Transformers Learn Shortcuts to Automata

https://arxiv.org/abs/2210.10749

Bingbin

Jordan T. Ash

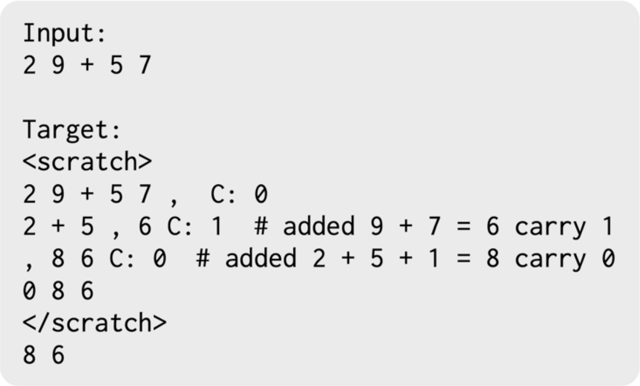
Surbhi Goel

Akshay Krishnamurthy

Cyril Zhang

Algorithmic reasoning

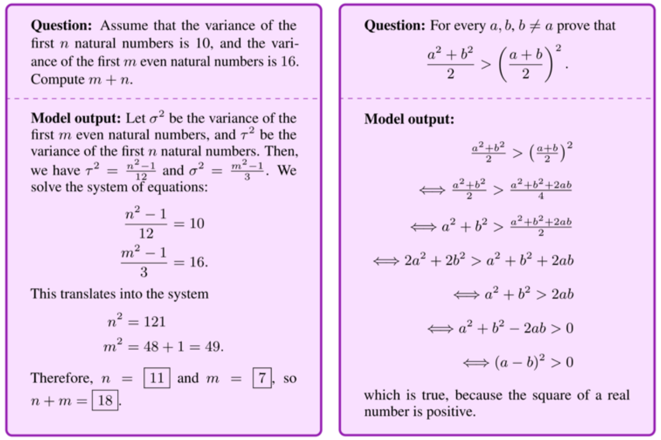
Want to understand the reasoning capabilities & internal mechanisms of self-attention models.



```
Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 ,  C: 0
2 + 5 , 6 C: 1  # added 9 + 7 = 6 carry 1
, 8 6 C: 0  # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6
```

Scratchpad (Nye et al. 22)



Question: Assume that the variance of the first n natural numbers is 10, and the variance of the first m even natural numbers is 16. Compute $m + n$.

Model output: Let σ^2 be the variance of the first m even natural numbers, and τ^2 be the variance of the first n natural numbers. Then, we have $\tau^2 = \frac{n^2-1}{12}$ and $\sigma^2 = \frac{m^2-1}{3}$. We solve the system of equations:

$$\frac{n^2-1}{12} = 10$$
$$\frac{m^2-1}{3} = 16.$$

This translates into the system

$$n^2 = 121$$
$$m^2 = 48 + 1 = 49.$$

Therefore, $n = \boxed{11}$ and $m = \boxed{7}$, so $n + m = \boxed{18}$.

Question: For every $a, b, b \neq a$ prove that

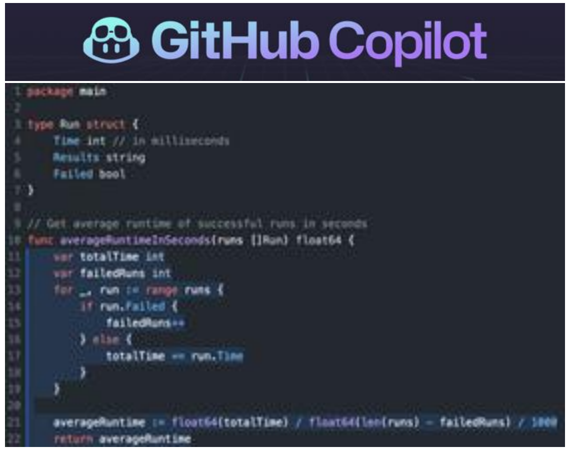
$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2.$$

Model output:

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2$$
$$\iff \frac{a^2+b^2}{2} > \frac{a^2+b^2+2ab}{4}$$
$$\iff a^2+b^2 > \frac{a^2+b^2+2ab}{2}$$
$$\iff 2a^2+2b^2 > a^2+b^2+2ab$$
$$\iff a^2+b^2 > 2ab$$
$$\iff a^2+b^2-2ab > 0$$
$$\iff (a-b)^2 > 0$$

which is true, because the square of a real number is positive.

Math: Minerva
(Lewkowycz et al. 22)



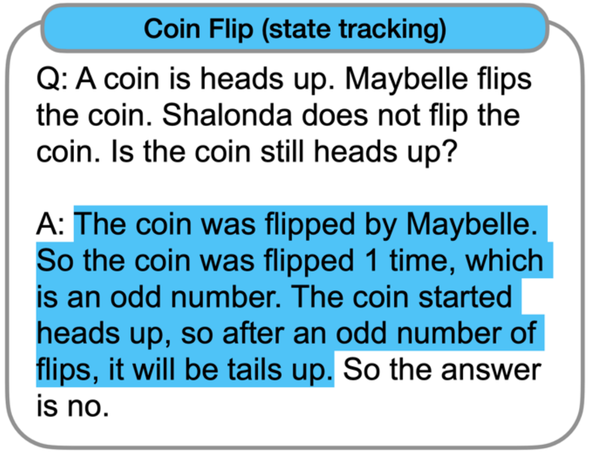
```
package main

type Run struct {
    Time int // in milliseconds
    Results string
    Failed bool
}

// Get average runtime of successful runs in seconds
func averageRuntimeInSeconds(runs []Run) float64 {
    var totalTime int
    var failedRuns int
    for _, run := range runs {
        if run.Failed {
            failedRuns++
        } else {
            totalTime += run.Time
        }
    }

    averageRuntime := float64(totalTime) / float64(len(runs) - failedRuns) / 1000
    return averageRuntime
```

Code: Codex / Copilot
(Chen et al. 21)



Coin Flip (state tracking)

Q: A coin is heads up. Maybelle flips the coin. Shalonda does not flip the coin. Is the coin still heads up?

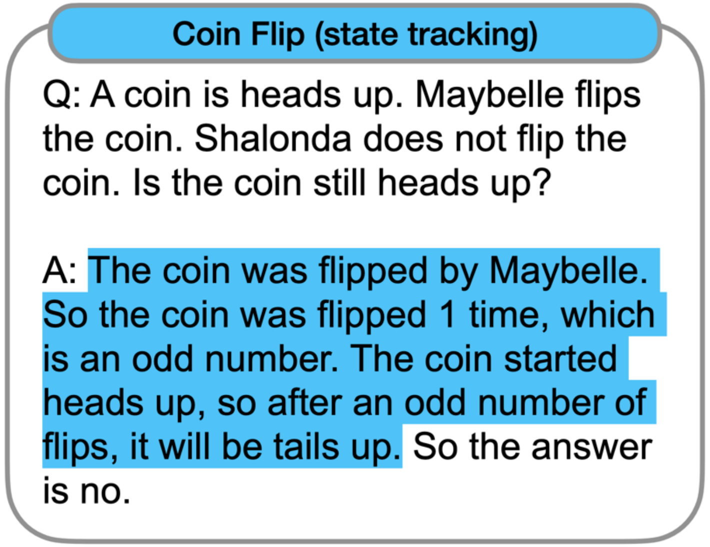
A: The coin was flipped by Maybelle. So the coin was flipped 1 time, which is an odd number. The coin started heads up, so after an odd number of flips, it will be tails up. So the answer is no.

Chain-of-Thought (Wei et al. 21)

Algorithmic reasoning

Coin flip = **parity**

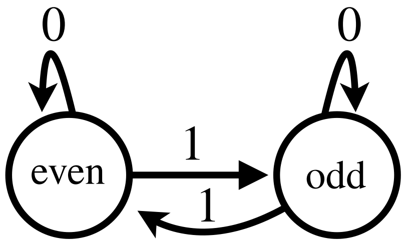
- Input (flip or not): $\Sigma = \{1, 0\}$.
- State (head/even or tail/odd): $Q = \{0, 1\}$.



Coin Flip (state tracking)

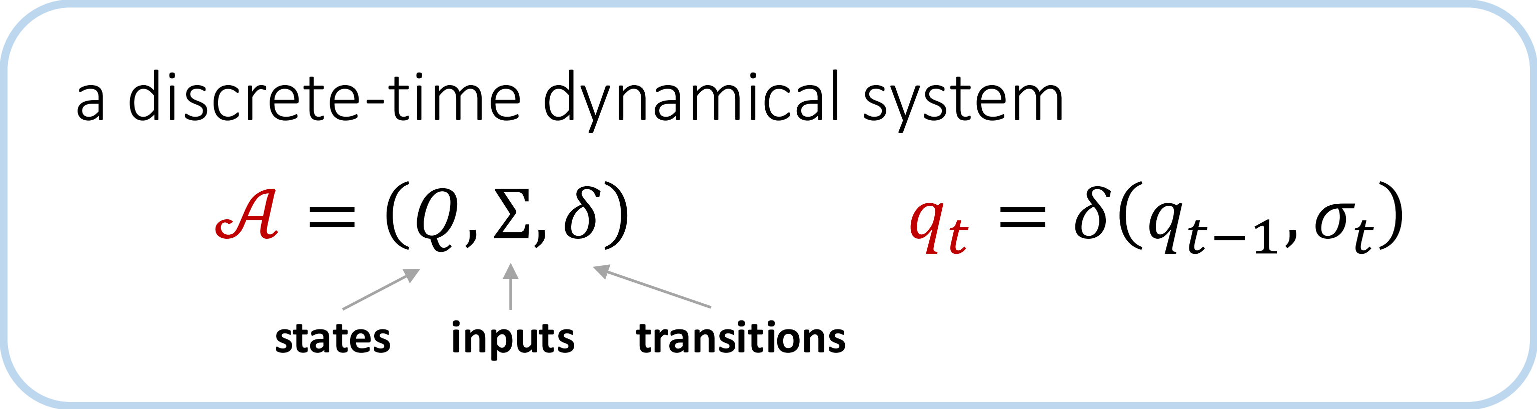
Q: A coin is heads up. Maybelle flips the coin. Shalonda does not flip the coin. Is the coin still heads up?

A: The coin was flipped by Maybelle. So the coin was flipped 1 time, which is an odd number. The coin started heads up, so after an odd number of flips, it will be tails up. So the answer is no.



$Q = \{\text{even}, \text{odd}\}$

$\Sigma = \{0, 1\}$



a discrete-time dynamical system

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

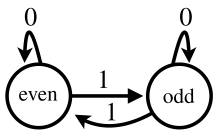
states **inputs** **transitions**

(Semi)Automata

- **Semiautomaton $\mathcal{A}$**: a discrete-time dynamical system

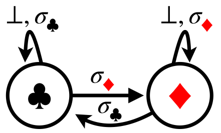
$$\mathcal{A} = (Q, \Sigma, \delta) \qquad\qquad q_t = \delta(q_{t-1}, \sigma_t)$$

states **inputs** **transitions**



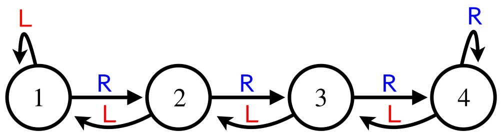
$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

parity counter



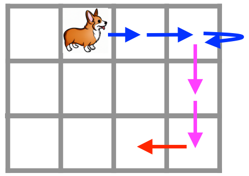
$Q = \{\clubsuit, \diamondsuit\}$
$\Sigma = \{\sigma_\clubsuit, \sigma_\diamondsuit, \perp\}$

1-bit memory unit



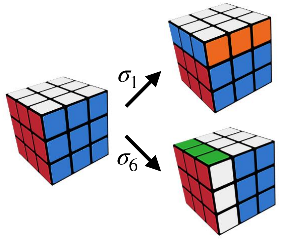
$Q = \{1, 2, 3, 4\}$
$\Sigma = \{\mathsf{L}, \mathsf{R}\}$

1D gridworld



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

2D gridworld



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

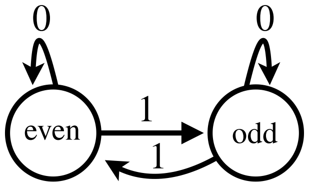
Rubik's Cube

- **Automaton:** $(Q, \Sigma, \delta, \varphi)$, outputs $\tilde{q}_t = \varphi(q_t)$ (acceptance function)

Simulating automata

(Semi)automata: a discrete-time dynamical system

$$\mathcal{A} = (\underset{\textbf{states}}{Q}, \underset{\textbf{inputs}}{\Sigma}, \underset{\textbf{transitions}}{\delta}) \qquad q_t = \delta(q_{t-1}, \sigma_t)$$



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

e.g. **parity counter**

- Task: **simulate** $\mathcal{A}$: learn a seq2seq function for length T.
  - Input: sequence of tokens $\sigma_1, \sigma_2, \cdots, \sigma_T \in \Sigma$.
  - Output: sequence of states $q_1, q_2, \cdots, q_T \in Q$.

e.g. parity counter: $\{\sigma_t\} = 01101$, $\{q_t\} = 01001$.

Simulating automata with Transformers

🤔 *Can Transformers learn automata?*

- Automata is *recurrent*; Transformers are *non-recurrent* *(parallel)* and shallow.

Yes – **Shortcut**: solutions with $o(T)$ sequential “steps/rounds”.

- Def: longest path in a computation graph (i.e. at most T).
- e.g. recurrent nets: sequence length.
- e.g. Transformers: number of layers.

Simulating automata with Transformers

🤔 *Can Transformers learn automata?*

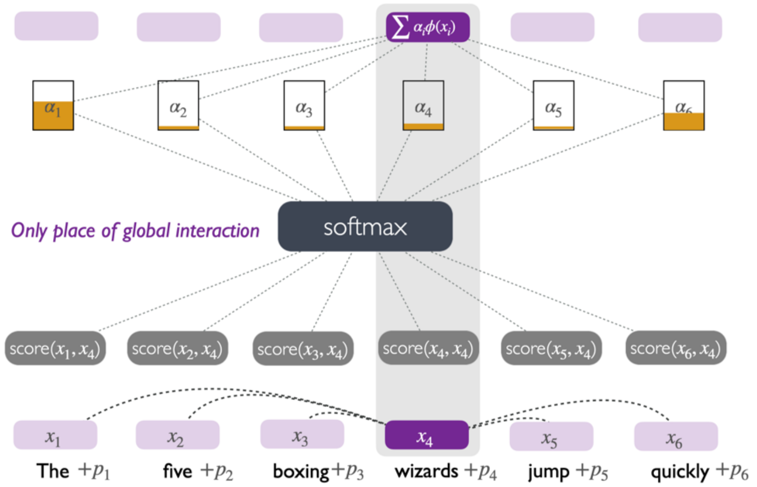
- Automata is *recurrent*; Transformers are *non-recurrent (parallel)* and shallow.

Yes – **Shortcut**: solutions with $o(T)$ sequential “steps/rounds”.

  - Def: longest path in a computation graph (i.e. at most T).
  - e.g. recurrent nets: sequence length.
  - e.g. Transformers: number of layers.

Transformer recap

- $\forall i \in [T], x_i^{(l+1)} = \sum_i \alpha_{ij}\, x_j^{(l)}$;
- $\alpha_{ij} \propto \exp(\langle W_Q x_i^{(l)}, W_K x_j^{(l)} \rangle)$.



Simulating automata with Transformers

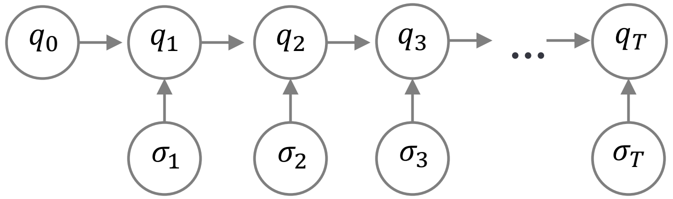
🤔 *Can Transformers learn automata?*

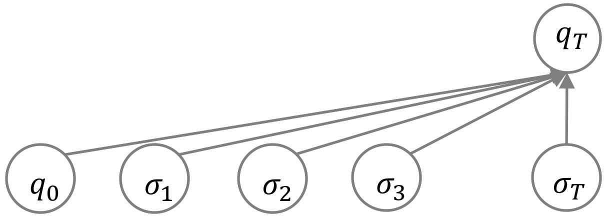
- Automata is *recurrent*; Transformers are *non-recurrent (parallel)* and shallow.

Yes – **Shortcut**: solutions with $o(T)$ sequential “steps/rounds”.

Example: parity (prefix sum): $q_t, \forall t \in [n]$,

- **Iterative solution**:
  $q_t = q_{t-1} \oplus \sigma_t \in \{0,1\}$.
- **Shortcut** (parallel) **solution**:
  $q_t = (\sum_{\tau \leq t} \sigma_\tau) \bmod 2 \in \{0,1\}$.

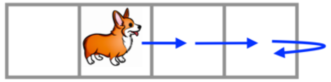




Why shortcuts:
- computational advantage;
- Unique & natural to Transformer.

Transformers Learn Shortcuts to Automata

TL;DR: Shallow Transformers can simulate $\mathcal{A} = (Q, \Sigma, \delta)$.

- **Theory:** for any length T, Transformers with $o(T)$ layers suffice.
  - $O(\log T)$-layer simulation for *all* $\mathcal{A}$ (also the lower bound for the general case)
  - $O(|Q|^2 \log |Q|)$-layer simulation for all *solvable* $\mathcal{A}$
  - $O(1)$-layer simulation for gridworld 

- **Empirical study**
  - Positive: training shallow models SGD finds shortcuts in practice.
  - Negative: they're brittle OOD (distr on input symbols; other lengths).
  - Fix: *Scratchpad* training: force a Transformer to learn the recurrent solution.

- **Discussions** 💡

Theory (as brain teasers)

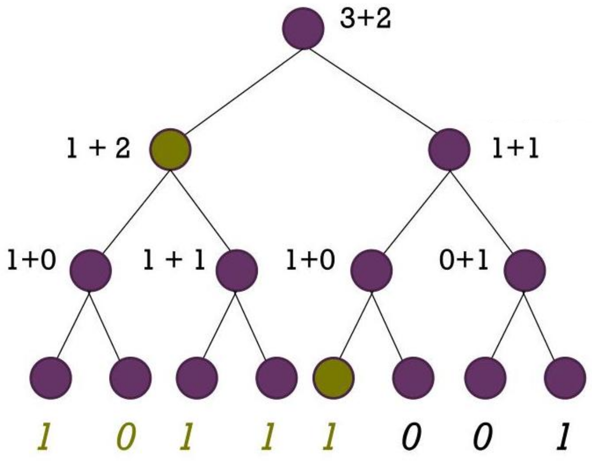
Solutions with $o(T)$ computation depth

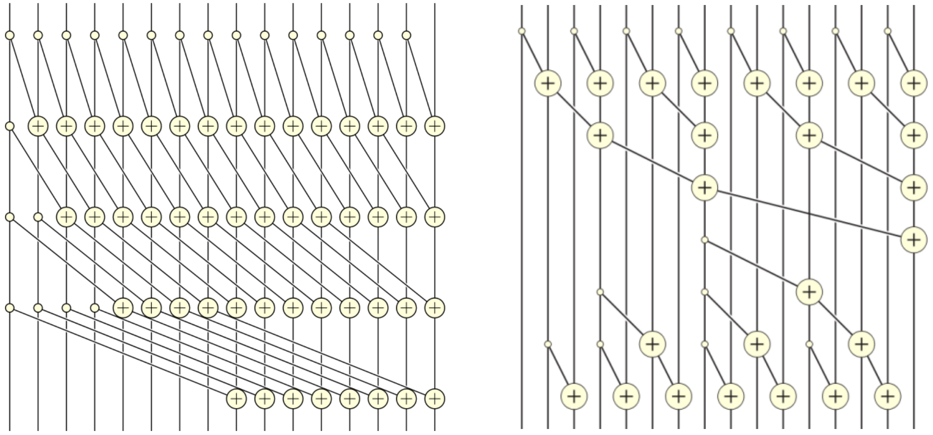
Puzzle 1: Do shortcuts exist?

$\mathcal{A} = (Q, \Sigma, \delta)$: (states, inputs, transitions)

Task: for each $t \in [T]$, compute $q_t = \boxed{\big(\delta(\cdot, \sigma_t) \circ \cdots \circ \delta(\cdot, \sigma_1)\big)}(q_0)$.

- Input token $\sigma \to$ a function $\delta(\cdot, \sigma): Q \to Q$.
- Function composition (associative).





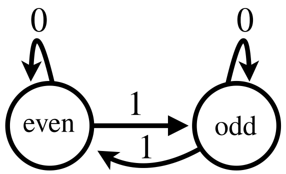
Parallel prefix sum (fig from Wikipedia)

Puzzle 1: Do shortcuts exist?

$\mathcal{A} = (Q, \Sigma, \delta)$: (states, inputs, transitions)

Task: for each $t \in [T]$, compute $q_t = \boxed{\left(\delta(\cdot, \sigma_t) \circ \cdots \circ \delta(\cdot, \sigma_1)\right)} (q_0)$.

- Input token σ → a function $\delta(\cdot, \sigma) \colon Q \to Q$.
- Function composition (associative) ⟷ matrix multiplication



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

parity counter

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

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

function composition

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

matrix multiplication

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

Representation theory (Chughtai 23): can explain $O(\log T)$, but not what's coming up.

Can we use $o(\log T)$ layers?

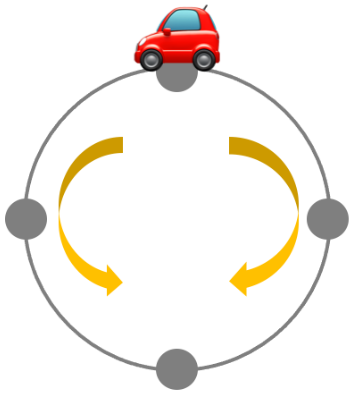
$\mathcal{A} = (Q, \Sigma, \delta)$: (states, inputs, transitions)

We already have positive result on some tasks.

- Parity: $q_t = \left(\sum_{i \in [t]} \sigma_i\right) \bmod 2$: only need to count #1s.
- Counting suffices, if the function composition is commutative.
  - Corollary: any commutative compositions (abelian groups--explained later) can be represented with $O(1)$ layers.

Question: Is there a solution with $o(\log T)$ layers when *non-commutative*?

Puzzle 2: $\tilde{O}(|Q|^2)$ layers



Reversible car on a circular road:

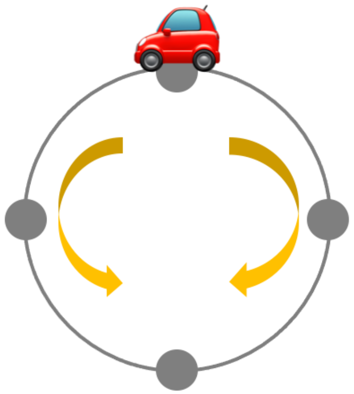
- $\Sigma = \{D, U\}$ (drive, U-turn), $Q = \{$🚗 , 🚘$\} \times \{0,1,2,3\}$.
- Consider input $DDDUDDUUD$…., starting from (🚗 , 0).
- How to decide the current direction and position?
  - Direction = a parity task on U. (parity: {1, -1} ↔ {0, 1})
  - Position = sum of signed counts (sign = parity of U) mod 4.

D D D U D D U U D

Parity: 1 1 1 -1 -1 -1 1 -1 -1 → 🚘

Signed counts: 1 1 1 0 -1 -1 0 0 -1 → 0

Solution 2: $\tilde{O}(|Q|^2)$ layers

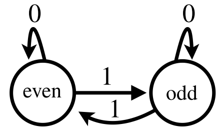


Reversible car on a circular road:

- $\Sigma = \{D, U\}$ (drive, U-turn), $Q = \{0,1,2,3\} \times$ 🚗 , 🚗}.
- Decompose: 1) direction (parity), 2) position (signed sum mod 4).

Is such decomposition always possible? *Yes!*

Transformation group: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition.



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

parity counter

$\xrightarrow{\mathcal{T}(\mathcal{A})}$

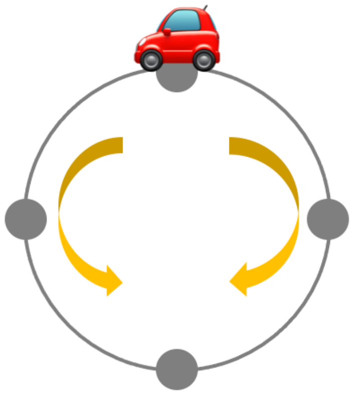
	0	1
0	0	1
1	1	0

cyclic group C_2

Group G: a set G with operation $G \times G \to G$.

- Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Identity: $a \cdot e = e \cdot a = a$
- Inverse: $\forall a \in G, \exists b \in G$ s.t. $a \cdot b = b \cdot a = e$

Solution 2: $\tilde{O}(|Q|^2)$ layers



Reversible car on a circular road:

- $\Sigma = \{D, U\}$ (drive, U-turn), $Q = \{0,1,2,3\} \times \{$🚘 , 🚗$\}$.
- Decompose: 1) direction (parity), 2) position (signed sum mod 4).

Is such decomposition always possible? *Yes!*

Transformation group: $\mathcal{T}(\mathcal{A}) \coloneqq \{\delta(\cdot, \sigma) : \sigma \in \Sigma\}$ under composition.

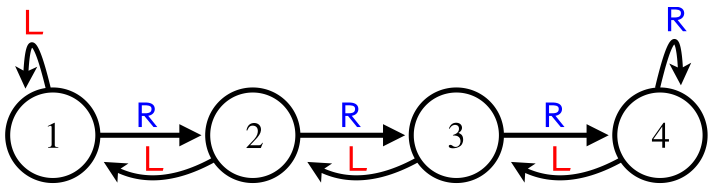
"Prime factorization" for groups: $G \rhd H_n \rhd \cdots \rhd H_1$ (Jorden & Hölder [~1880])

- $G \rhd H$: H is a normal subgroup of G (~factors).
- H_{i+1}/H_i are simple groups (~prime numbers).

What about *semi*groups?

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

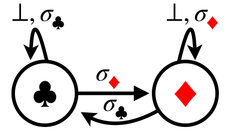
1d gridworld: σ_t = L/R steps, q_t = location in a bounded room.



$$q_0 = 1, \qquad q_t = \min\big(n, \max(1, q_{t-1} + 1)\big)$$

e.g. LRLLRRRRLL ↦ 1, 2, 1, 1, 2, 3, 4, 4, 3, 2

- $\delta(\cdot, L) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ is not invertible → $\mathcal{T}(\mathcal{A})$ is a *semigroup*.



$Q = \{\clubsuit, \diamondsuit\}$

$\Sigma = \{\sigma_\clubsuit, \sigma_\diamondsuit, \perp\}$

1-bit memory unit

$\xrightarrow{\mathcal{T}(\mathcal{A})}$

	$\sigma_\diamondsuit$	$\sigma_\clubsuit$	$\perp$
$\sigma_\diamondsuit$	$\sigma_\diamondsuit$	$\sigma_\diamondsuit$	$\sigma_\diamondsuit$
$\sigma_\clubsuit$	$\sigma_\clubsuit$	$\sigma_\clubsuit$	$\sigma_\clubsuit$
$\perp$	$\sigma_\diamondsuit$	$\sigma_\clubsuit$	$\perp$

flip-flop monoid

Semigroup G: a generalization of group.

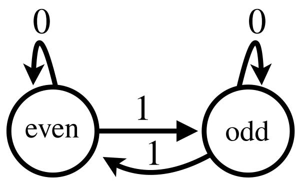
- Associativity.
- (+ Identity: a *monoid*.)

Solution 2: $\tilde{O}(|Q|^2)$ layers

$\mathcal{A} = (Q, \Sigma, \delta)$: (states, inputs, transitions)

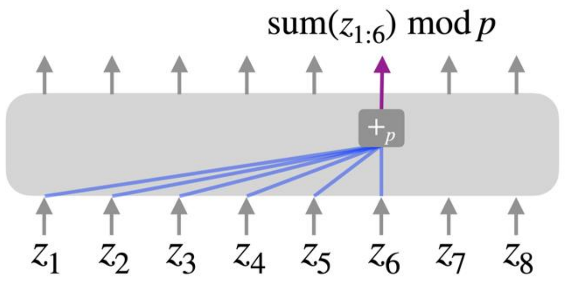
1d gridworld: σ_t = L/R steps, q_t = location in a bounded room.

- *Krohn-Rhodes* (Thm 2): solvable semiautomata decomposed into mod + reset.
  - *Solvable: H_{i+1}/H_i is commutative/abelian. [Recall Jorden & Hölder: $G \rhd H_n \rhd \cdots \rhd H_1$]*

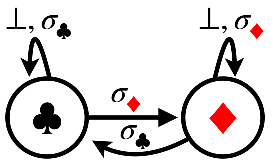


$\delta(q, \sigma) = q + \sigma \bmod S$

parity counter

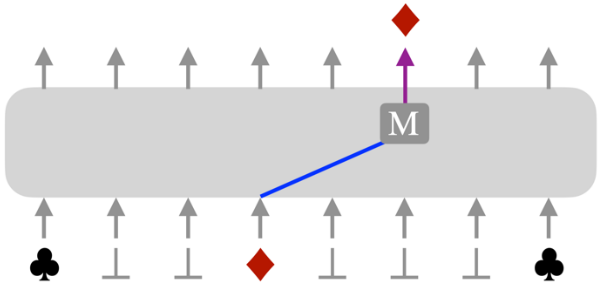


uniform attention



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

memory unit

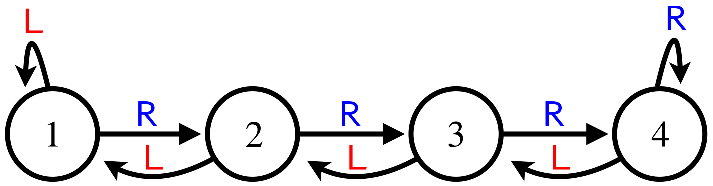


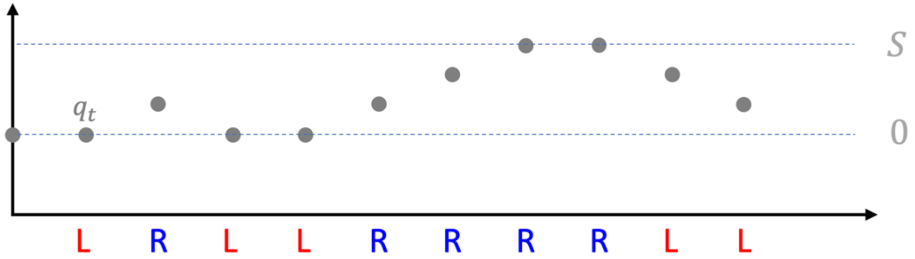
sparse attention

**We do not claim that Transformers learn these decompositions in practice.*

Puzzle 3: $O(1)$ layers?

1d gridworld: σ_t = L/R steps, q_t = location in a bounded room.

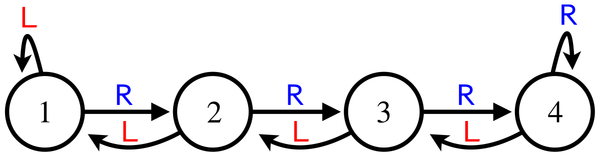




Puzzle: design a parallel algorithm to compute $\sigma_{1:T} \mapsto q_{1:T}$.

- Hint: *boundary detection*: discard history; easy afterwards (prefix sum).

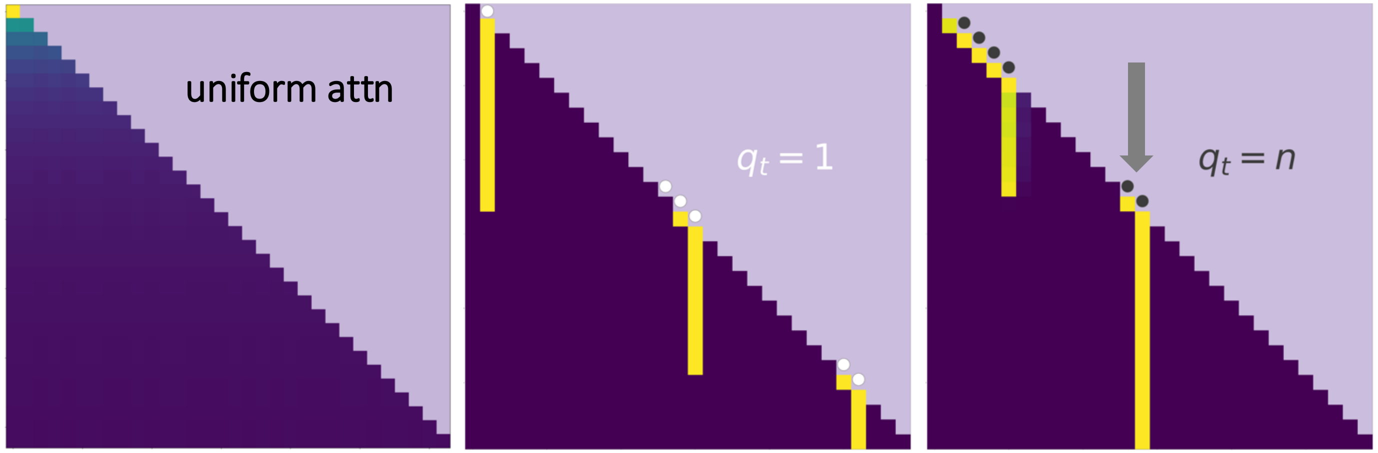
Solution 3: $O(1)$ layer for



~~“You can only figure out whether you’re at a wall if you know q_{t-1}.”~~

- Parallel boundary detection!

  - discard history; easy afterwards (prefix sum).



1st layer

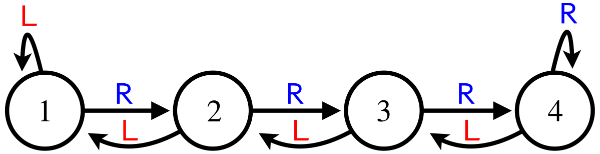
4th layer, left boundary

4th layer, right boundary

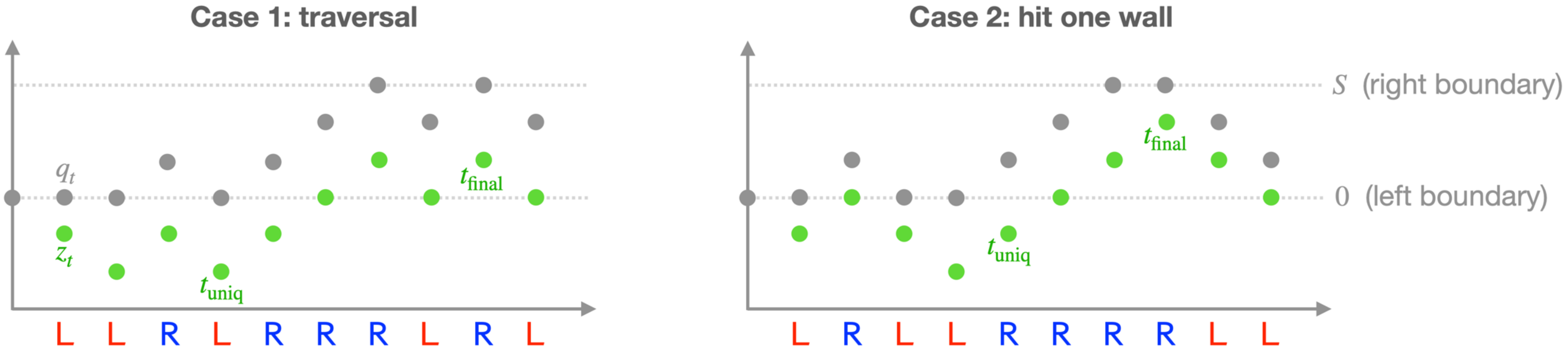
attention heatmaps

(GPT solved this before we did o o)

“mechanistic interpretability”
[Nanda & Lieberum ‘22]

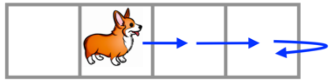
Solution 3: $O(1)$ layer for 

- **Parallel boundary detector**:
  - Compute prefix sums $z_t \coloneqq \sum \sigma_{1:t}$ (ignoring boundaries);
  - At each t, find most recent $t_{\mathrm{uniq}} < t$ such that $z_{t_{\mathrm{uniq}}:t}$ has n(#states) unique values;
  - Then $t_{\mathrm{final}} \coloneqq \max\left(\operatorname*{argmax}_{t_{\mathrm{uniq}} \le \tau \le t} z_\tau, \operatorname*{argmin}_{t_{\mathrm{uniq}} \le \tau \le t} z_\tau\right)$ is last boundary collision.



Takeaways

For any length T, Transformers can simulate $\mathcal{A}$ with $o(T)$ depth.

- Theorem 1: $O(\log T)$-layer simulation for *all $\mathcal{A}$*
  - Parallel prefix computation (divide and conquer); lower bound for general $\mathcal{A}$.
- Theorem 2: $O(|Q|^2 \log |Q|)$-layer simulation for all *solvable $\mathcal{A}$*
  - Factorization (Krohn-Rhodes).
- Theorem 3: $O(1)$-layer simulation for gridworld 
  - Specific $\mathcal{A}$: even shallower results (e.g. via parallel boundary detector).

Empirical findings

Positive results, challenges

Overview

$\sigma_{1:T}$ → Transformer → $q_{1:T}$

(Theory)

- Q0: are shallow non-recurrent nets sufficiently expressive? [Yes!]

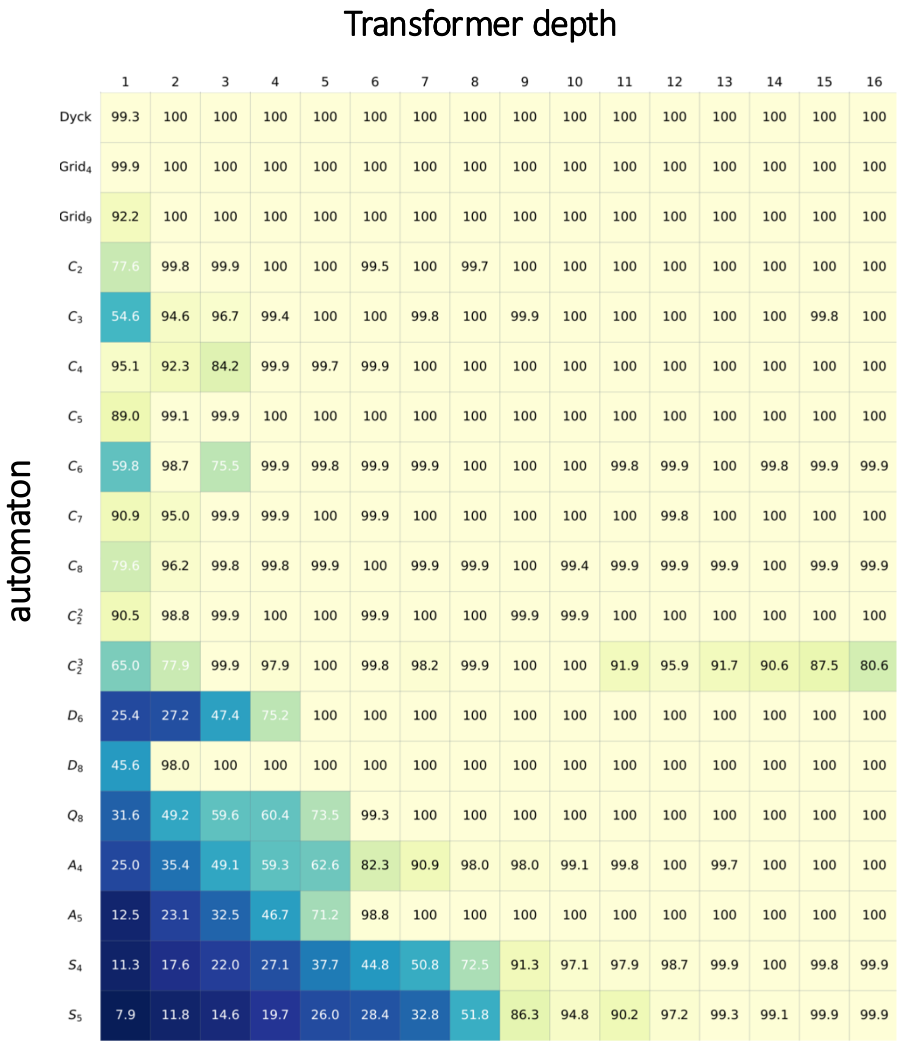
(Experiments)

- Q1: Can SGD find shortcuts in practice?
- Q2: Shortcomings of shortcuts?
  - Q2.1: Does SGD work with limited supervision?
  - Q2.2: Are shortcuts robust to unseen inputs?
- Q3: Solutions?

(Open question)

- Q4: How to interpret the shortcuts?
- Q5: How to design an architecture to *solve* reasoning tasks?

SGD works, under ideal supervision



Transformer depth

automaton	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Dyck	99.3	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Grid_4	99.9	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Grid_9	92.2	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
C_2	77.6	99.8	99.9	100	100	99.5	100	99.7	100	100	100	100	100	100	100	100
C_3	54.6	94.6	96.7	99.4	100	100	99.8	100	99.9	100	100	100	100	100	99.8	100
C_4	95.1	92.3	84.2	99.9	99.7	99.9	100	100	100	100	100	100	100	100	100	100
C_5	89.0	99.1	99.9	100	100	100	100	100	100	100	100	100	100	100	100	100
C_6	59.8	98.7	75.5	99.9	99.8	99.9	99.9	100	100	100	99.8	99.9	100	99.8	99.9	99.9
C_7	90.9	95.0	99.9	99.9	100	99.9	100	100	100	100	100	99.8	100	100	100	100
C_8	79.6	96.2	99.8	99.8	99.9	100	99.9	99.9	100	99.4	99.9	99.9	99.9	100	99.9	99.9
C_2^2	90.5	98.8	99.9	100	100	99.9	100	100	99.9	99.9	100	100	100	100	100	100
C_2^3	65.0	77.9	99.9	97.9	100	99.8	98.2	99.9	100	100	91.9	95.9	91.7	90.6	87.5	80.6
D_6	25.4	27.2	47.4	75.2	100	100	100	100	100	100	100	100	100	100	100	100
D_8	45.6	98.0	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Q_8	31.6	49.2	59.6	60.4	73.5	99.3	100	100	100	100	100	100	100	100	100	100
A_4	25.0	35.4	49.1	59.3	62.6	82.3	90.9	98.0	98.0	99.1	99.8	100	99.7	100	100	100
A_5	12.5	23.1	32.5	46.7	71.2	98.8	100	100	100	100	100	100	100	100	100	100
S_4	11.3	17.6	22.0	27.1	37.7	44.8	50.8	72.5	91.3	97.1	97.9	98.7	99.9	100	99.8	99.9
S_5	7.9	11.8	14.6	19.7	26.0	28.4	32.8	51.8	86.3	94.8	90.2	97.2	99.3	99.1	99.9	99.9

Q1: Does SGD on Transformers find shortcuts? *Yes!*

Group-free semiautomata (reset): Dyck, gridworld

- Easy to learn; generalizes Dyck results [Yao et al. '22]

Cyclic groups (mod-*n* counters): unstable training & o.o.d. eval

- possibly accounts for previous negative results [Bhattamishra et al. '20]
- **open challenge:** improve architectures and training.

Other abelian groups: worse instabilities; higher depth helps training

Harder groups: more results including D_8, A_5 (non-solvable), S_5 (NC^1-complete).

- Groups with deeper factorization requires mores layers
- **open challenge 1:** dissect/interpret the learned solutions
- **open challenge 2:** precisely characterize instance complexity

Training robustness: SGD beyond ideal supervision

Q2.1: Does SGD work under limited supervision? *Up to a point.*

- *Setup*: reduce the amount of state info during training; in-distribution eval.

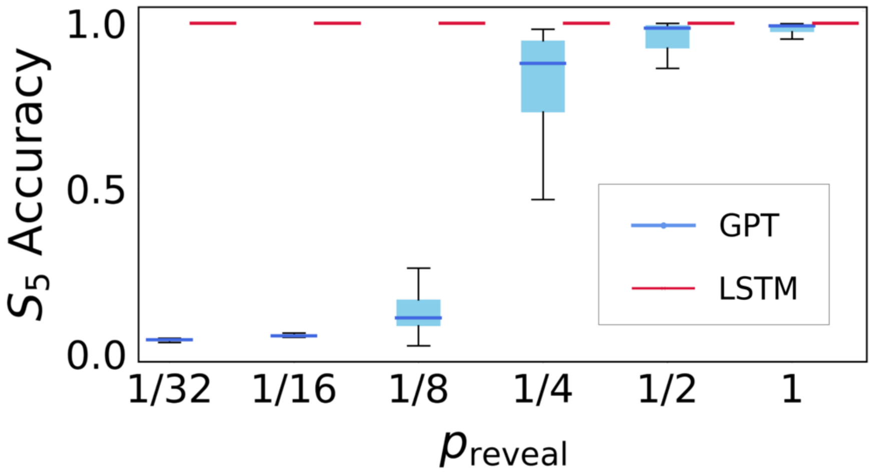
Indirect supervision

train & test on a function of q_t.

$\text{Dyck}_{4,8}$	Grid_9	S_5	C_4	D_8
stack top	$\mathbb{1}_{\text{boundary}}$	$\pi_{1:t}(1)$	$\mathbb{1}_{0 \bmod 4}$	location
100.0	99.8	99.8	99.7	99.8

Incomplete supervision

q_t is revealed w.p. $p_{\text{reveal}} \in [0,1]$.

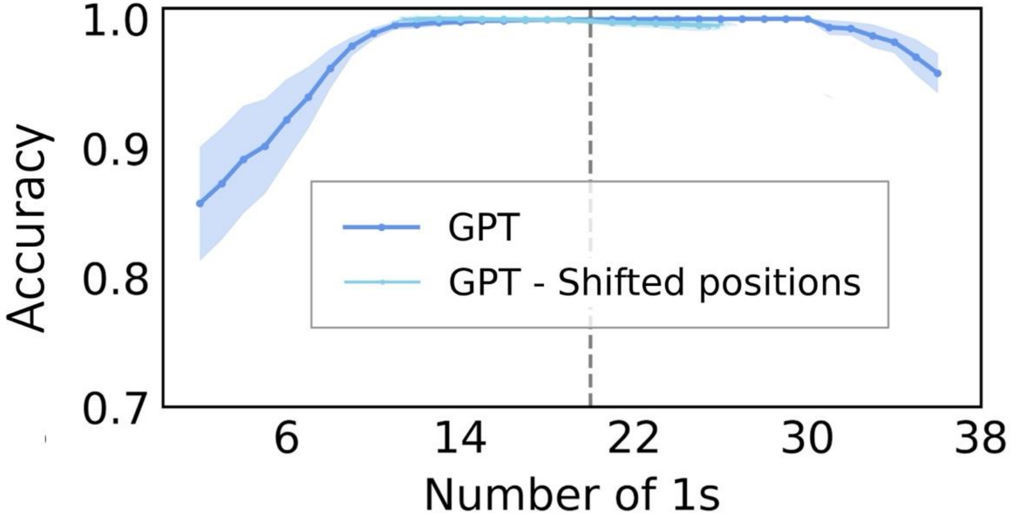


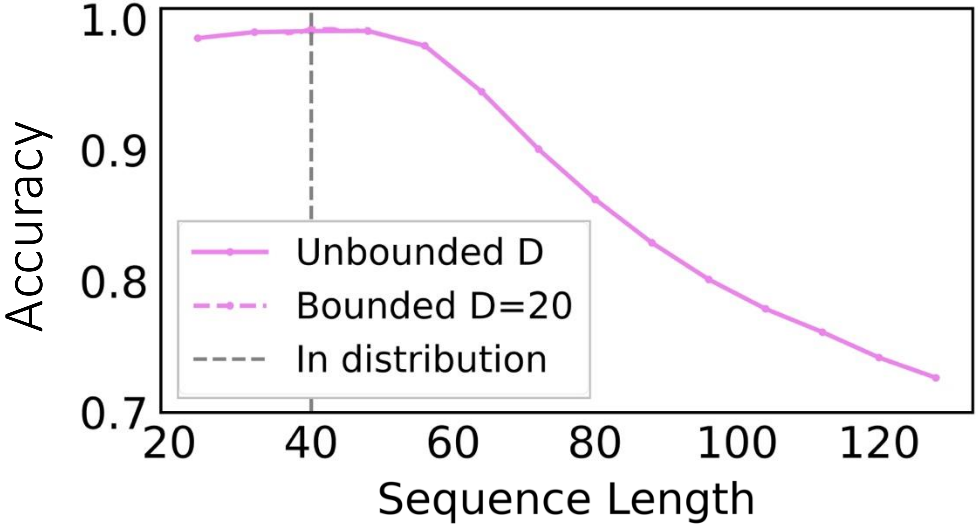
Takeaway: learning is still possible, but not as robust as RNNs (LSTM always 100%).

Testing robustness: hallucinated variables

Q2.2: Are shortcuts robust to unseen inputs? No.

- **Parity** (C_2): ideal training with $p(1) = 0.5$, evaluate under distribution shifts.
  - Shortcut $q_t = \left(\sum_{i\in[t]} \sigma_i\right) \bmod 2 \rightarrow \left(\sum_{i\in[t]} \sigma_i\right) \approx t/2$ with deviation $\sim\sqrt{t}$.
  - $\bmod 2$ is memorized by MLP: fail to generalize to diff $\left(\sum_{i\in[t]} \sigma_i\right)$.
- Transformer solutions? Same failure mode → maybe same solution.

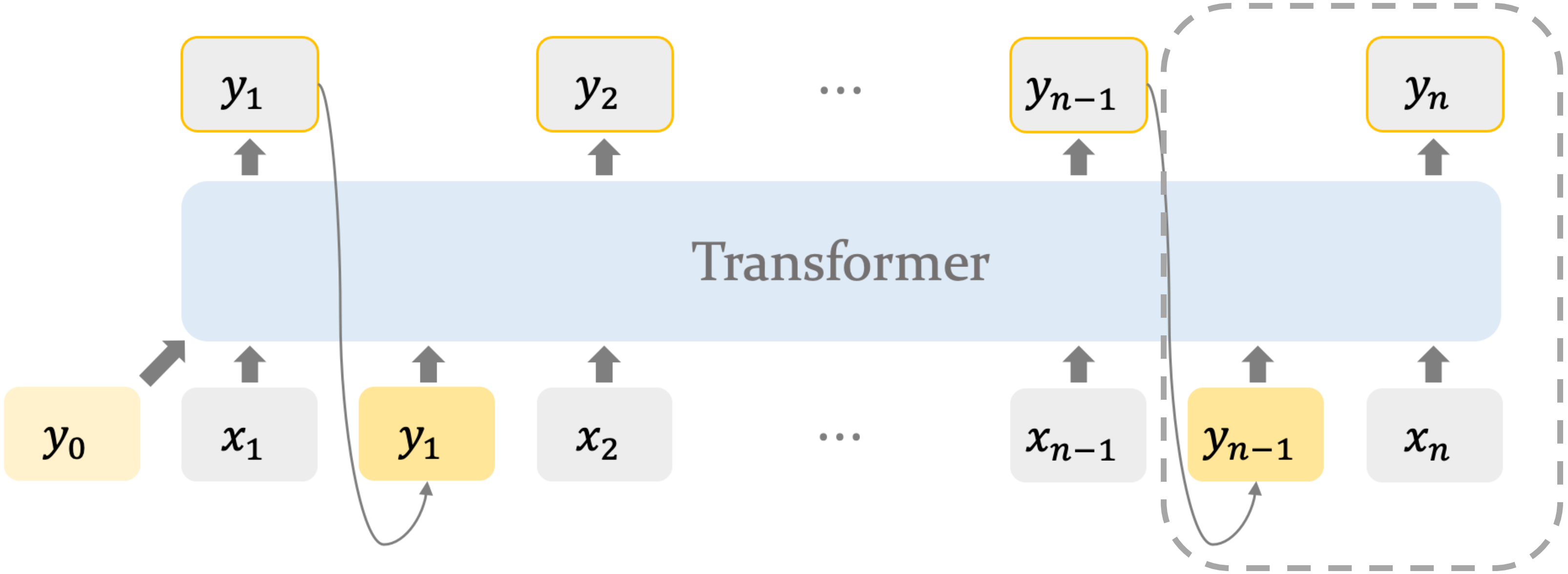




The recurrent mode of Transformers

Q3: Solutions? Guiding Transformers to learn recurrent solutions.

- *Setup*: train with previous states as inputs;



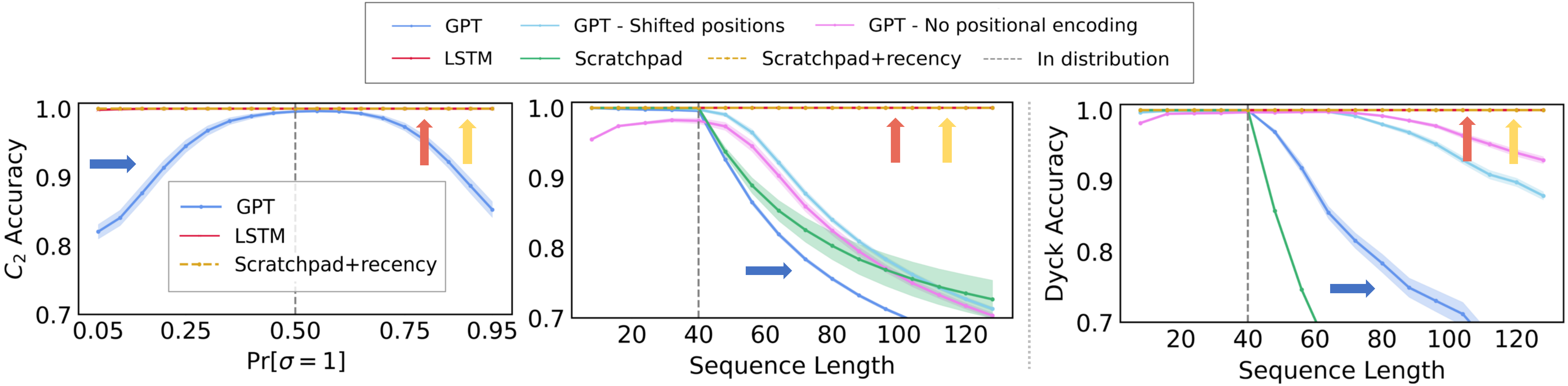
$q_t = \delta(q_{t-1}, \sigma_t)$

Scratchpad (Nye et al. 21)
(input $x_t = \sigma_t$, output $y_t = q_t$)

The recurrent mode of Transformers

Q3: Solutions? Guiding Transformers to learn recurrent solutions.

- *Setup*: train with previous states as inputs; eval at diff distributions or lengths.



Takeaway: Transformers turned recurrent with scratchpad [Nye et al. 22] + recency bias [Press et al. 21] → Open: *Can we learn shortcuts that generalize?*

Takeaway

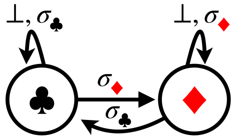
- Positive: Transformers with SGD can find good in-distribution solutions.
  - The trend roughly matches our beliefs on the difficulty of the tasks.

- Challenges: shortcuts are less robust than recurrent methods
  - Struggle at OOD generalization / changes in training setups.
    - Open question: How to thoroughly test for generalization?
  - Fix: made recurrent by scratchpad + recency bias.

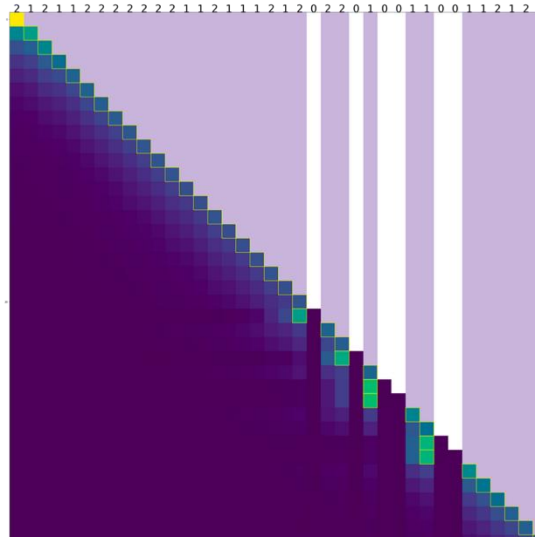
- Mechanistic understanding
  - Gridworld: boundary detection.
  - Parity: evidence for implementing $(\sum_t \sigma_t) \bmod 2$.

Discussions: Interpretability and generalization

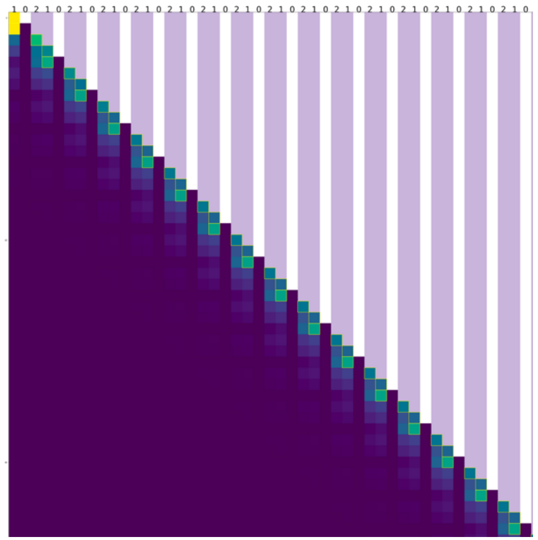
How to test for generalization? ∞ unseen inputs -- how much do we need to probe?

e.g. Flipflop  : train with $p(0) = 0.5, p(1) = p(2) = \frac{1-p(0)}{2}$.

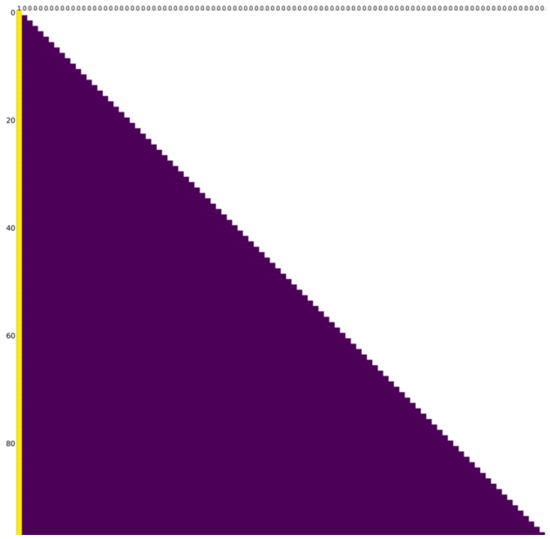
- Test: $p(0) \in \{0.1, 0.2, \dots, 0.9\}$ × (100k len-100 seqs): 100% accuracy



random with small $p(0)$



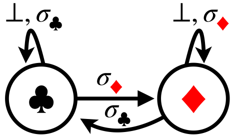
$[1,0,2] \cdot n$



$[1,0,0, \cdots]$

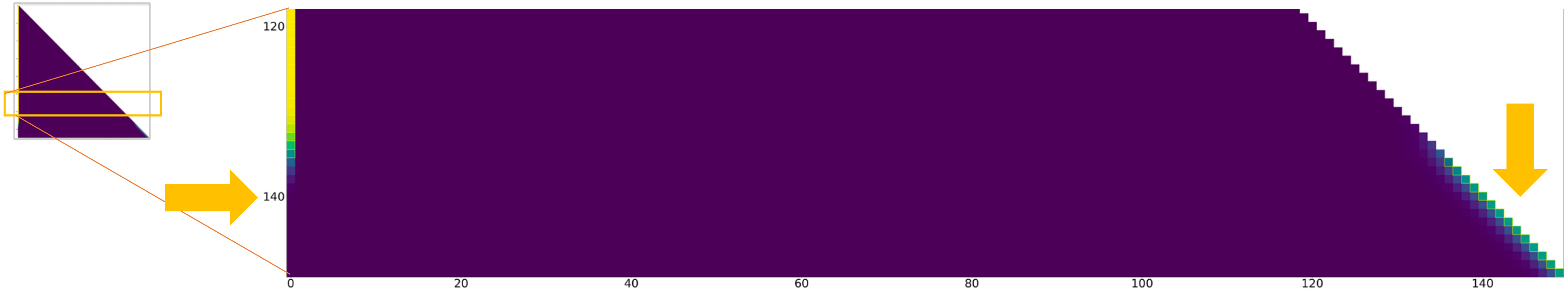
Discussions: Interpretability and generalization

How to test for generalization? ∞ unseen inputs -- how much do we need to probe?

e.g. Flipflop 

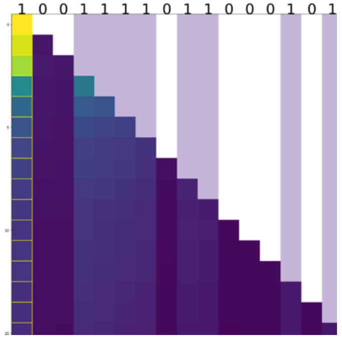
: train with $p(0) = 0.5, p(1) = p(2) = \frac{1-p(0)}{2}$.

- Test: 100% for $p(0) \in \{0.1, 0.2, \ldots, 0.9\}$, for 100k len-100 seqs per $p(0)$.
- $[1,0,0,0, \ldots, 0]$ is the hardest sample -- long-term dependency solved! ☺
- Not quite: failing on length generalization. ☹

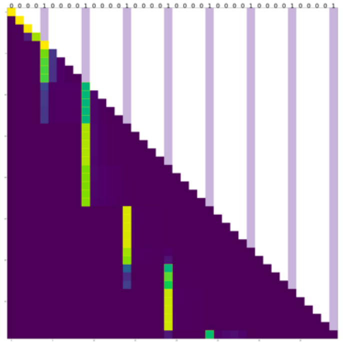


Discussions: (mechanistic) interpretability

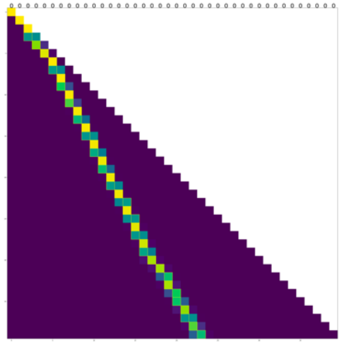
- Parity: what if not “sensible”? Variance (across randomness) in the patterns?



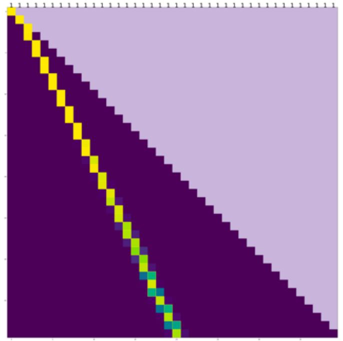
attend to 1s only



[0,0,0,1]*10

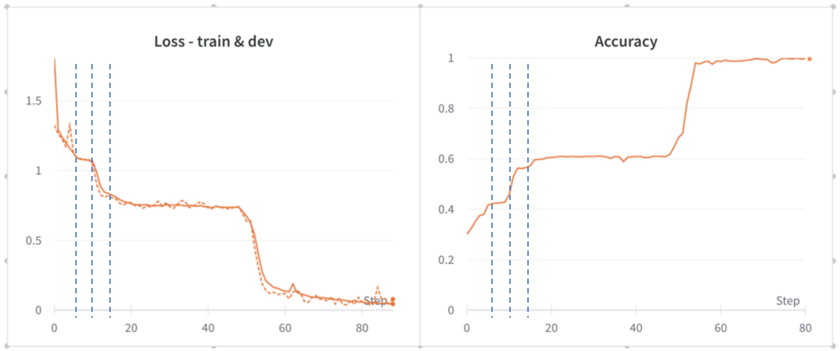


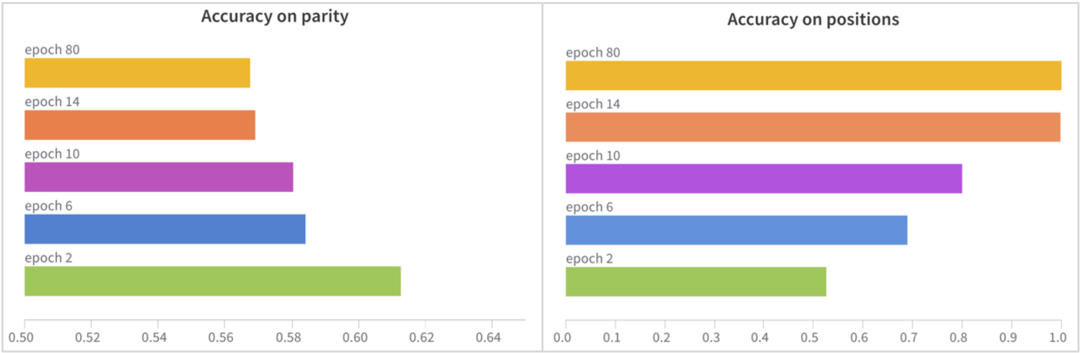
all-0s



all-1s

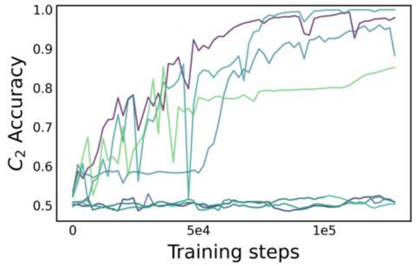
- Dihedral group : not learning the parity (on direction).



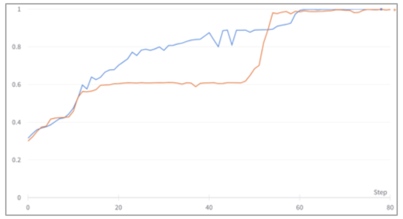


Discussions: optimization

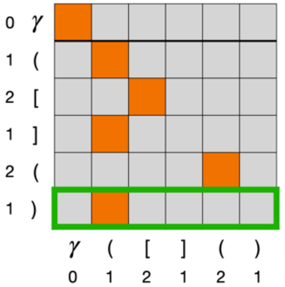
- Improve / stabilize training?
  - e.g. Parity: without positional encoding? With 1 layer?



- How to find (hidden) progress measures?
  - Barak 22, Nanda 23



- What solutions are preferred, e.g. among shortcuts?
  - Dyck: sparse in theory (Yao 20), uniform in practice.
  - Are some solutions *better* than others?



- Modified architecture: parity: no issue with mod if using $\sin(\sum \sigma_t)$.

  → How to optimize with sinusoidal activation?

Discussions

- **Generalization**
  - Is is possible to have *guarantees* via tests (e.g. a “complete” test set)?
  - Is interpretability required for true generalization?
- **Synthetic & real**
  - *Why synthetic*: clean setup + cheaper + infinite supply of data
    - Decomposition → understand basic units first
    - Easier to formalize & understand & diagnose & fix
  - *Why not synthetic*: not directly useful
    - What’s missing: factual aspects? “Complex” structures?
  - *Goal*: Transfer insights to code / math / languages.
- **Shortcut & recurrent**: best of both worlds? (RWKV?)

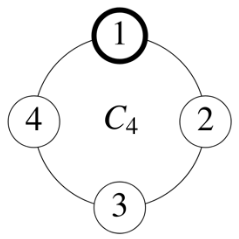
Transformers Learn Shortcuts to Automata

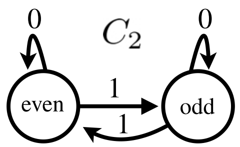
TL;DR: $o(T)$-depth Transformers can simulate $\mathcal{A}$, in theory and practice

- **Theory:** for any length T, Transformers can learn:
  - $O(\log T)$-layer for *all* $\mathcal{A}$ – parallel prefix computation
    - Non-solvable $\mathcal{A}$: matching lower bound (Barrington)
  - $O(|Q|^2 \log |Q|)$-layer for all *solvable* $\mathcal{A}$ – factorization with Krohn-Rhodes
  - $O(1)$-layer for gridworld – boundary detector
- **Empirical study:** shortcuts can be found but are brittle OOD.
  - Fix: make Transformers recurrent (scratchpad + recency bias).
- **Future work**: understanding and improving the model?

Appendix

Theorem 2: the glue

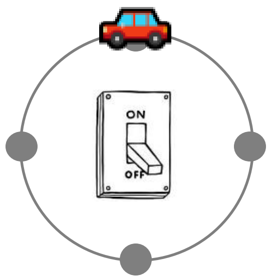




Direct product ×, e.g. 🦓 $C_4 \times C_2$

Two independent groups

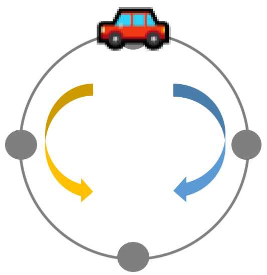
- $(g_1, h_1) \cdot (g_2, h_2) = (g_1 g_1, h_1 h_2)$
- e.g. car + a light switch

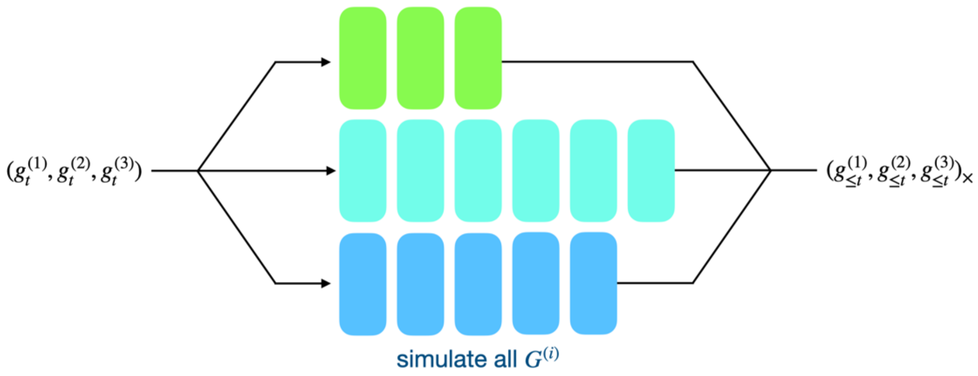


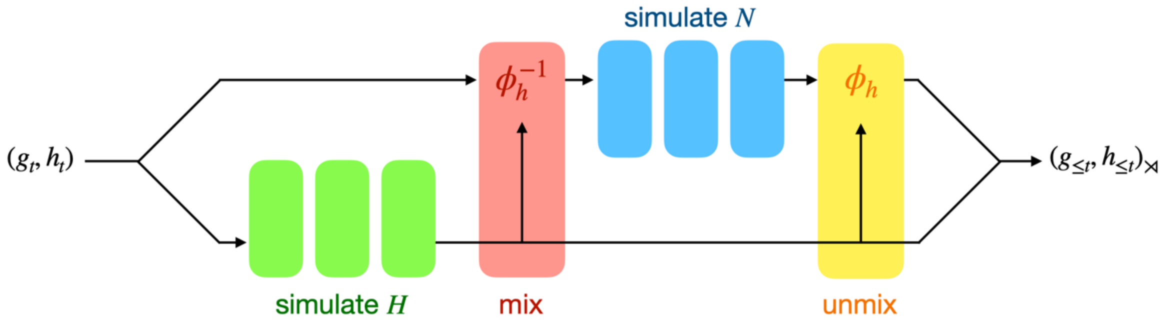
Semidirect product ⋊, e.g. 🦀 $D_8 \cong C_4 \rtimes C_2$

Two *interacting* groups

- $(g_1, h_1) \cdot (g_2, h_2) = (g_1 h_2 g_2 h_2^{-1}, h_1 h_2)$
- e.g. car + direction toggle




$(g_t^{(1)}, g_t^{(2)}, g_t^{(3)})$
$(g_{\le t}^{(1)}, g_{\le t}^{(2)}, g_{\le t}^{(3)})_\times$
simulate all $G^{(i)}$

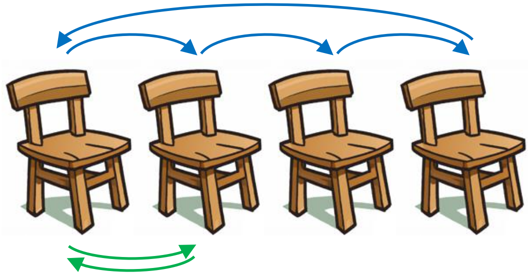


What about *semi*groups?

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

More complicated: rank collapses.

n-player musical chairs



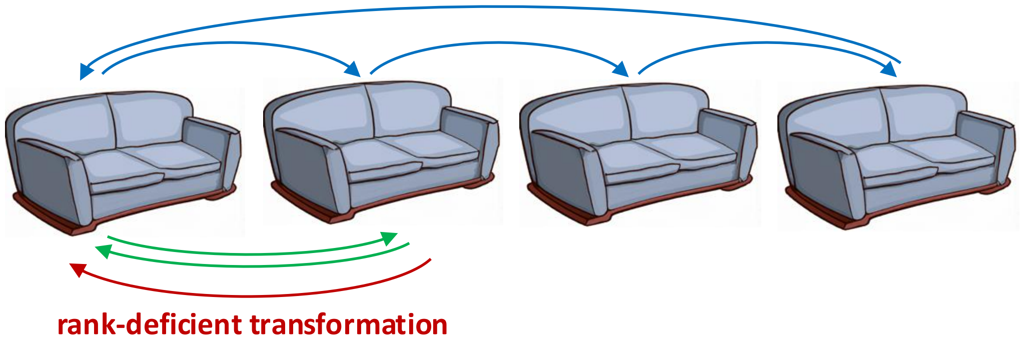
$Q = \{\text{positions of } n \text{ players}\}$

$\Sigma = \{\ \text{cycle},\ \ \text{swap}\ \}$

$\begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{bmatrix} \begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$

$\mathcal{T}(\mathcal{A}) = S_n$: all $n!$ permutations on $[n]$

n-player musical sofas



$Q = \{\text{positions of } n \text{ players}\}$

$\Sigma = \{\ \text{cycle},\ \ \text{swap},\ \ \text{merge}\ \}$

$\begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{bmatrix} \begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & & \\ & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$

$\mathcal{T}(\mathcal{A}) = T_n$: all n^n functions $[n] \to [n]$

Prime factorizations of algebraic objects

- Universal structure theorems in abstract algebra:
  - **Integers:** $N = p_1 \cdot p_2 \cdots p_n$, prime numbers p_i [Euclid ~300 BC]
  - **Groups:** $G \rhd H_n \rhd \cdots \rhd H_1$, simple groups H_{i+1}/H_i [Jordan & Hölder ~1880]
  - **Semigroups:** *do we know anything if we're only given associativity?*

- *Yes!* [Krohn & Rhodes '65]

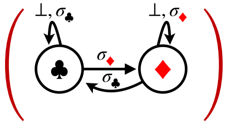
Krohn-Rhodes theorem (semigroups)

wreath product

$$G \leq H_n \wr M \wr \cdots \wr H_2 \wr M \wr H_1$$

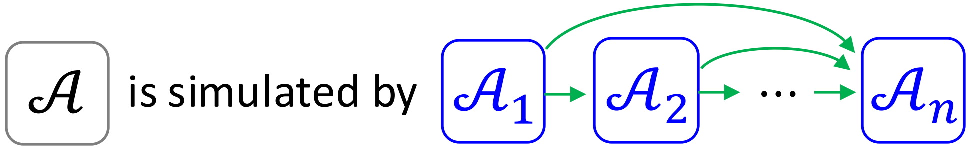
simple groups

flip-flop monoid $\mathcal{T}$



Krohn-Rhodes theorem (semiautomata)

feedforward cascade


$\mathcal{A}$ is simulated by $\mathcal{A}_1 \rightarrow \mathcal{A}_2 \rightarrow \cdots \rightarrow \mathcal{A}_n$

permutation-reset automata:
all $\delta(\cdot, \sigma)$ either invertible or constant

Factorization: from integers to groups

$$\boxed{8 = 2 \times 2 \times 2}$$

- Why groups get complicated: **combinatorial explosion**

🐛C_8: mod-8 addition
🐙$E_8 \cong C_2 \times C_2 \times C_2$: 3-bit vectors under XOR
🦓$C_4 \times C_2$: non-interacting mod-4 & parity
🦀$D_8 \cong C_4 \rtimes C_2$: rotations/reflections of a square
🦄Q_8: multiplication of unit quaternions

D_8, Q_8: **non-abelian:** $gh \neq hg$

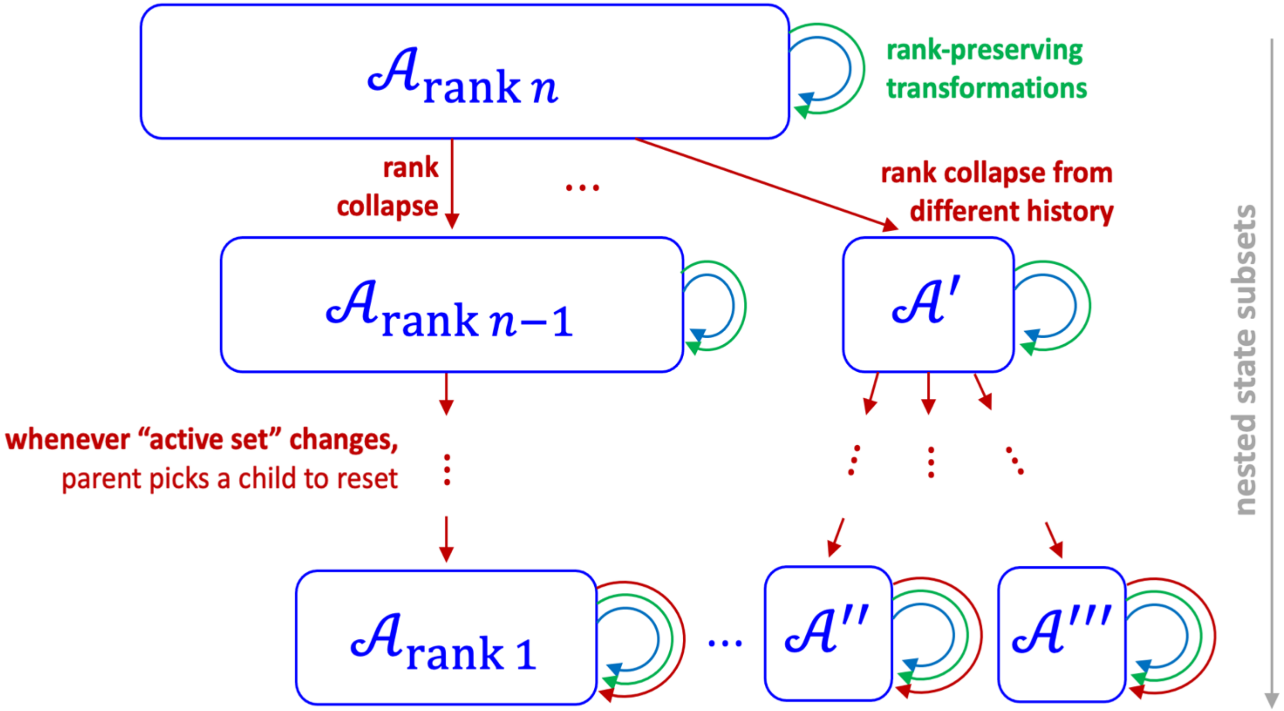
- **Finite group theory:** classical toolbox for understanding symmetries

$$C_8, E_8, C_4 \times C_2, D_8, Q_8 \leq (C_2 \wr C_2) \wr C_2$$

Jordan-Hölder factors (simple groups)
Krasner-Kaloujnine embedding (wreath product)

Krohn-Rhodes intuitions

Tracking rank collapses (*holonomy decomposition*)

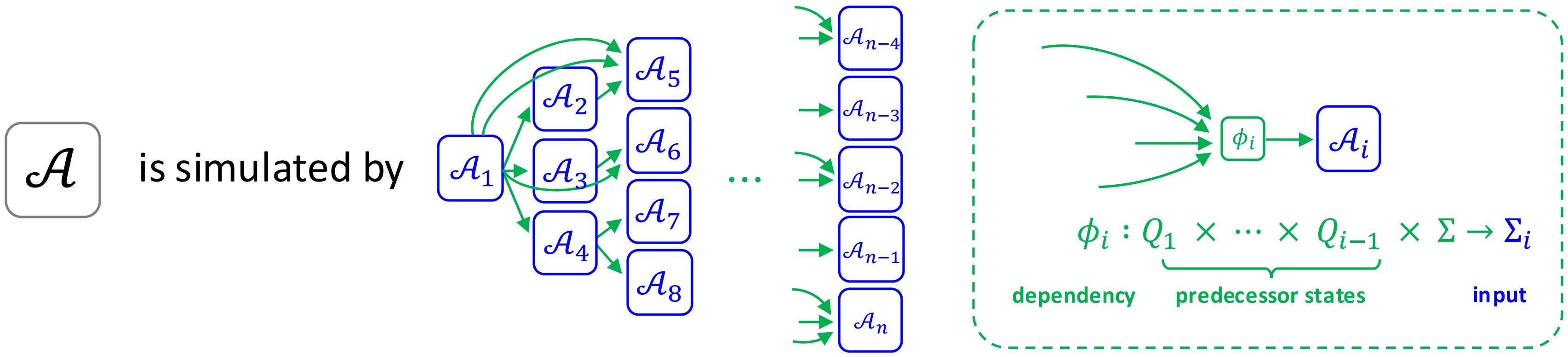

$\mathcal{A}_{\text{rank}\,n}$
rank-preserving transformations
rank collapse
…
rank collapse from different history
$\mathcal{A}_{\text{rank}\,n-1}$
$\mathcal{A}'$
whenever "active set" changes, parent picks a child to reset
$\mathcal{A}_{\text{rank}\,1}$
…
$\mathcal{A}''$
$\mathcal{A}'''$
nested state subsets

Number of layers:

- Solvable groups: $O(\log|G|)$
  - mod counter
- Permutation-reset semiautomaton: $O(\log|G|) + 2 \leq O(|Q| \log|Q|)$.
  - mod counter + memory unit
- Semiautomaton: $\leq |Q|$ levels of the above.

Proof of Krohn-Rhodes: key intuitions

- Holonomy decomposition “compiles” to a cascade

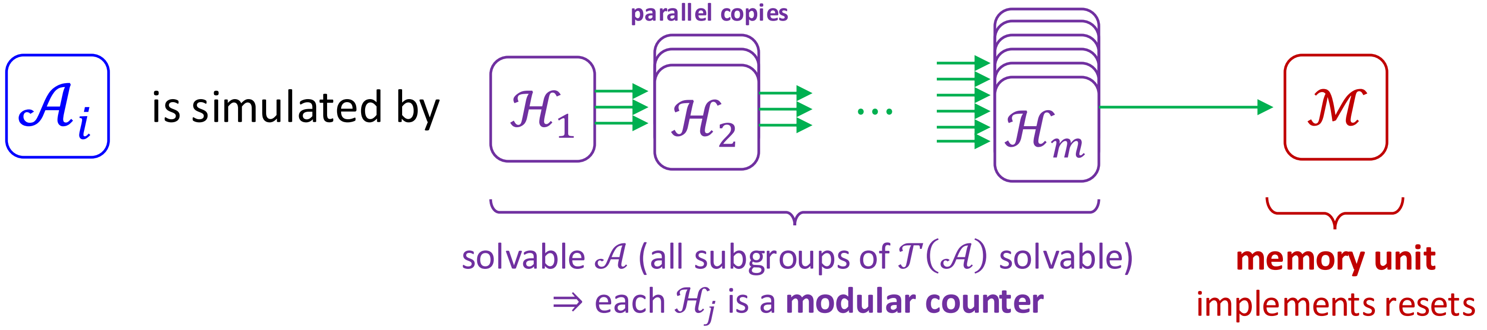


$\mathcal{A}$ is simulated by

$$\underbrace{\phi_i}_{\text{dependency}} : \underbrace{Q_1 \times \cdots \times Q_{i-1}}_{\text{predecessor states}} \times \underbrace{\Sigma}_{\text{input}} \to \Sigma_i$$

semigroup theory under the hood:
$\mathcal{T}(\mathcal{A}_j)$ are holonomy groups adjoined with left zero semigroups, i.e. *permutations* + *resets*

- Permutation-reset semiautomata factorize further



$\mathcal{A}_i$ is simulated by

solvable $\mathcal{A}$ (all subgroups of $\mathcal{T}(\mathcal{A})$ solvable)
⇒ each $\mathcal{H}_j$ is a **modular counter**

memory unit
implements resets

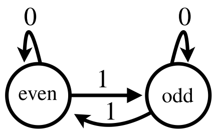
group theory under the hood:
$\mathcal{T}(\mathcal{H}_j)$ are simple groups from Jordan-Hölder composition factors, which are all cyclic if $\mathcal{A}$ is solvable

realized by **universal embedding theorem**
[Krasner & Kaloujnine ‘51]

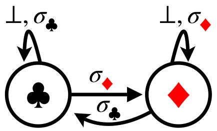
Implications for simulating automata

- **Krohn-Rhodes:** *all* $\mathcal{A}$ decompose into permutation-reset $\{\mathcal{A}_i\}$
- **Jordan-Hölder:** *all* $\mathcal{A}_i$ decompose into simple group machines $\{\mathcal{H}_i\}$

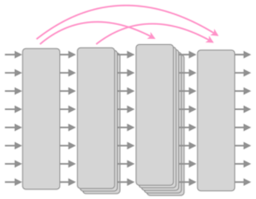
- **Recall:** *when can we find a solution like* $\Sigma\sigma_{1:t} \bmod 2$ *for parity?*
- **Answer:** whenever the *"atoms"* and *"glue"* are implementable



mod-n counter



n-state memory unit



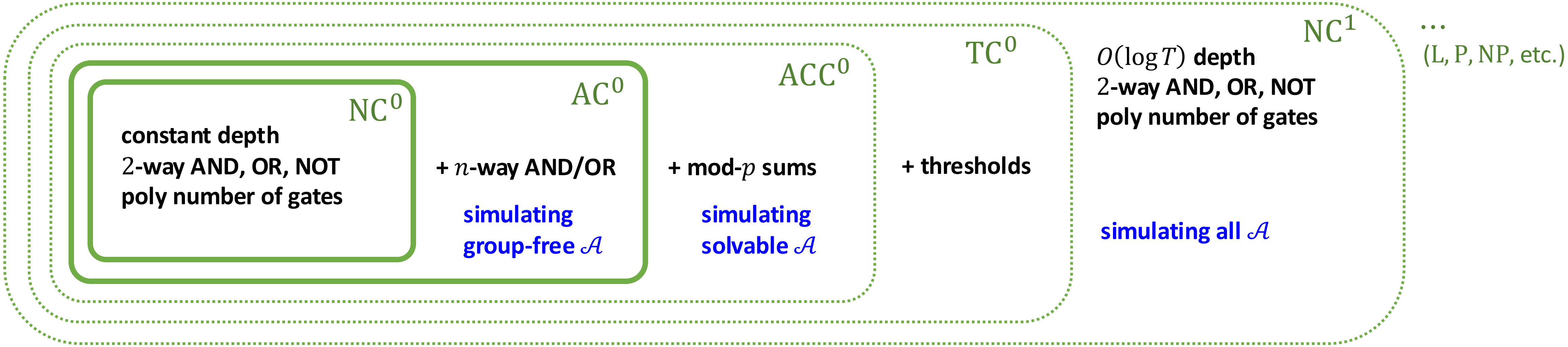
transformation cascade

sufficient when all groups within $\mathcal{T}(\mathcal{A})$ are **solvable**

- Need circuit complexity to formalize the question…

Quantifying efficient parallel circuits

- **Goal:** formalize *"Krohn-Rhodes implies efficient simulation"*
- Low-depth parallel algorithms are best captured by **circuit complexity**



NC^0

constant depth
2-way AND, OR, NOT
poly number of gates

AC^0

+ n-way AND/OR

simulating group-free $\mathcal{A}$

ACC^0

+ mod-p sums

simulating solvable $\mathcal{A}$

TC^0

+ thresholds

NC^1

$O(\log T)$ **depth**
2-way AND, OR, NOT
poly number of gates

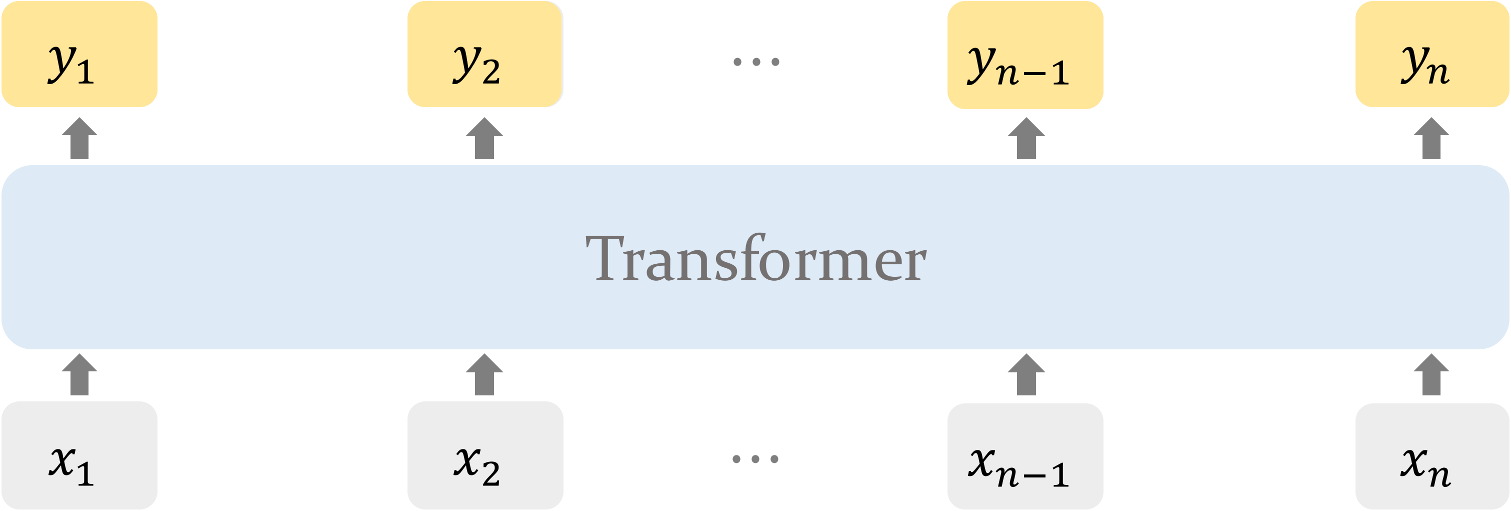
simulating all $\mathcal{A}$

…
(L, P, NP, etc.)

Embarrassingly open: are any of the ⬚ proper? $\mathrm{ACC}^0 \overset{?}{=} \mathrm{NP}$?

Scratchpads (modification of Nye et al. 21)

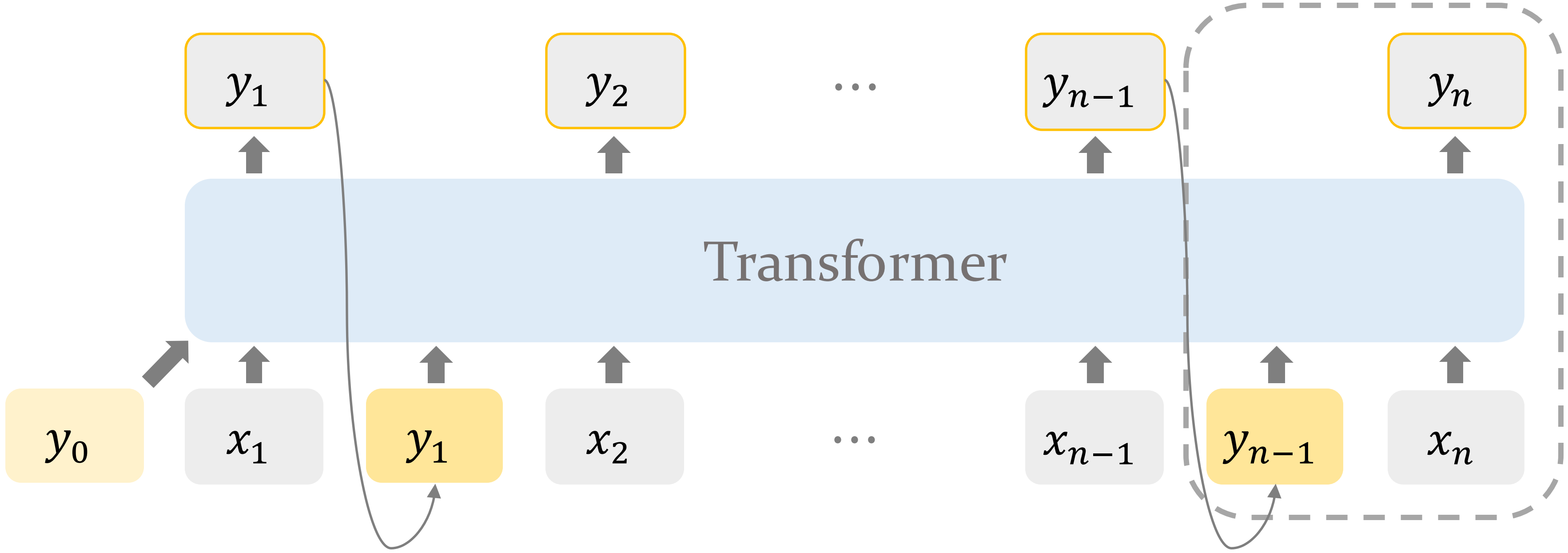
Idea: make the model recurrent with scratchpad (~buffer): explicitly modeling states.



want: $y_i = x_i \oplus y_{i-1}$

Scratchpads (modification of Nye et al. 21)

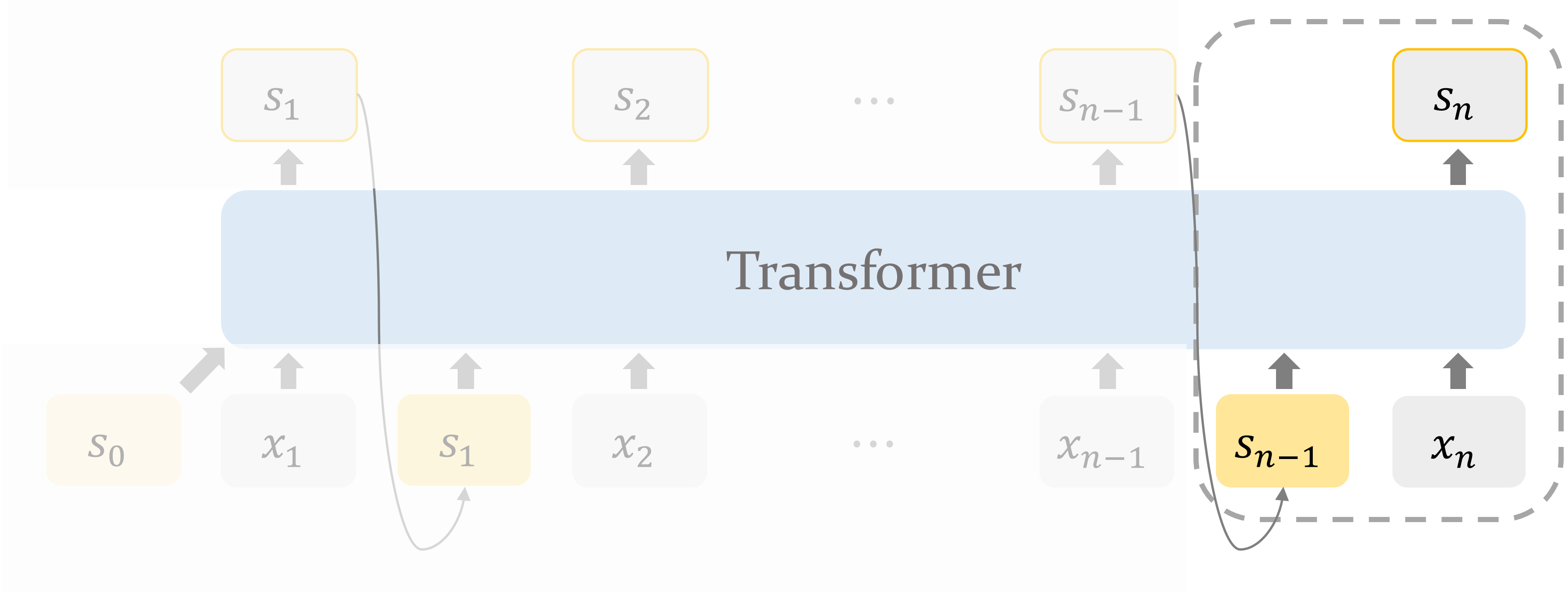
Idea: make the model recurrent with scratchpad (~buffer): explicitly modeling states



want: $y_i = x_i \oplus y_{i-1}$

Scratchpads (modification of Nye et al. 21)

Idea: make the model recurrent with scratchpad (~buffer): explicitly modeling states



want: $s_i \coloneqq y_i = x_i \oplus s_{i-1}$