In Context-Learning and Finite Automata

Keshab Agarwal*, Evan Davis*, Yuri Lee*, Trenton O'Bannon**

November 2025

Abstract

This research investigates the capability of neural sequence models to perform in-context learning (ICL) of deterministic finite automata using state-action demonstration trajectories, taking inspiration from literature in the computational social sciences. Models are provided with alternating sequences of states and actions as prompt demonstrations, followed by a query sequence where the model must predict the next states. We evaluate Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and transformer variants, comparing them in predictive accuracy and computational efficiency. Our experiments reveal that standard RNNs largely fail to learn the underlying automata, whereas LSTMs succeed, but require substantial parameter counts to achieve high predictive accuracy. In contrast, transformers demonstrate superior performance with significantly fewer parameters, but we find that a progressive curriculum learning strategy is essential since direct training proved ineffective. These findings underscore the parameter efficiency of transformers for ICL tasks while highlighting the important role of training methodology in mastering structured sequence prediction.

1 Introduction

Generative artificial intelligence, and computer science in general, are set to potentially revolutionize the computational social sciences. Economics is no different; tools from theoretical computer science, state of the art deep neural networks, and reinforcement learning have made tasks such as agent-based modeling, using large language models to simulate human behavior, and formalizing notions of complexity tractable.

Complexity, which refers to cognitive and procedural costs individuals face when implementing or following rules, algorithms, or procedures, is a central concept in behavioral economics. One novel approach to formalizing complexity in decision theory is representing instructions as finite-state machines, or finite automata[1] (Oprea, 2020). Formal characteristics of the automata can be used to rigorously measure complexity. Human experimental

*All authors did equal work on this paper; ordering is alphabetical by last name. All were undergraduate or Master's students at UC Berkeley studying Computer Science.

[1]As well as more powerful models of computation, such as deterministic pushdown automata or Turing machines.

subjects can then be tasked with implementing instructions of varying levels of complexity, allowing researchers to empirically measure how complexity affects human behavior. Furthermore, economists have displayed an interest in using Large Language Models (LLMs) to simulate human behavior (Horton, 2023).

Given this interest in using artificial intelligence's cognitive capabilities as a proxy for humans, we draw inspiration from Oprea (2020)'s formalization of human instructions via finite automata to study neural networks' abilities to learn automata.

We tackle this question by evaluating the ability of small neural network architectures to infer, manage state, and perform computations of finite automata in-context. We feed Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), and 2-layer transformer models input-output pairs corresponding to the actions and state transitions that occurred in an automaton and evaluate its ability to output the state transitions from the input-output sequences in context. This is a little different from explicitly giving the model an embedding of the rules of the automaton itself and seeing if it can interpret it, but this still tests how they can infer and apply rules.

We find that LSTMs are better able to learn more complex automata, while transformers have to engage in curriculum learning by training on easier automata before evaluating more complex ones. However, this tradeoff comes at the cost of increased parameter complexity for the models.

This paper is structured as follows: the first section is the introduction. The second section covers inspirational and related literature from both deep neural networks and computational economics research. The third section discusses formal characteristics of baseline architectures and finite automata. The fourth summarizes the data, and the fifth discusses specific architectural design considerations and our results. Concluding statements are made in the sixth and final section.

2 Literature Review

2.1 Motivational Context

In this paper, we are inspired by work experimentally measuring the procedural complexity of tasks formalized as algorithms, as well as novel computational social science literature using large language models as proxies for humans.

Oprea (2020) utilizes finite automata and deterministic pushdown automata to model instructions, and tasks experimental subjects with correctly implementing instructions of varying complexities. He utilizes different measures of complexity[2] to track how much profit opportunity subjects were willing to forgo in order to avoid implementing instructions of varying complexity, formalizing the cognitive costs of complexity in this manner. To measure the cognitive costs of complexity monetarily, subjects are tasked with a choice of implement-

[2]In particular, the hypotheses he tested were as follows. Considering rules representable as finite or deterministic pushdown automata (a) ceterus paribus, rule A is more complicated than rule B if it is representable with fewer states (b) ceterus paribus, rule A is more complicated than rule B if it is representable with fewer transitions in excess of states (c) ceterus paribus, rule A is more complicated than rule B if rule B contains an absorption state, and A does not.

ing a more complex algorithm for more money, or a less complex algorithm for less money. He finds that (a) both transitions and states are strong predictors of complexity, (b) that absorption states reduce complexity, (c) that subjects fail to reduce rules with redundant components to algorithmically equivalent, simpler ones[3], (d) that familiarity with a learned rule[4] reduces complexity, (e) that rules represented as pushdown automata[5] are significantly less complex, and (f) that having to reason through a rule is much more complex than simply having to implement it.

Horton (2023) utilizes large language models as a model of an economic actor he calls *homo silicus*. He finds that LLMs, when placed in classic behavioral experiments, make choices that closely resemble human decisions. This further motivates the view (emphasized in computational social science literature) that large language models may serve as proxies for human subjects and highlights the possibility of using their behavior to simulate human reasoning under economic constraints. More broadly, such findings suggest that studying LLMs can reveal insights about the structure and limits of human cognition, and vice versa.

Naturally, the question arises regarding to what extent neural networks and LLMs treat complexity similarly to humans. While they may not experience the same disutility of cognitive processing that humans subjectively do, this does not rule out the possibility of structural complexity traits making learning itself difficult for both artificial intelligence and humans. Given the use of automata for modeling procedures in the social sciences, this motivates an investigation into how well sequential models and transformers can in-context learn finite automata.

2.2 Related Literature

Here we review some relevant literature on transformers' and other state-space models' abilities to perform in-context learning (ICL). Garg, Tsipras et al. (2022) show that standard transformers can be trained from scratch to in-context learn various function classes, including linear functions, sparse linear functions, neural networks, and decision trees, by conditioning on input-output examples without parameter updates. Their work serves as a useful proof of concept for transformers' ability to learn complex function approximation encodings through pure ICL.

In terms of generalization capabilities, Fan et al. (2023) demonstrate that transformers can acquire meta skills, which are high-level compositional abilities enabling generalization to unseen tasks. Through extensive in-context learning experiments on function composition, the authors find that models trained with limited composite examples generalize from sums of two basis functions to five unseen ones. The results indicate that transformers learn these meta-skills in a sample-efficient and unsupervised manner and suggest that their

[3]In the context of Oprea (2020), rule A is *reducible* to another rule B if A and B recognize the same language, but A is otherwise more complex.

[4]A learned rule is one implemented multiple times by a subject. On repeat implementations, subjects are familiar with the rule, thus having learned it.

[5]Representing a rule as a deterministic pushdown automaton (a finite automaton with a "stack" for tracking memory) rather than a simple finite automaton provides subjects with a smaller of conditional instructions, and instead asks them to count a sequence of inputs, in effect replacing states with working memory.

weak-to-strong generalization capabilities emerge from compositional reasoning rather than memorization, a phenomenon we hope to replicate in this work.

Furthermore, Goddard et al. (2025) study transformers' abilities to generalize across tasks outside the sample space of training tasks (the tasks in particular being linear functions), finding that increasing the diversity of the training tasks helps with generalization. When task space is represented as an n-dimensional hypersphere with a particular radius, training data are samples from a ϕ-degree "slice" of this space, and test data are drawn fro a disjoint slice of the hypersphere. If training data is drawn from a sample slice of $\phi > 120°$ degrees, the transformer switches from learning specialized linear functions to being able to generalize across the entire sample space and even outside of it (tasks drawn from a larger-radius hypersphere). In this paper, we pursue an analogous strategy through two finite-automaton configurations, denoted $3S3A$ and $5S5A$, each with varying structural complexity. This approach to finite automata is similarly intended to ensure that transformers are capable of generalization, a result supported by their reliance on curriculum learning.

Our primary inspiration is Akyürek et al. (2024). They study how decoder-only transformers can meta-learn formal languages generated by random finite automata by conditioning on example input sequences and generating new valid sequences directly in-context without parameter updates. It demonstrates that transformers develop specialized attention mechanisms, such as n-gram heads, to capture underlying rules from few examples, enabling strong in-context generalization on complex regular language tasks. This work relates to our study on in-context learning of deterministic Moore machines by providing a transformer framework and empirical evidence that such models can infer and generalize state-based sequence-to-sequence mappings, key for learning the input-output behavior of Moore machines within prompts.

While in contrast to Akyürek et al. (2024) we are not studying regular languages but instead the transition functions of finite automata, their baseline architectures serve as primary inspiration for ours.

Further work exists approaching finite automata from an ICL or mechanistic interpretability perspective. Liu et al. (2023) demonstrate that transformers can simulate finite automata by learning shallow shortcuts that replace recurrent transitions. Through formal analysis based on the Krohn-Rhodes decompostion, the authors prove that automata can be represented by constant or logarithmic-depth transformer layers, and experiments on synthetic tasks confirm theoretical predictions. However, results also reveal a trade-off between computational efficiency and the robustness of such parallelized architectures, due to their lack of explicit recurrence.

Adriaensen and Maene (2024) investigate how transformers trained on regular languages learn and represent finite automata from a mechanistic interpretability perspective. Using an extension of the classical L^* algorithm, the authors extract Moore machines from trained transformers and show that one-layer transformers can learn and generalize certain regular languages well, especially when the automaton's state is determined by a finite number of input symbols. Since we are not using L^*, we restrict our analysis to 2-layer transformers.

3 Formal Models

3.1 Architectures

In this section, we briefly overview the base formal architectures we trained at a high-level from a mathematical perspective. Specific implementation details are covered in Section 5.

Recurrent Neural Networks

We utilize two primary architectures: standard recurrent neural networks (RNNs) in the style of Elman networks (Jordan, 1986; Elman, 1990), and Long Short Term Memory (LSTM) via Hochreiter and Schmidhuber (1997).

Given an input sequence of embeddings $(x_1, \ldots, x_T)$ with $x_t \in \mathbb{R}^{d_{\text{model}}}$, a standard recurrent neural network (referred to interchangeably as a vanilla or Elman RNN) updates a hidden state $h_t \in \mathbb{R}^{d_h}$ according to

$$h_t = \phi(W_{hx}x_t + W_{hh}h_{t-1} + b_h)\,,$$

where ϕ is a pointwise nonlinearity (we use tanh), and h_0 is initialized to zero. A linear readout layer produces token-wise logits

$$z_t = W_{\text{out}}h_t + b_{\text{out}},$$

which are trained with cross-entropy loss. The recurrence forces the model to compress all past information into h_t, making RNNs a natural baseline for stateful sequence modeling tasks.

The LSTM augments the hidden state h_t with a cell state c_t that enables longer-range memory. Given input embedding x_t, the LSTM computes four gates:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f),\\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i),\\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o),\\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c),
\end{aligned}$$

where f_t is the forget gate, i_t the input gate, o_t the output gate, and $\tilde{c}_t$ the candidate cell update. The cell and hidden states evolve as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \qquad h_t = o_t \odot \tanh(c_t),$$

with elementwise multiplication $\odot$. As with RNNs, a linear layer maps h_t to logits:

$$z_t = W_{\text{out}}h_t + b_{\text{out}}.$$

LSTMs typically handle longer dependencies than vanilla RNNs, making them a stronger baseline for in-context learning of stateful automata.

Transformers

We utilize variations of Vaswani et al. (2017)'s original transformer model, with 2 and 3 transformer blocks instead of 6. Let $X^{(\ell)} \in \mathbb{R}^{T \times d_{\text{model}}}$ be the input to layer ℓ. We use pre-layer normalization, multi-head self-attention with a causal mask M, and a position-wise feedforward network.

$$\tilde{X}^{(\ell)} = \text{LayerNorm}_1\big(X^{(\ell)}\big)$$

$$Q_h = \tilde{X}^{(\ell)} W_Q^{(h)}, \quad K_h = \tilde{X}^{(\ell)} W_K^{(h)}, \quad V_h = \tilde{X}^{(\ell)} W_V^{(h)} \quad \text{for } h = 1, \ldots, H$$

$$\text{Attention}_h(\tilde{X}^{(\ell)}) = \text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}} + M\right) V_h$$

$$\text{MultiHeadAttention}(\tilde{X}^{(\ell)}) = \Big[\text{Attention}_1(\tilde{X}^{(\ell)}); \ldots; \text{Attention}_H(\tilde{X}^{(\ell)})\Big] W_O$$

$$X_2^{(\ell)} = X^{(\ell)} + \text{MultiHeadAttention}(\tilde{X}^{(\ell)})$$

$$\hat{X}^{(\ell)} = \text{LayerNorm}_2\Big(X_2^{(\ell)}\Big)$$

$$\text{FeedForwardNetwork}(\hat{X}^{(\ell)}) = \sigma\Big(\hat{X}^{(\ell)} W_1 + b_1\Big) W_2 + b_2$$

$$X^{(\ell+1)} = X_2^{(\ell)} + \text{FeedForwardNetwork}(\hat{X}^{(\ell)})$$

where H is the number of heads, $d_k = d_{\text{model}}/H$, $[\cdot, \cdot]$ denotes concatenation over the head dimension, σ is a nonlinearity (e.g. GELU), and M is a causal mask with $M_{ij} = -\infty$ for $j > i$ and 0 otherwise.

3.2 Finite Automata

Definition 1. A **deterministic finite automaton (DFA)**, (for our purposes also called a **finite automaton** or **finite state machine (FSM)**, although neither term is technically equivalent[6]), is a 5-tuple $(Q, \sum, \delta, q_0, F)$ (Sipser, 2013).

Q is the **set of states** in the automaton. $\sum$ is the **alphabet** in the automaton, or the set of "actions" that can be taken from each state. $\delta : Q \times \sum \to Q$ is the **transition function**, a function that maps from each conceivable state, action pair to another state, specifying the transitions in the automaton. $q_0 \in Q$ is the **starting state**, where the finite automaton begins. $F \subseteq Q$ is the **set of accept states** in the automaton. An automaton **accepts** a sequence of actions if starting from the start state, said sequence of actions ends in an accept

[6]Technically, a finite automaton or finite state machine could be deterministic or non-deterministic. Non-deterministic finite automata need not fully specify their transition function, although they are technically each equivalent to a deterministic finite automaton in the languages they recognize. We do not consider nondeterministic finite automata in this paper.

state. A **language** (set of sequences) is called a **regular language** if it comprises exactly the set of every sequence accepted by a deterministic finite automaton.

Although start and accept states are needed for specifying regular languages, we do not consider them in this paper; instead we start from a random state and do not distinguish between accept states and non-accept states. We are primarily concerned with the abilities of sequential models and transformers to learn a finite state machine's transition function.

Definition 2. An **absorption state** is a state in a DFA where all possible transitions from the state lead back to it.

Intuitively, once you enter an absorption state, you can never leave it. Oprea (2020) finds that the presence of an absorption state in the DFA-representation of a procedure was a significant predictor of reduced cognitive complexity, and one could similarly hypothesize that sequential models and transformers would have an easier time successfully learning DFAs with absorption states, since entering an absorption state would necessitate a model simply predicting that same state as the output after every transition for the remainder of the DFA's specific run.

Figure 1 shows an example of a DFA formally defined as: $(Q_a, \sum_a, \delta_a, q_a, F_a)$. $Q_a = \{Start, q_1, q_2, q_3\}, \sum_a = \{0, 1\}, q_a = Start, F_a = \{q_1, q_2\}$. δ_a can be represented by Table 1. In this automaton, $Start$ is the start state, q_1 and q_2 are accept states, and q_3 is an absorption state.

State/Action	0	1
$Start$	q_2	q_1
q_1	q_3	q_1
q_2	q_2	q_3
q_3	q_3	q_3

Table 1: δ_a.

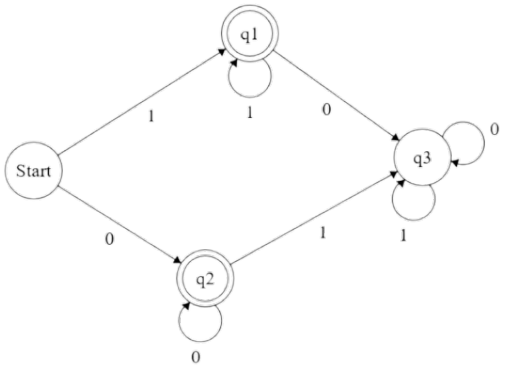


Figure 1: An example of a deterministic finite automaton $(Q_a, \sum_a, \delta_a, q_a, F_a)$ that accepts the regular language $\{0^+ \cup 1^+\}$, which contains exactly the set of sequences of N consecutive 1's or N consecutive 0's, where N is any positive integer.

4 Data

4.1 Data Generation and Sequence Configuration

We developed a synthetic data pipeline using a DFA generator and a trajectory sampler. The sampler produces state-action sequences by executing actions against a deterministic finite automaton (DFA) and recording the resulting state transitions. Each training sample follows an in-context learning format consisting of three demonstration trajectories separated by padding tokens, followed by a target query trajectory:

$$(s_0, a_0, s_1, \ldots, s_T)_1 \quad <\texttt{pad}> \quad (s_0, a_0, s_1, \ldots, s_T)_2 \quad \ldots \quad <\texttt{query}> \quad (s_0, a_0, \ldots, s_k)_{\text{query}}$$

To determine the optimal sequence lengths required for identifiability, we implemented a baseline solver.[7] We defined the target length as the minimum number of steps required to deterministically recover the underlying FSM with $\geq 99.5\%$ accuracy. For the 5-state, 5-action (5S5A) configuration, 3 demonstrations of 90 steps each met this threshold; for the 3-state, 3-action (3S3A) configuration, 3 demonstrations of 30 steps sufficed. To improve the robustness of the model, we introduced a random variation $\pm 20\%$ to these demonstration lengths during generation.

4.2 Dataset Composition

We generated 10,000 unique DFA samples for each configuration, partitioned into 6,000 training, 2,000 validation, and 2,000 test samples. Two iterations of the dataset were created: an initial set generated with completely random transition dynamics, and a subsequent set of 10,000 samples constrained to contain exactly one absorption state, maintaining the same train-test splits.

To evaluate length generalization, we generated supplementary test datasets comprising 2,000 samples each, with query lengths ranging from 200 to 600. These datasets allowed us to assess whether model performance remains consistent on longer queries when demonstration lengths are held constant.

4.3 Training Objective

The architecture leverages in-context learning, where the demonstration trajectories provide the implicit transition dynamics of the specific DFA. Following these demonstrations, the model processes query sequences (100 steps for 5S5A, and 30 steps for 3S3A) to predict state transitions based on the inferred logic. Training utilized full teacher-forcing; however, we applied a masking strategy to the loss function. The loss was computed exclusively on the state outputs of the query sequence, ignoring the demonstration tokens. This ensures that the model is optimized strictly on its ability to generalize the learned DFA rules to new inputs.

[7] Note that since we are simply tracking states as output, we need not distinguish between Mealy or Moore machines in this paper, both of which have a separate output alphabet but function identically with regard to states and actions.

5 Results & Discussion

5.1 Sequential Models

5.1.1 Training

We conducted an exhaustive experimental sweep to evaluate LSTM performance relative to the Transformer baseline. We trained every architecture listed in Table 2 across all hyperparameter configurations detailed in Table 3.

Hidden Dimension	Number of Layers	Parameters
32	1	9,024
32	2	17,472
64	1	34,432
64	2	67,712
128	1	134,400
128	2	266,496
256	1	530,944
256	2	1,057,280

Table 2: LSTM Architecture Exploration

Hyperparameter	Values Swept
Learning Rate	10^{-4}, 5×10^{-4}, 10^{-3}, 2×10^{-3}, 5×10^{-3}, 10^{-2}
Batch Size	16, 24, 32
Weight Decay	0, 10^{-5}, 10^{-4}

Table 3: Hyperparameter Grid Search Space for LSTMs

5.1.2 Results

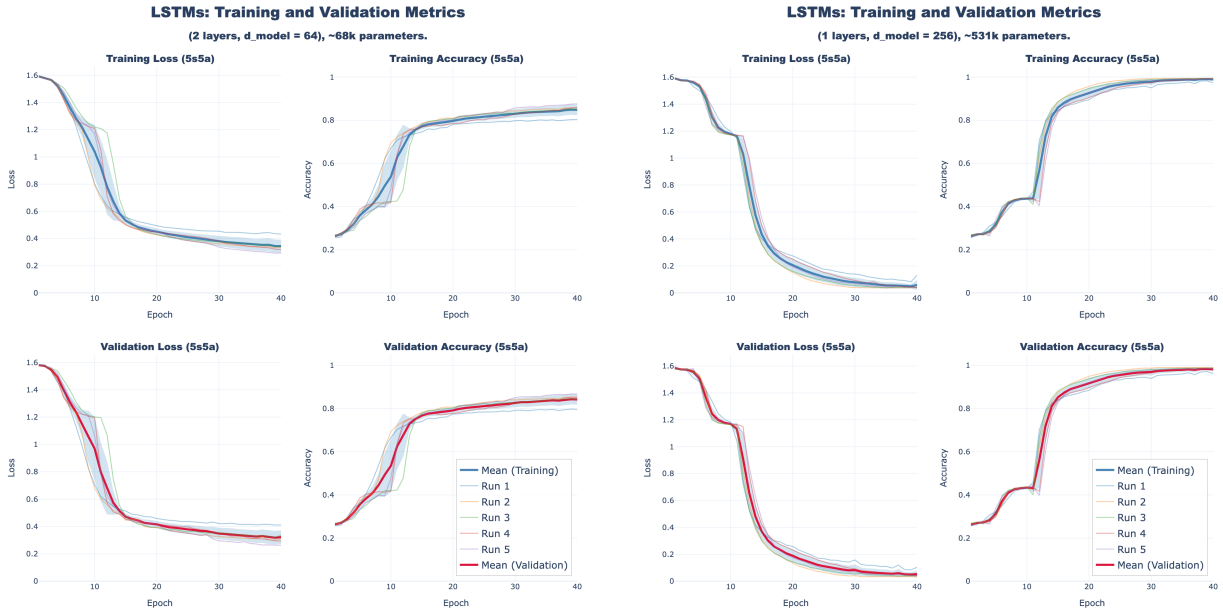
From the comprehensive set of trained models, we report on two distinct configurations that highlight the trade-off between parameter efficiency and accuracy:

1. **Small LSTM (Scale-Matched):** The configuration with 64 hidden units and 2 LSTM layers ($67,712$ parameters). This model was selected because it is comparable in size to our transformer model, which achieved near-perfect validation accuracy with $\approx 52k$ parameters.

2. **Large LSTM (Performance-Matched):** The configuration with 256 hidden units and 1 LSTM layer ($530,944$ parameters). This was the smallest LSTM architecture capable of matching the transformer's high validation accuracy.

The sweep identified distinct optimal settings for these two models. The Small LSTM performed best with a learning rate of 5×10^{-3}, while the Large LSTM required 2×10^{-3}. Both utilized a batch size of 24 and 0 weight decay.

Training results are visualized in Figure 2a and Figure 2b. For the LSTM experiments, each curve represents one of five independent training runs, whereas the RNN baseline was evaluated over three runs. The shaded region denotes one standard deviation around the mean, providing a visual measure of variability. Critically, while we explored curriculum learning for the LSTM models, we observed no significant improvement in convergence speed. The LSTM required substantial epochs to converge even on the simpler 3S3A task, and upon transitioning to the 5S5A task (Stage 2), the loss immediately reverted to a high value of ≈ 1.4–1.5 (near-random guessing). This indicates that the model retained little transferable knowledge, effectively restarting the learning process. Consequently, we report results from direct training on the 5S5A dataset. As shown in Figure 2a, the Small LSTM failed to converge to a competitive accuracy (plateauing at $\approx$ 84%), despite having a parameter count slightly larger than the transformer ($68k$ vs $52k$). In contrast, Figure 2b shows that the Large LSTM successfully achieved near-perfect accuracy. However, this performance came at a significant cost: the LSTM required approximately $10\times$ the parameters ($531k$) to match the performance the transformer achieved with only $52k$.



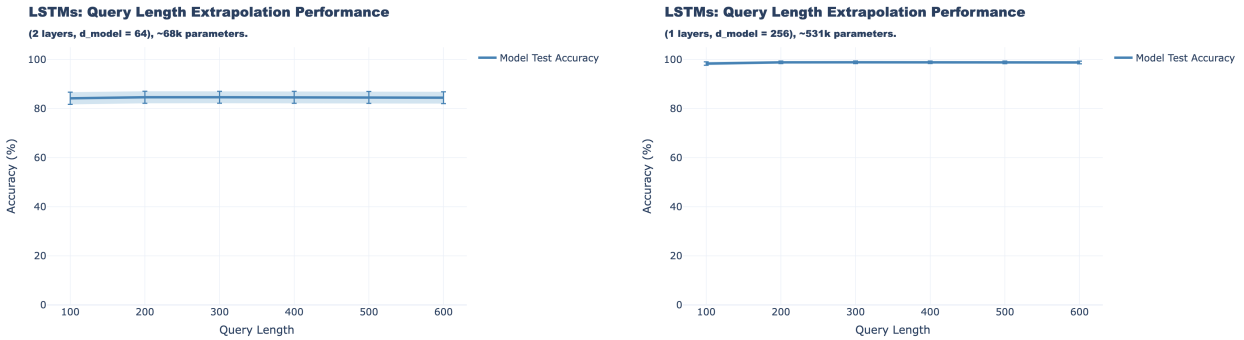
(a) Small LSTM Model Training

(b) Large LSTM Model Training

Figure 2: Small vs. Large LSTM Model Training

We evaluated next-state prediction accuracy across all five experimental runs on query lengths ranging from 200 to 600. As illustrated in Figures 3a and 3b, query length did not significantly impact performance; the small LSTM maintained an accuracy close to 84.5%, while the large model maintained an accuracy close to 98.9%. Furthermore, full-sequence prediction was evaluated specifically (auto-regressive generation in contrast to teacher forcing) at the training query length of 100. In this setting, both models yielded 0% accuracy.

Failures typically occurred within the first few states of the generated sequence. We hypothesize that this early failure is due to the models' sensitivity to the transition from the prompt to the generation phase, where recent exposure to padding and query tokens may disrupt the initial auto-regressive steps, causing cascading errors.



(a) Small LSTM Query Length Extrapolation (b) Large LSTM Query Length Extrapolation

Figure 3: Small vs. Large LSTM Query Length Extrapolation

Query Length	Small LSTM		Large LSTM		Deterministic Solver Accuracy
	Mean	Std	Mean	Std	
100	84.23%	2.49%	98.39%	0.70%	99.61%
200	84.64%	2.45%	98.89%	0.52%	99.60%
300	84.65%	2.43%	98.91%	0.52%	99.51%
400	84.63%	2.43%	98.91%	0.50%	99.54%
500	84.58%	2.43%	98.87%	0.53%	99.56%
600	84.47%	2.41%	98.83%	0.55%	99.63%

Table 4: Query length extrapolation mean accuracy and standard deviation for Small vs. Large LSTM models. Note: Query Length = 100 was the original query length on which the models were trained.

5.1.3 RNNs

We applied the same exhaustive hyperparameter and architecture sweep to Recurrent Neural Networks (RNNs) as described for LSTMs. It is important to note that due to the simplified architecture (a single computational unit versus four gates), an RNN possesses approximately one-quarter the number of parameters of an LSTM with the equivalent hidden dimension and layer count.

To ensure a fair comparison, we present results for the RNN configuration with a hidden dimension of 128 and 2 layers. This specific configuration was selected because it results in

approximately $68k$ parameters, matching the scale of the Transformer ($52k$) and the Small LSTM ($68k$).

Despite this parameter parity, RNNs failed to train effectively on any of the FSM tasks. To contextualize our results, we note that a cross-entropy loss of ≈ 1.1 ($-\ln(1/3)$) corresponds to completely random prediction across 3 states, while a loss of ≈ 1.6 ($-\ln(1/5)$) corresponds to random prediction across 5 states.

As illustrated in Figure 4, the RNN loss on the simpler 3S3A dataset plateaued near 0.8. While this is slightly better than the random baseline of 1.1, the model failed to improve further. On the more complex 5S5A dataset, the loss remained stuck at ≈ 1.6, indicating that state selection was indistinguishable from random guessing.

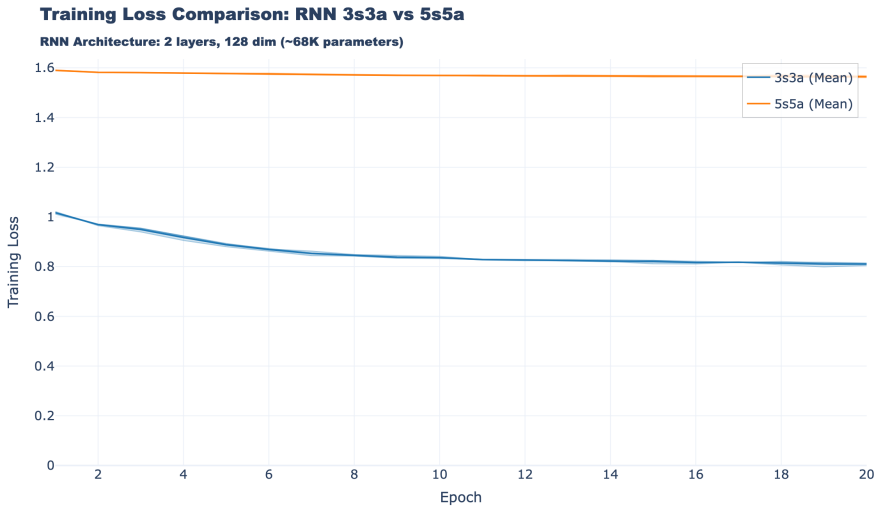


Figure 4: RNN Training Losses.

5.2 Transformers

5.2.1 Experimental Setup: Architecture and Constraints

We explored a variety of transformer architectures to determine the minimal model complexity required for learning FSM transition functions. Architectural details are provided in Table 5 and visualized in Figure 5.

Depth Constraint and Induction Heads Mechanistic interpretability research demonstrates that while 2-layer Transformers are the minimal architecture capable of forming "induction heads" for simple pattern copying (Olsson 2022, and Elhage 2021), they lack the expressivity for higher-order circuit composition found in deeper networks. By limiting the depth to 2 layers, we restrict the model to the simplest possible mechanism for sequence processing, testing whether it learns the specific generative rules of the DFA rather than relying on the complex, multi-step memorization strategies available to deeper architectures.

Positional Embeddings and Masking We utilized Rotary Positional Embeddings (RoPE) rather than absolute positional embeddings. Preliminary experiments indicated that RoPE yielded superior performance, likely due to its ability to encode relative positions, which are critical for understanding state transitions in a sequence. Furthermore, a causal mask was applied to the self-attention mechanism to ensure the model operates in an autoregressive manner, preventing it from attending to future tokens.

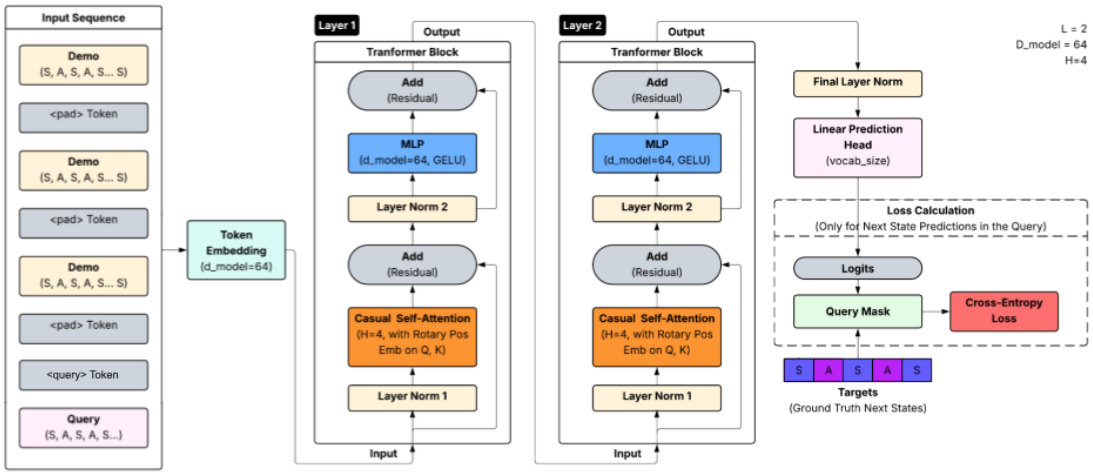


Figure 5: **Transformer Architecture.** Diagram of the 2-layer configuration used in experiments.

Encoding Dim	Num Heads	FFN Length	Parameters
32	2	64	18,088
32	4	64	18,088
64	4	64	51,717
64	4	128	68,936
64	8	128	68,936
128	8	256	268,936
256	8	512	1,062,152

Table 5: Transformer Architecture Exploration (Fixed Depth: 2 Layers)

Hyperparameter	Values Swept
Learning Rate	10^{-4}, 5×10^{-4}, 10^{-3}, 2×10^{-3}, 5×10^{-3}, 10^{-2}
Batch Size	16, 24, 32
Weight Decay	0, 10^{-5}, 10^{-4}

Table 6: Hyperparameter Grid Search Space for Transformers

5.2.2 Optimization and Metrics

We universally utilized the AdamW optimizer and conducted a hyperparameter sweep over the values listed in Table 6.

In addition to the theoretical motivations for limiting model depth, our experimental scope was also bounded by computational constraints. Consistent with our LSTM analysis, loss was computed solely on predicted output states. We report next-state prediction accuracy for our query-length extrapolation tests rather than full-sequence accuracy. This metric provides insight into the model's ability to approximate the DFA's next-state transition function, preventing a single early error from invalidating the evaluation of an entire sequence.

5.2.3 Results and Analysis

The hyperparameter sweep identified the optimal settings as a learning rate of 10^{-3}, 0 weight decay, and a batch size of 24.

Model Selection: The results for the swept architectures are displayed in Figure 6. We identified the "smallest effective model" as the configuration with an encoding dimension of 64, 4 attention heads, and a feed-forward length of 64. This model, containing approximately $52k$ parameters, was the most parameter-efficient architecture capable of achieving near-perfect validation accuracy. We show the training curves for this model architecture in Figure 6. Note that as earlier, the shaded regions in the transformer loss and accuracy charts indicate one standard deviation around the mean.

Curriculum Learning and Phase Transitions: A key finding is the necessity of curriculum learning for Transformers. Unlike the LSTM baselines, the Transformer models failed to learn the complex 5S5A task when trained directly, plateauing at a training and validation loss of ≈ 1.59 (equivalent to random guessing across 5 states) for the duration of training. This mirrors findings by Oprea (2020), suggesting that minimal structural increases in automata complexity create disproportionately harder learning landscapes for attention mechanisms.

However, when trained via a curriculum (first on 3S3A, then 5S5A), the Transformers successfully converged. This suggests that pre-training on simpler DFAs allows the attention heads to learn generalizable patterns, specifically, attending to the relevant (state, action) pairs immediately preceding the current step, which can then be transferred to more complex tasks.

We also observed a distinct "phase transition" phenomenon, visualized in Figure 6, which occurred around epochs 3–4 for the 3S3A task and epochs 1–2 for the 5S5A task. During the initial epochs, the loss remained consistently high, indicating that the model performed little better than random guessing. This plateau was followed by an abrupt convergence, where the loss collapsed to near-zero within the span of a single epoch or two. This sharp discontinuity suggests that the transformer model does not acquire the transition function incrementally; instead, it appears to search for a specific attention mechanism (likely an

induction head), and once this mechanism is discovered, the model solves the task almost instantaneously.

Length Generalization: Finally, as seen in Figure 7 and Table 7, we evaluated the optimized Transformer on extended query lengths ranging from 200 to 600 tokens. Unlike the LSTM models, which maintained robustness across lengths, the transformer's accuracy tended to somewhat degrade as the query length increased. We hypothesize that this is because the LSTM's recurrent update rule is inherently time-invariant, allowing it to apply the same learned transition logic indefinitely. In contrast, the transformer model relies on learned positional embeddings (RoPE) which, despite their relative nature, may suffer from distributional shift when evaluated on sequence lengths significantly exceeding the 100-token window seen during training. We also calculated the full-sequence accuracy on the 5S5A task for the best transformer model (where a sequence is correct only if all 100 predictions are perfect when generated auto-regressively). The transformer achieved a full-sequence accuracy of 94.05%.

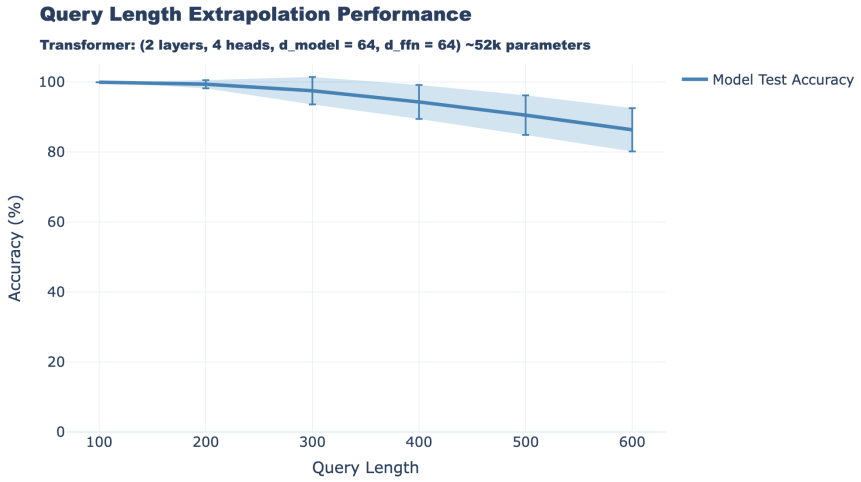


Figure 7: Transformer accuracy on query length extrapolation

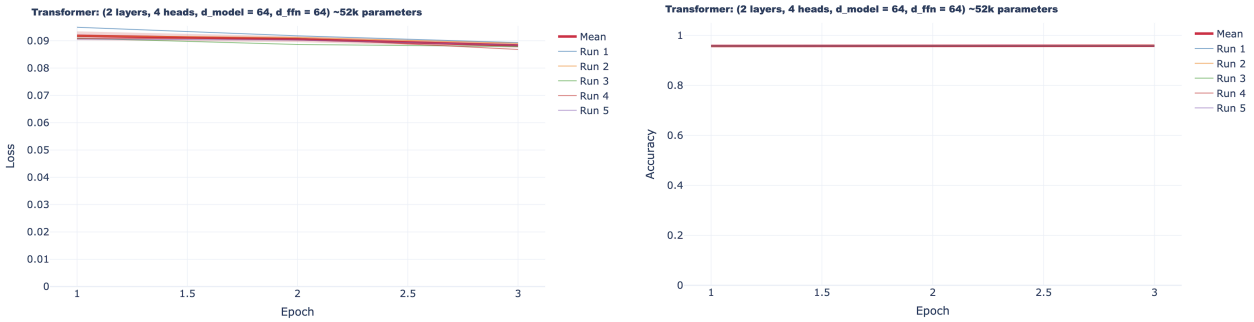
Query Length	Transformer Mean	Transformer Std	Deterministic Solver Accuracy
100	99.92%	0.01%	99.61%
200	99.36%	1.17%	99.60%
300	97.51%	3.92%	99.51%
400	94.30%	4.87%	99.54%
500	90.53%	5.65%	99.56%
600	86.34%	6.19%	99.63%

Table 7: Query length extrapolation performance for the Transformer model. Note: Query Length = 100 was the original query length on which the models were trained.

Figure 6: **Transformer Training and Validation.** Training dynamics showing the necessity of curriculum learning and the "phase transition" phenomenon.

5.2.4 Absorption States

We solely trained the transformer models on DFAs with at least one absorption state. The losses were immediately very low for both the training stages. Figures 8a and 8b show the validation losses and accuracies for the second stage of learning for these datasets. On the surface, this matches with Oprea (2020)'s results and our earlier intuition that automata with absorption states would be easier to learn. However, the training demonstrations were largely dominated by absorption states, suggesting that the model probably collapses to a trivial solution of repeating the absorption state, casting doubt on these transformers' abilities to generalize to automata without absorption states. Training large finite automata with absorption states may be a fruitful avenue for future research, one unavailable to us due to compute limits.



(a) validation loss saturated at around 0.09 (b) validation accuracy saturated at around 96%

Figure 8: Validation Losses and Accuracies for Stage 2 (5S5A) of Curriculum Learning on Samples with Absorption States

6 Conclusion

In this paper, we take inspiration from literature in the computational social sciences and study how well sequential and 2-layer transformer models can in-context learn finite automata. Unlike existing literature in the area, we restrict our attention to learning the transition functions of relatively small finite automata from observed transition sequences going from one state to another through actions. We find that 2-layer transformers are better at ICL in this case and use fewer parameters in doing so, but must learn larger automata through curriculum learning with smaller automata, while LSTM models can likely brute-force learn larger automata, likely because of their hard-coded sequential structure. Additionally, transformers in particular display phase transition affects as the attention heads learn how to emulate transition functions across DFAs, while LSTM models display relatively more gradual improvements.

There are ways future work could build upon our results. A more robust set of neural network architectures and DFA sizes in terms of state and action counts could yield more detailed insights into what structural markers of complexity make automata difficult to learn. For instance, training models on larger finite automata with absorption states or full representations of an accept state-containing automaton's transition function could improve

the reliability of those results by reducing the domination of absorption states during training. This could also lead to more rigorous identification of markers of learning complexity common to humans and neural networks[8].

Additionally, training neural network architectures on computational models higher on the Chomsky hierarchy, such as pushdown automata or Turing machines, would be an interesting approach. A lot of work has been done studying in-context learning for mathematical functions based on primitive recursive functions, an equivalent computational model to Turing machines, but potentially quite structurally different in terms of learning complexity for neural networks.

References

Rik Adriaensen and Jaron Maene. *Extracting Moore Machines from Transformers using Queries and Counterexamples*. 2024. arXiv: 2410.06045 [cs.LG]

Ekin Akyürek et al. *In-Context Language Learning: Architectures and Algorithms*. 2024. arXiv: 2401.12973 [cs.CL]

Ying Fan et al. "Transformers Can Learn Meta-skills for Task Generalization in In-Context Learning". In: *NeurIPS 2024 Workshop on Compositional Learning: Perspectives, Methods, and Paths Forward*. 2024

Shivam Garg et al. *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes*. 2022. arXiv: 2208.01066 [cs.CL]

Chase Goddard et al. *When can in-context learning generalize out of task distribution?* 2025. arXiv: 2506.05574 [cs.LG]

John J. Horton. *Large Language Models as Simulated Economic Agents: What Can We Learn from Homo Silicus?* 2023. arXiv: 2301.07543 [econ.GN]

Bingbin Liu et al. *Transformers Learn Shortcuts to Automata*. 2023. arXiv: 2210.10749 [cs.LG]

Ryan Oprea. "What Makes a Rule Complex?" In: *American Economic Review* 110.12 (2020), pp. 3913–3954. DOI: 10.1257/aer.20200019

Kishan Padayachy et al. *In-Context Learning Enhanced Credibility Transformer*. 2025. arXiv: 2509.08122 [cs.LG]

Sipser M., (2013). *Introduction to the Theory of Computation.* Melbourne: Cengage Learning.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems* 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[8]Back in the computational social science sphere, giving large languages models prompts explaining instructions represented by automata (as they were in Oprea (2020)), asking them to obey said instructions, and tracking hallucination rates could be an interesting research direction. In general, simulating economic experiments with LLMs previously done on people always has the potential to yield interesting insights for computer scientists, economists, and cognitive scientists.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Elman, J.L., (1990). Finding Structure in Time. *In Cognitive Science, A Multidisciplinary Journal.* https://doi.org/10.1207/s15516709cog1402_1.

Jordan, M.I. (1986). *Serial order: A parallel distributed processing approach* (Tech. Rep. No. 8604). San Diego : University of California, Institute for Cognitive Science.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah, (2022). *In-Context Learning and Induction Heads. ArXiv Preprint*, abs/2209.11895. URL https://arxiv.org/abs/2209.11895.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann† Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, Chris Olah, (2021). *A Mathematical Framework for Transformer Circuits.* Anthropic. URL https://transformer-circuits.pub/2021/framework/index.html.

Appendix A: Poster Feedback

The primary point of feedback we received was that the axes on the transformer charts were not identical. We fixed this, so every x-axis now tracks 7 epochs, loss ranges from 0 to 1.4 on the y-axis of the loss graphs, and the y-axis for the accuracy graphs was of course 0 to 1.

Appendix B: Addressing Reviewers

Here, we address some notable feedback points from the reviewers.

1. We tried curriculum learning for LSTMs, but results performed as well as random initialization in the second phase, so we decided to train them directly on $5S5A$ automata rather than relying on curriculum learning.

2. Transformers could not reduce loss across multiple epochs for $5S5A$ automata without curriculum learning.

3. Keep in mind that we did the best training possible for RNNs, LSTMs, and transformers. Curriculum learning worked well for transformers, but not LSTMs. So, this is an apples to apples comparison. We did not just engage in curriculum learning for transformers alone. We made this more explicit above.

4. We added a table for transformers on the query length extrapolation test.

5. In the tables, where we have a query length of 100, we explicitly mentioned it is the baseline test loss, since these were the inputs the models were trained on.

6. When mentioning test accuracy, we textually mentioned standard deviation.

7. We reworded the "grokking" phenomenon, instead referring to it as "phase transition".

8. We evaluate models using next state prediction rather full-sequence prediction because the latter conflates DFA transition function learning $\delta(q, a)$ learning with compounding autoregressive error. We have added measures of full sequence accuracy for 1 run. Even with a 99% prediction accuracy the chances of successfully getting the sequence correct is about 36%, so our results glean us much greater levels of insight with next-state prediction accuracy rather than full sequence prediction accuracy, which would treat 99% correct sequences the same as 0% correct sequences.

9. Reviewer #4 likely used an LLM, which seems to be hallucinating. For instance, "nite" was nowhere to be found in our paper. Additionally, some of their suggestions were clearly intractable for a project of our scope, particularly "mechanistic interpretability analysis of attention patterns".