Computability and Transformers:
A Circuit Complexity Revival?

Michaël Cadilhac

Joint work with Andy Yang and David Chiang (Notre Dame)
Appearing at NeurIPS 2025

Computability and Transformers:
A Circuit Complexity Revival?

THE T of GPT

Michaël Cadilhac

DePaul University

Joint work with Andy Yang and David Chiang (Notre Dame)
Appearing at NeurIPS 2025

Goals of this presentation

- Succinctly present transformers (as recognizers)

Goals of this presentation

- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:

Goals of this presentation

- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:
  - Transformers are Turing-complete *when allowed to "think"* for an unbounded number of steps

Goals of this presentation

- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:
  - Transformers are Turing-complete *when allowed to "think"* for an unbounded number of steps
  - Transformers are in small circuit-complexity classes as classifiers

Goals of this presentation

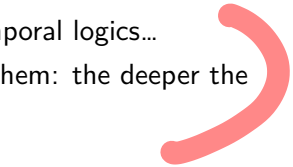
- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:
  - Transformers are Turing-complete *when allowed to "think"* for an unbounded number of steps
  - Transformers are in small circuit-complexity classes as classifiers
- In the latter case, tie transformers to temporal logics...

Goals of this presentation

- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:
  - Transformers are Turing-complete *when allowed to "think"* for an unbounded number of steps
  - Transformers are in small circuit-complexity classes as classifiers
- In the latter case, tie transformers to temporal logics...
- ...and a derive depth-hierarchy result for them: the deeper the transformer, the more it can express

Goals of this presentation

- Succinctly present transformers (as recognizers)
- Tie them to two notions of computability:
  - Transformers are Turing-complete *when allowed to "think"* for an unbounded number of steps
  - Transformers are in small circuit-complexity classes as classifiers
- In the latter case, tie transformers to temporal logics...
- ...and a derive depth-hierarchy result for them: the deeper the transformer, the more it can express



Outline

Transformers

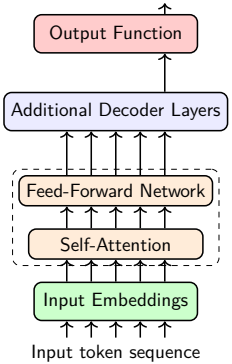
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

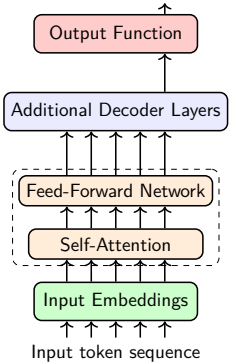
Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



Transformers

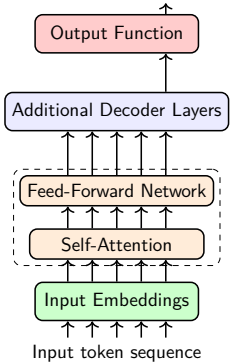
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$E : \Sigma \to \mathbb{R}^d$

Transformers

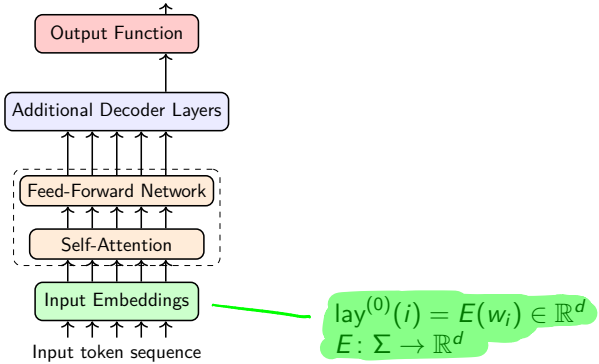
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$$\text{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$
$$E : \Sigma \to \mathbb{R}^d$$

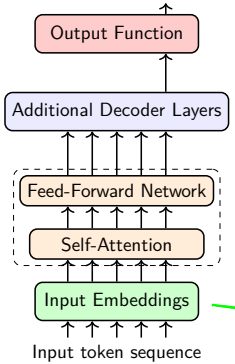
Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



Transformers

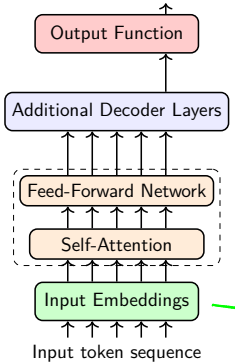
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$$k^{(1)}(i) = K^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$q^{(1)}(i) = Q^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$\mathrm{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$
$$E : \Sigma \to \mathbb{R}^d$$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

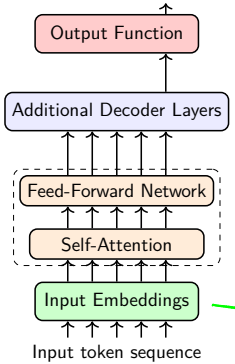


$$k^{(1)}(i) = K^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$q^{(1)}(i) = Q^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$\mathrm{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^{d}$$
$$E : \Sigma \to \mathbb{R}^{d}$$

$\in \mathbb{R}^{d \times d^{\mathrm{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

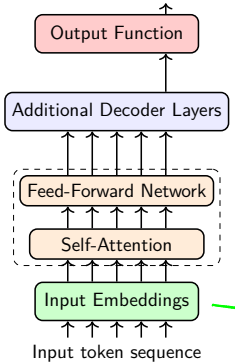


$$\mathrm{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$
$$k^{(1)}(i) = K^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$q^{(1)}(i) = Q^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$\mathrm{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$
$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\mathrm{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

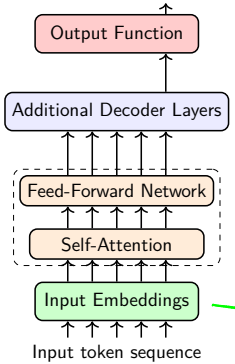


$$v^{(1)}(i) = V^{(1)} \cdot \mathrm{lay}^{(0)}(i) \in \mathbb{R}^d$$
$$\mathrm{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$
$$k^{(1)}(i) = K^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$q^{(1)}(i) = Q^{(1)}\mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$\mathrm{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$
$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\mathrm{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

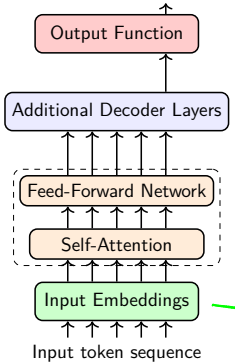


$$v^{(1)}(i) = V^{(1)} \cdot \mathrm{lay}^{(0)}(i) \in \mathbb{R}^d$$
$$\mathrm{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$
$$k^{(1)}(i) = K^{(1)} \mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$q^{(1)}(i) = Q^{(1)} \mathrm{lay}^{(0)}(i) \in \mathbb{R}^{d^{\mathrm{hid}}}$$
$$\mathrm{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$
$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\mathrm{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$$\text{pre-lay}^{(1)}(i) = \frac{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j)) v^{(1)}(j)}{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j))} \in \mathbb{R}^d$$

$$v^{(1)}(i) = V^{(1)} \cdot \text{lay}^{(0)}(i) \in \mathbb{R}^d$$

$$\text{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$

$$k^{(1)}(i) = K^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

$$q^{(1)}(i) = Q^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

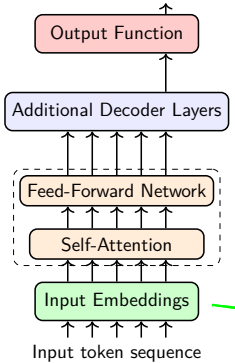
$$\text{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$

$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\text{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$$\text{pre-lay}^{(1)}(i) = \frac{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j)) v^{(1)}(j)}{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j))} \in \mathbb{R}^d$$

$$v^{(1)}(i) = V^{(1)} \cdot \text{lay}^{(0)}(i) \in \mathbb{R}^d$$

$$\text{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$

$$k^{(1)}(i) = K^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

$$q^{(1)}(i) = Q^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

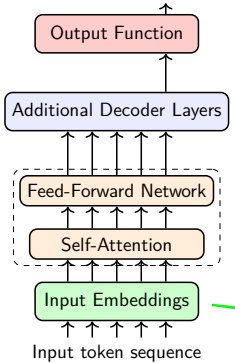
$$\text{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$

$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\text{hid}}}$

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



$$\text{lay}^{(1)}(i) = \text{ffnn}^{(1)}(\text{pre-lay}^{(1)}(i)) \in \mathbb{R}^d$$

$$\text{pre-lay}^{(1)}(i) = \frac{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j)) v^{(1)}(j)}{\sum_{j \le i} \exp(\text{att}^{(1)}(i,j))} \in \mathbb{R}^d$$

$$v^{(1)}(i) = V^{(1)} \cdot \text{lay}^{(0)}(i) \in \mathbb{R}^d$$

$$\text{att}^{(1)}(i,j) = q^{(1)}(i) \cdot k^{(1)}(j) \in \mathbb{R}$$

$$k^{(1)}(i) = K^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

$$q^{(1)}(i) = Q^{(1)} \text{lay}^{(0)}(i) \in \mathbb{R}^{d^{\text{hid}}}$$

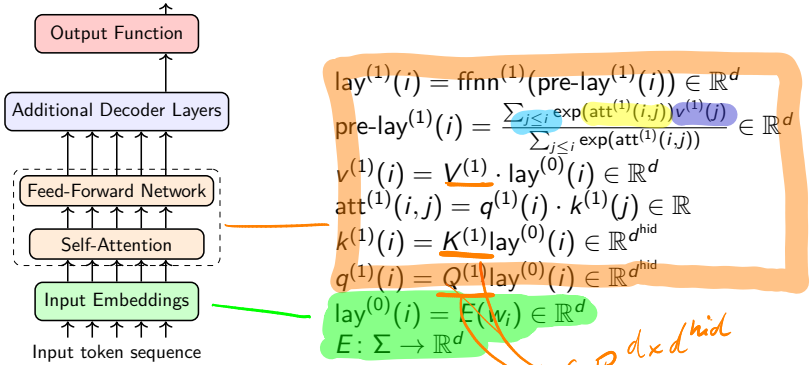
$$\text{lay}^{(0)}(i) = E(w_i) \in \mathbb{R}^d$$

$$E : \Sigma \to \mathbb{R}^d$$

$\in \mathbb{R}^{d \times d^{\text{hid}}}$

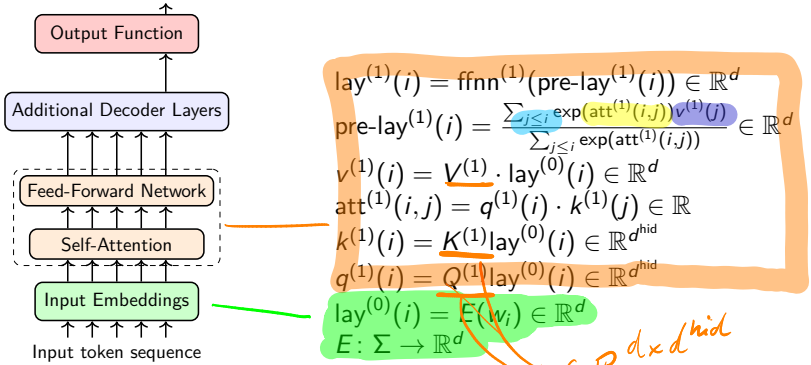
Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

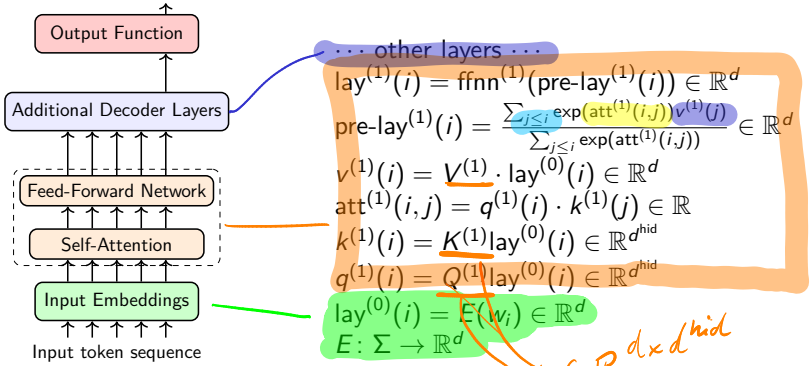


Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

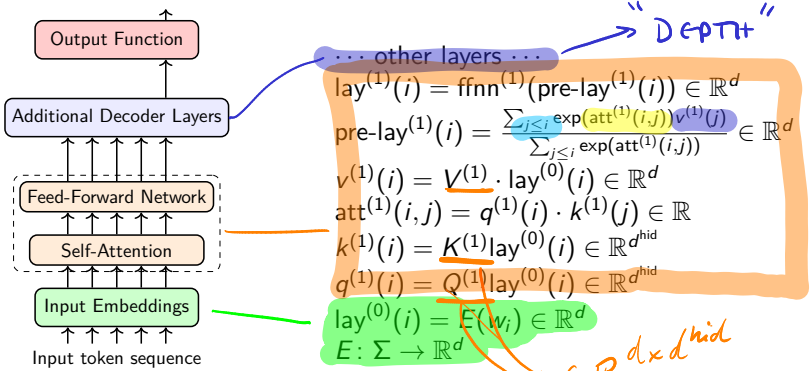


Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

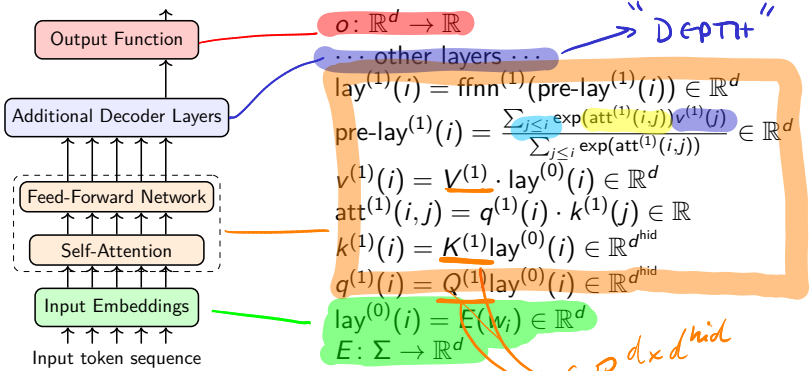


Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

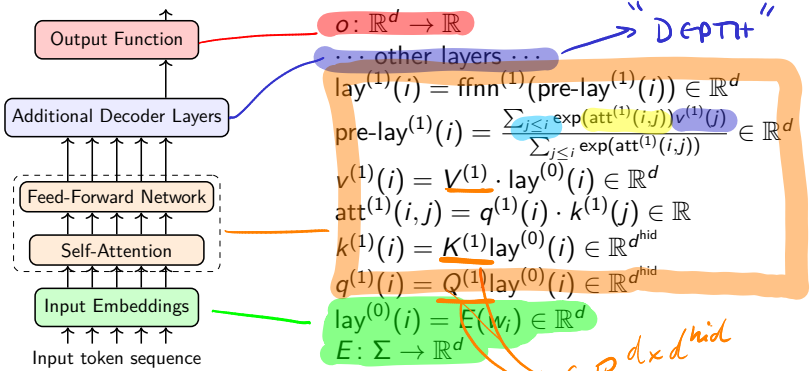


Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)



Transformers as Recognizers

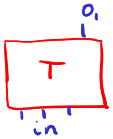
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)

Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"

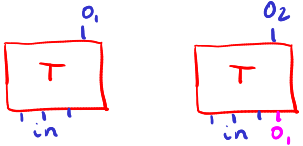
Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



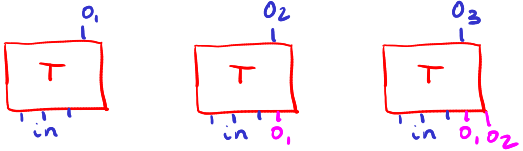
Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



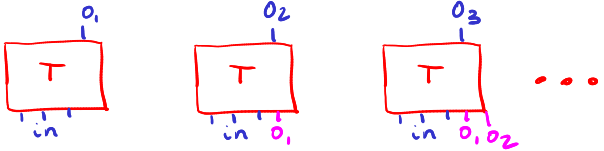
Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



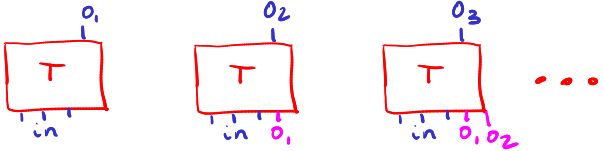
Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



Transformers as Recognizers

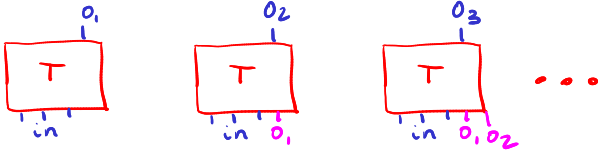
- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



  - *Transformer language:* $\{w \in \Sigma^* \mid T(w) > 0\}$

Transformers as Recognizers

- Transformers compute functions $T : \Sigma^* \to \mathbb{R}$ using a bunch of (mostly) linear transformations
- Transformers have multiple *layers*, each of them accessing the previous layers ("attention" mechanism)
- Language-theoretic objects:
  - *Chain of thought language:* let T "generate" tokens, until it generates "accept/reject"



  - *Transformer language:* $\{w \in \Sigma^* \mid T(w) > 0\}$
  - *What is the complexity of these languages?*

Outline

Chain of thought languages

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

- Based on first character of input, output first transition of M

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

- Based on first character of input, output first transition of M
- After that, at each generation step, count number of left/right moves using attention

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

- Based on first character of input, output first transition of M
- After that, at each generation step, count number of left/right moves using attention
- Track what is cell content at that position

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

- Based on first character of input, output first transition of M
- After that, at each generation step, count number of left/right moves using attention
- Track what is cell content at that position
- Generate the next transition of the Turing machine □

Chain of thought languages

Theorem (Pérez et al. 2019)

"Transformers are Turing-complete"

Idea.

Let M be a TM; a transformer with chain-of-thought can simulate it:

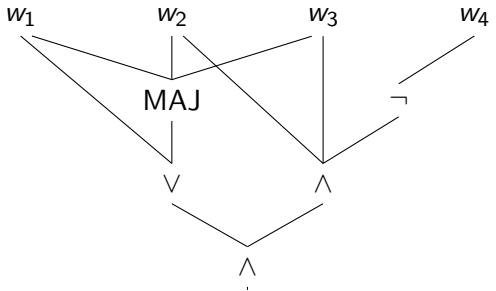
- Based on first character of input, output first transition of M
- After that, at each generation step, count number of left/right moves using attention
- Track what is cell content at that position
- Generate the next transition of the Turing machine □

- Crucially relies on chain-of-thought!

Outline

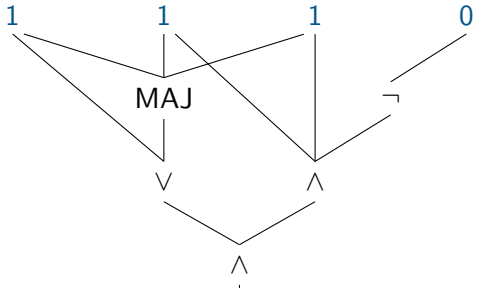
Transformer languages

Preliminary 1: Circuit complexity



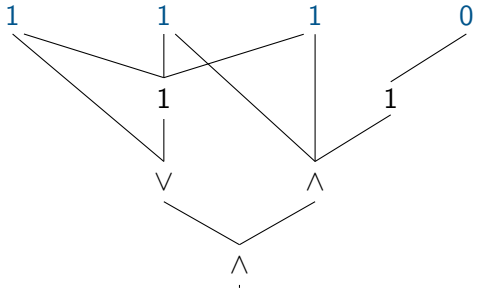
Transformer languages

Preliminary 1: Circuit complexity



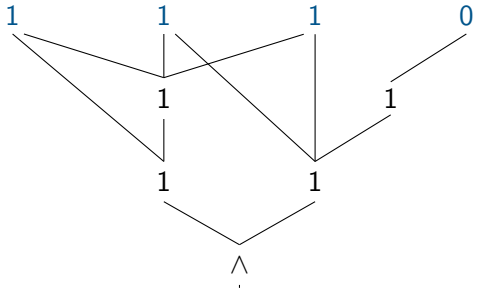
Transformer languages

Preliminary 1: Circuit complexity



Transformer languages

Preliminary 1: Circuit complexity



Attention layer

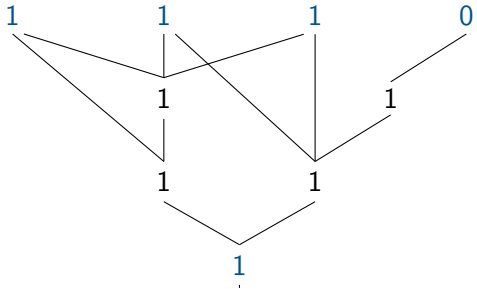
For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Transformer languages

Preliminary 1: Circuit complexity



Attention layer

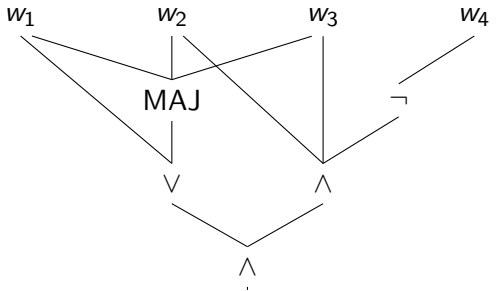
For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Transformer languages

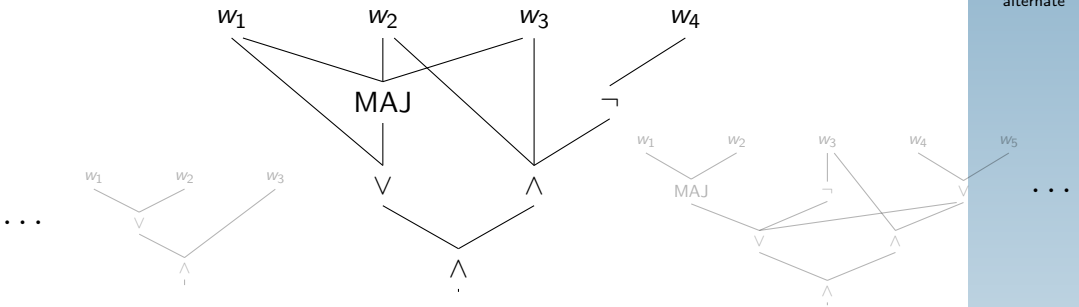
Preliminary 1: Circuit complexity



$$L_4 = \{1110, \ldots\}$$

Transformer languages

Preliminary 1: Circuit complexity



$L_4 = \{1110, \ldots\}$ $\quad L = \bigcup_{i \in \mathbb{N}} L_i$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

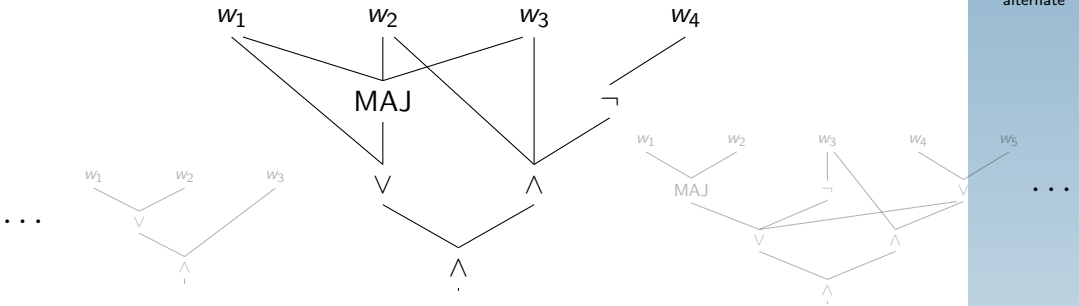
Transformer

- Input encoding
- Attention layers/FFNN alternate

Transformer languages

Preliminary 1: Circuit complexity

size(n) = number of gates

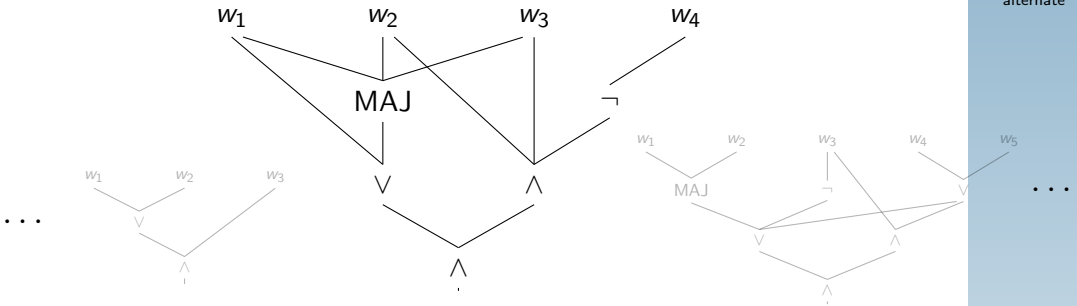


$L_4 = \{1110, \ldots\}$ $\quad L = \bigcup_{i \in \mathbb{N}} L_i$

Transformer languages

Preliminary 1: Circuit complexity

$\text{size}(n)$ = number of gates
= **poly**(n)



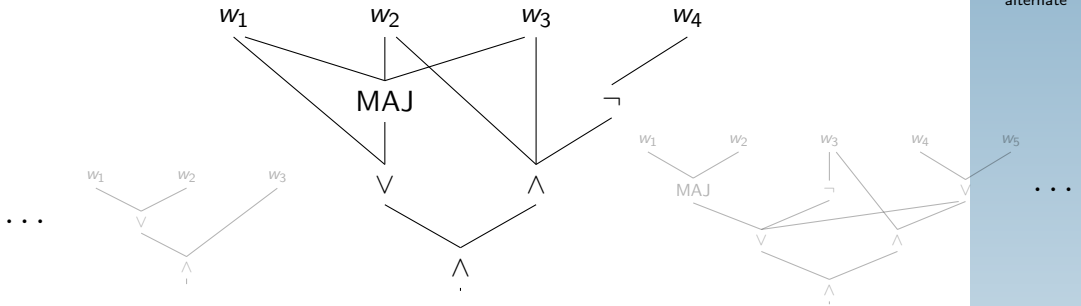
$L_4 = \{1110, \ldots\}$ $\quad$ $L = \bigcup_{i \in \mathbb{N}} L_i$

Transformer languages

Preliminary 1: Circuit complexity

$\text{size}(n)$ = number of gates
= **poly**(n)

$\text{depth}(n)$ = maximal length of a path



$L_4 = \{1110, \ldots\}$ $\quad L = \bigcup_{i \in \mathbb{N}} L_i$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

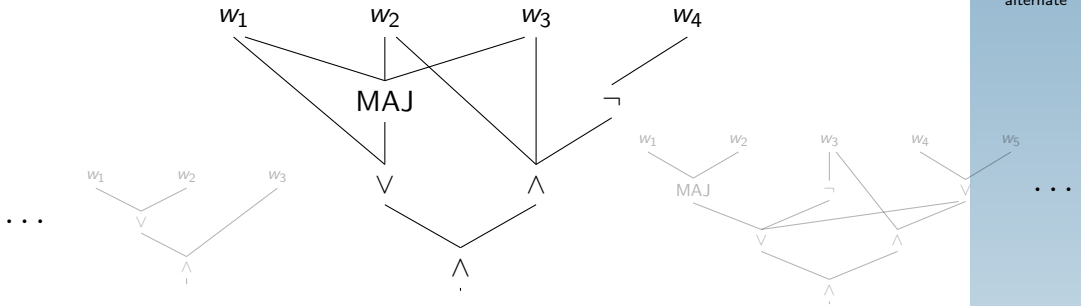
- Input encoding
- Attention layers/FFNN alternate

Transformer languages

Preliminary 1: Circuit complexity

$\text{size}(n)$ = number of gates
= **poly**(n)

$\text{depth}(n)$ = maximal length of a path
= **constant**



$L_4 = \{1110, \ldots\}$ $L = \bigcup_{i \in \mathbb{N}} L_i$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

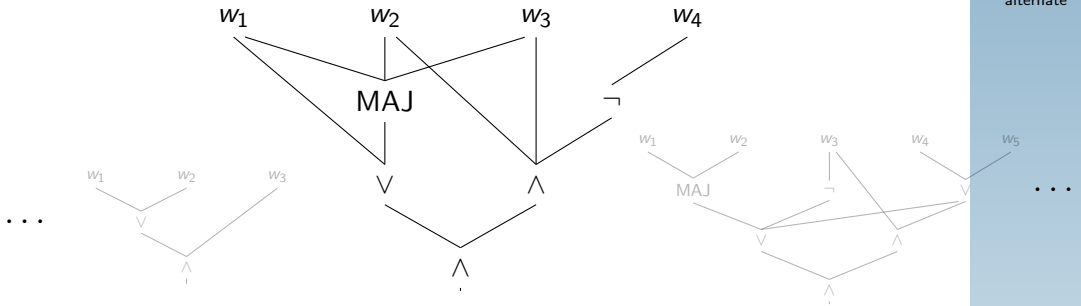
- Input encoding
- Attention layers/FFNN alternate

Transformer languages

Preliminary 1: Circuit complexity

$\text{size}(n)$ = number of gates
= **poly**(n)

$\text{depth}(n)$ = maximal length of a path
= **constant**



$L_4 = \{1110, \ldots\}$ $L = \bigcup_{i \in \mathbb{N}} L_i$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

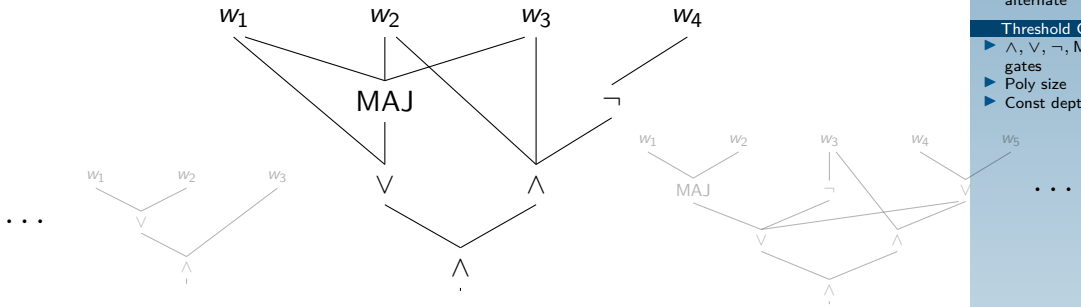
- Input encoding
- Attention layers/FFNN alternate

Transformer languages

Preliminary 1: Circuit complexity

$\text{size}(n)$ = number of gates
= **poly**(n)

$\text{depth}(n)$ = maximal length of a path
= **constant**



$L_4 = \{1110, \ldots\}$

$L = \bigcup_{i \in \mathbb{N}} L_i$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

Transformer languages

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:
  - Addition, multiplication, division, max (for FFNN), exp, iterated addition

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:
  - Addition, multiplication, division, max (for FFNN), exp, iterated addition
  - On floating point numbers (precision $\sim$ log of input length)

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:
  - Addition, multiplication, division, max (for FFNN), exp, iterated addition
  - On floating point numbers (precision $\sim$ log of input length)
- Circuit depth correlates with transformer layer-depth □

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:
  - Addition, multiplication, division, max (for FFNN), exp, iterated addition
  - On floating point numbers (precision $\sim$ log of input length)
- Circuit depth correlates with transformer layer-depth □
- Threshold circuits are in LogSpace, which is in PolyTime

Transformer languages

Theorem (Merrill and Sabharwal, 2023)

Any transformer language is implementable as a threshold circuit family

Idea.

- This requires bounding precision (but not to a constant)
- All the operations used in "executing" a transformer are implementable by threshold circuits:
  - Addition, multiplication, division, max (for FFNN), exp, iterated addition
  - On floating point numbers (precision $\sim$ log of input length)
- Circuit depth correlates with transformer layer-depth □

- Threshold circuits are in LogSpace, which is in PolyTime
- What about a converse? Or something exact?

Transformer languages

Preliminary 2: Temporal logic with counting

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\text{true}] = 1 \wedge a$

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

→ WELL PARENTHESIZED

a (b)

- TL[$\overleftarrow{\#}$]: temporal logic with $\overleftarrow{\#}$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

- TL$[\overleftarrow{\#}]$: temporal logic with $\overleftarrow{\#}$

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

- TL$[\overleftarrow{\#}]$: temporal logic with $\overleftarrow{\#}$
- TL$[\overleftarrow{\#}]$ cannot express $(a + b)^* aa (a + b)^*$ (Behle & Krebs 2008)

Transformer languages

Preliminary 2: Temporal logic with counting

By examples with $w \in \{a, b\}^*$:

- $w \models a$ The *last* letter of w is a
- $w \models (\overleftarrow{\#}[a] = \overleftarrow{\#}[b])$ As many a's as b's
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[\text{true}] = 1 \wedge a] = 1$ First letter is a
- $w \models \overleftarrow{\#}[\overleftarrow{\#}[a] < \overleftarrow{\#}[b]] = 0$ No prefix has more b's than a's
- Language of ϕ: $\{w \mid w \models \phi\}$

- TL$[\overleftarrow{\#}]$: temporal logic with $\overleftarrow{\#}$
- TL$[\overleftarrow{\#}]$ cannot express $(a + b)^* aa (a + b)^*$ (Behle & Krebs 2008)

Proposition (Behle & Krebs 2008)

TL$[\overleftarrow{\#}]$ *languages are all implementable as threshold circuits of linear size*

Transformer languages

An exact characterization

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Idea.

- A more careful study of the behavior of attention

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Idea.

- A more careful study of the behavior of attention
- Rounding/precision is done in a slightly different way

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Idea.

- A more careful study of the behavior of attention
- Rounding/precision is done in a slightly different way
- The nesting of $\overleftarrow{\#}$ corresponds to the depth of the transformer □

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Idea.

- A more careful study of the behavior of attention
- Rounding/precision is done in a slightly different way
- The nesting of $\overleftarrow{\#}$ corresponds to the depth of the transformer □

Lies, a bunch of lies! Certainly transformers can do $(a+b)^*aa(a+b)^*$*!*

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

An exact characterization

Theorem (Yang, Cadilhac, Chiang 2025)

Transformer languages are exactly those of $\mathsf{TL}[\overleftarrow{\#}]$

Idea.

- A more careful study of the behavior of attention
- Rounding/precision is done in a slightly different way
- The nesting of $\overleftarrow{\#}$ corresponds to the depth of the transformer □

Lies, a bunch of lies! Certainly transformers can do $(a+b)^*aa(a+b)^*$*!*

- The *main* lie: *positional encoding*
- Some other lesser lies (fixed precision at some places)

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

$aaabbaa \in L_3$

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

$aaabbaa \in L_3$

Theorem (Yang, Cadilhac, Chiang 2025)

L_k is in depth-k $\mathrm{TL}[\overleftarrow{\#}]$ but not in depth-$(k-1)$ $\mathrm{TL}[\overleftarrow{\#}]$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

$aaabbaa \in L_3$

Theorem (Yang, Cadilhac, Chiang 2025)

L_k is in depth-k $\mathrm{TL}[\overleftarrow{\#}]$ *but not in depth-$(k-1)$* $\mathrm{TL}[\overleftarrow{\#}]$

Idea.

- Depth-1 $\mathrm{TL}[\overleftarrow{\#}]$ languages are commutative (letters commute)

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

$aaabbaa \in L_3$

Theorem (Yang, Cadilhac, Chiang 2025)

L_k is in depth-k $\mathrm{TL}[\overleftarrow{\#}]$ *but not in depth-$(k-1)$* $\mathrm{TL}[\overleftarrow{\#}]$

Idea.

- Depth-1 $\mathrm{TL}[\overleftarrow{\#}]$ languages are commutative (letters commute)
- A reduction lemma: if L_m is in depth n, then L_{m-1} is in depth $n-1$ □

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy

- L_k: all of words over $\{a, b\}$ that alternate k times between a's and b's

aaa bb aa $\in L_3$

Theorem (Yang, Cadilhac, Chiang 2025)

L_k is in depth-k $\mathrm{TL}[\overleftarrow{\#}]$ but not in depth-$(k-1)$ $\mathrm{TL}[\overleftarrow{\#}]$

Idea.

- Depth-1 $\mathrm{TL}[\overleftarrow{\#}]$ languages are commutative (letters commute)
- A reduction lemma: if L_m is in depth n, then L_{m-1} is in depth $n-1$ □

Corollary

Tranformers of depth k can express L_k but not L_{k+1}

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy in Practice: Expressing vs Training

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Transformer languages

A depth hierarchy in Practice: Expressing vs Training

Accuracy on $[101, 150]$

depth →	1	2	3	4	5	6	7	8	9	10
L_3	100	100	100	100	100	100	100	100	100	100
L_4	44	100	100	100	100	100	100	100	100	100
L_5	42	67	100	100	100	100	100	100	100	100
L_6	36	46	61	100	100	100	100	100	100	100
L_7	36	52	55	65	100	100	100	100	100	100
L_8	30	37	42	42	65	100	100	100	100	100
L_9	8	35	40	46	43	38	87	75	100	80
L_{10}	5	8	25	33	25	27	44	30	44	47
L_{11}	5	10	12	17	16	17	24	18	24	17
L_{12}	0	0	0	9	13	0	4	4	19	16

Accuracy on $[151, 200]$

depth →	1	2	3	4	5	6	7	8	9	10
L_3	100	100	100	100	100	100	100	100	100	100
L_4	37	100	100	100	100	100	100	100	100	100
L_5	30	60	100	100	100	100	100	100	100	100
L_6	19	30	54	98	98	98	98	99	98	100
L_7	26	34	35	48	100	99	99	98	98	99
L_8	17	16	18	16	40	99	100	99	99	100
L_9	3	14	11	18	19	32	80	63	99	69
L_{10}	2	6	5	9	7	5	13	8	21	30
L_{11}	2	3	3	4	4	3	4	3	5	3
L_{12}	0	0	0	3	6	0	2	2	11	6

Accuracy on $[201, 250]$

depth →	1	2	3	4	5	6	7	8	9	10
L_3	99	100	100	100	100	100	100	100	100	100
L_4	26	99	99	100	100	100	100	100	100	100
L_5	24	50	99	99	98	100	100	100	100	100
L_6	8	15	38	93	89	92	90	94	91	96
L_7	18	20	19	31	96	93	96	89	92	95
L_8	10	7	8	7	23	97	98	96	97	98
L_9	1	7	3	7	9	26	75	60	98	62
L_{10}	0	7	1	2	1	1	3	2	7	22
L_{11}	0	0	1	1	1	0	1	0	1	0
L_{12}	0	0	0	0	0	0	0	0	1	1

Accuracy on $[251, 300]$

depth →	1	2	3	4	5	6	7	8	9	10
L_3	99	100	100	100	100	100	100	100	100	100
L_4	20	96	96	99	100	100	100	100	100	99
L_5	19	41	97	96	91	99	99	99	100	100
L_6	5	8	27	84	77	83	78	85	80	90
L_7	13	13	12	23	87	84	89	72	82	88
L_8	5	2	2	2	13	91	94	91	89	94
L_9	1	4	1	3	5	25	71	58	95	57
L_{10}	0	4	0	0	0	5	1	0	3	20
L_{11}	0	0	0	0	0	0	0	0	0	0
L_{12}	0	0	0	0	0	0	0	0	0	0

Figure 4: Experimental results. Corollary 5.2 predicts that a transformer with depth k can recognize language L_{k+2} but not L_{k+3} (demarcated by the black line). Up to L_9, this closely predicts our experimental results (shown as numbers and colors).

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Outline

Transformers

What can transformers compute, if we give them time to think?

What can transformers compute in one evaluation?

Conclusion

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

TL[$\overleftarrow{\#}$]

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge$, $\vee$, $\neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathrm{TL}[\overleftarrow{\#}]$

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathrm{TL}[\overleftarrow{\#}]$
  - A depth hierarchy can be deduced

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathrm{TL}[\overleftarrow{\#}]$
  - A depth hierarchy can be deduced

Questions:

- Main question: Learnability vs expressiveness

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathrm{TL}[\overleftarrow{\#}]$
  - A depth hierarchy can be deduced

Questions:

- Main question: Learnability vs expressiveness
- Fix the lies (positional encoding!)

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathsf{TL}[\overleftarrow{\#}]$
  - A depth hierarchy can be deduced

Questions:

- Main question: Learnability vs expressiveness
- Fix the lies (positional encoding!)
- Can transformers contribute to circuit complexity?

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathsf{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$

Conclusion

We saw:

- Transformers = just a bunch of (mostly) linear operations
- Allowed to "think:" Turing-complete
- As language acceptors:
  - Expressible using threshold circuits
  - Exactly equivalent (more or less!) to $\mathrm{TL}[\overleftarrow{\#}]$
  - A depth hierarchy can be deduced

Questions:

- Main question: Learnability vs expressiveness
- Fix the lies (positional encoding!)
- Can transformers contribute to circuit complexity?

> **On the circuit lower bound barrier.** It was suggested in [Hah20] that unconditional lower bounds against *encoder-only* transformer would imply breakthrough circuit lower bounds against linear threshold circuits.[1] Our result avoids this barrier by exploiting the information bottleneck and autoregressive nature of *decoder-only* models. Namely, our proof crucially depends on the fact that in decoder-only models, each token cannot attend to any token after it.

Attention layer

For each pos i, compute score against j, weigh by a value, then take avg (softmax)

Transformer

- Input encoding
- Attention layers/FFNN alternate

Threshold Circuits

- $\wedge, \vee, \neg$, MAJ gates
- Poly size
- Const depth

$\mathrm{TL}[\overleftarrow{\#}]$

- Temporal logic
- Evaluated at end
- With counting in the past $\overleftarrow{\#}$