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

author names withheld

Under Review for the Workshop on High-dimensional Learning Dynamics, 2025

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, 12, 18, 21, 22, 32–35]. There has also been work that focuses on understanding ICL for simpler toy problems [7–9, 24, 25, 28]. Toys (e.g., linear regression [9, 10, 25]) 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.

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 [9] — 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 [31] 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 [15, 19, 34] 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 [31, 32] and using OLMo checkpoints [20], 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

Consider predicting the continuous-state of an *unknown* linear dynamical system from the orthogonally evolved family [26], where the system $U \in \mathbb{R}^{5\times5}$ is a uniformly drawn-at-random [16] 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

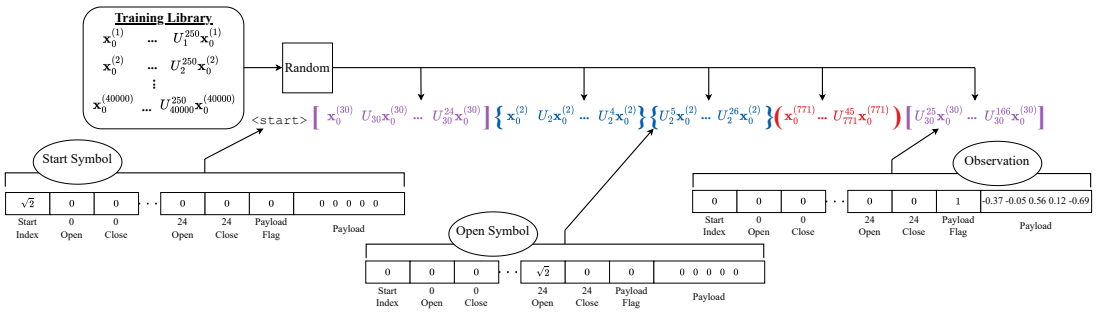


Figure 1: Generating a training example — Notice in this example the continuation from the first segment to the last (system U_{30}), and from the 2nd segment to the 3rd (system U_2). The "parentheses" (symbolic punctuating labels) are encoded as special tokens as shown.

further detail in the Appendix 5.1 and illustrated in Figure 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 5.2.

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 5.3. From a training dynamics point of view, this ability seems to develop steadily during training — no evidence of "emergence."

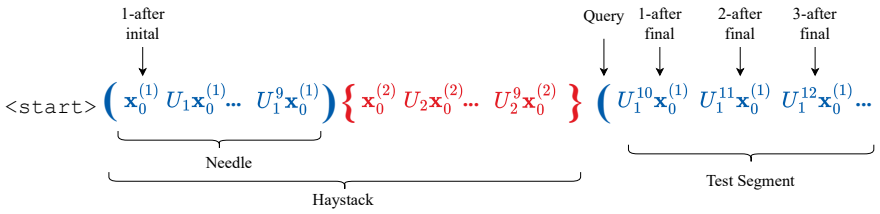
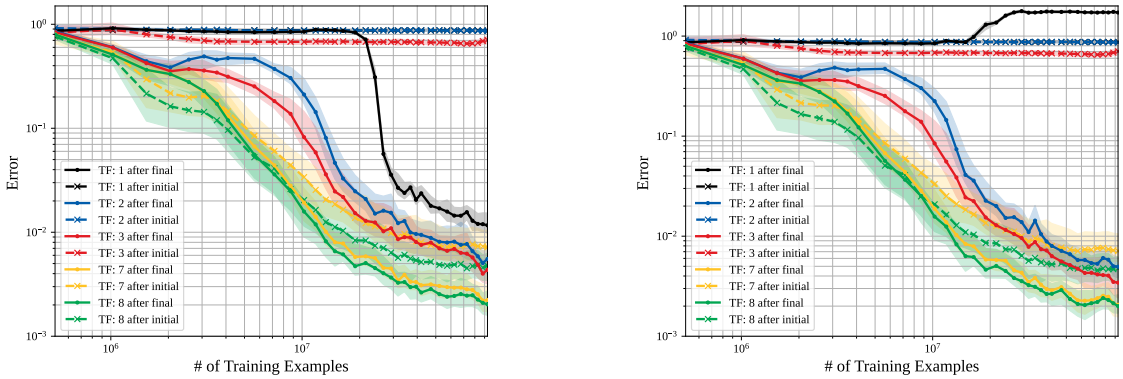


Figure 2: Test format with a two-system haystack and a query to resume the first system.

To study associative-recall, we use structured test traces as depicted in Figure 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 mean-squared error. The key results can be seen with $N = 2$ in Figure 3. (Results for some other N are in fig. 9.) 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 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. 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 Figure 3 with or without the misdirection — the performance on continuing the queried sequence 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 token 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 5.7.) 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 [20] 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, 30]. In Figure 4 (right), we see a similar phase transition in the first token prediction task, a parallel of the 1-after recall dynamics in our toy model. Meanwhile, the second-token performance is both better and more gradual in its improvement across training. This again matches what we saw in our toy problem.[2]

1. We use Spanish instead of the International Phonetic Language (IPA) as IPA has tokens that are not compatible with the OLMo 2 tokenizer. We also change the in-context labels to have no semantic meaning in light of [31].
2. One natural question is whether what we are seeing 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

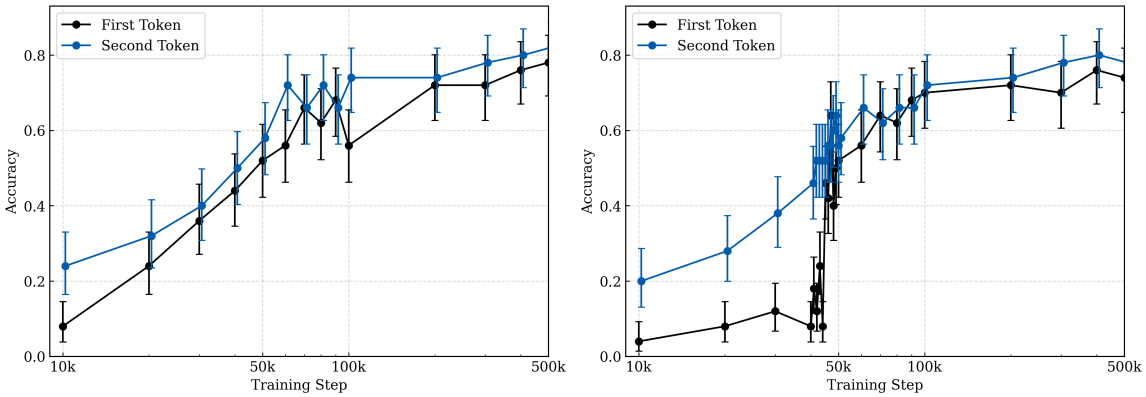


Figure 4: Comparative example of in-weights associative recall (left) and in-context associative recall (right) in a 2-shot prompting setting. Each point is a separate OLMo-2 7B model at different training steps. We report 95% confidence intervals using a Jeffrey's prior.

4. Discussion

At this point, the community understands that ICL is rich and nuanced [11, 12, 18, 23, 29]. 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 no reason to suspect it otherwise. And because the toy is simple, we have hope that it will help the community understand the why and how more deeply 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 [34] means that this 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 grown/improved) even if it can't solve the task completely. The very success of this mechanism during training will further reduce the overall gradient pressure for alternative and potentially better mechanisms, potentially forcing them to develop more slowly. However, structurally, there are certain aspects of tasks that are likely to require the use of these 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 [28].

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 during training that occurs around step 50k.

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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

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

5. Appendix

5.1. Training Details

Generating a library of training sequences: We first compile a training library by generating 40000 orthogonal matrices iid uniformly over all 5×5 orthogonal matrices $U_1, \ldots, U_{40000} \overset{iid}{\sim} \mu(\mathrm{O}(5))$ and generating 40000 iid initial states that will correspond to each training system $\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(40000)} \overset{iid}{\sim} \mathcal{N}\left(0, \frac{1}{5}I\right)$. We then roll out the states to get observation sequences that are each 251 entries long, and compile the sequences as our training library as depicted in figure 1.

Cutting and interleaving training sequences To form a training example, we interleave segments of observation sequences from the library into a context window of length 251. The max number of systems that could be inserted into this training example is drawn according to a $\mathrm{Zipf}(1.5, 25)$ depicted graphically[3] in Fig. 5. Fig. 1 shows the format of a training example. The first token of a training example is always a start symbol encoded as a one-hot vector. The number of cuts is then drawn according to a Poisson distribution (with parameter twice the number of systems), and this many cuts are placed uniformly at random within the 250 long context window (excluding the start symbol).

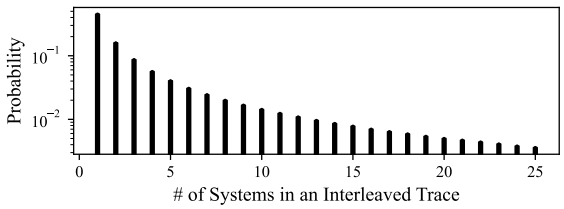


Figure 5: The $\mathrm{Zipf}(1.5, 25)$ used to generate the number of systems to be interleaved in a training trace.

Symbolic punctuating labels (SPLs) At every cut, a symbolic close label is inserted that denotes the end of the previous segment of observations. This is immediately followed by a symbolic open label that denotes the start of the next segment of observations. Each distinct system in a training example has its own randomly assigned symbolic open and close labels. Within a single training example, segments of a particular system, say System 30, always start with the same open token and always end with the 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*. The symbolic punctuating open and close labels are also one-hot encoded vectors.

Input structure and embedding The input dimension of our models is 57. There are 50 dimensions for encoding 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. 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.

3. The Zipf distribution was chosen for its ubiquity in nature [3] and natural language [27], along with recent work pointing to its importance in modern neural networks [17].

Model and embedding Building off of the codebase in [7], which was influenced by [9], we train a 9.1M parameter GPT-2 style transformer to perform this task. Our model has hidden dimension 128, 12 layers, and 8 heads. Our model's input embedding is 128×57 dimensional, since our model has a 128-dimensional hidden dimension. The model's output embedding is 5×128 to ensure that the model makes 5-dimensional predictions. The input and output embeddings are untied.

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. 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 [7], 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 [14] and trained using mean-squared error loss.

5.2. Pretraining loss dynamics

In figure 6, 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 section 5.1. The black curve is the model's performance on freshly drawn interleavings of traces from a held-out test library, while the blue curve is on freshly drawn interleavings of traces from the training library. The red dotted line gives a lower bound baseline for the prediction error of the model by computing the average MSE of a predictor that computes $\hat{\mathbf{x}}_{i+1} = \hat{U}\mathbf{x}_i$, where

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

and $X^\dagger$ denotes the Moore-Penrose pseudoinverse of X. This perfect baseline also correctly unbraids the interleaved system so it knows exactly where it is in which sequence. 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.

Figure 6 shows us that when prediction error is averaged over all timesteps, it appears that the model is making steady gradual progress during training towards learning to predict interleaved traces. On these training-style loss curves, there is a potential kink visible around $4 \sim 5 \times 10^6$ training examples seen. But this could also just be seen as approaching the irreducible loss as given by the dashed red curve at around 0.04 MSE loss.

To actually see what is going on during training, we have to look at the model's behavior with more careful probes. Our study of the model's performance on specific indices using structured needle-in-a-haystack test traces in Section 2 uncovers multiple mechanisms for prediction with distinct learning dynamics.

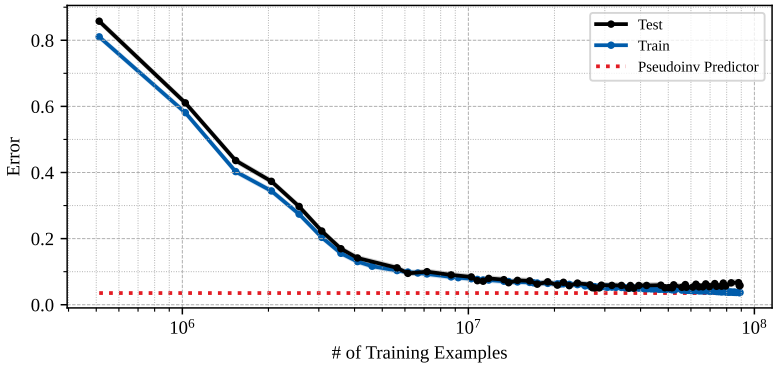
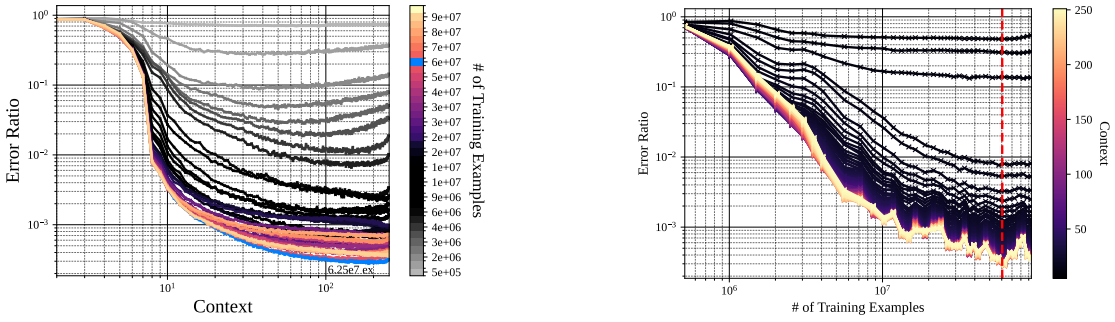


Figure 6: Pretraining Loss—The MSE of the transformer model's predictions on traces interleaved in the style of the training data averaged over each timestep of the trace. The error bars are the standard deviation. For both the train and test data, averaging was done over 40,000 different interleaved traces, each of length 251. The dotted red line is the averaged MSE of a predictor that computes an estimate of the underlying system dynamics by using the Moore-Penrose pseudoinverse of the observed data.

5.3. Can a model learn the dynamics in context? Yes

Before testing the recall performance, we first confirm that our trained model is able to learn to predict long sequences from unseen systems in-context. We generate 100 held-out systems and 1000 different held-out initial states for each system as our test set. We plot the median MSE over these initializations and test systems. Figure 7 shows the performance when predicting 249 entry-long sequences.



(*a*) (*b*)

Figure 7: Performance on the first segment—Fig. 7 (left) and fig. 7 (right) depict the in-context learning performance of the model on 248 entry long test examples. The red line in fig. 7 (right) at 6.5×10^7 training examples denotes the checkpoint that we use for early stopping.

Notice in figure 7 (left) that early checkpoints saturate out, and cannot continue to make better predictions with more context. In figure 7 (right) the model is gradually learning to make better predictions as training continues and emergence is not present. The best performance of the model is at the end of the context window with an MSE of $\approx 2 \times 10^{-4}$. According to [13], this is near the precision threshold for transformer models. Additionally, we notice in figure 7 (right) that the model suffers from overfitting late in training. We use these baseline experiments to set our early stopping checkpoint of 6.5×10^7 training examples, as denoted by the red vertical line in figure 7 (right) which corresponds to the blue curve in figure 7 (left).

5.4. Learning to restarting predictions on a new system

In figure 8, we see that at the beginning of training, the model has not learned to restart its predictions for a new system when being told that a new system is starting. Its MSE in segment 3 is at least 0.05 above its counterpart predictions in segment 1 for 2 through 8 steps into each segment at 2×10^6 training examples. The model then gradually learns to reset and begin predicting a new system as the MSE for each step in segment 3 converges to the value of its segment 1 counterpart.

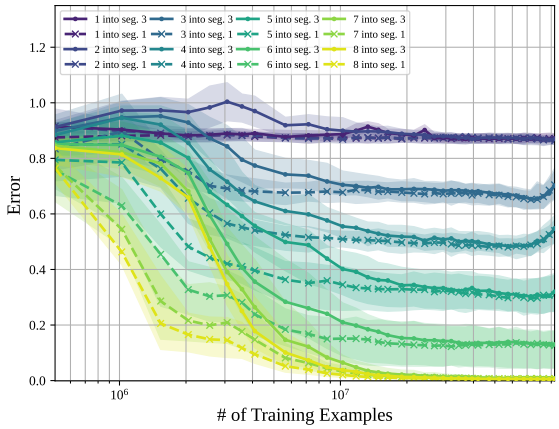


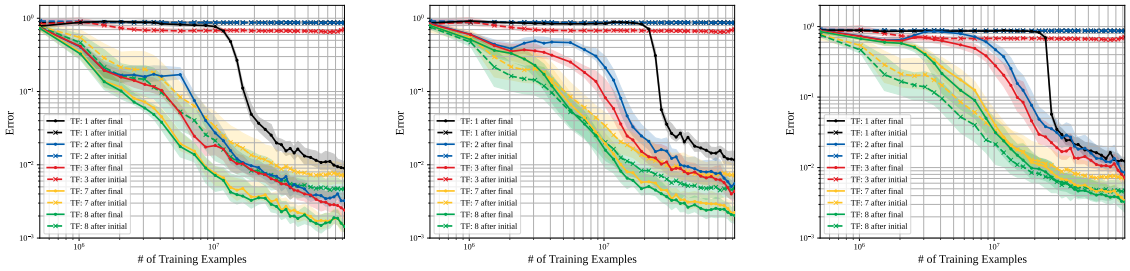
Figure 8: Performance on a new subsequent segment—The above plot shows the MSE of the orthogonal model on the 1st system segment and the 3rd system segment that it has seen in its context. For all three of these segments, a sequence from their respective systems appeared for the first time.

5.5. Further Plots

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 Figure 9. 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.



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

Figure 9: 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.

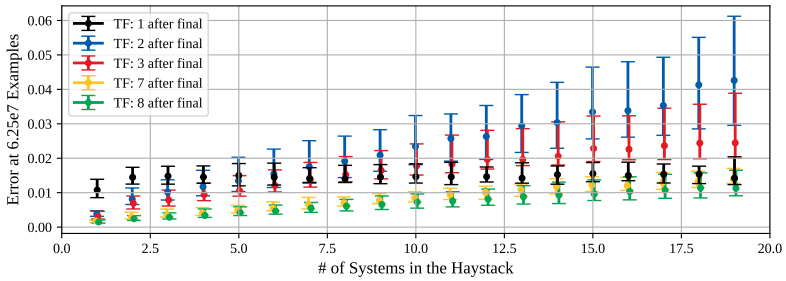


Figure 10: The 25th, 50th, and 75th quartiles of the MSE after 6.25×10^7 training examples as the number of systems in the haystack increases.

In Figure 10, 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.

5.6. Multiple Mechanisms for Prediction

Testing the hypotheses presented in section 1, we explore whether the mechanisms by which the associative recall task is performed is mediated by **label-based recall** or **observation-based Bayesian recall**. Under label-based recall, the model uses in-context learning of the association of symbolic labels to systems, and then performs inference based on recalling the queried system and continuing its evolution. Under observation-based Bayesian recall, the model ignores the symbolic-labels and instead leverages the seen payload, to figure out which system it could have come from. The model then performs Bayesian prediction based on previous observations to make future predictions.

5.6.1. FURTHER OUT-OF-DISTRIBUTION INFERENCE-TIME EXPERIMENTS

In order to better understand the mechanisms governing how our model is performing associative recall, we conducted four out of distribution experiments at inference-time.

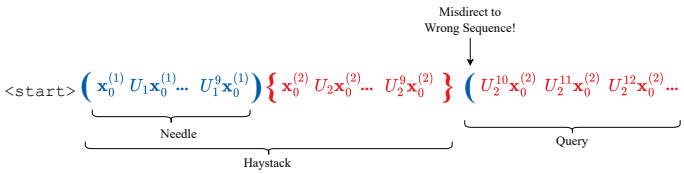
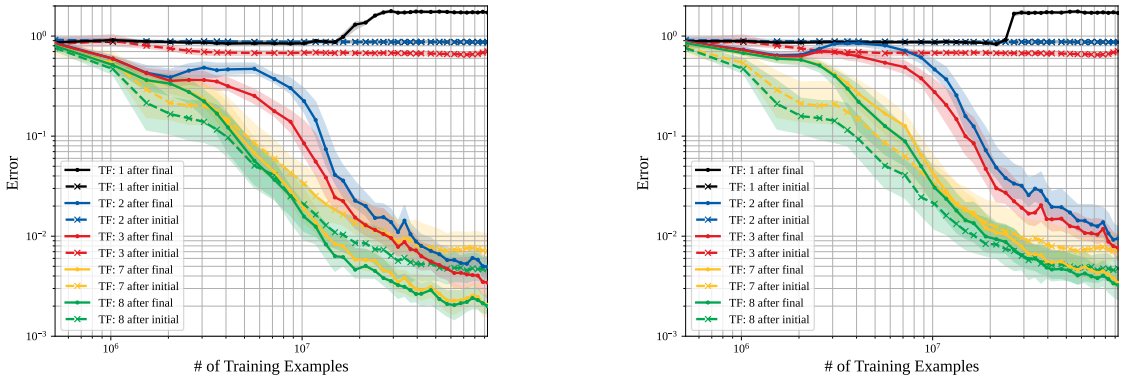


Figure 11: Misdirecting the model towards the incorrect sequence in the haystack.

Misdirecting the model towards the incorrect sequence in the haystack falsifies pure label-based recall and provides strong evidence for observation-based Bayesian recall. In order to test the label-based recall hypothesis, we provide the model with the incorrect symbolic label as depicted in Figure 11. The goal of this experiment was to test if the model would apply the recalled system to the observed payload or leverage the observation to perform Bayesian prediction. Our results show that the model ignores the misdirection and leverages the observation to perform prediction. The baseline performance without misdirection for the 2+ after tasks shown in Figure 12 matches exactly what we see with misdirection in Figure 3. This is strong evidence that the model applies a Bayesian approach for the 2+ after tasks when being shown a symbolic label it has seen in context.



(*a*) 2 systems in the haystack.

(*b*) 5 systems in the haystack.

Figure 12: Misdirection towards incorrect sequence — we find that for the 2+ after final payloads, the model ignores the misdirection. This suggests that the model leverages a Bayesian approach when performing the 2+ after tasks.

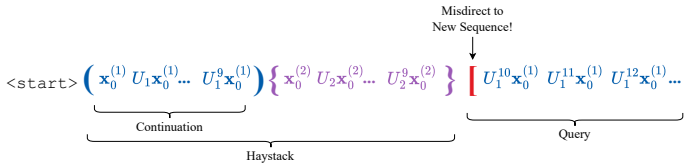
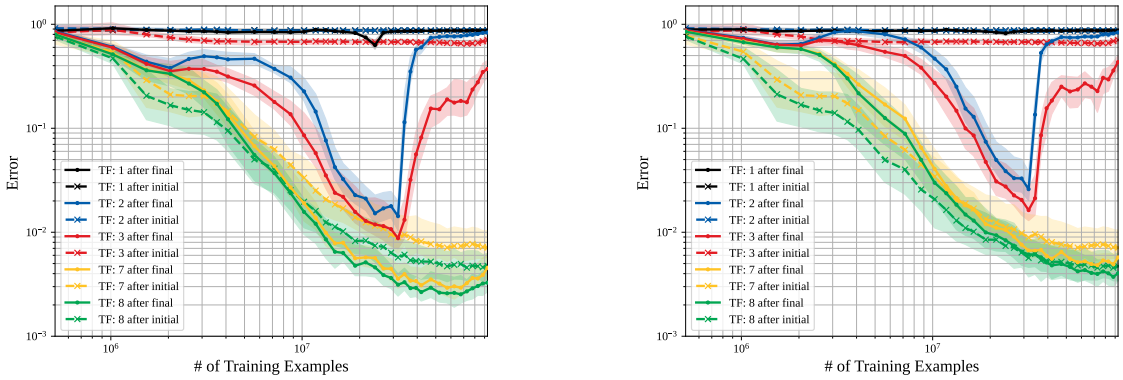


Figure 13: Misdirecting the model with an unseen symbolic label.

Misdirecting with an unseen symbolic label highlights a phase transition in model behavior: mediating between observation-based disambiguation and label-based disambiguation. We attempt to misdirect the model towards applying a new symbolic label to the continuation of a sequence it has already seen (Figure 13). Through the initial stages of training, the model ignores the symbolic label and continues the sequence it has already seen as shown in Figure 14. After associative recall emerges later in training, and the model learns how to leverage the labels, it moves towards treating a continuation of the old sequence as a brand new sequence corresponding to the new label. This suggests that the model performs unmediated observation-based Bayesian recall through the initial part of training, but after emergence, leverages the unseen label to treat the new observations as a new sequence.



(*a*) 2 system in the haystack.

(*b*) 5 systems in the haystack.

Figure 14: Misdirection towards unseen system—When trying to misdirect the model towards an unseen system, the model experiences a phase transition where it moves from using payload to mediate its generation to using the symbolic label to mediate its generation. This is seen in the model initially performing well on continuing the existing system to eventually treating it as an unseen system. We note that this transition happens shortly after the emergence of associative recall suggesting the model learns the meaning behind the labels.

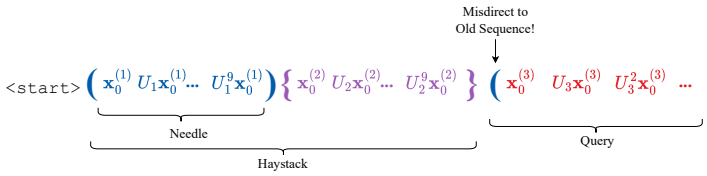
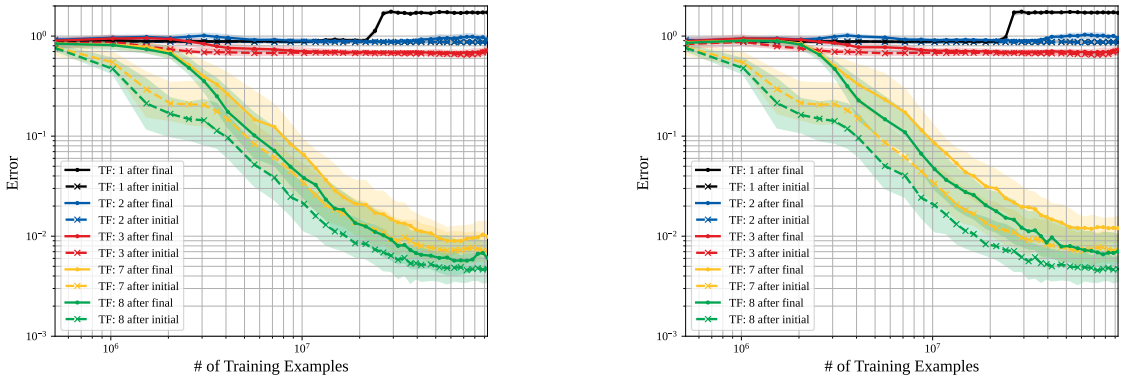


Figure 15: Misdirecting the model with a previously seen symbolic label.

Misdirecting with a previous symbolic label displays the model's Bayesian tendencies even with label interference. We finally attempt to misdirect the model towards treating a brand new sequence as an old one by providing a symbolic label it has seen in its context (Figure 15). We find that when controlling for the position in the haystack (sequences introduced later in context perform worse early in training) *the 2+ after final tasks perform equivalently*: when given the correct new symbolic label (Figure 8), and when shown a symbolic label misdirecting to a sequence that has already been observed in context (Figure 16). This supports the idea that the model is label-ignoring Bayesian. It recognizes that the sequence it is seeing does not correspond to a sequence it has already seen. However, the point in training where we see the model start leveraging the label in the unseen symbolic label experiment (Figure 13) is approximately when performance for 2 after final in this setup starts to degrade, suggesting the model may be starting to try to leverage the symbolic label.



(*a*) 2 system in the haystack.

(*b*) 5 systems in the haystack.

Figure 16: Misdirection towards a seen system—We attempt to misdirect the model by providing a symbolic label for a system seen in context while feeding the model a new system. We see the model performs Bayesian inference on the system and matches the performance when correctly prompted with a new symbolic label for a new sequence for the third position in the haystack (Figure 8). We note that at the same point in training that the model begins to leverage the symbolic token for the misdirection towards an unseen system (Figure 14) the model starts to perform worse on the 2-after task in this setting.

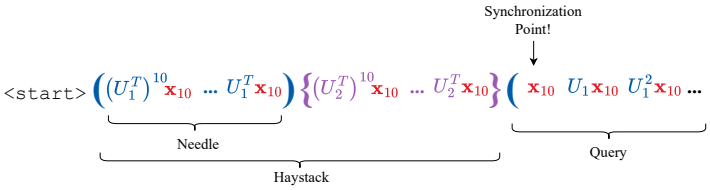
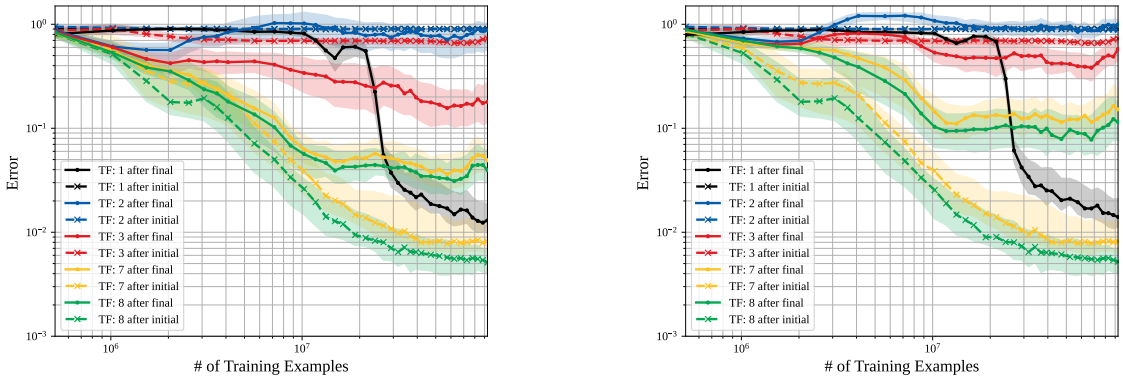


Figure 17: Synchronizing previous systems in the haystack to require symbol-based disambiguation.

Synchronizing previous systems in the haystack to necessitate symbol-based disambiguation displays that the model does not exclusively leverage a Bayesian approach. We construct an experiment where two different systems have equivalent payloads on the one-after observation, effectively making the 2 after task ambiguous if the models leverage Bayes. To do this we generate a single payload at $x_{10} \sim \mathcal{N}\left(0, \frac{1}{5}I\right)$ for all systems and generate the haystack by "rewinding" our systems back to x_0 by $x_{i-1} = U^T x_i$ as is shown in Figure 17. In this case given the ambiguity of which system x_{10} (the 1-after observation) corresponds to, the model must leverage the symbolic label to disambiguate the sequence. We find that not only does the model fail to do the 2-after task, *but in fact the model continues to struggle even through 8-after* where it under performs the 8 after initial baseline as shown in Figure 18. This indicates that the model is not strictly performing Bayesian prediction either, as after seeing multiple examples following the symbolic label, it should be able to disambiguate between the systems.



(*a*) 2 systems in the haystack.

(*b*) 5 systems in the haystack.

Figure 18: Synchronizing rotations—Synchronizing the rotations of the two systems makes the 2-after task ambiguous without the symbolic label. We observe the model performance is completely broken for our two after task indicating it is unable to utilize the symbolic label. Furthermore, we find that this degradation in performance persists with 8 after final being significantly worse than 8 after initial.

5.7. Transformer Circuit Analysis

Edge Pruning is 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 One-after and Two-after tasks. As our model is solely trained with MSE on payloads, we remove the KL objective and optimize on a scaled up MSE added to the original edge loss. The loss is $\mathcal{L}' = k \cdot \mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{edge},s}$

We report the size of the circuits and the MSE of the ground truth with the predictions outputted from both the final checkpoint of the trained model and the pruned circuits in Table 1. Importantly, we find high accuracy and *0% edge overlap* between the 1-after and 2-after circuits, indicating that *our model leverages mechanistically different learned mechanisms for consecutive tokens*.

Circuit	# Edges	One After MSE	Two After MSE
Orthogonal Sys Full Model	32936	0.002	0.004
Orthogonal Sys One-after Circuit	200	0.008	0.73
Orthogonal Sys Two-after Circuit	40	0.35	0.02

Table 1: Edge pruning finds sparse transformer circuits with high evaluation accuracy in our GPT2 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 MSE of the circuits' predictions and the ground truth payloads for both the One-after and Two-after tasks. We visualize the One-after circuit in Figure 19.

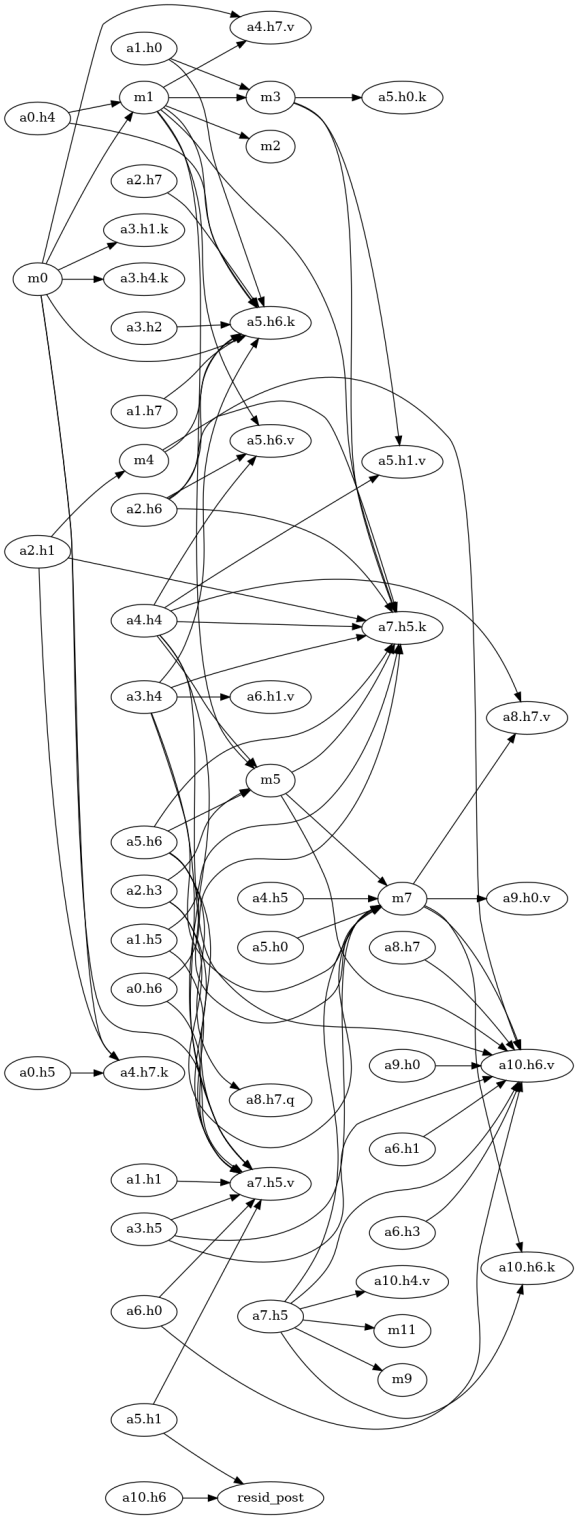


Figure 19: One-after circuit in orthogonal systems

6. Limitations

Our core problem is very much a toy, but that's the point. It enabled us to discover the phenomenon of multi-mechanism ICL and the fact that these different mechanisms have different training dynamics. However, the model we used (based on the GPT-2 architecture) is outdated in multiple key ways, especially the lack of gating-style multiplicative nonlinearities and the lack of a modern PE like RoPE. This limitation is partially mitigated by the use of OLMo-2 in Section 3, since that is a modern dense architecture and manifests the same phenomenon.

A fuller scientific treatment would have ablated many of the architectural components, but we lacked the compute budget. Similarly, and for the same reason, the circuit discovery experiments were not carefully ablated or tuned to look at the tradeoff curve of the performance with the size of the circuit.

More NLP experiments could have been run — we just ran and reported one — to see how widely applicable is our conjecture of multiple simultaneous ICL mechanisms for individual tasks.