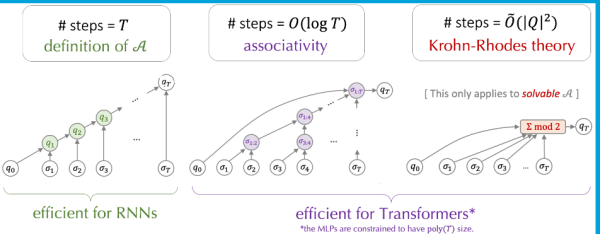
Transformers Learn Shortcuts to Automata

Yash Chauhan [1] & Harisankar B [1]

[1]National Institute of Science Education and Research

April 2025

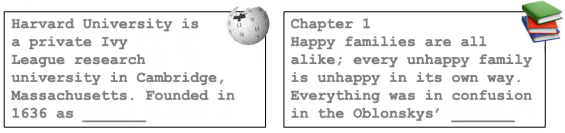


Outline

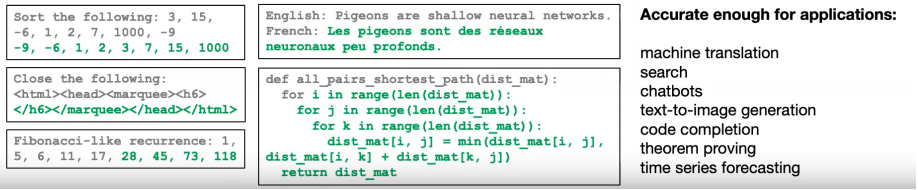
Section 1. Introduction

Emergence of Neural Algorithmic Reasoning

- **Training:** next-word prediction on ~45TB of Internet text [Brown et al. '20]



- **Evaluation:** open-ended "reasoning" capabilities



Have We Achieved Universal Reasoning?

Have We Achieved Universal Reasoning?

- **Of course not...**

Here is a sketch of the proof that P != NP.

1. Assume for contradiction that P = NP.
2. Assume that there exists an algorithm A that can solve any NP-complete problem in polynomial time.
3. Since A can solve any NP-complete problem, it can solve the SAT problem, which is known to be NP-complete.
4. Since A can solve the SAT problem in polynomial time, it can solve any instance of the SAT problem in polynomial time.
5. This implies that the Boolean satisfiability problem is in P, which contradicts the known fact that it is NP-complete.
6. Therefore, P ≠ NP.

hallucinated logic

P != NP was proven in 1975 by Stephen Cook.

The formal statement of the proof is called the Cook-Levin theorem, which states that any problem in the complexity class NP can be efficiently reduced to a problem in the complexity class P. This theorem was the first major result in the field of computational complexity theory and effectively established the first examples of NP-complete problems. The Cook-Levin theorem is widely considered to be one of the most important contributions to the field of theoretical computer science.

hallucinated knowledge

Problem Statement

- Transformers have achieved remarkable success in various sequence modeling tasks.

Problem Statement

- Transformers have achieved remarkable success in various sequence modeling tasks.
- Despite lacking recurrence, they can perform algorithmic reasoning tasks that traditionally require recurrent models like Turing machines.

Problem Statement

- Transformers have achieved remarkable success in various sequence modeling tasks.
- Despite lacking recurrence, they can perform algorithmic reasoning tasks that traditionally require recurrent models like Turing machines.
- This raises the question: **How do shallow, non-recurrent Transformers emulate computations that seemingly require deep, recurrent architectures?**

Motivation

Why Study Shortcut Learning in Transformers?

- Understanding the mechanisms that allow Transformers to simulate automata can shed light on their internal workings.
- Insights into these "shortcut" solutions can inform the design of more efficient models and training procedures.
- Exploring the limitations and brittleness of these shortcuts is crucial for ensuring model robustness, especially in out-of-distribution scenarios.

Section 2. Preliminaries

Finite-State Automata

Finite-State Automaton

A **finite-state automaton** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states,
- Σ is a finite input alphabet,
- $\delta : Q \times \Sigma \to Q$ is a transition function,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states.

FSAs define regular languages, recognize patterns, and model bounded-memory computations.

Semiautomata and Simulation

Semiautomaton

A **semiautomaton** is a triple (Q, Σ, δ), where Q, Σ, and δ are as defined in FSAs. It evolves deterministically, without designated accept states or outputs.

State Evolution : Given an input string $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_T) \in \Sigma^T$ and an initial state q_0, the semiautomaton computes a trajectory $(q_1, q_2, \ldots, q_T) \in Q^T$ by recursive application of the transition function:

$$q_t = \delta(q_{t-1}, \sigma_t), \quad \text{for } t = 1, \ldots, T.$$

This is called the **simulation problem** - given the initial state q_0 and an input sequence $\boldsymbol{\sigma}$, compute the final state q_T. Intuitively, this corresponds to executing a chain of T reasoning steps.

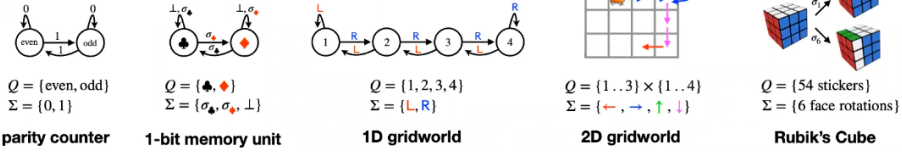
Semiautomata and Simulation

- **Semiautomaton:** a discrete-time dynamical system

$$(Q, \Sigma, \delta) \qquad q_t = \delta(q_{t-1}, \sigma_t)$$

states inputs transitions

recurrent dynamics



$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

parity counter

$Q = \{\spadesuit, \diamondsuit\}$
$\Sigma = \{\sigma_\spadesuit, \sigma_\diamondsuit, \perp\}$

1-bit memory unit

$Q = \{1, 2, 3, 4\}$
$\Sigma = \{\mathsf{L}, \mathsf{R}\}$

1D gridworld

$Q = \{1..3\} \times \{1..4\}$
$\Sigma = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$

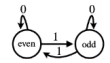
2D gridworld

$Q = \{54 \text{ stickers}\}$
$\Sigma = \{6 \text{ face rotations}\}$

Rubik's Cube

An Interesting Example

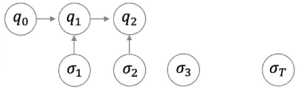
- Consider the problem of simulating the **parity counter** semiautomaton:



$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

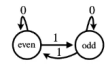
iterative state emulation

```
q ← q_0
for t = 1..T:
  if σ_t == 0:
    q ← 1 − q
return q
```



An Interesting Example

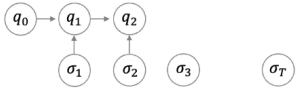
- Consider the problem of simulating the **parity counter** semiautomaton:



$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

iterative state emulation

```
q ← q_0
for t = 1..T:
  if σ_t == 0:
    q ← 1 − q
return q
```

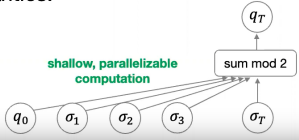


- Dynamics of $\mathcal{A} = (Q, \Sigma, \delta)$ induce global regularities:

This is a mod-2 counter!

"shortcut" solution

```
sum ← q_0 + Σ_{t=1:T} σ_t
return sum % 2
```



Algebraic Structure behind Automata

Semigroup

A **semigroup** is a set S equipped with an associative binary operation $\cdot : S \times S \to S$. That is, for all $a, b, c \in S$, we have:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

Group

A **group** is a semigroup $(G, \cdot)$ with an identity element $e \in G$ such that:

- $e \cdot a = a \cdot e = a$ for all $a \in G$,
- Every $a \in G$ has an inverse $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Input Symbol as Transformation

In a semiautomaton (Q, Σ, δ), each input symbol $\sigma \in \Sigma$ induces a transformation $f_\sigma : Q \to Q$ defined by:

$$f_\sigma(q) = \delta(q, \sigma).$$

This means that each symbol acts as a function on the set of states- specifically, a **state transformation**. If we denote by $\mathcal{T}_Q$ the set of all functions from Q to Q, then:

$$f_\sigma \in \mathcal{T}_Q.$$

The set $\mathcal{T}_Q$ with function composition forms a **semigroup** called the full transformation semigroup on Q.
The transformations $\{f_\sigma\}_{\sigma \in \Sigma}$ generate a subsemigroup of $\mathcal{T}_Q$, called the **transition semigroup** of the automaton.

From Input Symbols to Transformations

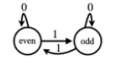
- Each input symbol $\sigma \in \Sigma$ induces a transformation:

$$\delta(\cdot, \sigma) : Q \to Q$$

- These transformations can be represented as functions or matrices acting on state vectors.

Example: Parity Counter Automaton

- Each input symbol $\sigma \in \Sigma$ induces a **transformation** $\delta(\cdot, \sigma) : Q \to Q$



$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

parity counter

$$\delta(\cdot, 0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \textit{(identity)}$$

$$\delta(\cdot, 1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \textit{(swap)}$$

function composition

$$q_t = \left(\delta(\cdot, \sigma_t) \circ \cdots \circ \delta(\cdot, \sigma_1)\right) q_0$$

matrix multiplication

$$e_{q_t} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} e_{q_0}$$

- These functions form a **semigroup:** $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition
  - Function composition is *associative:* $(f \circ g) \circ h = f \circ (g \circ h)$
- Understand $\mathcal{A}$ via its **transformation semigroup** $\mathcal{T}(\mathcal{A})$

From Input Words to State Updates

- A word $\sigma_1\sigma_2\cdots\sigma_t$ induces a composed transformation:

$$q_t = (\delta(\cdot,\sigma_t)\circ\cdots\circ\delta(\cdot,\sigma_1))(q_0)$$

- In matrix form:

$$e_{q_t} = \delta_{\sigma_t}\cdot\delta_{\sigma_{t-1}}\cdots\delta_{\sigma_1}\cdot e_{q_0}$$

The Transformation Semigroup

- The set of all such transformations forms a **semigroup** under composition:

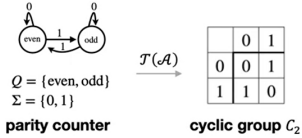
$$\mathcal{T}(\mathcal{A}) := \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$$

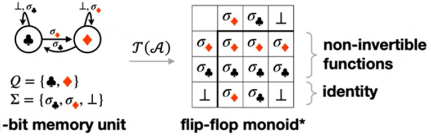
- Composition is associative:

$$(f \circ g) \circ h = f \circ (g \circ h)$$

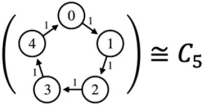
- We understand the automaton $\mathcal{A}$ via its transformation semigroup $\mathcal{T}(\mathcal{A})$

Examples of Transformation Semigroups

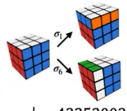




$$\mathcal{T}\left(\right) \cong C_5$$

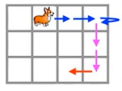


$$\mathcal{T}\left(\right) \cong G_{\text{Rubik}}$$



$|G_{\text{Rubik}}| = 43252003274489856000$

$\mathcal{T}\left(\right)$: not a group!



Normal Subgroup

Normal Subgroup

A subgroup $N \leq G$ is called a **normal subgroup** of G if it is invariant under conjugation by elements of G, i.e.,

$$\forall g \in G, \quad gNg^{-1} = N.$$

This is denoted as $N \trianglelefteq G$.

Quotient Group

Quotient Group

Given a normal subgroup $N \trianglelefteq G$, the **quotient group** G/N is the group formed by the cosets of N in G, with the operation:

$$(aN)(bN) = (ab)N.$$

This captures the “collapse" structure of G modulo N.

Simple Group

Simple Group

A group G is **simple** if it is nontrivial and has no proper nontrivial normal subgroups. That is, the only normal subgroups of G are:

$$\{e\} \quad \text{and} \quad G.$$

Simple groups are the "building blocks" of all finite groups.

Solvable Group

Solvable Group

A group G is **solvable** if there exists a finite sequence of subgroups:

$$\{e\} = G_0 \trianglelefteq G_1 \trianglelefteq \cdots \trianglelefteq G_n = G$$

such that each quotient G_{i+1}/G_i is abelian. These groups generalize the idea of "solving by radicals" in Galois theory.

Semidirect Product

Semidirect Product

Given groups N and H, and a homomorphism $\varphi : H \to \mathrm{Aut}(N)$, the **semidirect product** $N \rtimes_\varphi H$ is defined on the set $N \times H$ with multiplication:

$$(n_1, h_1)(n_2, h_2) = (n_1 \cdot \varphi(h_1)(n_2),\ h_1 h_2).$$

This construction allows H to act on N nontrivially.

Wreath Product

Wreath Product

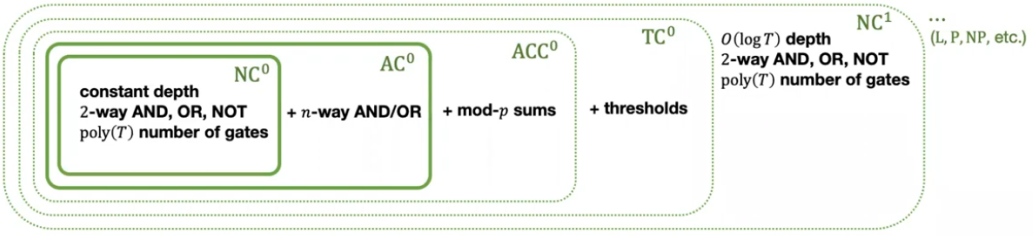
Let A and B be groups, and let B act on a set X. The **wreath product** $A \wr B$ is defined as the semidirect product:

$$A^X \rtimes B,$$

where A^X is the direct product of copies of A indexed by X, and B acts on A^X by permuting coordinates.

Circuit Complexity

- **Boolean circuits:** Computational graphs for functions $\{0,1\}^T \to \{0,1\}$
- **Circuit complexity:** Which functions can simple circuits represent?



e.g. Since $\mathcal{T}(\mathcal{A}_{\text{parity}}) \cong C_2$, parity can be simulated in ACC^0

Neural Architectures

- **Neural architectures:** computational graphs with trainable parameters.
- **Recurrent Neural Nets (RNNs):** emulate state by feeding hidden state forward.

RNN Update Rule

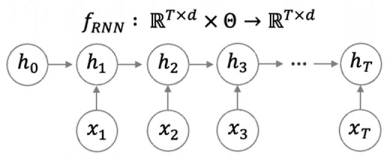
Let $\Theta = \{A, B \in \mathbb{R}^{d \times d}, h_0 \in \mathbb{R}^d\}$. Given input sequence $x_1, x_2, \ldots, x_T$, the hidden states evolve as:

$$h_t := \sigma(Ah_{t-1} + Bx_t)$$

$$f_{\text{RNN}} : \mathbb{R}^{T \times d} \times \Theta \to \mathbb{R}^{T \times d}$$

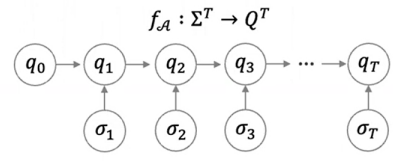
RNNs: modeling automata with automata

- RNNs operate over $\mathbb{R}^d$ (continuous state space)
- Automata operate over Q (finite set)
- RNNs **generalize** classical automata



⊃

subsume by design



Transformers as Learned Transition Systems

- Transformers process sequences using stacked layers of attention and feedforward blocks.
- Unlike fixed automata, transitions are learned and real-valued.
- Each layer applies a transformation to the entire input sequence.
- Central mechanism: **self-attention**.

Self-Attention Mechanism

Scaled Dot Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Queries (Q), Keys (K), Values (V) are learned linear projections of input.
- Each token attends to all other tokens.

Transformer Layer Structure

- Each layer has:
  - Multi-Head Self-Attention
  - Feedforward Network
  - Residual Connections
  - Layer Normalization
- Position encodings break input symmetry.

Token Representation Update

$$\mathbf{x}_t^{(\ell+1)} = \text{Layer}_\ell(\mathbf{x}_{1:T}^{(\ell)})$$

Algebraic Interpretation

- Automata: each input induces a transformation of states.
- Transformer: each layer induces a transformation over $\mathbb{R}^n$.
- Sequence of attention layers = composed functions (a semigroup!).
- Each head acts like a soft transition map over the input space.

Section 3. Theoretical Results

Can shallow Transformers simulate deep reasoning by learning the right computational shortcuts?

Theorem I: Simulation is generically parallelizable

Transformers can simulate all semiautomata $\mathcal{A} = (Q, \Sigma, \delta)$ at length T, with depth $O(\log T)$, embedding dimension $O(|Q|)$, attention width $O(|Q|)$, and MLP width $O(|Q|^2)$.

Why Can We Hope for Shortcuts?

Core Idea: The composition of automaton transitions is *associative*, enabling a divide-and-conquer strategy.

- To simulate an automaton over input $\sigma_{1:T}$, we compute a function composition:

$$q_T = \delta(\cdot, \sigma_T) \circ \cdots \circ \delta(\cdot, \sigma_1)(q_0)$$

- Each $\delta(\cdot, \sigma)$ is a function $Q \to Q$, which can be represented as a matrix. (e.g. $\delta(\cdot, 0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\delta(\cdot, 1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$).
- Function composition becomes matrix multiplication - which is associative.
- We can parallelize this computation using a binary tree of depth $\log T$.

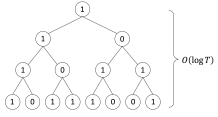


Figure: Associativity allows divide-and-conquer parallelism; tree depth is $O(\log T)$

Theorem II: Transformer Krohn-Rhodes

Transformers can simulate all solvable semiautomata $\mathcal{A} = (Q, \Sigma, \delta)$, with depth $O(|Q|^2 \log |Q|)$, embedding dimension $2^{O(|Q| \log |Q|)}$, attention width $2^{O(|Q| \log |Q|)}$, and MLP width $|Q|^{O(2^{|Q|})} + 2^{O(|Q| \log |Q|)} \cdot T$.

“Prime Factorization of Automata”

Universal Structure Theorems (Algebraic Analogy):

- **Integers:** $N = p_1 \cdot p_2 \cdots p_n$, prime numbers p_i
- **Groups:** $G \rhd H_n \rhd \cdots \rhd H_1$, simple groups H_{i+1}/H_i
- **Semigroups:** Do we know anything if we only assume associativity?

Yes! *(Krohn & Rhodes, 1965)* - Every finite semigroup can be decomposed into simple groups and memory components via the wreath product.

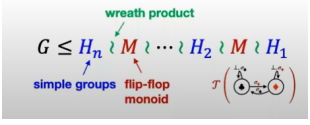


Figure: Semigroup view: Wreath product of simple groups and flip-flop monoids

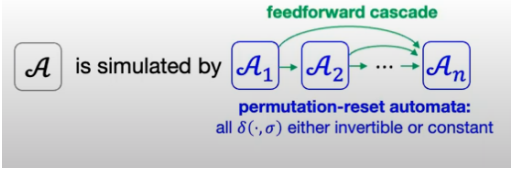


Figure: Semiautomata view: Simulated by a cascade of permutation-reset automata

Theorem III: Depth-2 shortcut for gridworld

For all positive integers n, T, Transformers can simulate Grid_n at length T, with depth 2, embedding dimension $O(1)$, attention width $O(n)$, and MLP width $O(T)$.

Simulating a Non-Abelian Group: Car-on-a-Circle

Setup: A car moves around a circular track of 4 stations with direction (left/right) and two actions:

- D (drive): move forward 1 unit based on current direction
- U (U-turn): reverse direction in place

Challenge: The operations do **not** commute:

$$\texttt{drive} \circ \texttt{U-turn} \neq \texttt{U-turn} \circ \texttt{drive}$$

Goal: Find the final state of the car after executing a sequence of D and U actions without simulating each step.

$Q = \{$🚗, 🚗$\} \times \{0,1,2,3\},$

$\Sigma = \{D(\text{drive}), U(\text{U-turn})\}.$

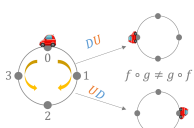


Figure: Car-on-circle setup with non-commutative transitions

Decomposing the Car-on-Circle Automaton

Even though the full automaton is non-abelian, we can simulate it by composing two simpler automata:

- Count **number of U-turns** $\Rightarrow$ get direction (left/right)
- Use that direction to compute a **signed sum of drives** (mod 4)

Each subtask is parallelizable:

- 1 layer for parity of U, 1 layer for modular sum $\Rightarrow$ total: $\mathcal{O}(1)$ layers

Figure: Step 1: U-parity gives direction

Figure: Step 2: Signed drive sum (mod 4)

Krohn-Rhodes Decomposition with Transformers

Any solvable semiautomaton $\mathcal{A}$ can be decomposed into two fundamental components:

- **Mod counters** - track sums modulo p (e.g., parity).
- **Memory units** - track discrete memory updates via sparse attention.

Implementation in Transformers:

- *Mod counters:* handled via uniform attention $\Rightarrow$ compute prefix sum + mod with MLP
- *Memory units:* implemented with sparse attention $\Rightarrow$ attend to last write location

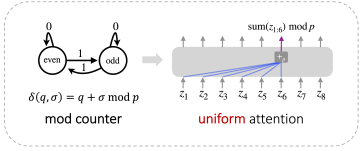


Figure: Mod counter using uniform attention

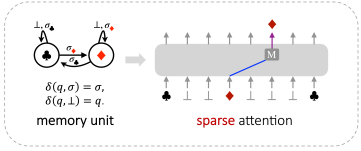


Figure: Memory unit using sparse attention

Summary of the Theoretical Results

For any sequence length T, Transformers with polynomial width can simulate automata with the following depth guarantees:

- **Theorem 1 (Example 1):** Any semiautomaton $\mathcal{A}$ can be simulated using $O(\log T)$ layers.
  Moreover, $\Omega(\log T)$ layers are necessary unless $\mathrm{TC}^0 = \mathrm{NC}^1$.
- **Theorem 2 (Examples 2 & 3):** Any *solvable* semiautomaton can be simulated with $O(|Q|^2 \log |Q|)$ layer complexity,
  where depth depends only on $|\mathcal{A}|$ (not on T).
- **Theorem 3 (Example 4):** Structured automata such as gridworlds can be simulated in constant depth $O(1)$.
  Even shallower circuits may suffice for special automata (e.g. Parity Counter).

Section 4. Empirical Results

Setup

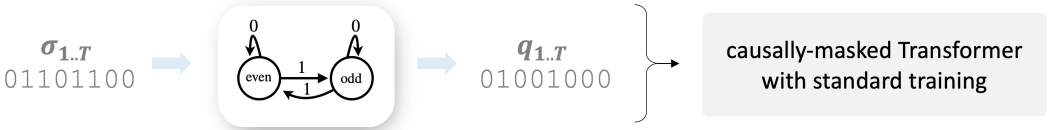


Figure: A causally-masked Transformer (GPT2-style) is trained on semiautomaton trajectories. Each input sequence $\sigma_{1:T} \in \Sigma^T$ is mapped to a corresponding state sequence $q_{1:T} \in Q^T$, generated by applying the automaton's transition rule starting from a fixed $q_0 \in Q$.

Can shortcuts be found in practice?

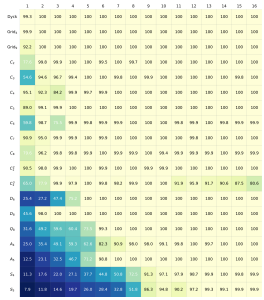


Figure: Accuracy heatmap over depth.

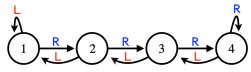
- **Shallow models perform well** on solvable and structured automata (e.g., C_n, Dyck, Gridworld).
- **Deeper networks are needed** for automata with non-solvable transition groups (e.g., A_5, S_5).
- **Theoretical depth aligns with accuracy**, though shortcut generalization can be brittle.

1d gridworld with $\mathcal{O}(1)$-layer solution

Setup: Consider a 1D gridworld with n states, inputs $\Sigma = \{L, R\}$ and state updates:

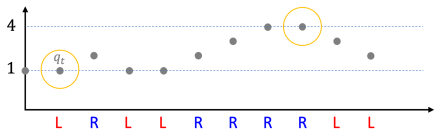
$$q_t = \max(1, \min(n, q_{t-1} + \sigma_t)), \quad \text{where } \sigma_t \in \{-1, +1\}$$

Goal: Predict the final position after a sequence of left/right moves starting from q_0.
Krohn-Rhodes: Guarantees a shortcut with $O(|Q|^2)$ operations.
Surprising result: GPT-2 learns a much shorter $O(1)$-layer solution, independent of n.



$Q = \{1, 2, 3, 4\}$, $\Sigma = \{$L/-1, R/1$\}$.

$\delta(q_t, \sigma_t) = \max(n, \min(1, q_t + \sigma_t))$



Boundary-Aware Shortcut via Attention

Core idea: Instead of simulating all transitions, the model learns to detect the most recent boundary state ($q_t = 1$ or $q_t = n$), after which movement is linear.
Implementation: One layer of uniform attention is sufficient to compute the net displacement after boundary, enabling exact simulation.

Transformer attention aligns with the last boundary hit - where position clips.

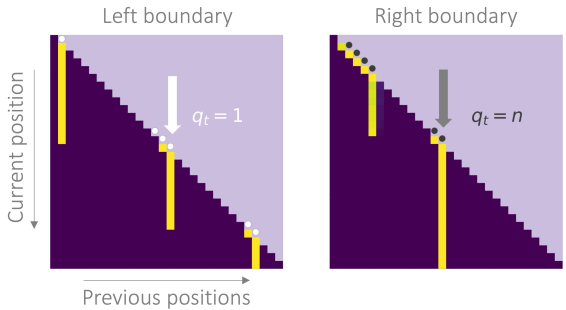


Figure: attention maps: yellow = sharp focus on last boundary state ($q_t = 1$ or $q_t = n$).

Are shortcuts strictly better than iterative solutions?

Shortcuts offer shallow, parallel solutions to simulate automata - faster and often easier to train.

But how do they hold up under limited supervision or when tested out-of-distribution?

Shortcuts Can Be Brittle: Limited Supervision

Setup: Reduce supervision in two ways:

- **Indirect supervision:** Train on a function of q_t instead of q_t directly.
- **Incomplete supervision:** Reveal q_t at each step with probability p_{reveal}.

Observation:

- Transformers perform well under indirect supervision.
- But fail sharply with sparse q_t reveals - especially on complex automata like S_5.
- LSTM remains stable across all p_{reveal} values due to recurrent inductive bias.

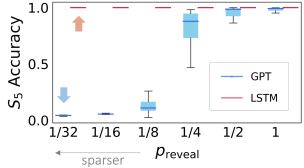


Figure: Transformer accuracy drops as supervision gets sparser; LSTM remains robust.

Shortcuts Fail Out-of-Distribution (OOD)

OOD Test: Train parity automaton on sequences of length 40 with $p(\sigma_t = 1) = 0.5$. At test time, change the number of 1s.

Observation:

- Transformer performs well near training distribution (around 20 ones).
- Performance drops when the number of 1s deviates - especially for underrepresented cases.
- Likely due to reliance on positional encoding correlations that don't hold OOD.

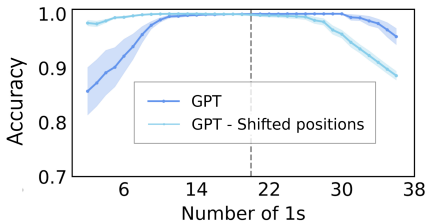


Figure: Accuracy drops for unseen numbers of 1s. Shifted position encoding helps.

Section 5. Discussion and Conclusion

Discussion

- ⟶ Transformers can simulate automata efficiently - often with surprisingly shallow circuits.
- ⟶ Algebraic structure (e.g., solvability) predicts whether shortcut solutions exist.
- ⟶ However, shortcut-based solutions are fragile:
  - Fail under sparse supervision or shifted distributions.
  - Tend to overfit patterns instead of rules.
- ⟶ Recurrent models (like LSTMs) handle these brittle cases better due to their inductive biases.

Conclusion

- Transformers can learn algorithmic shortcuts to simulate automata - efficiently and often correctly.
- Theoretical results match empirical trends, especially for solvable automata.
- Shallow solutions generalize poorly under realistic conditions.
- Recurrent or autoregressive solutions offer better robustness - at the cost of depth and parallelism.

What the paper does not address:

- No formal characterization of when learned shortcuts generalize beyond training distribution.
- Limited discussion on scaling to real-world, large-scale structured data or noisy settings.

Bibliography

Eilenberg, S. (1974). Automata, Languages, and Machines, volume A. Academic Press.

Liu, B. (2023). Transformers learn shortcuts to automata - blog summary. https://clarabing.github.io/shortcut_automata/. Accessed April 2025.

Liu, B., Zhang, C., Weiss, G., Karthik, N., Shih, K., Jaakkola, T., and Sontag, D. (2023). Transformers learn shortcuts to automata. arXiv preprint arXiv:2306.15595.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Å., and Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Zhang, C. (2023). How do transformers reason? first principles via automata, semigroups, and circuits. https://www.youtube.com/watch?v=g8zdumOAWzw. Accessed April 2025.