Formal Languages and Neural Networks on Sequences

 **William Merrill**

 Ashish Sabharwal

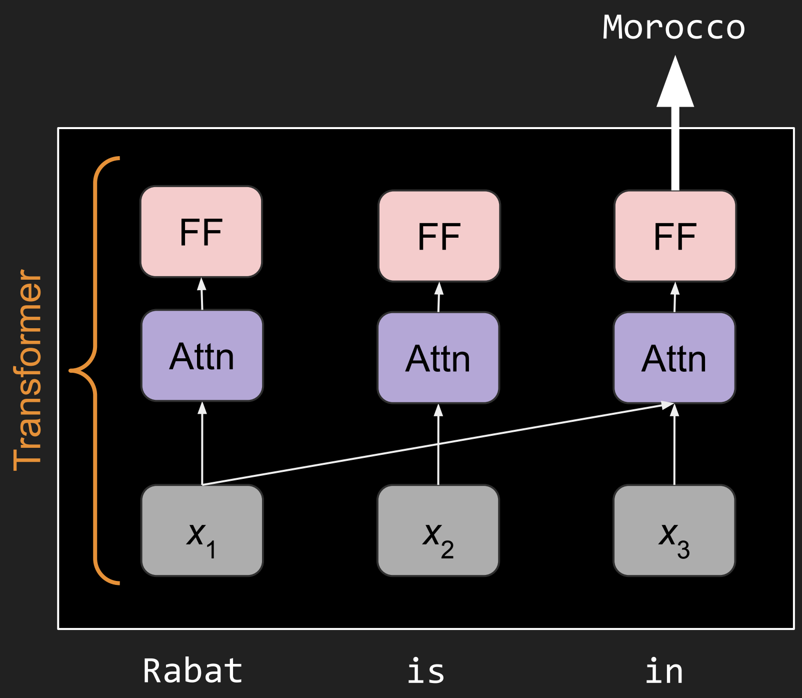
New York University
New York, NY, USA

Allen Institute for AI
Seattle, WA, USA

ICGI 2023
Rabat, Morocco

Recent trend in natural language processing: large language models (LMs)

- **Language model:** system that predicts next word
- Modern large LMs are implemented as (large) neural networks called a **transformers**
- **Use:** Generate text or use as building block for larger systems



Goal: understand the computational power of transformers

1. What kind of computation are transformers doing and where does it fall in the Chomsky hierarchy?

2. What problems can transformers *not* solve? I.e., can we derive **upper bounds**?

Why care about transformers' computational power?

Transformer LMs are impressive; how do they work?

They can output coherent text involving logical inferences ("reasoning")

W

Can you prove the Myhill-Nerode theorem for me?

Certainly! The Myhill-Nerode theorem is a fundamental result in formal language theory. It provides a necessary and sufficient condition for a language to be regular, based on the concept of equivalence classes. Let's go through the proof together.

Before we begin, let's define some key terms:

1. Language: A language is a set of strings over some alphabet.
2. Equivalence relation: A relation that partitions a set into disjoint subsets called equivalence classes.
3. Equivalence class: A subset of elements in a set that are all equivalent to each other under a given equivalence relation.

Now, let's state the Myhill-Nerode theorem:

W

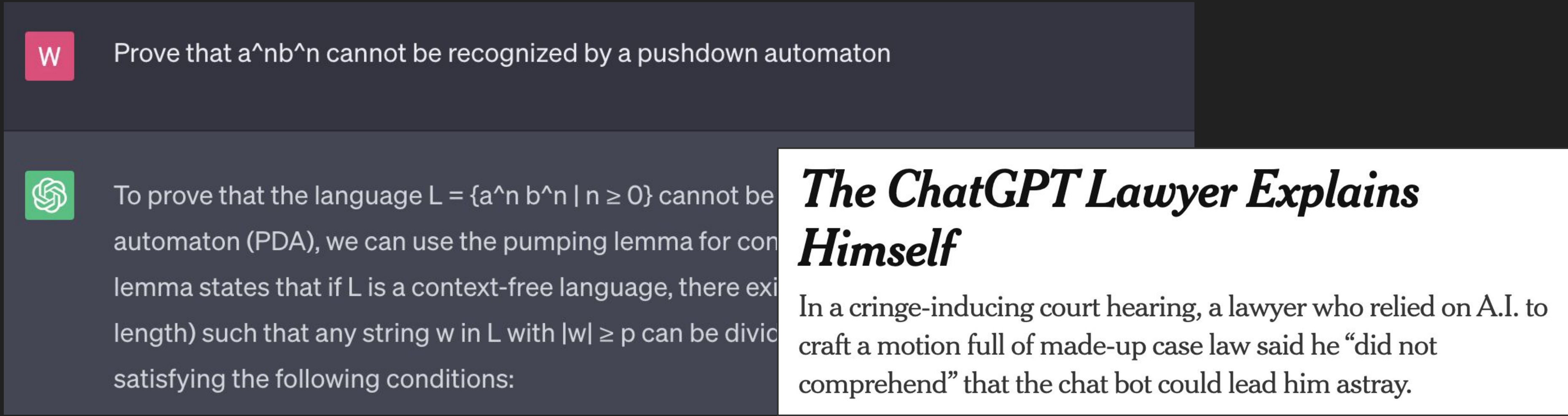
Define the parity of a language to be whether the smallest string in a language has even or odd size. Let bloop(L) be the complement of L if L has even parity and L + "bloop" (where + is concatenation) otherwise. Prove that the regular languages are closed under bloop.

To prove that the regular languages are closed under the operation bloop, we need to show that given a regular language L, the language bloop(L) is also regular.

Let's consider the cases separately:

1. If L has even parity:
   In this case, the smallest string in L has an even size. The complement of L, denoted as L', consists of all strings that are not in L. Since L is regular, L' is also regular because regular languages are closed under complementation. Now, to obtain bloop(L), we need to take

Transformer LMs often reason poorly (despite the hype!)



1. Systematic understanding of when LMs fail
2. Which reasoning errors will go away in bigger LMs, and which are a *fundamental limitation* of any transformer LM?

Main results: expressive power of transformers

2. Transformers can only recognize languages in **uniform TC^0**
    a. Cannot recognize all regular languages
    b. Simple problems that break transformers *(in theory and in practice)*
3. Extensions and related results
    a. Logical upper and lower bounds on transformers *(tight characterization open)*
    b. Chain-of-thought adds substantial power to transformers

Tutorial outline

1. Part 1: Expressive power of recurrent neural networks (RNNs)
2. Part 2: Expressive power of transformers via circuit complexity

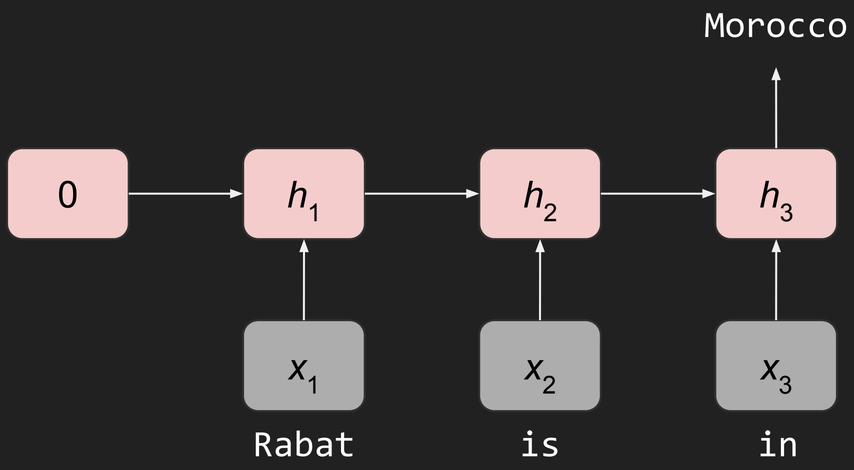
Break (5 minutes)

3. Part 3: Logical and algebraic characterization, as well as chain-of-thought
4. Part 4: Learning biases of transformers

Part 1:
Expressive Power of
Recurrent Neural Networks

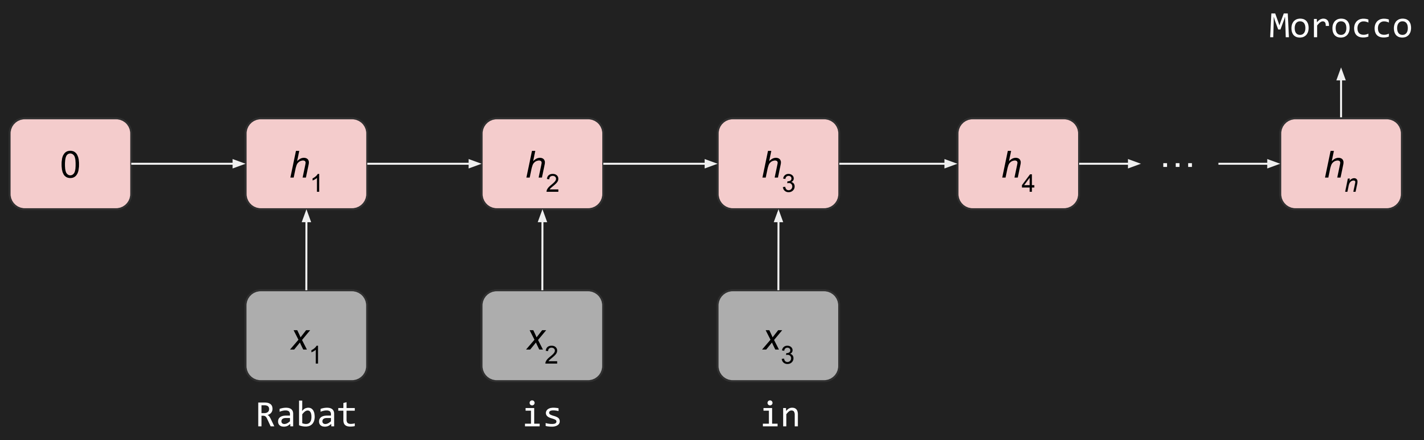
Recurrent neural networks (RNNs)

- Update state vector *h* according to previous state and input token



Non-realtime RNNs

- First model studied in the 1990s
- Can keep computing past the end of input!
- Important parameter: datatype (binary, rational, real)

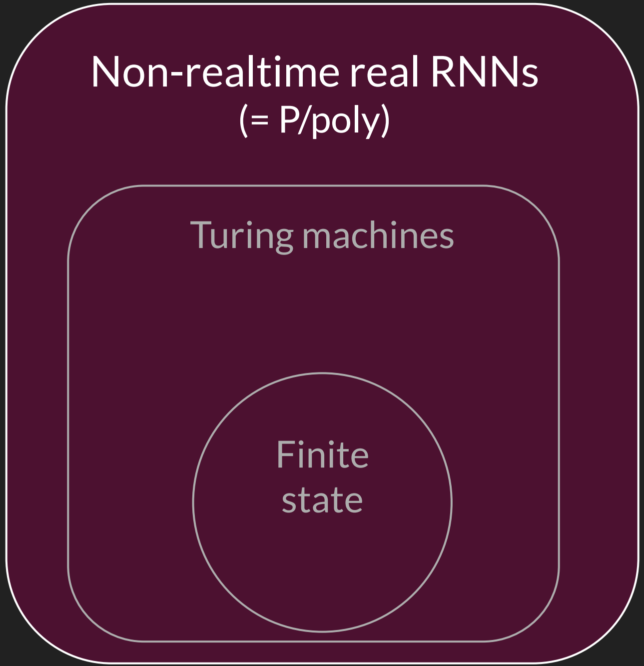


Non-realtime RNNs with real weights are (too) powerful

(Siegelmann and Sontag, 1994)

- Specifically, they recognize P/poly
  - Includes undecidable problems
  - Use infinite-precision reals to memorize infinite lookup table

⇒ Infinite-precision weights are unrealistic!



Natural idea: real → rational weights

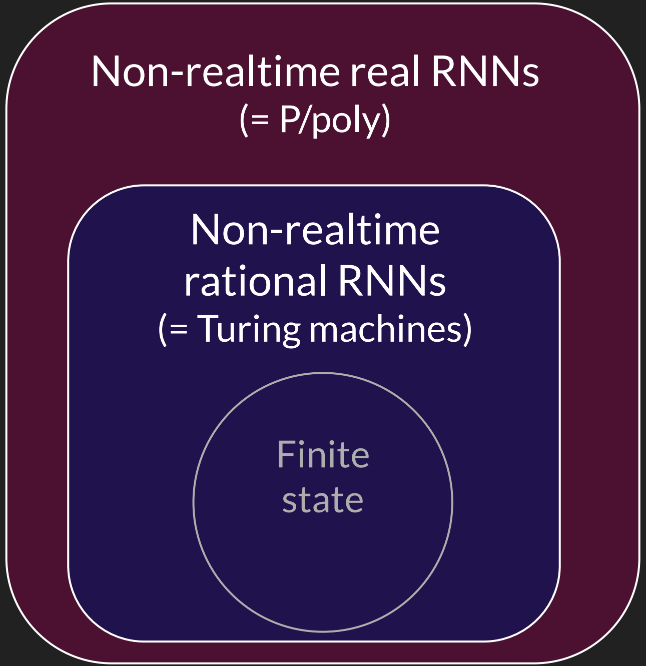
- Unlike real numbers, rational numbers have a finite description
- The RNN will not be able to solve undecidable problems

Non-realtime RNNs with rational weights are Turing-complete

(Siegelmann and Sontag, 1995)

- Proof idea: simulate a two-stack Turing machine
    - Can use 3 hidden-state neurons for a stack

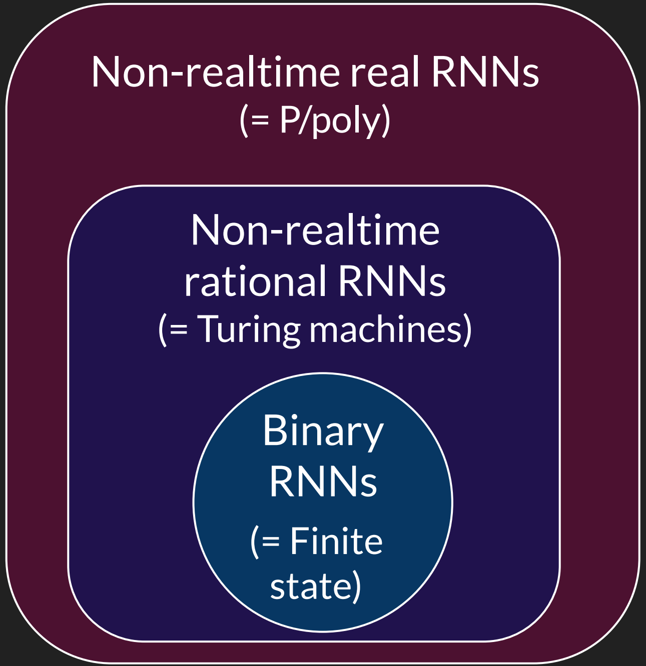
- Relies on unbounded precision weights and runtime



RNNs with binary weights are equivalent to automata

(Siegelmann and Sontag, 1995)

- Proof idea: Only a finite number of possible hidden state vectors
- Applies with/without realtime
- Closest model to practical RNNs (realtime, bounded-precision)



Discussion: standard realtime RNNs ≈ automata

- Turing-completeness requires unbounded precision and runtime
- Empirically, simple RNNs cannot reliably learn beyond regular languages

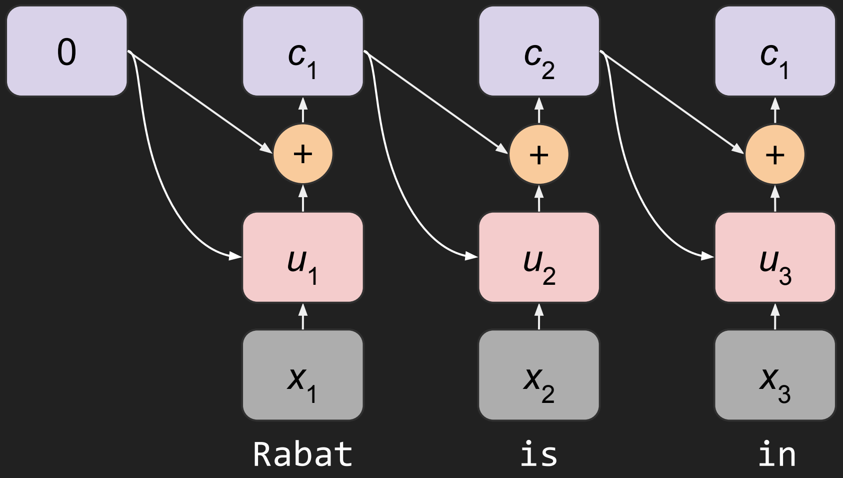
(Weiss et al., 2018; Delétang et al., 2023, *inter alia*)

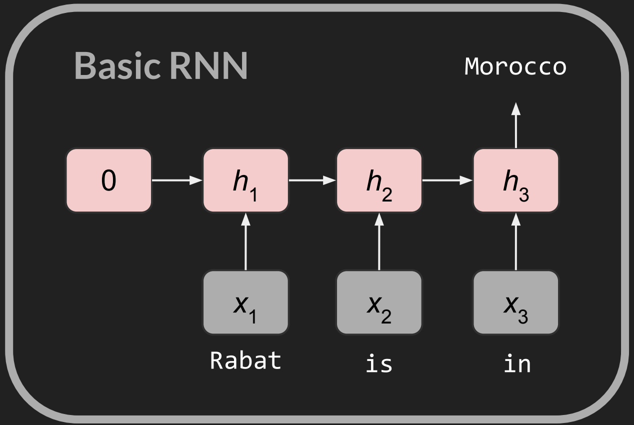
⇒ What about generalized gated RNNs like LSTMs?

Long short-term memory networks (LSTMs)

(Hochreiter & Schmidhuber, 1997)

- Extension of normal RNN with **additive update:**



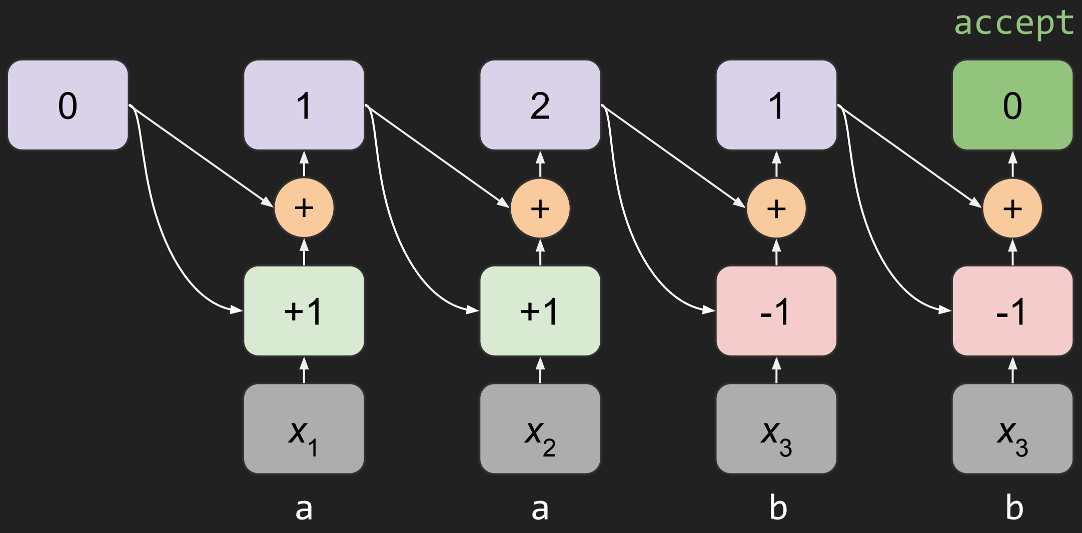


- Empirically better than normal RNNs, popular before transformers

LSTMs can count (unlike basic RNNs)

(Weiss et al., 2018)

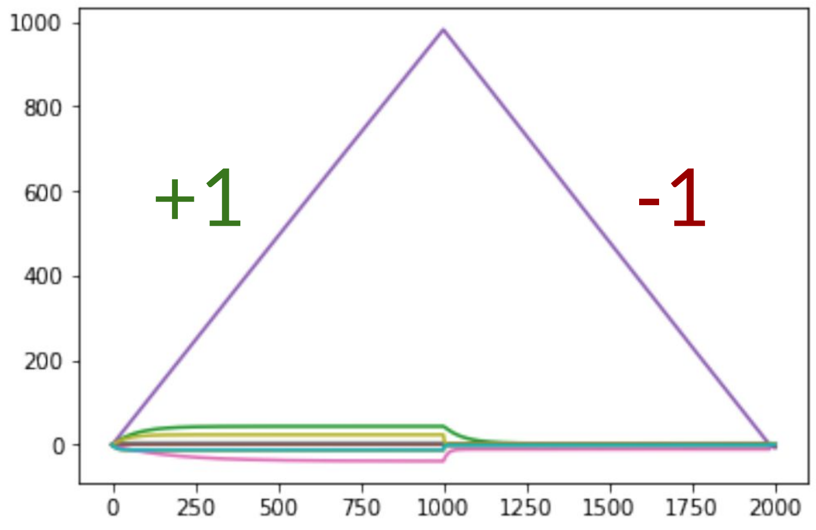
- Recognizing $\mathtt{a}^n\mathtt{b}^n$:



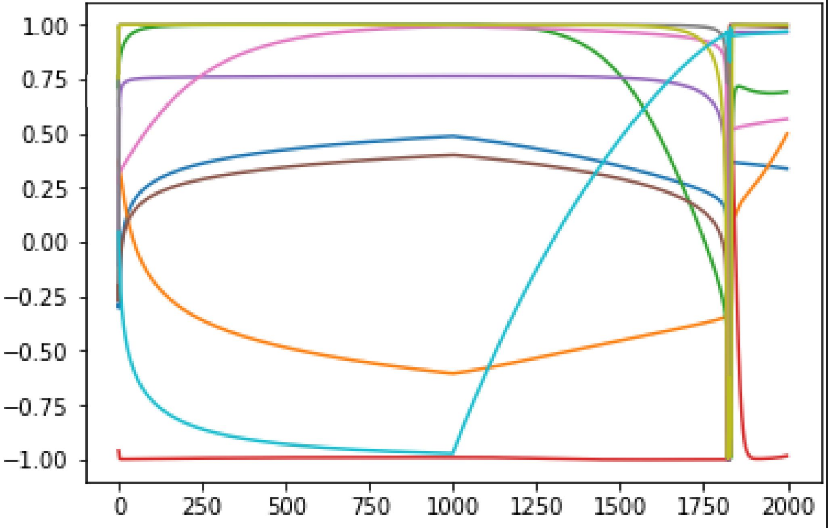
- Use a counter, not a stack

LSTMs can learn to count in practice

(Weiss et al., 2018)



(a) $a^n b^n$-LSTM on $a^{1000} b^{1000}$



(c) $a^n b^n$-GRU on $a^{1000} b^{1000}$

(Variant of simple RNN)

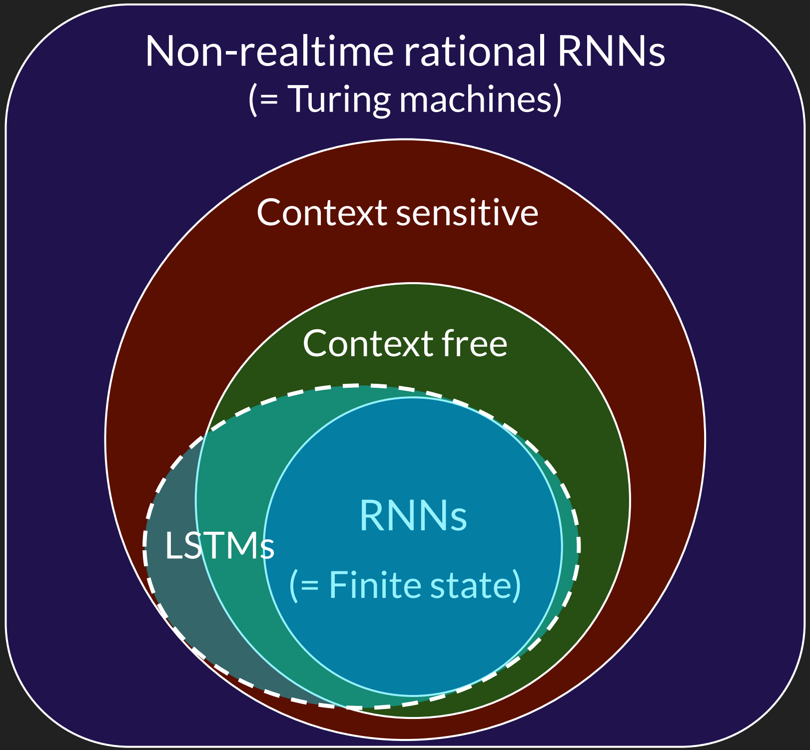
Counting makes LSTMs more powerful than RNNs (but far from Turing-complete)

(Merrill, 2019)

- LSTMs cross-cut context-free within context-sensitive languages
  - ≈ Counter automata

- Further finegrained comparison of RNN variants (cf. Merrill et al., 2020)

⇒ But what about transformers?



Questions?

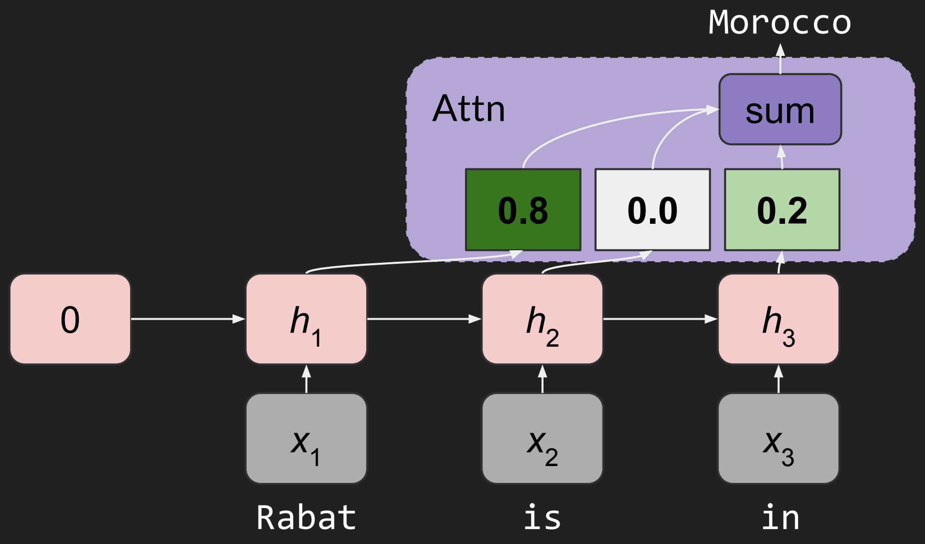
Part 2:
Expressive Power of Transformers

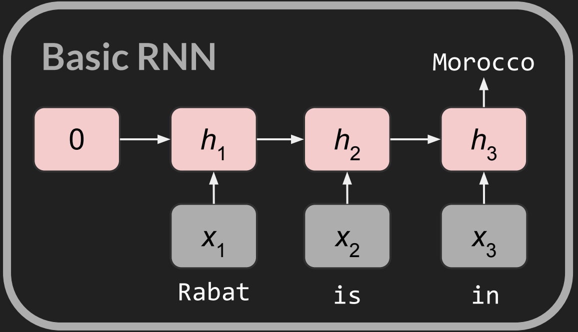
From RNNs to transformers

RNNs → RNNs w/ attention → Transformers

RNNs with attention

- Predict output based on **weighted average** of previous states
- Key/value lookup with query based on current state



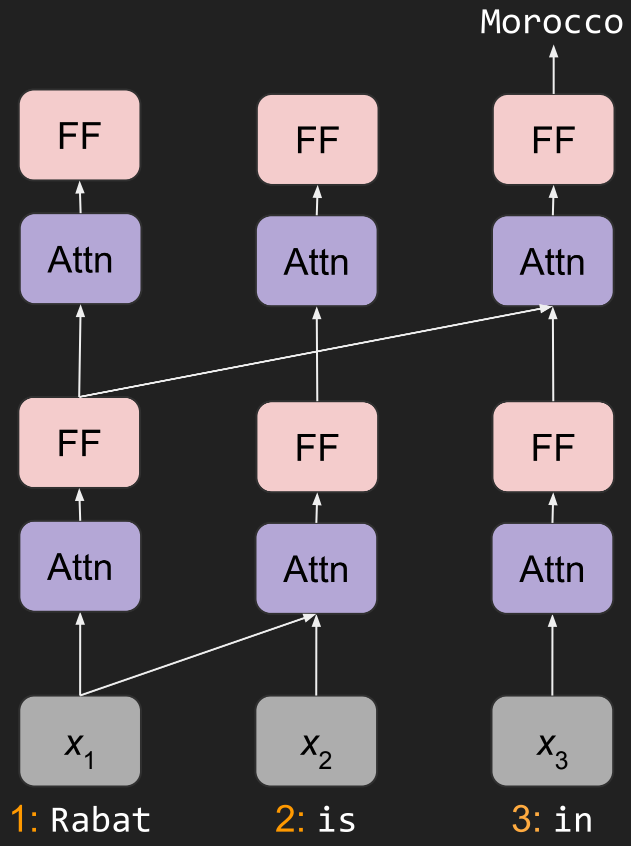


Transformers ≈ stacks of attention layers

"Attention is all you need"

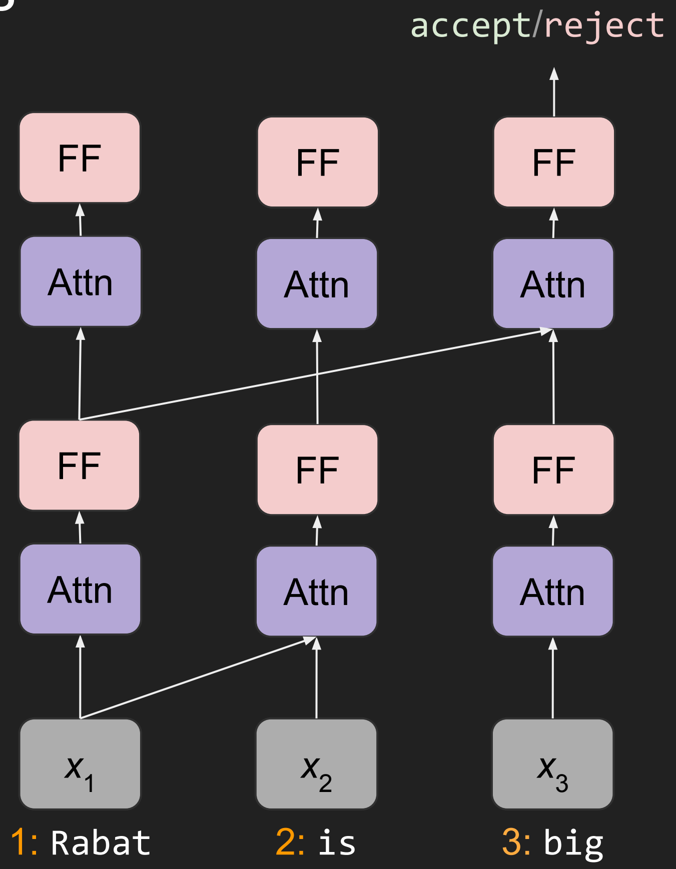
-Vaswani et al. (2017)

- Local processing by feedforward (FF) nets

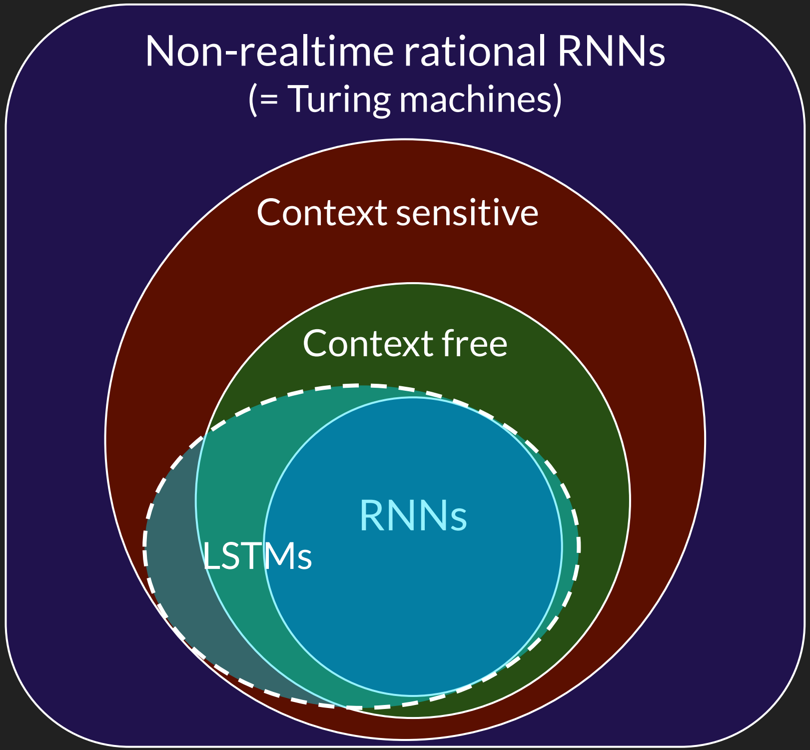


Transformers as language recognizers

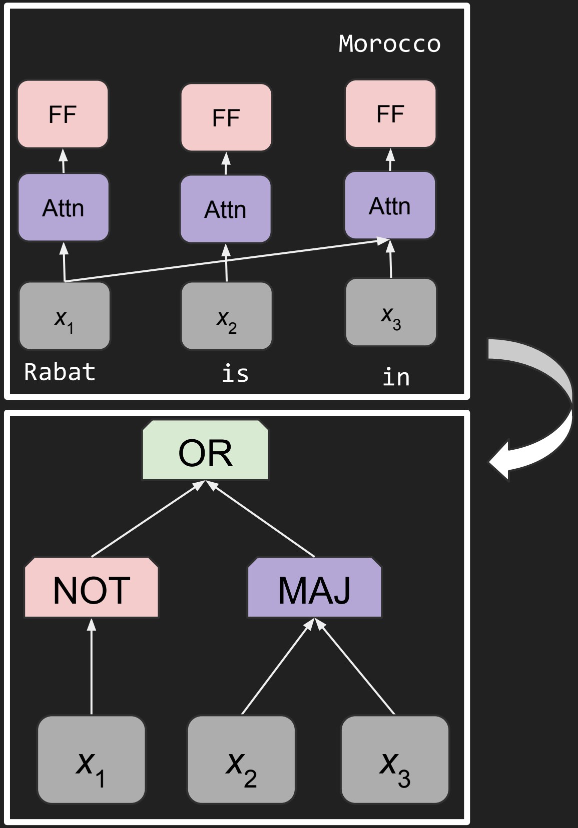
- Predict `accept` or `reject` after reading the full string



Transformers in the Chomsky hierarchy?



How to derive upper bounds on transformers?



Problem: no recurrent structure in transformers

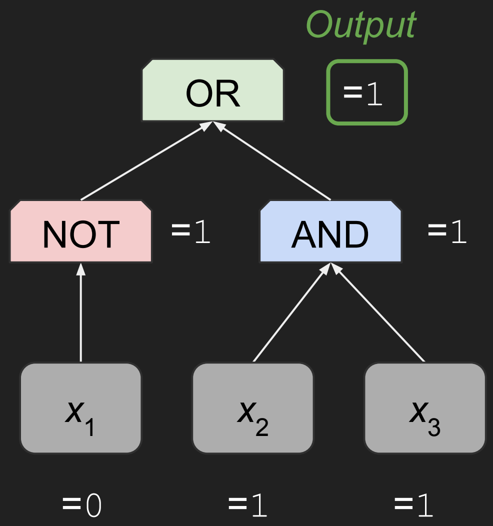
Transformers (likely) don't fit nicely in Chomsky hierarchy

Instead: place transformer in hierarchy of circuit complexity classes

Both transformers and circuits are doing parallel computation over sequences

Detour:
Circuit Complexity

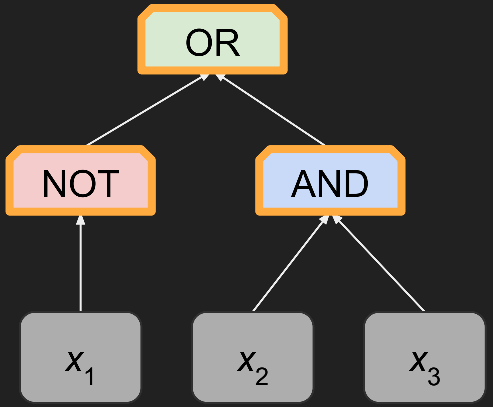
Circuits



- Directed acyclic computation graph
- Input nodes represent bits (0/1)
- Internal nodes: AND, OR, NOT
- Output node (0/1)

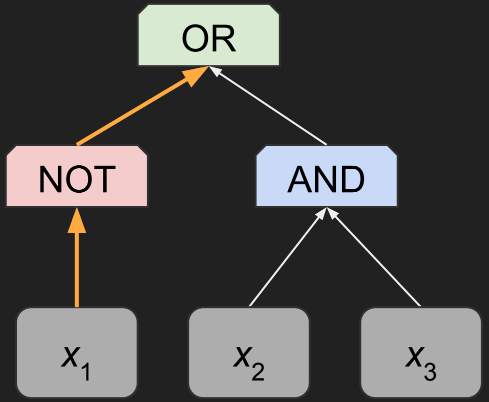
Example: evaluate on x = 011

Circuit size



- Number of non-input nodes
  - Here, size = 3
- Runtime without parallelism

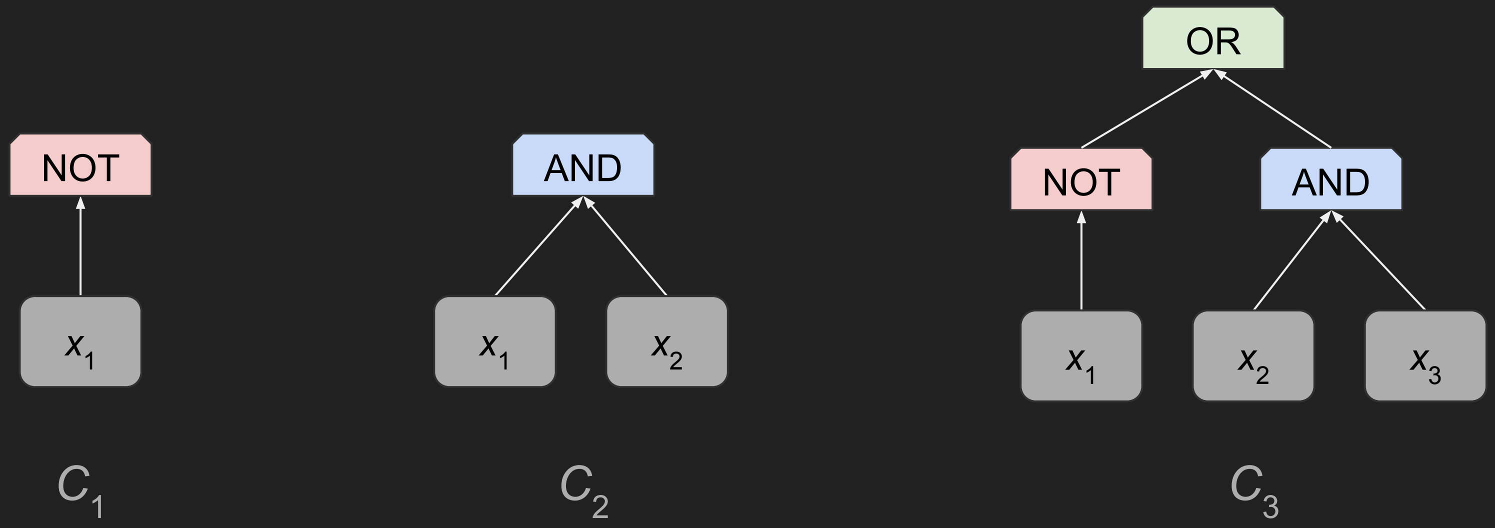
Circuit depth



- Longest path from input to output
  - Here, depth = 2
- Runtime with parallelism

Circuit families

- Circuit with n inputs is function $C_n : \Sigma^n \rightarrow \{0, 1\}$
- Circuit family: sequence of circuits C_1, C_2, ... defining formal language



Aside: uniformity and circuit families

- **Nonuniform:** Circuit C_n can change arbitrarily with input size *n*
  - Allows circuit families to solve some undecidable problems (like real-valued RNNs)

- This issue is fine for now, but will fix later by enforcing *uniformity*
  - **Uniform:** circuits for different input sizes cannot be too different

AC^0: constant-depth, poly-size circuit families

Problems parallelizable to very high (constant) degree

- **Polynomial size**: size(C_n) is polynomial of input sequence length n
- **Constant depth**: depth(C_n) fixed with respect to input sequence length n

AC^0 is a very limited complexity class

(Furst et al., 1984)

Classically, many simple languages outside AC^0:

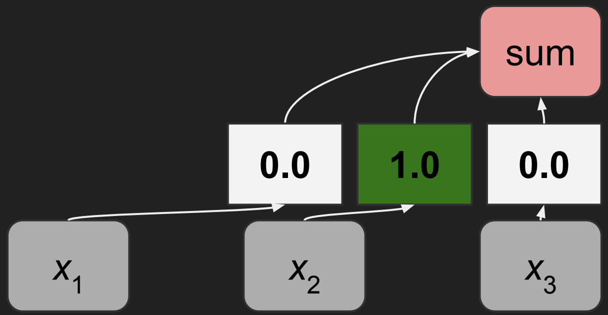
- PARITY: XOR of sequence of bits
  - `100` ∈ PARITY
  - `101` ∉ PARITY
- MAJORITY: majority vote of sequence of bits
  - `110` ∈ MAJORITY
  - `100` ∉ MAJORITY

Simulating (hard-attention) transformers in AC^0

Hard Attention Upper Bound (*Hao et al., 2022*)
Transformers with hard attention can be simulated by constant-depth, poly-size circuit families

I.e., hard-attention transformers only recognize languages in AC^0

Hard attention:



Implications: hard-attention transformers are weak!

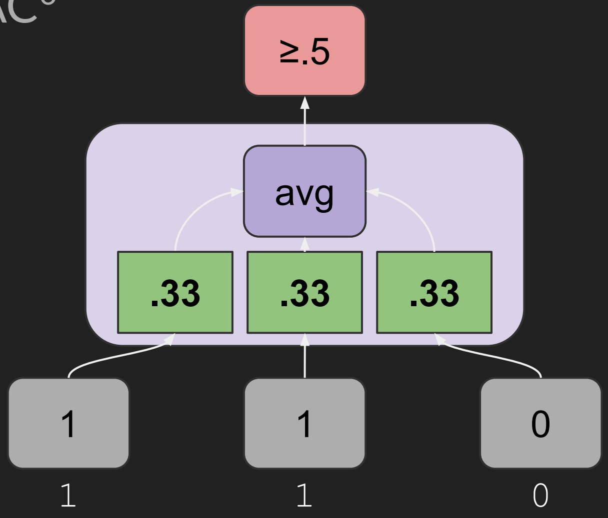
- Hard-attention transformers can't solve PARITY, MAJORITY, etc. outside AC^0
  - Cannot recognize all regular languages

- But hard attention is strong simplifying assumption
  - Transformers use soft attention to count, like LSTMs (Bhattamishra et al., 2020)
  - Empirically, transformers *can* recognize MAJORITY!

Recognizing MAJORITY with soft attention

(Pérez et al., 2021)

- Reminder: MAJORITY (majority vote of bits) $\notin AC^0$



- Soft attention adds power compared to hard attention

⇒ Upper bound for transformers with *soft attention*?

Non-Uniform Upper Bound for Soft-Attention Transformers

Log-precision soft-attention transformers

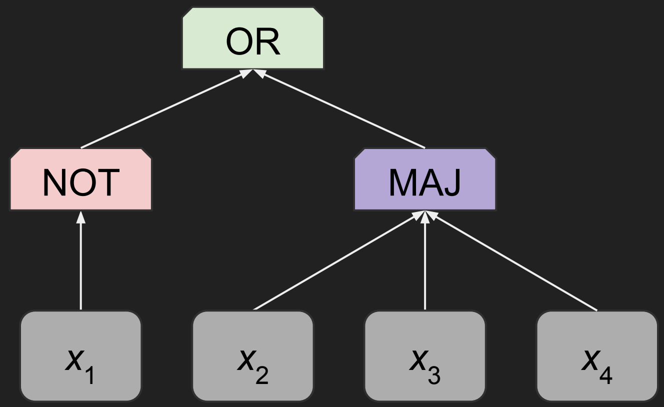
- Soft attention (unlike hard attention) relies on taking averages/arithmetic
- **Log-precision:** Arithmetic in transformer gets O(log n) precision on length n
- Good middle ground to avoid unrealistic unbounded precision constructions without making transformers too weak (skipping details)

Result: upper bound on log-precision transformers

Upper Bound (*Merrill et al., 2022*)
Log-precision transformers can be simulated by poly-size, constant-depth threshold circuit families

I.e., they only recognize languages in TC^0

What are threshold circuits and TC^0?



- MAJ gate computes MAJORITY
- TC^0: poly-size, constant-depth threshold circuits

⇒ TC^0 more powerful than AC^0

Proof sketch: log-precision transformers in TC^0

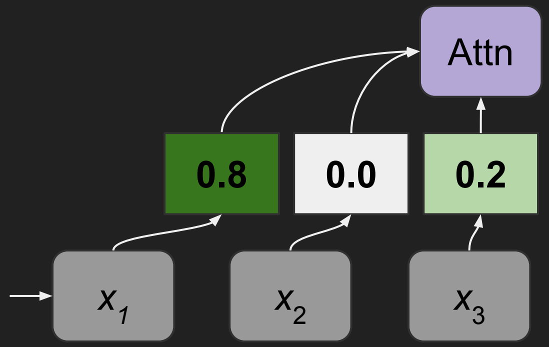
Upper Bound (*Merrill et al., 2022*)
Log-precision transformers can be simulated by poly-size, constant-depth threshold circuit families

Proof overview: simulate each transformer component in TC^0

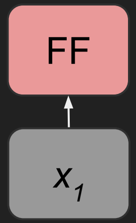
1. **Attention**
2. **Feedforward**

Proof sketch: log-precision transformers in TC^0

1. **Attention:** roughly, a weighted average
   a. Summation and division are in TC^0



2. **Feedforward:** local function of one state
   a. Any boolean function of c log *n* bits is in $AC^0 \subset TC^0$ (Hao et al., 2022)



Implications: transformers are inherently parallel

- Transformers can only solve problems that can be parallelized to a very high degree (TC^0)
- Majority/counting are exactly the additional power that soft attention has over hard attention
  - Soft vs. hard attention roughly analogous to LSTM vs. RNN

- **Missing:** To get concrete problems transformers cannot solve
  - Need a **uniform** upper bound

Problems Transformers Cannot Solve (Uniform Upper Bound)

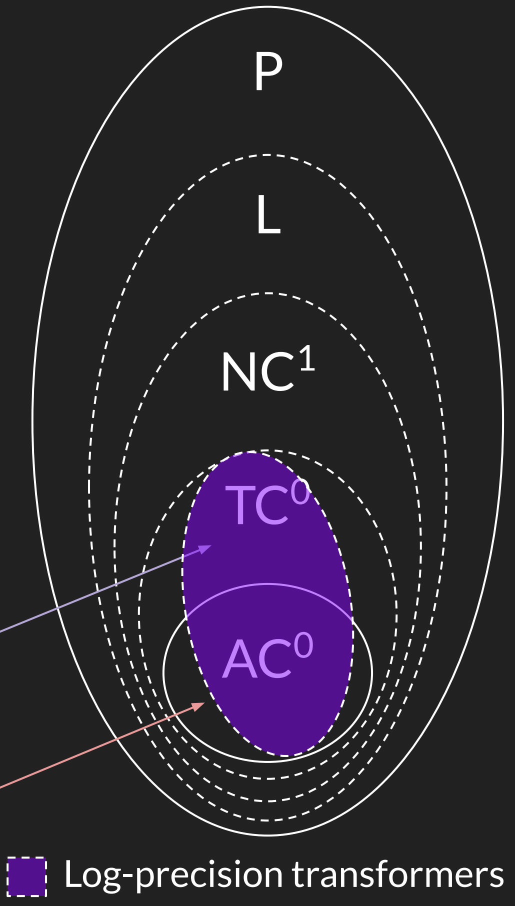
Goal: *uniform* TC^0 upper bound

- **Uniform** ⇒ circuits for different input sizes can be constructed by resource-bounded Turing machine
    - **Log-space uniform** ⇒ TM gets log space in input length

- Uniform circuit classes fall in hierarchy with other natural complexity classes

Uniform constant-depth, poly-size threshold circuits

Uniform constant-depth, poly-size circuits



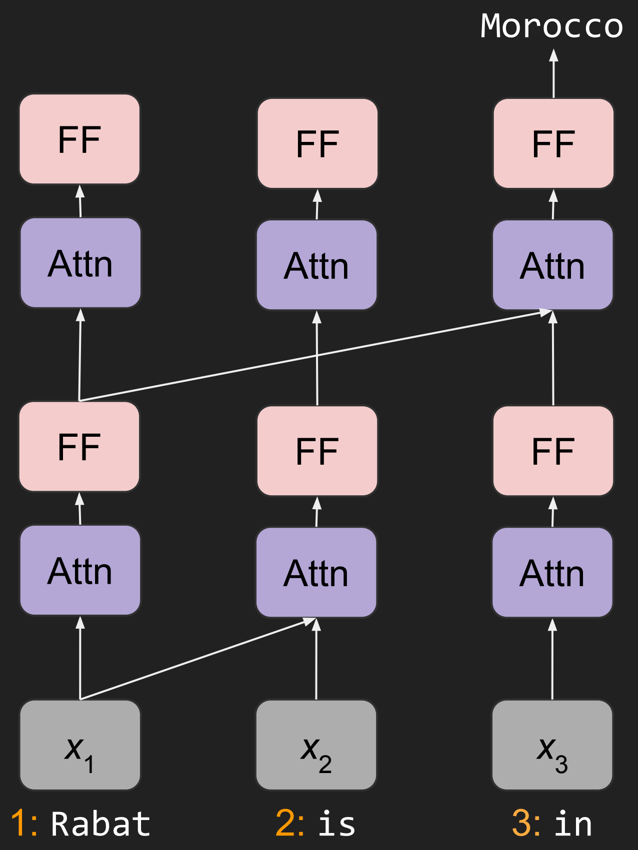
Simulating transformer with *uniform* threshold circuits

Uniform Upper Bound (*Merrill & Sabharwal, 2023a*)
Log-precision transformers can be simulated by poly-size, constant-depth, log-space-uniform threshold circuits

I.e., transformers are in log-space-uniform TC^0

Intuition: transformer is inherently uniform

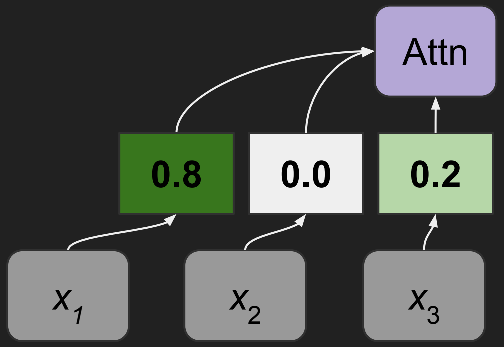
- Recall: parameters shared across columns
- To build transformer for input size *n*, simply copy one column *n* times



Proof sketch: log-precision transformers in uniform TC^0

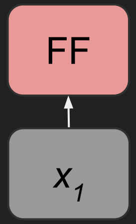
1. **Attention:**

   Summation and division are in uniform TC^0



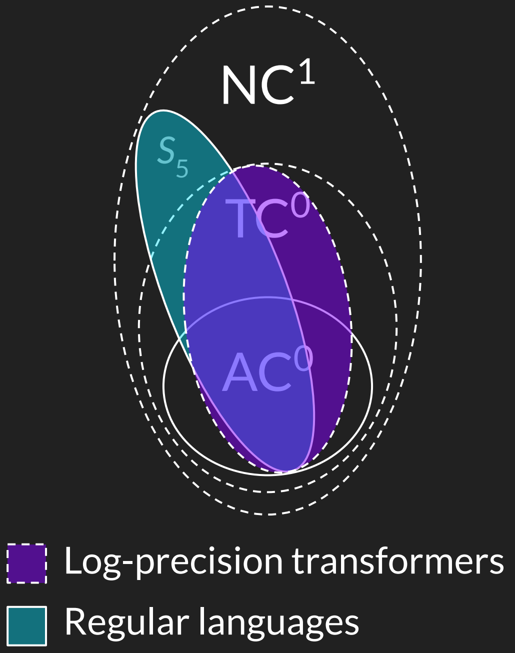
2. **Feedforward:**

   We prove: boolean function of c log n bits is in uniform $AC^0 \subset$ uniform TC^0 if it is computable in space c log n



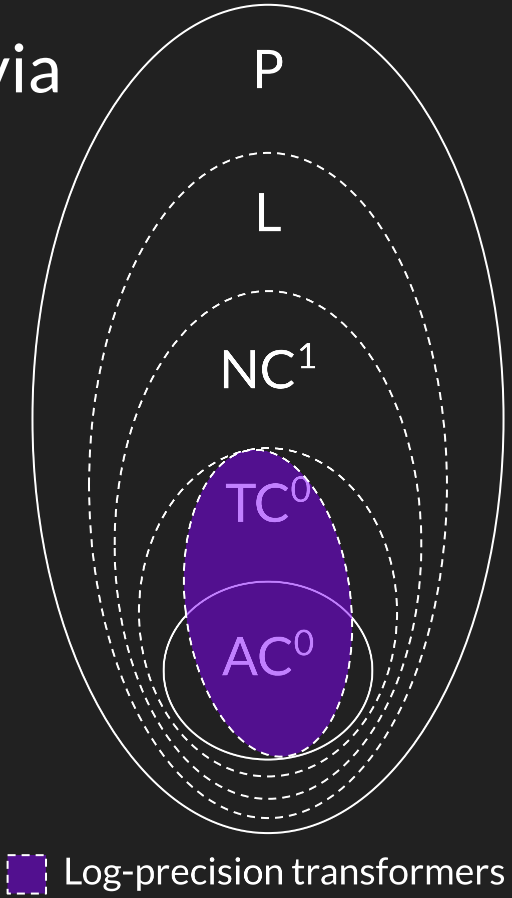
Implications: transformers and automata (also RNNs) are (likely) incomparable

- Transformers can clearly recognize non-regular languages (e.g., MAJORITY)
- By our result, transformers cannot recognize all regular languages unless $TC^0 = NC^1$
  - Since word problem for S_5 is regular and NC^1-complete (cf. Barrington and Maciel, 2000)
- **Intuition:** Some automata are too recurrent for transformers

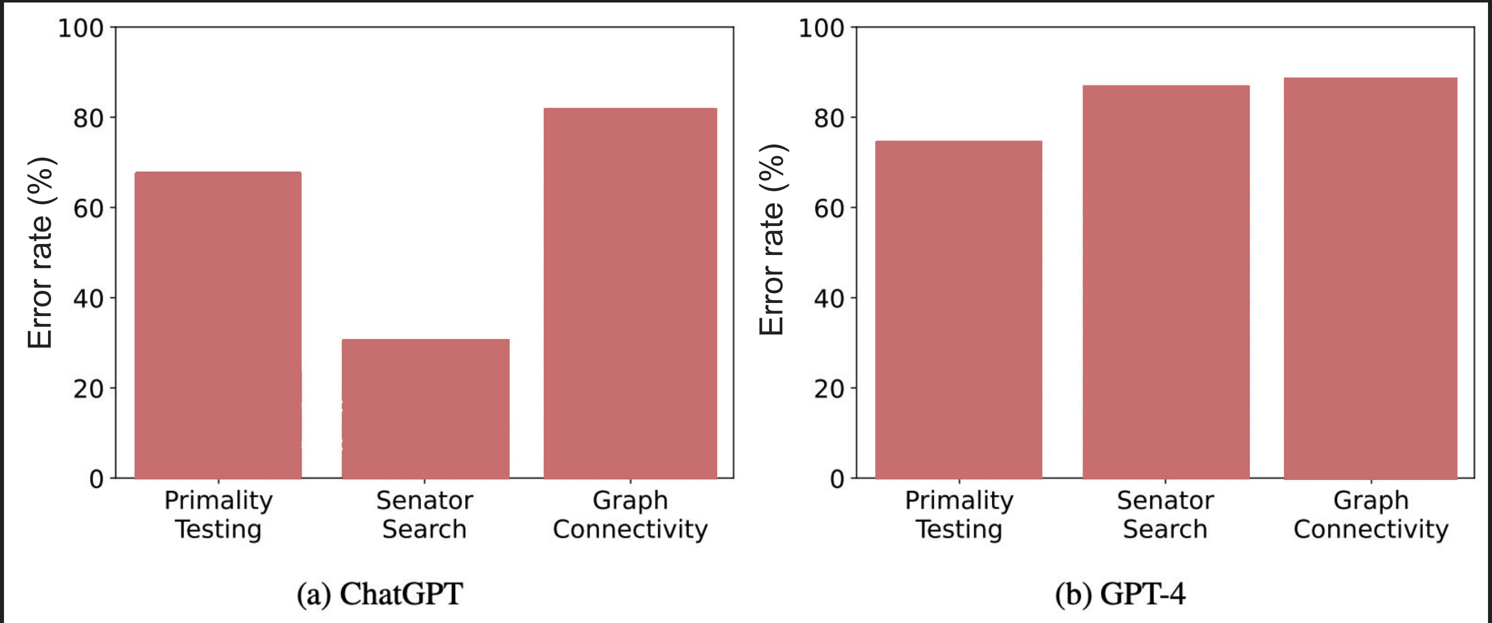


Implications: limitations of transformers via circuit complexity hierarchy

Problem transformers cannot solve	Conjecture
Boolean formula evaluation	$TC^0 \neq NC^1$
Linear equalities (find x s.t. $Ax = b$)	$TC^0 \neq P$
Horn satisfiability	$TC^0 \neq P$
Given CFG + string, does CFG → string?	$TC^0 \neq P$
Undirected/directed graph connectivity	$TC^0 \neq L, NL$
Primality testing	PRIMES $\notin$ L

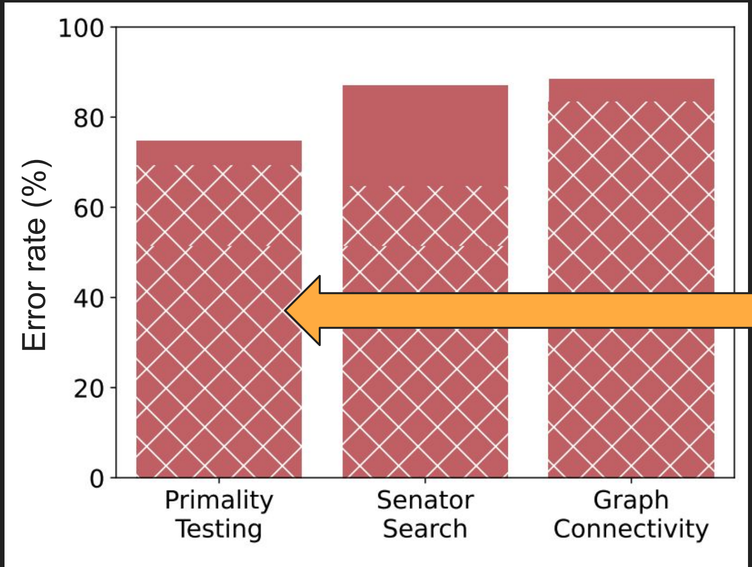


ChatGPT/GPT-4 can't solve graph connectivity/primes

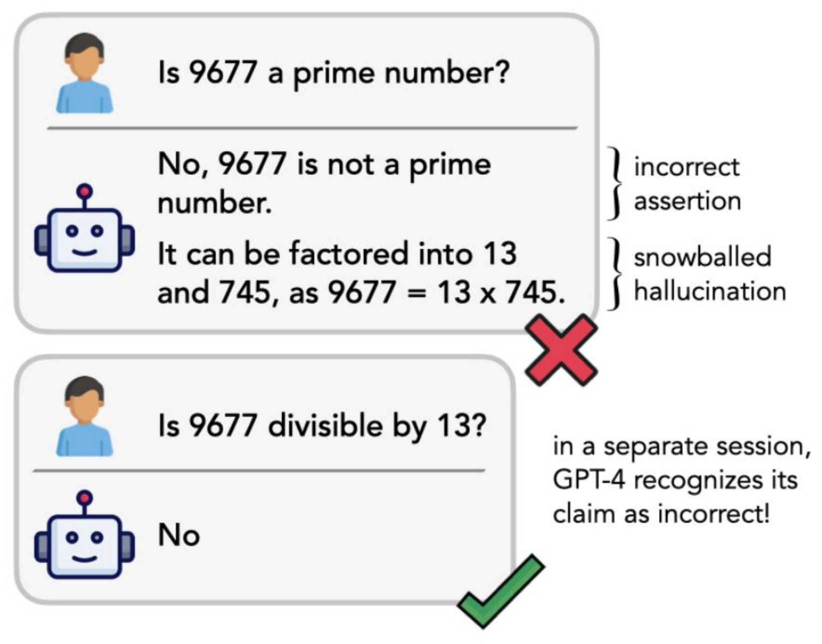


(Zhang et al., 2023)

LMs “hallucinate” wrong explanations for wrong answers



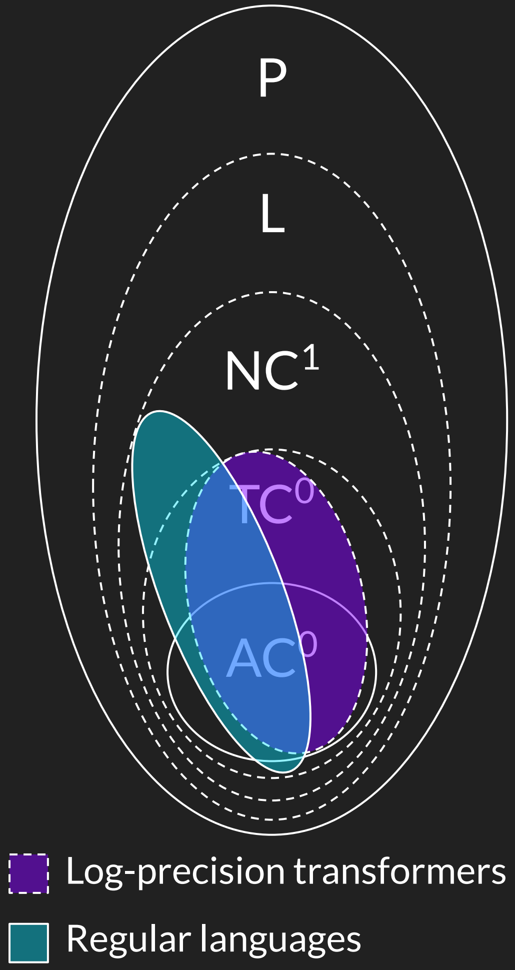
(b) GPT-4



(Zhang et al., 2023)

New takeaways: transformers and circuits

1. RNNs/LSTMs recognize regular/counter languages
2. Transformers recognize languages in **uniform TC^0**
   a. Can't recognize all regular languages
   b. Can't solve graph connectivity, etc. (in theory and in practice)



Questions?

Break (5 Minutes)

Part 3: Extensions to Circuit View

Logical and Algebraic Characterizations of Transformers
+ Power of Chain of Thought

First-order logic (FO) over strings

- Logical sentences can be used to define sets of strings:

$$\exists\, i.\ \mathtt{a}(i) \wedge \mathtt{b}(i+1)$$

"Contains bigram `ab`"

- First order: can quantify over positions in string (like above)
- FO[<] defines the star-free regular languages (McNaughton & Papert, 1971)

Can we capture transformers with some string logic?

Logical Upper Bound (*Merrill & Sabharwal, 2023b*)
Any language recognized by a log-precision transformer can be defined in first-order logic with majority (FO[M])

Example: defining $\mathtt{a}^n\mathtt{b}^n$ in FO[M]:

$$Mi.\ \mathtt{a}(i) \quad \wedge \quad Mj.\ \mathtt{b}(j) \quad \wedge \quad \neg \exists k.\ [\mathtt{b}(k) \wedge \mathtt{a}(k+1)]$$

1. Most tokens are $\mathtt{a}$
2. Most tokens are $\mathtt{b}$
3. $\mathtt{ba}$ does not occur

Where does this result come from?

- Strengthen TC^0 upper bound for transformers to **log-time** uniform TC^0 (as opposed to log-space)

- Log-time uniform TC^0 = FO[M] (Barrington et al., 1990)

Logical characterization: lower bound

Logical Lower Bound (*Chiang et al., 2023*)
Any language definable in counting logic with + and MOD (FOC[+; MOD]) can be recognized by a transformer

$$\begin{tikzcd} \mathrm{FOC}[+;\ \mathrm{MOD}] \arrow[r, "a^n b^n"'] \arrow[rr, bend left] & \text{Transformer} \arrow[r, "\text{Nested quantifiers}"'] & \mathrm{FO}[\mathrm{M}] \end{tikzcd}$$

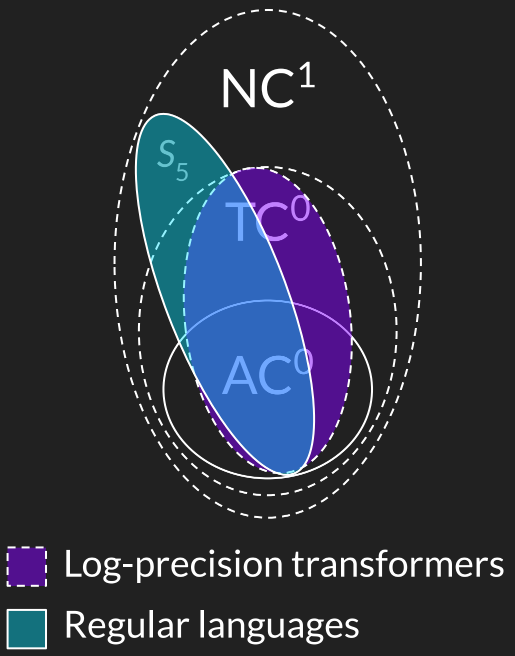
Open: tight (or tighter) logical characterization of transformers

Algebraic View on Transformers’ Expressive Power

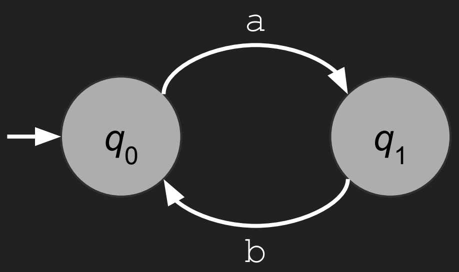
Back to transformers and regular languages

- Recall: transformers cannot recognize all regular languages unless $TC^0 = NC^1$

- Q: Can we characterize which automata transformers can simulate?
- A: Yes, via **solvability of the syntactic monoid** (Liu et al., 2023)



Syntactic monoid of an automaton



- Each word has transition function δ: $Q \to Q$
  - How the word changes the state

$$\delta_{\mathtt{a}} = \{q_0 \to q_1\}$$

$$\delta_{\mathtt{b}} = \{q_1 \to q_0\}$$

- Syntactic monoid:
  - Set of all word transition functions
  - Forms monoid under composition

$$\delta_{\mathtt{ab}} = \{q_0 \to q_0\}$$

Solvable groups and monoids

- **Solvable group:** group can be constructed from Abelian groups via extension
- **Solvable monoid**: all subgroups are solvable (Liu et al. define slightly differently)

- **Intuition:** solvable ≈ “almost commutative”
  - Rich theory for decomposing (simplifying) solvable semigroups (Krohn & Rhodes, 1965)

Transformers can't simulate automata with non-solvable syntactic monoids

Solvability Upper Bound (*Liu et al., 2023*)
Log-precision transformers cannot simulate the state transition sequence for automata with non-solvable syntactic monoids

- Word problem for non-solvable monoids is NC^1-complete: cannot be parallelized!

Linear-width transformers *can* simulate any automaton with a solvable syntactic monoid

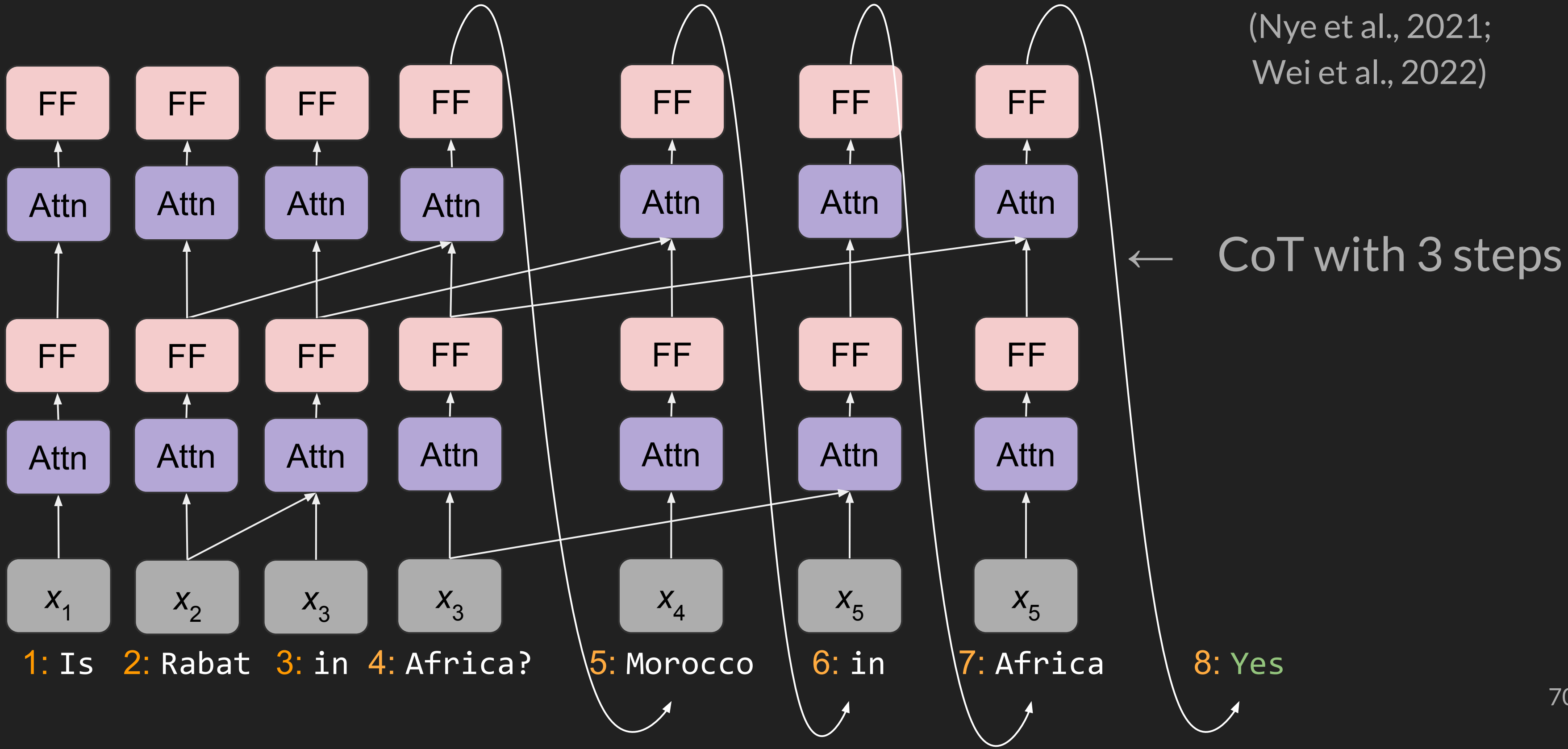
Solvability Lower Bound (*Liu et al., 2023*)
Log-precision transformers with feedforward width $o(n)$ can simulate the transition sequence for automata with solvable syntactic monoids

Note: Allowing width to grow with input is a generous assumption (nonuniform!)

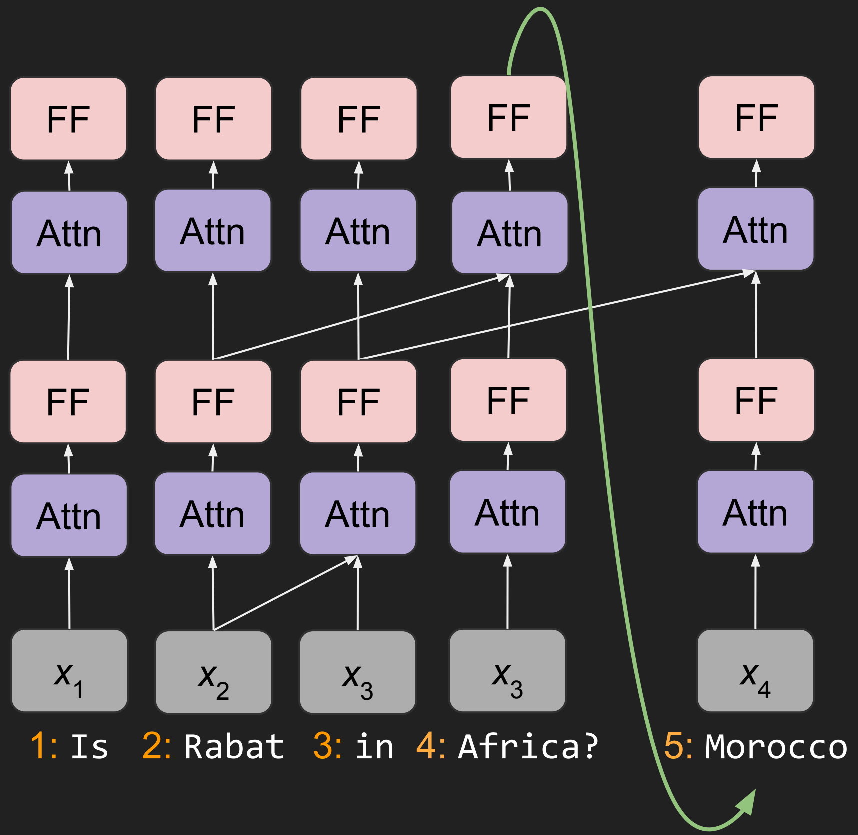
Transformers with Chain of Thought (Preliminary Results)

Chain of thought (CoT): generate tokens *before* answer

(Nye et al., 2021; Wei et al., 2022)



CoT: a way to transcend TC^0



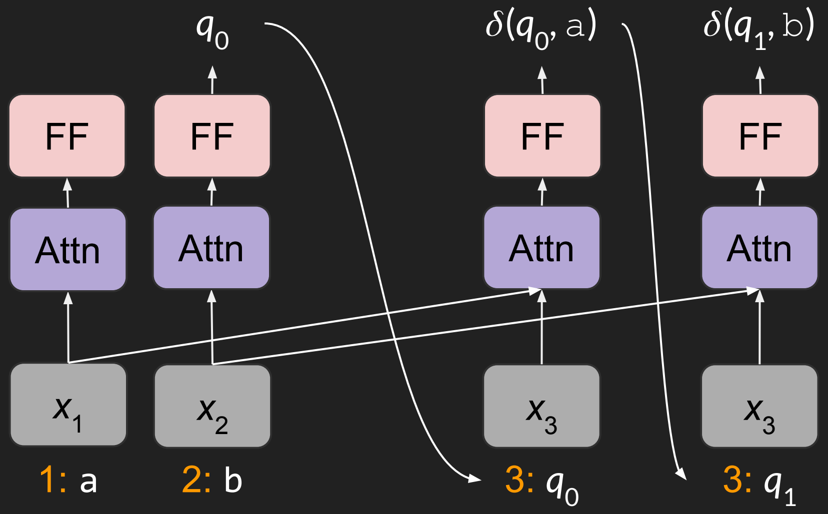
- TC^0: constant-depth/lacks recurrence
- CoT adds depth/recurrence

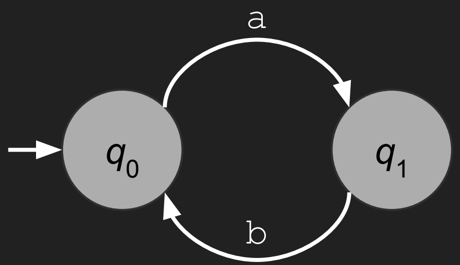
⇒ How much power beyond TC^0 do $T(n)$ CoT steps give?

CoT with linear steps can recognize all regular languages

- DFA simulation: write current state to CoT

(Merrill & Sabharwal, in progress)



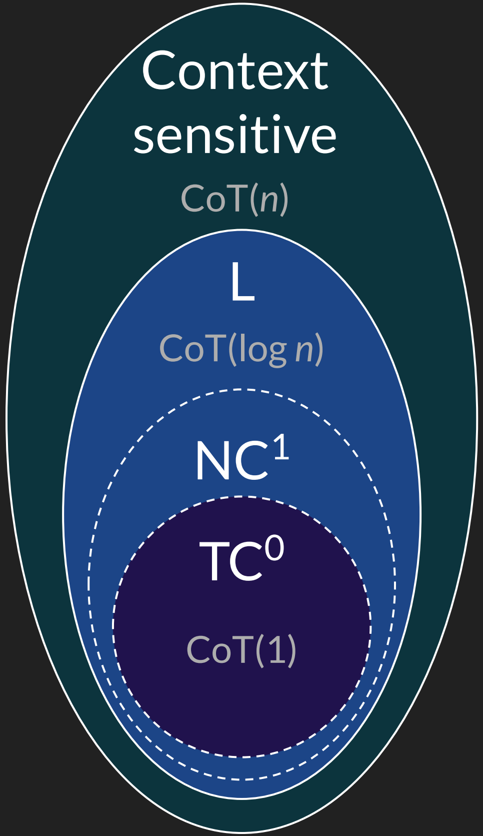


- Chain of thought adds power beyond TC^0
- Can extend to simulate 2-counter automaton (Turing-complete, but slow)

Upper bound for CoT transformers

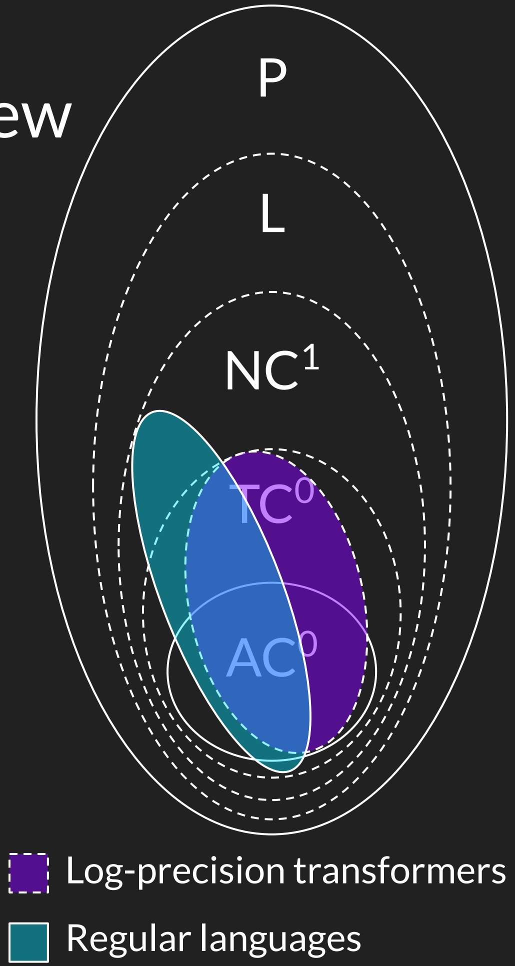
CoT Upper Bound (*Merrill & Sabharwal, in progress*)
Transformers with $T(n)$ CoT steps are in $\mathrm{SPACE}(T(n) + \log n)$

- **With log steps:** transformers can transcend TC^0 but remain in L (logspace)
- **With linear steps:** upper bound of *context-sensitive languages*



New takeaways: extensions to the circuit view

1. RNNs/LSTMs recognize regular/counter languages
2. Transformers recognize languages in **uniform TC^0**
   a. Can't recognize all regular languages
   b. Can't solve graph connectivity, etc. (in theory and in practice)
3. Logical upper and lower bounds for transformers — tight characterization open
   a. Transformers cannot simulate non-solvable automata
   b. Chain-of-thought adds substantial power to transformers



Questions?

Part 4:
Learning Biases of Transformers

Expressive power vs. learnability

- Past section: mostly upper bounds on **expressive power**
    - Upper bounds ⇒ not learnable
    - Lower bounds ⇏ learnable

⇒ What functions can transformers learn via gradient descent?

Outline: learning biases of transformers

1. Transformers biased to low-sensitivity functions

2. Norm growth biases transformers to sparse functions and simplifies attention

What is sensitivity?

- Robust complexity measure for boolean functions $\{0, 1\}^n \to \{0, 1\}$
  - Polynomially related to many other notions of complexity
  - Sensitivity Conjecture recently proven by Huang (2019)

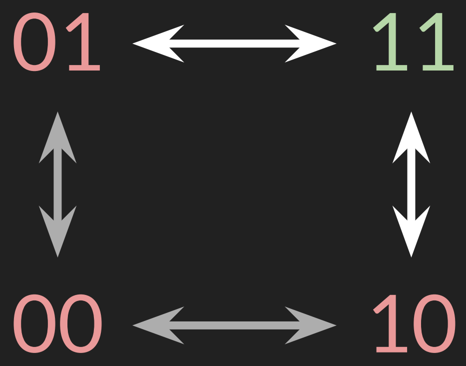
- **Sensitivity (at input):** number of input bits that can be flipped to change output
- **Average sensitivity:** average across all inputs in $\{0, 1\}^n$

Sensitivity example: AND

1. Sensitivity of AND at 01 is 1
   - AND(01) = 0 but AND(11) = 1
   - AND(01) = AND(00)

2. What is the sensitivity of AND at 00 and 11?
   - 00: 0; neither bit changes output when flipped
   - 11: 2; both bits change output when flipped

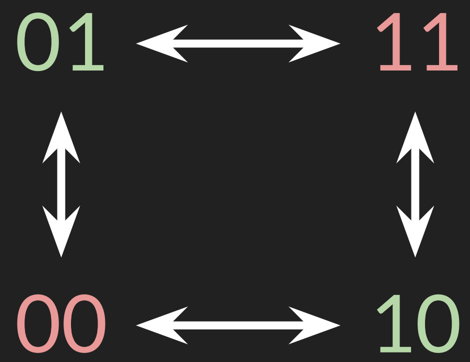
⇒ Average sensitivity of AND is 1
   - (1 + 1 + 0 + 2) / 4



Sensitivity example: PARITY

- On input length *n*, sensitivity of PARITY is *n* for any input
- Average sensitivity of PARITY is also *n*

⇒ PARITY is maximally sensitive



LSTMs and transformers have bias for low sensitivity

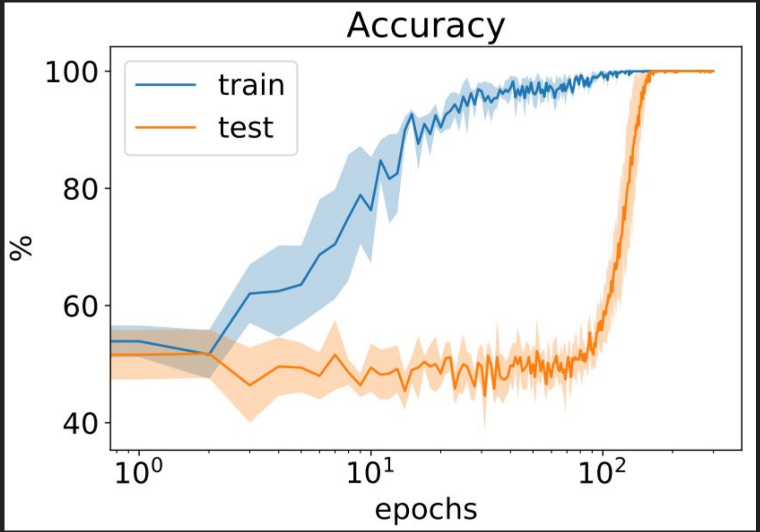
(Hahn et al., 2021; Bhattamishra et al., 2022)

1. At initialization, LSTMS and transformers have low sensitivity
2. After training, lower sensitivity than the function they were trained on
3. Transformers have an *even stronger* low-sensitivity bias than LSTMs

Similar findings for feedforward neural networks (Franco, 2006)

Norm Growth and Saturation

What happens if you train neural nets on (sparse) parity?



(Merrill et al., 2023)

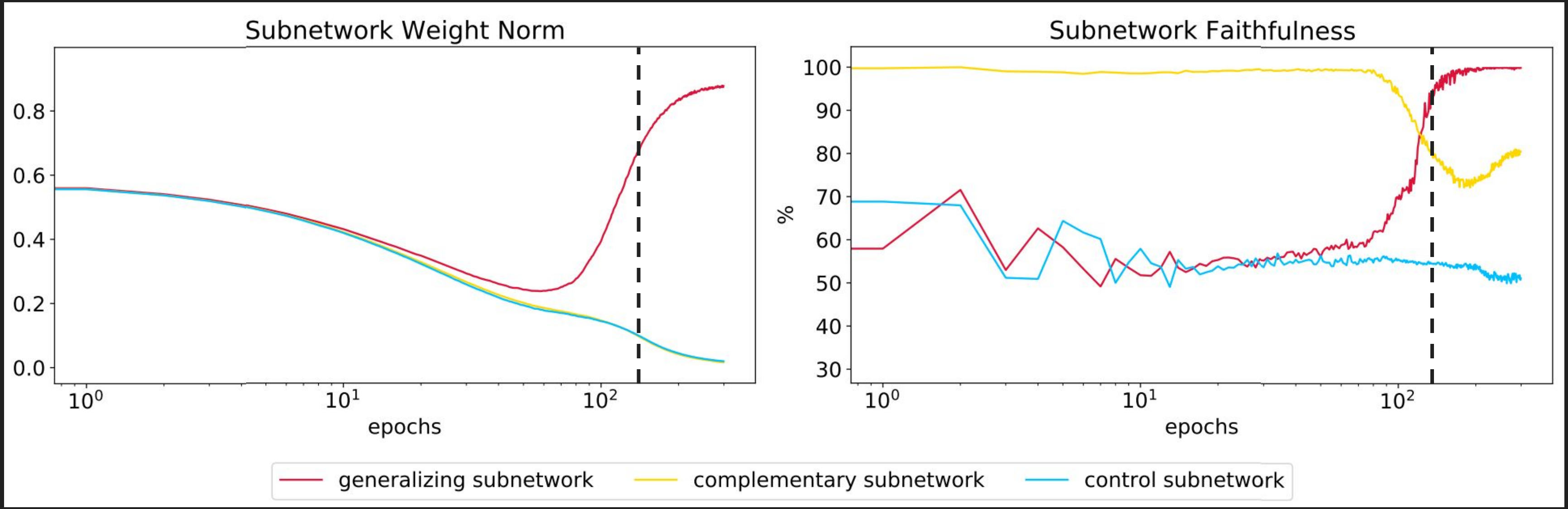
- Initially overfits but magically starts to generalize (without new data!)

- Trend referred to as **grokking**
  (Powers et al., 2022)

What's going on inside the model during grokking?

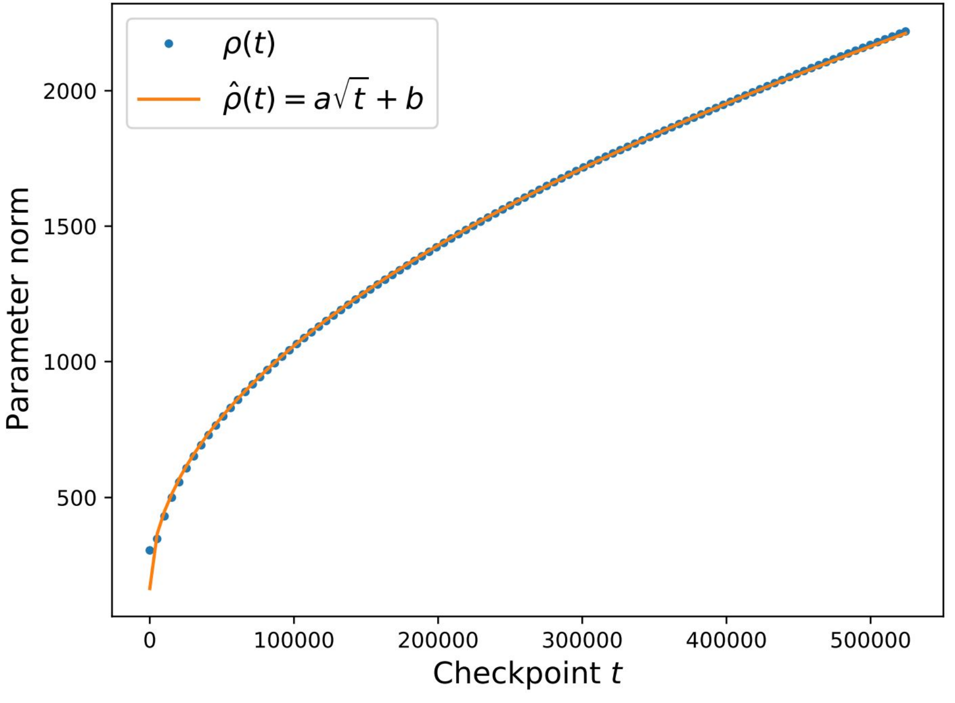
(Merrill et al., 2023)

- Grokking coincides with rapid **norm growth**
- Norm growth ⇒ sparse subnetwork controls network prediction



Norm growth also happens in transformer LMs

(Merrill et al., 2021)



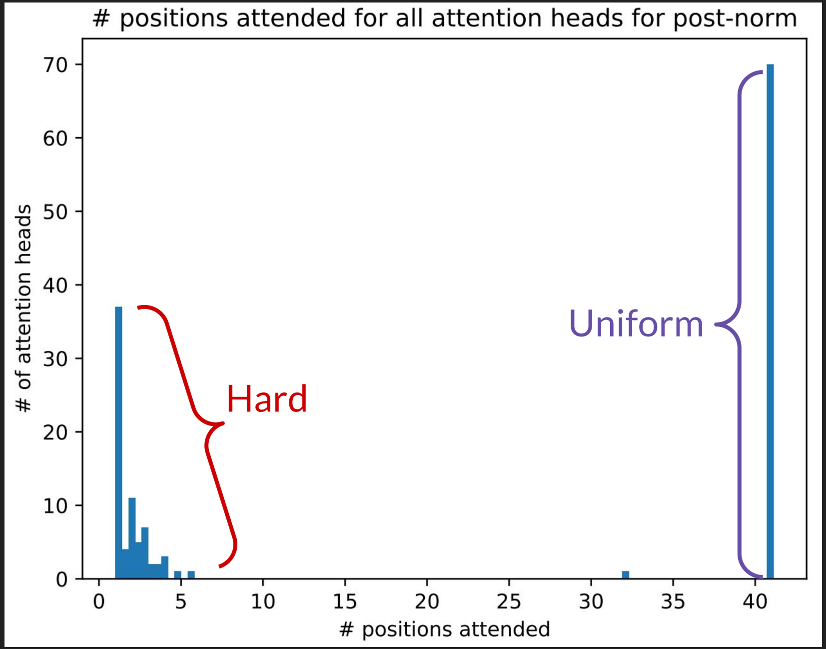
- Norm growth focused at specific weights
  (Dettmers et al., 2022)

Norm growth simplifies attention in transformers

(Merrill et al., 2021)

- Norm growth ⇒ Attention patterns are either hard or uniform

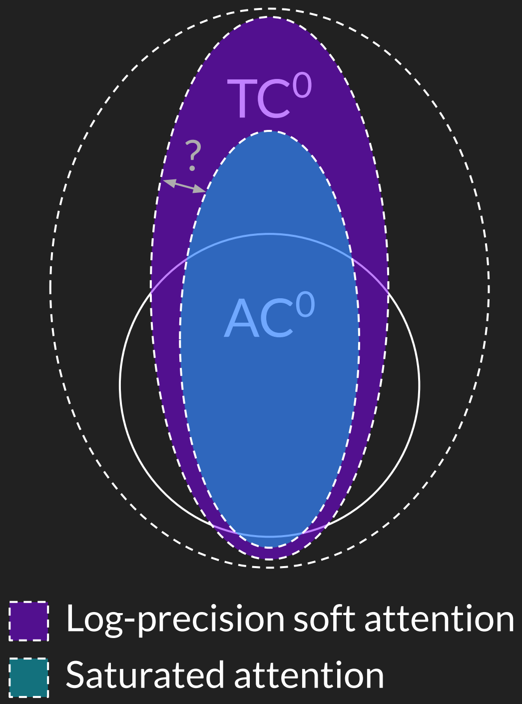
- Why?
  - Norm growth causes transformers to approximate saturated attention



Open question: power of saturated attention

- Saturated attention: roughly, hard + uniform
- Norm growth causes transformers to approximate saturated attention
- Saturated attention also in TC^0 (like soft)

⇒ Is there a separation between soft/saturated?



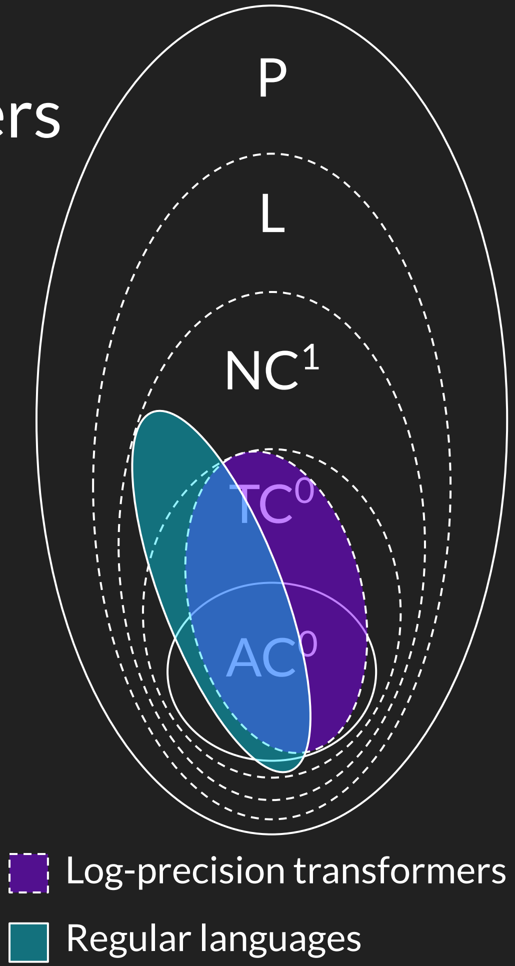
Insights about learning biases of transformers

1. **Sensitivity:** Transformers are biased to low-sensitivity functions

2. **Norm growth:** Caused by gradient descent, induces sparsity and simplifies attention

New takeaways: learning bias of transformers

1. RNNs/LSTMs recognize regular/counter languages
2. Transformers recognize languages in **uniform $\mathbf{TC^0}$**
   a. Can’t recognize all regular languages
   b. Can’t solve graph connectivity, etc. (in theory and in practice)
3. Logical upper and lower bounds for transformers — tight characterization open
   a. Transformers cannot simulate non-solvable automata
   b. Chain-of-thought adds substantial power to transformers
4. Transformers are biased towards low sensitivity and saturation



Conclusion and Takeaways

Themes: transformers' learning biases and limits

Part 4: *What kinds of functions are transformers biased towards learning?*

Part 2 + 3: *What are the limits on the languages can transformers recognize?*

- Lots of recent progress!
- Rich connections to different areas of TCS

Conclusion

1. RNNs/LSTMs recognize regular/counter languages
2. Transformers recognize languages in **uniform TC^0**
   a. Can't recognize all regular languages
   b. Can't solve graph connectivity, etc. (in theory and in practice)
3. Logical upper and lower bounds for transformers — tight characterization open
   a. Transformers cannot simulate non-solvable automata
   b. Chain-of-thought adds substantial power to transformers
4. Transformers are biased towards low sensitivity and saturation