that leads to an integer when dividing 72. To go from "medium" to "big", we doubled the number of layers, multiplied the model dimension by 1.5, maintained the same head dimension, and multiplied the learning rate by $\frac{5}{6}$. It was also brought to our attention that the learning rate should also scale with the batch size. For our later orthogonal runs, we additionally adopted the square-root learning-rate scaling as indicated in [22]. Specifically, we took the learning rate we had scaled by model size and multiplied it by $\sqrt{\frac{batch\_size}{512}}$. However, we have not verified if this scaling is the best way to proceed with our training and further testing is still required. Furthermore, we trained on one L40S GPU for the "tiny", "small", and "medium" models and we trained on two L40S GPUs for the "big" model. One L40S GPU has 48GB of RAM. This is the reason for the differing batch sizes across different model types.

Appendix D. Pretraining Loss Dynamics

In [13], it was shown that many emergent abilities emerge for models of different sizes once a model's pretraining loss performance on a broad corpus of held-out data in the style of their pretraining data reaches a certain threshold. We want to see if this holds true in our toy setting of interleaved time-series prediction.

We evaluate four different model sizes (see Table 1 in Appendix C for model parameter details) and show how their pretraining loss dynamics differ throughout training. In Fig. 10, we see the performance of model training checkpoints on data that is in the style of what the model saw during training as specified in Appendix A.2. The dotted curves are the models' performance on freshly drawn interleavings of traces from a held-out test library, while the crossed curves are on freshly drawn interleavings of traces from the training library.

In Fig. 10, it is clear that larger models decrease their pretraining loss earlier in training than smaller models. Furthermore, classic overfitting behavior is evident, especially in the larger models. We see that the pretraining error on the held-out dataset deteriorates late in training, while the error on the training data continues to decrease. For the "big" model, its error on the training data even goes lower than the fundamental lower bound imposed by the error of the perfect pseudoinverse predictor.

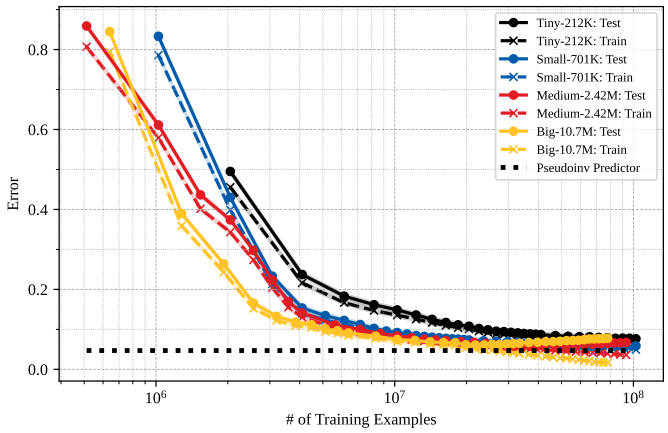
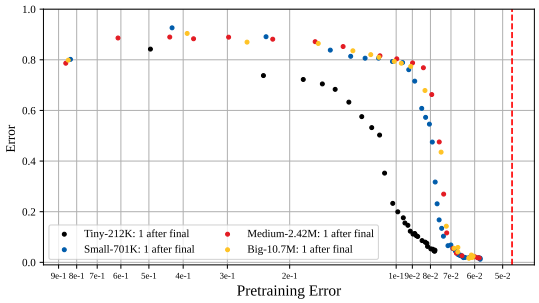


Figure 10: Pretraining loss — The squared-error of each transformer model's predictions on traces interleaved in the style of the training data averaged over each time step of the trace. For both the train and test data, averaging was done over 40,000 different interleaved traces, each of length 251. The horizontal black dotted line is the averaged squared-error of a predictor that computes an estimate of the underlying system dynamics by using the Moore-Penrose pseudoinverse of the observed data.

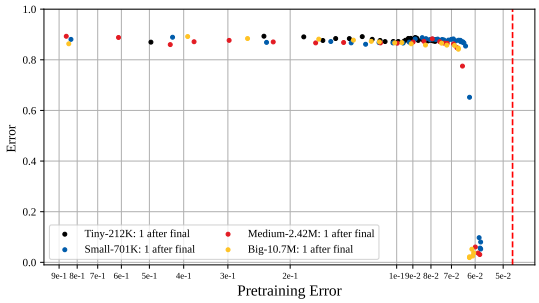
D.1. Training dynamics with respect to the pretraining loss

Taking inspiration from [13], in Fig. 11 we look at when the ability to resume a recalled sequence, the ability to continue predicting a recalled sequence, and the ability to restart ICL on a previously unseen system emerge with respect to the pretraining loss as opposed to the number of training examples. This is to see if these abilities emerge at a specific pretraining loss threshold that is independent of the model size. This pretraining loss corresponds to the held-out loss specified in Appendix D. The red vertical line in the plots is the pretraining loss achieved by the pseudoinverse predictor specified in (3) averaged over all context indices. The blue horizontal line in Figs. 11(e) and 11(f) is the median error of the pseudoinverse predictor at the specific index corresponding to the curve plotted.

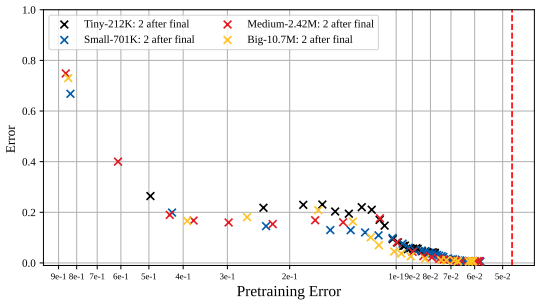
Although in [10] it is shown that larger models develop these emergent abilities before smaller models with respect to the number of training examples, in Fig. 11(b) we see that all the models exhibit the "1-after final" phase transition at a pretraining loss of around 6.5×10^{-2}. Furthermore, in Fig. 11(d), the "2-after final" phase transition occurs for all models begin at a pretraining loss of a bit more than 1×10^{-1}. In Fig. 11(a) the different models are again tightly linked, except for the "tiny" model whose emergence for "1-after final" occurs at a pretraining loss that is around 0.02 larger than the rest. Overall, these results seems to qualitatively match the conclusions in [13] that pretraining loss is a good indicator for when emergent abilities emerge.



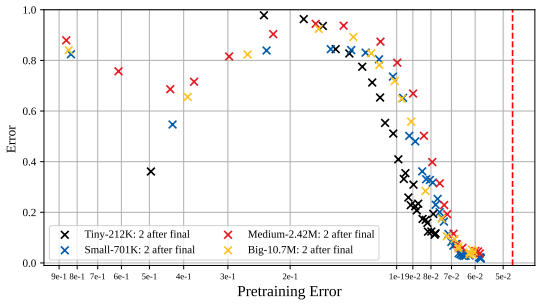
(*a*) Recall 1 step after final — 1 system haystack.



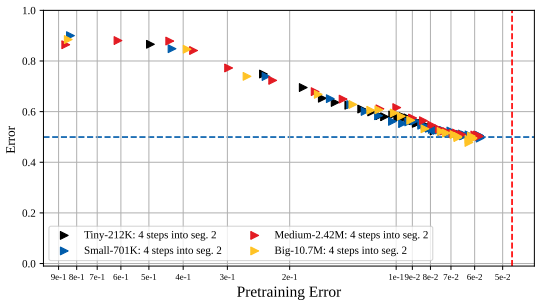
(*b*) Recall 1 step after final — 7 system haystack.



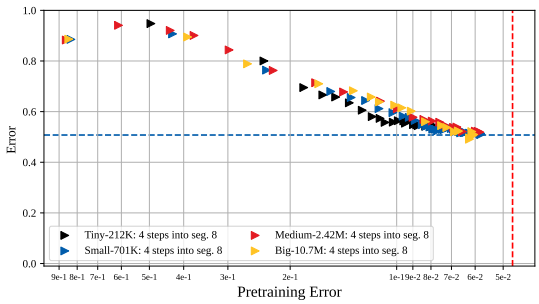
(*c*) Recall 2 steps after final — 1 system haystack.



(*d*) Recall 2 steps after final — 7 system haystack.



(*e*) Restart 4 steps into segment 2.

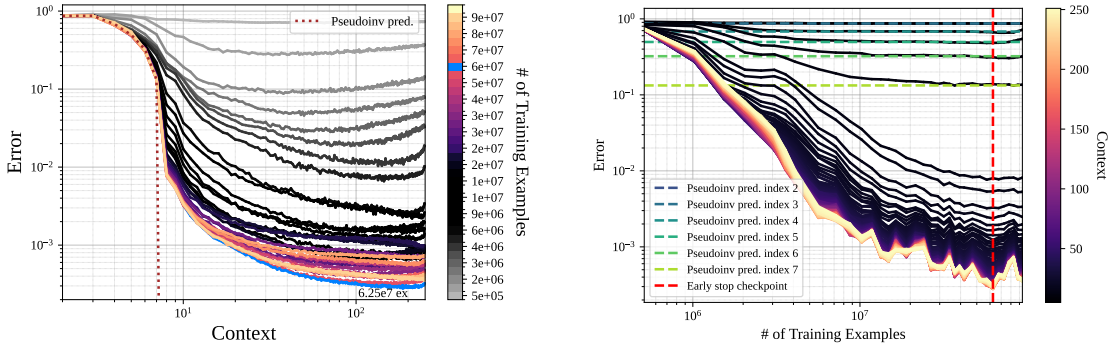


(*f*) Restart 4 steps into segment 8.

Figure 11: Recall and restart performance vs pretraining loss on held-out data. The red vertical line is the fundamental lower bound pretraining loss achieved by the pseudoinverse predictor. The blue horizontal line in Figs. 11(e) and 11(f) is the median error of the pseudoinverse predictor at the specific index plotted in each figure.

Appendix E. Can a model learn the first system in-context? Yes

We first confirm that our trained model is able to in-context learn the first system seen in its context. We evaluate the model on the uninterleaved traces from the testing library specified in Appendix B.1. In Fig. 12(a), we plot the median squared-error over these test traces vs the context index, and the color of each curve represents how far along the model is in training. The dotted line in this figure is the median squared-error of the pseudoinverse predictor in (3) over the same test traces. In Fig. 12(b) we plot the median squared-error over these test traces as training proceeds (measured by the number of training examples seen so far), the color of each solid curve represents the context index, the blue and green dotted horizontal lines are the optimal pseudoinverse predictor's median squared error for the specific early context indices.



(*a*) (*b*)

Figure 12: Performance on a long uninterleaved trace — 12(a) and 12(b) depict the in-context learning performance of the model on long uninterleaved test examples. The median squared-error is plotted against the context index in Fig. 12(a), and against the number of training examples seen in Fig. 12(b). The red line in Fig. 12(b) at 6.25×10^7 training examples, and the blue curve in Fig. 12(a) denote the checkpoint that we use for early stopping. In Fig. 12(a) the optimal pseudoinverse predictor is the brown dashed curve, while in Fig. 12(b) it is denoted by the blue and green horizontal lines for each of the early context indices. In Fig. 12(b), notice that the model gradually learns to make optimal predictions as opposed to the sudden emergence of associative recall ability seen in Appendix F, since the prediction errors for early indices slowly converge towards the pseudoinverse predictor's performance.

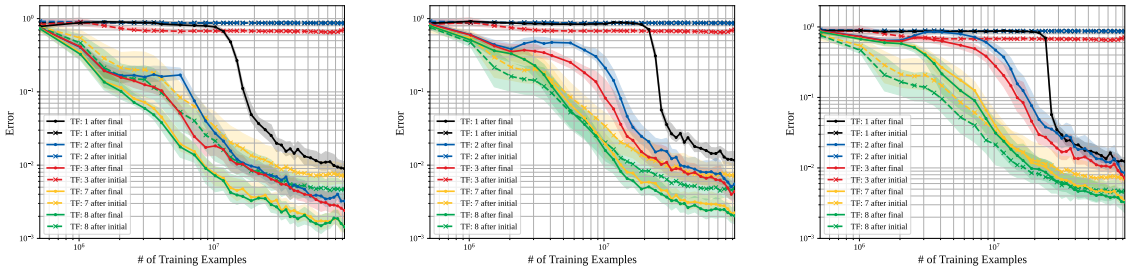
Notice in Fig. 12(a) that the early model checkpoints saturate out and cannot continue to make better predictions with more context. Nevertheless, after seeing 6.25×10^7 training examples, the model's prediction error on indices 2 through 7 closely match those of the pseudoinverse baseline from (3). This is also seen in Fig. 12(b) where, as training proceeds, the dark curves representing the model's prediction errors on early indices converge to the prediction error of the pseudoinverse predictor for the corresponding index given by the blue and green curves. The best performance of the model is at the end of the context window with a median squared-error of $\approx 2 \times 10^{-4}$. According to [24], this is near the practical precision threshold for transformer models.

Notice in Fig. 12(b) the model is gradually learning to make better predictions as training continues. This ICL ability for the first system seen is the same ability that is studied by [12, 47] and using this evaluation metric, we see that sudden emergence is not present. Nonetheless, using the same evaluation metric, the ability to restart ICL for a new system (Appendix G) and to recall a previously seen system (Appendix F) both exhibit emergence.

Additionally, we notice in Fig. 12(b) that the squared-error bottoms out after 6.25×10^7 training examples, showing that the model suffers from overfitting late in training. Having seen this, we set our early stopping checkpoint at 6.25×10^7 training examples, as denoted by the red vertical line in Fig. 12(b) which corresponds to the blue curve in Fig. 12(a).

In summary, the model is able to use context to make better predictions of state observations from held-out test systems. The model's ability gradually develops during training, as opposed to how associative recall develops suddenly later in training as seen in Appendix F.

Appendix F. Further Plots



(*a*) 1 system haystack. (*b*) 2 system haystack. (*c*) 5 system haystack.

Figure 13: Training dynamics—All haystack segments are of length 10 (excluding delimiting tokens). The test set consisted of 1,000 traces from each of 50 systems. The above curves are the quartiles of the mean-squared error of the transformer model's predictions versus the number of training examples it has seen for indices 1, 2, 3, 7, and 8 steps after the initial and final open tokens.

Because of space limitations, we only included the plots for a "haystack length" of 2 in the main paper. However, interesting behavior is visible if one considers $N = 1$ and $N = 5$ as well, illustrated together with $N = 2$ for easy comparison in Fig. 13. Notice that:

- $N = 1$ shows interesting trivial-recall behavior emerging earlier for the black curve: soon after 1×10^7 training samples.
- $N = 1$ also shows two interesting transitions during training in the blue curve for predicting the second position in a resumed sequence: first around 2×10^6 training samples where improvements stop and second around 6×10^6 training samples where improvements begin to happen much faster. By contrast, the behavior of the red curve for predicting the third position is much smoother.

- $N = 5$ puts the behavior of $N = 2$ into a clearer light by illustrating that whatever is happening before 1×10^7 training samples for the red and blue curves clearly gets progressively worse when there are more systems in the haystack.

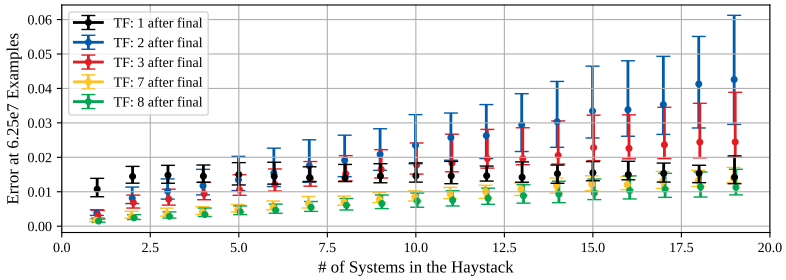


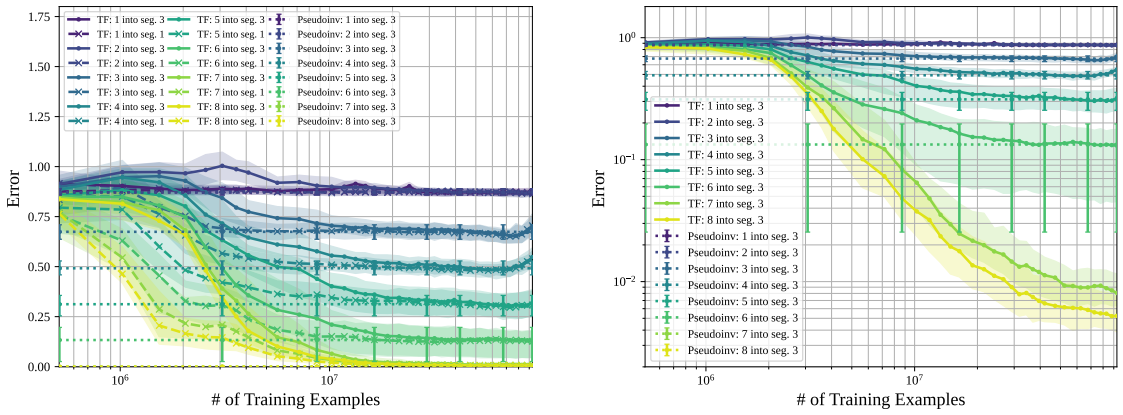
Figure 14: The 25th, 50th, and 75th quartiles of the MSE after 6.25×10^7 training examples as the number of systems in the haystack increases.

In Fig. 14, if the symbolic label is used to perform the task, we expect predictions in the final query sequence to be unaffected by the number of systems in the haystack. Contrarily, performing a symbol agnostic Bayesian prediction gets progressively more difficult with more systems in the haystack, since there are more candidate orthogonal matrices to average over. Observe that the 1-after final symbolic label performance remains steady with more systems, while the predictions for the later indices get progressively worse.

Appendix G. Emergence of the ability to restart ICL on new systems

We now show the training dynamics for the ability to restart in-context learning for a new system that was not the first system seen in-context. We find that learning this restart ability is not gradual, as it was for learning the first system seen (Appendix D). Instead, we see that the model begins to transition from poor restart performance to good restart performance early in training as compared to when associative recall emerges in training which we will see in Appendix F. We study the model's performance on the haystack segments of "needle-in-a-haystack" test examples (see the diagram in Fig. 2).

In Fig. 15, the median squared-error vs the number of training examples seen is plotted for steps 1 through 8 into the first system segment in the haystack and the third system segment in the haystack. Specifically, in Fig. 15(a) we see that at the beginning of training the model has not learned to restart its predictions for the third system segment, as its median squared-error in segment 3 is well above its counterpart predictions in segment 1 for all steps into each segment except for step 1. This shows that early on in training, there is clearly substantial interference from earlier segments when the model tries to learn to predict the behavior for a new sequence (explicitly labeled as such) that it is seeing in the third position in the haystack. As training proceeds, we see that the median squared-error for each step in segment 3 converges to the value of its segment 1 counterpart. Fig. 15(b) shows that the model transitions towards restarting ICL correctly when training has processed $\approx 2 \times 10^6$ training examples.



(*a*) (*b*)

Figure 15: Performance on new subsequent segments. (a) is the squared-error of predictions on steps 1 through 8 into the first and third system segments, where each segment is seen for the first time in context. (b) is the squared-error for steps 1 through 8 into the third system segments on log scale. The error bars show the 25^{th} and 75^{th} percentiles across trace configurations of the model's prediction error across the medians over the 1000 initial states in each trace configuration. The horizontal dotted lines are the median squared-error of the optimal pseudoinverse predictor.

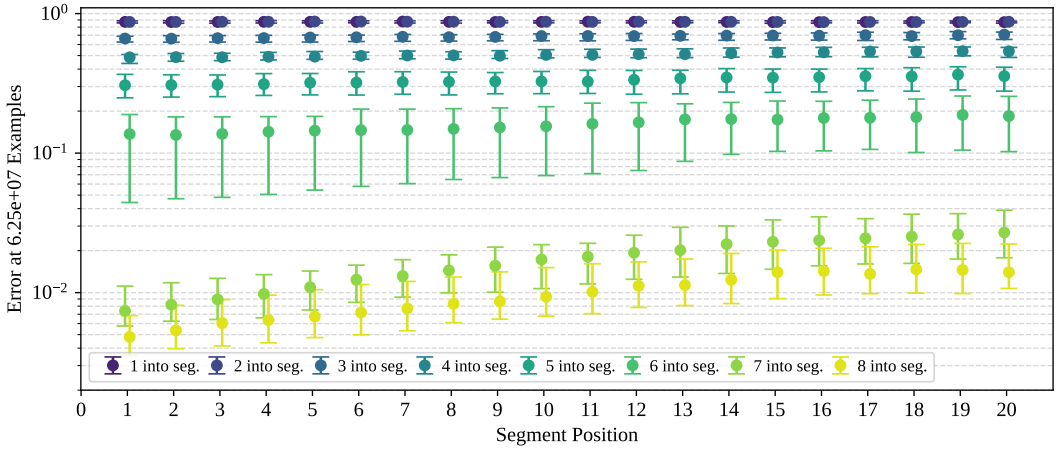


Figure 16: Restarting for a new system at the early-stopping checkpoint after seeing 6.25×10^7 training examples.

In Fig. 16, we show the median squared-error of the model's predictions for up to 8 steps into each new segment at the early-stopping checkpoint of 6.25×10^7 training examples. This log-scale plot shows that even at the end of training, the model's ability to restart ICL for new systems slowly degrades when predicting indices 6, 7 and 8 as more new systems are presented in the context window. This is seen as the upward trend in the green and yellow curves in Fig. 16.

Appendix H. Multiple Mechanisms for Prediction

As discussed in Section 1, we explore whether the trained model performs the associative recall task through a **label-based recall (H1)** mechanism or a **observation-based Bayesian recall (H2)** mechanism. If the mechanism for recall was label-based, the model would in-context learn the association of symbolic labels to their corresponding systems, and then perform inference based on recalling the queried system and continuing its evolution. If the mechanism for recall was observation-based and Bayesian, the model would ignore the symbolic labels and instead would use the state observation to figure out which system the observation could have come from. The model then performs Bayesian prediction based on previous observations to make future predictions.

H.1. Out-of-Distribution Inference-Time Experiments

To decide if H1 or H2 is a valid hypothesis for the associative recall mechanism, we conducted three out-of-distribution experiments at inference-time: misdirecting the model towards the incorrect sequence in the haystack (Appendix H.1.1), synchronizing sequences in the haystack from different systems so that they would all have the same state after the final open symbol (Appendix H.1.2), and misdirecting the model towards an unseen sequence not present in the haystack (Appendix H.1.3). Through these three out-of-distribution experiments, we found that neither H1 nor H2 fully describe the model's mechanism for associative recall. In fact, the evidence indicates that model uses H1 for predicting 1-after the final open symbol, and a suboptimal version of H2 for predicting two or more steps after the final open symbol.

To further explore the mechanism for restarting ICL on a new system, we conduct a fourth out-of-distribution experiment: misdirecting the model towards a seen sequence in the haystack (Appendix H.1.4). This experiment's results provide evidence that the model ignores the open symbol when restarting ICL on a new system.

H.1.1. EXPERIMENT 1: MISDIRECTION TOWARDS THE INCORRECT SEQUENCE IN THE HAYSTACK

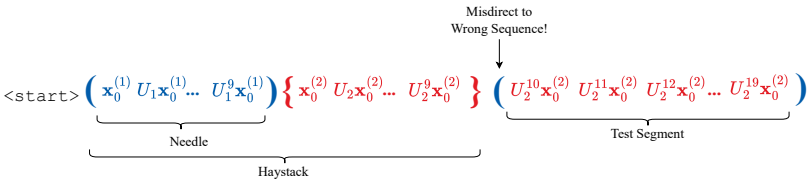
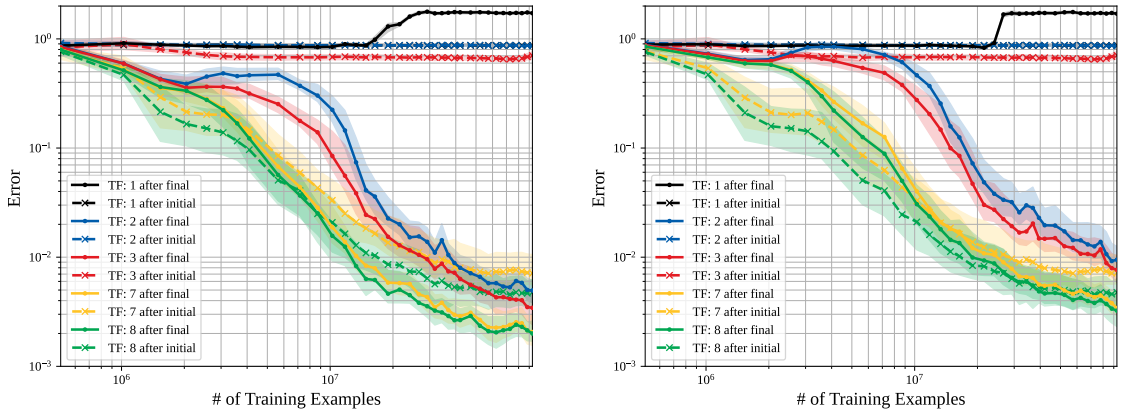


Figure 17: Misdirecting the model towards the incorrect sequence in the haystack.

For this out-of-distribution experiment, we test the model on "needle-in-a-haystack" test traces generated as specified in Appendix B.2, except we swap the final open symbol with another open

symbol that corresponds to a segment in the haystack that is not the "needle" as seen in Fig. 17. If H1 were true, the model would be using label-based recall and we would expect it to make predictions on the test segment for the wrong system. If H2 were true, the swapping of the label would not affect the prediction performance of the model, since the model would be using the seen states to make its predictions rather than the symbolic labels.



(*a*) 2 systems in the haystack.

(*b*) 5 systems in the haystack.

Figure 18: Misdirection towards incorrect sequence — The median squared-error of the model's predictions on the test segment after observing the incorrect final open label vs the number of training examples seen so far. The solid black curve sharply increases in these figures where it would have sharply decreased for a normal test trace as seen in Fig. 13, suggesting H1 is true for predicting the first index of the test segment. In contrast, we find that the model ignores the misdirection when predicting two or more indices into the test segment, since the solid blue, red, yellow, and green curves are almost identical to the corresponding curve in Fig. 13. This suggests that the model is using a Bayesian approach when predicting these indices.

Misdirecting the model towards the incorrect sequence in the haystack falsifies pure label-based recall and provides strong evidence that an observation-based Bayesian recall mechanism is present. The results of this experiment are shown in Fig. 18 for $N = 2$ and $N = 5$, where the median squared-error of the model's predictions are plotted against the number of training examples seen, mirroring the format of the Fig. 13. The first thing to notice in Fig. 18 is that the solid black curve now sharply rises late in training, as opposed to its sharp fall for a normal "needle-in-a-haystack" test trace as shown in Fig. 13. Presumably, the model is using the label-based recall to predict the first index into the test segment.

In contrast, the solid curves in Fig. 18 for 2, 3, 7, and 8 after the final open symbol look almost identical to the corresponding solid curves in Fig. 13. See the blue curve in Fig. 18(a) and 13(b) at 2×10^7 training examples. Their median squared-error both sit at around 2×10^{-2}. The same correspondence can be seen for the red, yellow, and green curves as well. This shows that the model's predictions for 2, 3, 7, and 8 after the final open symbol are not very affected by the open symbol. Instead, they must be using the state observations after the final open symbol to decide which system

to use for making predictions. This strengthens the case that the model uses an observation-based Bayesian recall mechanism for predicting the indices after the first index into the test segment.

H.1.2. EXPERIMENT 2: SYNCHRONIZING "ROTATIONS" IN THE HAYSTACK

In Appendix H.1.1, the misdirection towards the incorrect sequence experiment showed that the model can make accurate predictions for indices 2 and further into the test segment without using the final open symbol. However, in that experiment the observation that is 1 index after the final open symbol can determine which system should be applied for predicting the subsequent indices, since if a predictor has observed $\mathbf{x}_9^{(1)}$ from system 1 and $\mathbf{x}_9^{(2)}$ from system 2, when it observes $\mathbf{x}_{10}$ from either system 1 or system 2, it can check whether $\mathbf{x}_{10} = U_1\mathbf{x}_9^{(1)}$ or $\mathbf{x}_{10} = U_2\mathbf{x}_9^{(2)}$. Now, there is still the question of whether the model can use the final open symbol to predict the later indices in a situation where the 1 after final observation does not provide the necessary information. To answer this question, we conduct a synchronizing rotations experiment, where all of the sequences in the haystack from different systems all have the same state at the 10^{th} index, which corresponds to 1 index after the final open symbol. To do this, we first generate a single vector $x_{10} \sim \mathcal{N}\left(0, \frac{1}{5}I\right)$ for all systems and generate the haystack by "rewinding" our systems back to their initial state x_0 by $x_{i-1} = U^T x_i$ as is shown in Fig. 19. The ambiguity of which system the first observation in the test segment comes from, means that the model must use the symbolic label to make an accurate prediction on the second index into the test segment.

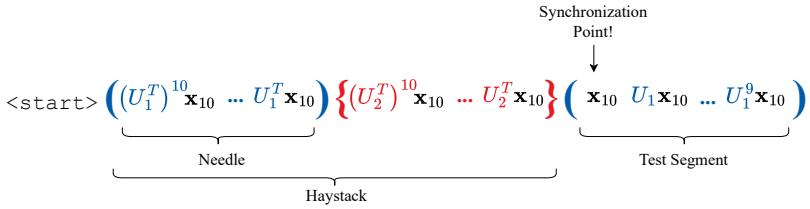
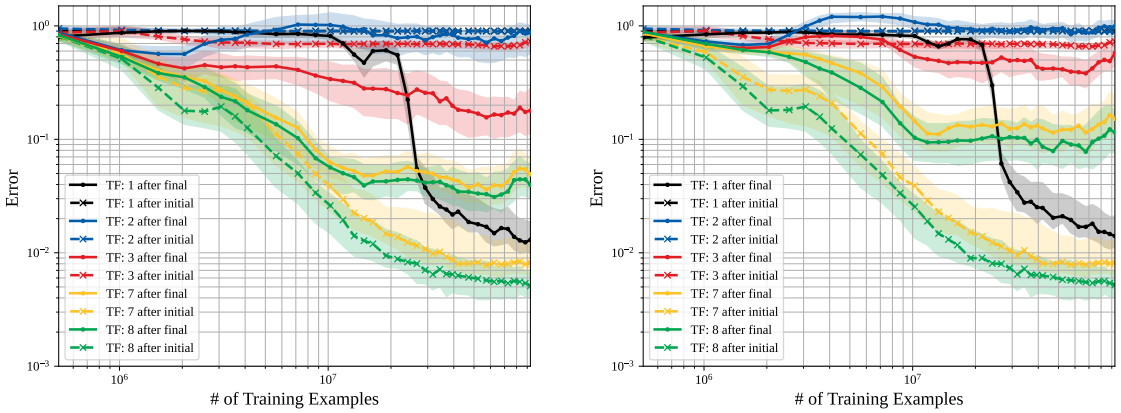


Figure 19: Synchronizing previous systems in the haystack, so the first observation in the test segment no longer reveals the system that is being continued.

Synchronizing the sequences in the haystack so the observation at the first index in the test segment is not informative of the continuing system shows that the model has not learned to use the symbolic label to make accurate predictions for two or more indices into the test segment. In Fig. 20, which plots the median squared-error of the predictions on the synchronizing test traces vs. the number of training examples seen, we see that the solid black curve still sharply decreases late in training[9], as it does in Fig. 13, showing that the model is able to use the final open symbol to predict $\mathbf{x}_{10}$. On the other hand, the solid blue curves are almost horizontal throughout all of training, and the solid red, yellow, and green curves are have a significantly higher squared-error than their counterpart curves in Fig. 13. This means that the model is unable to make accurate predictions on the subsequent indices in the test segment after $\mathbf{x}_{10}$, although the final open symbol provides all of the necessary information to do so. This strengthens the case for H2 being the right hypothesis for predicting the indices after $\mathbf{x}_{10}$, although the model is clearly not a Bayes optimal predictor since

9. Due to the synchronization, the first observation in the test segment is a valid continuation for all systems in the haystack and therefore is always predictable.

seeing $\mathbf{x}_{10}$ and the next observation $U\mathbf{x}_{10}$ would disambiguate which system U is being continued. Despite the system being disambiguated after seeing $U\mathbf{x}_{10}$, the red solid curves in Fig. 20 are still much worse than their counterparts in Fig. 13.



(*a*) 2 systems in the haystack.

(*b*) 5 systems in the haystack.

Figure 20: Synchronizing rotations — The median squared-error of the model's predictions on the test segment when the first index into the test segment is a valid continuation for all haystack segments vs how many training examples have been processed during training. Synchronizing the rotations of the haystack segments means that after the first observation in the test segment, a predictor cannot determine which system is being continued. The solid black curve still shows emergence at the same point in training as it did in Fig. 13, supporting the validity of H1 for predicting the first index into the test segment. Otherwise, the model performance is significantly degraded for the rest of the indices into the test segment. For example, the solid blue curve in these plots for prediction errors on the second index into the test segment is near 1.0 in Fig. 20(a) where it is near 1.0×10^{-2} in Fig. 13(b). This indicates that the model is unable to use the symbolic label well enough to make accurate predictions.

H.1.3. EXPERIMENT 3: MISDIRECTION TOWARDS AN UNSEEN SEQUENCE

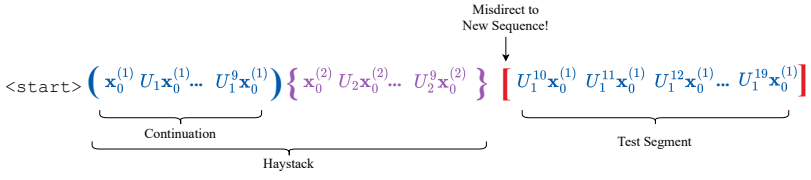
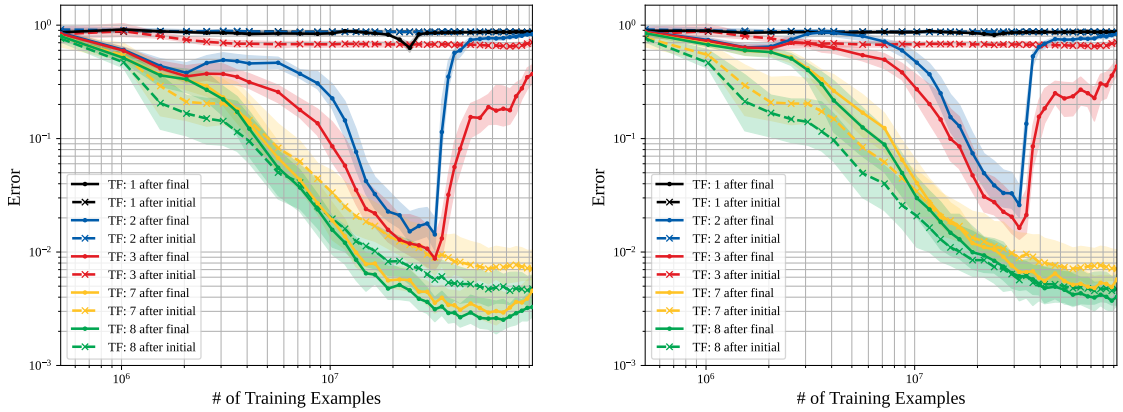


Figure 21: Misdirecting the model with an unseen symbolic label indicating a new sequence.

To further study whether the model uses the symbolic labels at all to continue its predictions on a recalled sequence, we misdirect the model to restart ICL for a sequence that has not appeared in the haystack so far. Fig. 21 illustrates how we swap out the final open symbol of a normal

"needle-in-a-haystack" test trace with an open symbol that does not correspond to any of the systems in the haystack. Given the results from the previous two sections, we expect the model to predict zero for the first index as that is optimal for restarting a new sequence, but we expect the model to make accurate predictions on the subsequent indices that are continuing a segment that is in the haystack. This is almost what happens, but late in training, the model suddenly transitions to (at least partially) restarting ICL on a new sequence.



(*a*) 2 systems in the haystack.

(*b*) 5 systems in the haystack.

Figure 22: Misdirection towards an unseen system — The median squared-error of the model's predictions on the test segment when the final open symbol does not correspond to any haystack systems vs the number of training examples seen so far. The solid blue and red curves match their counterparts in Fig. 13 until $\approx 3 \times 10^7$ training examples, at which point they sharply increase. This means the model continues predicting the existing system in early training then suddenly starts treating it more like an unseen system late in training. We note that this transition happens shortly after the emergence of associative recall, suggesting the model has learned that unseen labels also require ICL to be restarted.

Misdirecting with an unseen symbolic label highlights a new abrupt phase transition in model behavior late in training from disregarding the symbolic label and continuing to predict the observed test segment to restarting ICL for a new system. Fig. 22 shows the median squared-error of the model predictions after being presented with a final open symbol that does not correspond to any system in the haystack while the test segment is actually a continuation of a haystack segment vs how far along the model is in training. In the same figure, the first after final predictions match the expected behavior of restarting predictions as seen by the solid black curve being a horizontal line near 1. For the rest of the indices, the model ignores the symbolic label through the initial stages of training and continues to correctly predict the test segment, since the solid curves in Fig. 22 match their counterparts in Fig. 13 up to around 3×10^7 training examples. Once associative recall fully emerges later in training, and the model learns how to use the symbolic labels, the model transitions to treating a continuation of the old sequence as a brand new sequence corresponding to the new label, since the blue and red solid curves abruptly increase after 3×10^7 training examples in Fig. 22. This shows that the model predictions for two or more indices into the test segment *are* affected by

the final open symbol, although Appendix H.1.2 showed that the model is unable to use it to make accurate predictions for recall on those indices.

H.1.4. EXPERIMENT 4: MISDIRECTION TOWARDS A SEEN SEQUENCE IN THE HAYSTACK

The first three out-of-distribution experiments studied the model's mechanisms for performing associative recall, but this section will study the mechanism for restarting its prediction on a new system. Again, we devise a misdirection experiment. This time we provide a final open symbol that points toward the "needle" in the haystack, but the test segment is a sequence from a system that is not in the haystack (Fig. 23).

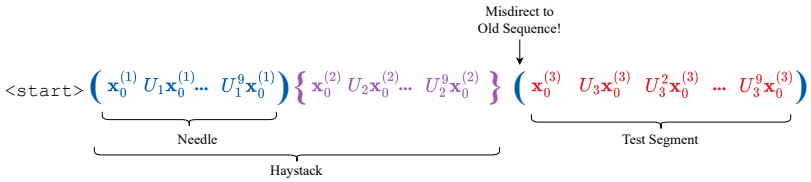
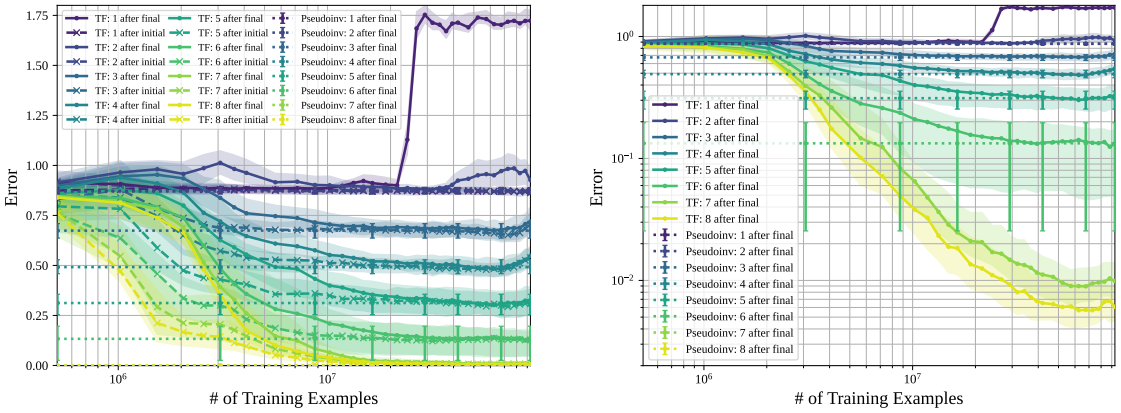


Figure 23: Misdirecting the model with a previously seen symbolic label.



(*a*) Linear scale. (*b*) Log scale.

Figure 24: Misdirection towards a seen system — The median squared-error of the model's predictions on the third segment that is an unseen sequence while the final open symbol corresponds to the first segment vs the number of training examples seen so far. The solid curves other than the 1-after final curve match the solid curves in Fig. 15. This supports the hypothesis that the model uses the state observations to continue its prediction on a newly seen segment, rather than the open symbolic label.

Misdirecting with a previously seen symbolic label does not stop the model from restarting its predictions on the new sequence for two or more indices into the test segment. In Fig. 24, the median squared-error on this misdirection experiment is plotted against the number of training examples seen so far during training. We find that when given the correct unseen symbolic label (Fig. 15(a)), and when shown a symbolic label misdirecting to a sequence that has already been observed in context (Fig. 24), the model performs equivalently on predicting two or more indices

into the newly seen segment, as the solid curves match except for the 1-after final curve that is using the symbolic open label. The model recognizes that the sequence it is seeing does not correspond to a sequence it has already seen. This supports a hypothesis that that model does not use the symbolic labels to restart ICL on a new system. Interestingly, misdirecting the model with a previously unseen symbolic label indicating a new system (Appendix H.1.3) shows that these unseen symbolic labels do affect the model's predictions in the test segment. This evidence shows that the mechanism for restarting ICL may not be purely label-based nor purely observation-based.

H.1.5. SUMMARY OF OUT-OF-DISTRIBUTION EXPERIMENT RESULTS

After conducting the four out-of-distribution experiments, we find that neither H1 nor H2 can fully explain the model's mechanism for predicting interleaved time-series. The misdirection to the incorrect sequence experiment (Appendix H.1.1) supports H1 for predicting the first index into the test segment, but it also shows that the model can make accurately continue its predictions in the test segment without using the final open symbolic label. Therefore, H1 is insufficient. The synchronizing rotations in the haystack experiment (Appendix H.1.2) further showed that the model is unable to use the final open symbolic label well enough to accurately continue its predictions in the test segment, providing strong evidence that the model performs a suboptimal version of H2 for this subtask. This evidence is further strengthened by the misdirection towards a seen sequence in Appendix H.1.4. Finally, the misdirection to an unseen sequence experiment (Appendix H.1.3) showed that even for the later indices, the final open symbolic label can signal the model to restart its predictions for a new segment, invalidating H2 as the sole mechanism for continuing predictions. It is clear from these results that the conjecture C3, where the transformer model uses multiple mechanisms for the single task of interleaved time-series prediction holds true.

H.2. Transformer circuit analysis

Circuit	# Edges	1-After squared-error	2-After squared-error
Orthogonal Sys Full Model	32936	0.002	0.004
Orthogonal Sys 1-after Circuit	200	0.01	0.64
Orthogonal Sys 2-after Circuit	40	0.32	0.02

Table 2: Edge pruning finds sparse transformer circuits with high evaluation accuracy in our GPT2-style model. We prune late checkpoints of a model using interleaved traces and data consisting of 5 systems in the haystack. We report the number of edges in the final circuit and the mean squared-error of the circuits' predictions and the ground truth state observations for both the 1-after and 2-after tasks. We run an edge thresholding binary search to reach the target edge sparsity and set all pruned edges to have weights of 0, then run inference. We visualize the 1-after circuit in Fig. 25.

Now that it is clear the model uses multiple mechanisms for the single task of interleaved time-series prediction, we study if these different mechanisms have different computation graphs in the weights of the learned transformer model. To do this we use Edge Pruning, a transformer circuit-discovery method that optimizes over continuous masks over a disentangled transformer to

find a sparse representation of a task [5]. We run a modified version of Edge Pruning to distinguish the circuits being used by our model for the 1-after and 2-after tasks. As our model is solely trained with squared-error, we remove the KL objective in the original Edge Pruning method's loss function in [5] and optimize on a scaled up squared-error added to the original edge loss. The loss function that we optimize to find a sparse computation graph is $\mathcal{L}' = k \cdot \mathcal{L}_{\text{squared}-\text{error}} + \mathcal{L}_{\text{edge},s}$. Our dataset for the edge pruning method consisted of one "needle-in-a-haystack" trace configuration generated in the same way as in Appendix B.2 for 5 systems in the haystack.[10]

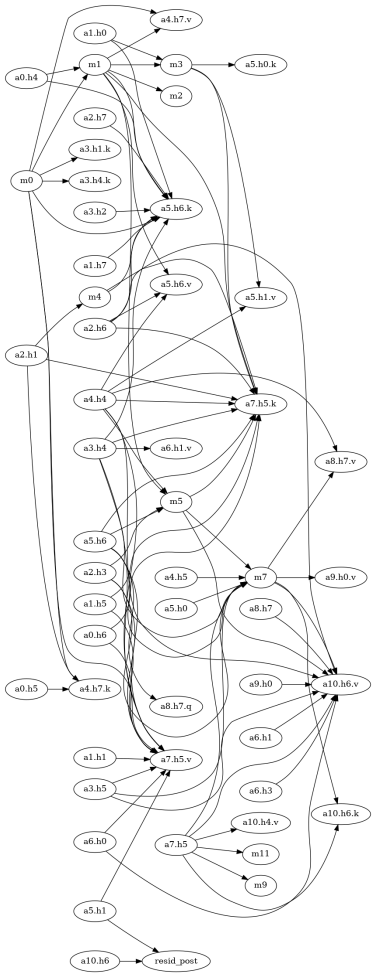


Figure 25: 1-after final open symbol circuit in orthogonal systems. The output of the residual stream is labeled as `resid_post`, MLPs are labeled as `m{layer_num}`, attention heads are labeled as `a{layer_num}.h{head_num}`, and `.q`, `.k`, and `.v` stand for QKV matrices respectively.

10. Further experiments should test more "needle-in-a-haystack" configurations, and would have a larger testing library of traces to create full train and test splits for training and evaluating the pruning method.

We report the size of the circuits and the squared-error of the ground truth with the predictions outputted from both the final checkpoint of the trained model and the pruned circuits in Table 2.[11] Since the pruning method optimizes continuous gates for each node in the computation graph, these continuous gates must be quantized to 0 or 1 to get a pruned circuit.[5] Our results in Table 2 show the accuracy of the pruned model *after* this quantizing has been done, which differs from how [5] reports their results. Post-quantization performance ensures that edges that we believe to be irrelevant truly have no contribution to the output. Importantly, we find high accuracy and *0% edge overlap* between the 1-after and 2-after circuits, indicating that *our model mostly leverages mechanistically different learnt mechanisms for consecutive tokens*.

Recall how optimizing over continuous masks on disentangled transformer nodes leads to a sparse computation graph after the continuous masks are quantized to 0 or 1. To perform this quantization, the quantization threshold must be set. We use binary search to find the smallest threshold (within 1e-5 precision) such that the pruned model's edge sparsity is close to the target edge sparsity of 0.98. Importantly, this does not force a viable path from the start and end of the residual stream, it only captures significant contributions over edges with respect to the loss. In Fig. 25, the pruned circuit is visually presented in a directed computation graph. Nodes in this computation graph are specific QKV matrices of attention heads, entire attention heads, MLPs at a specific layer, or the output of the residual stream. The output of the residual stream is labeled as `resid_post`, MLPs are labeled as `m{layer_num}`, attention heads are labeled as `a{layer_num}.h{head_num}`, and `.q`, `.k`, and `.v` stand for QKV matrices respectively.

11. The trained model that was used for these results is from an earlier training run than the "orthogonal medium" model that is throughout the rest of this paper. This earlier training run was trained on 5×5 orthogonal matrices that were sampled from a non-uniform distribution over all 5×5 orthogonal matrices [30].