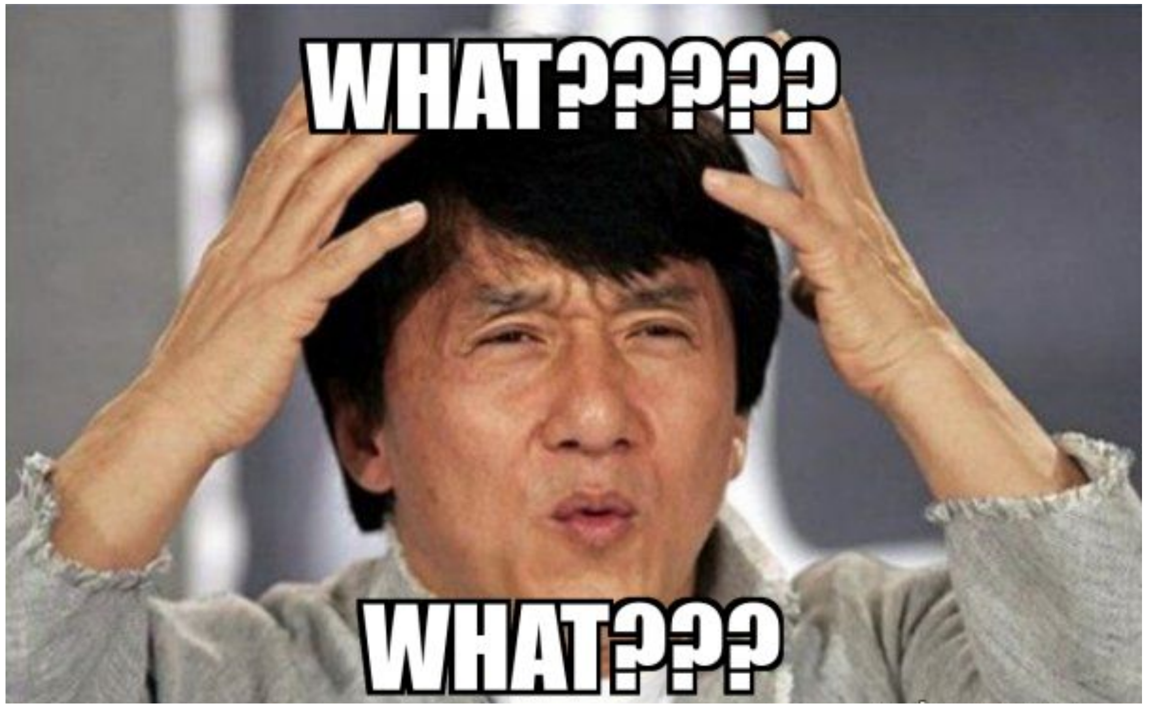
Sandbox for the Blackbox

How language models learn structured data

Ashok Vardhan Makkuva

(EPFL ⟶ Télécom Paris)


WHAT?????
WHAT???

LLMs are part of daily lives

The good and the bad

The good and the bad

Impressive language skills

Programming

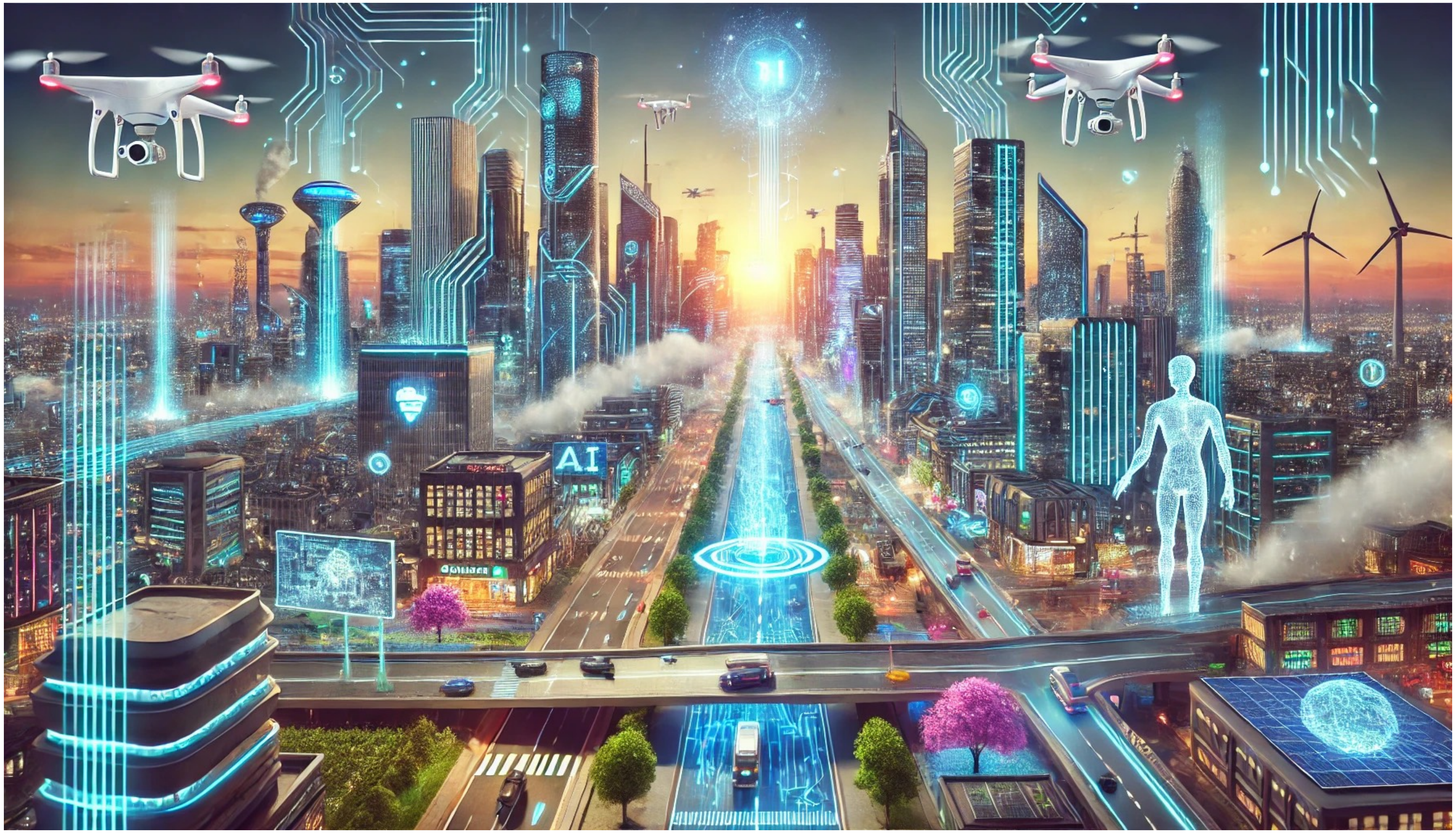
✓

Arithmetic reasoning

Hallucinations

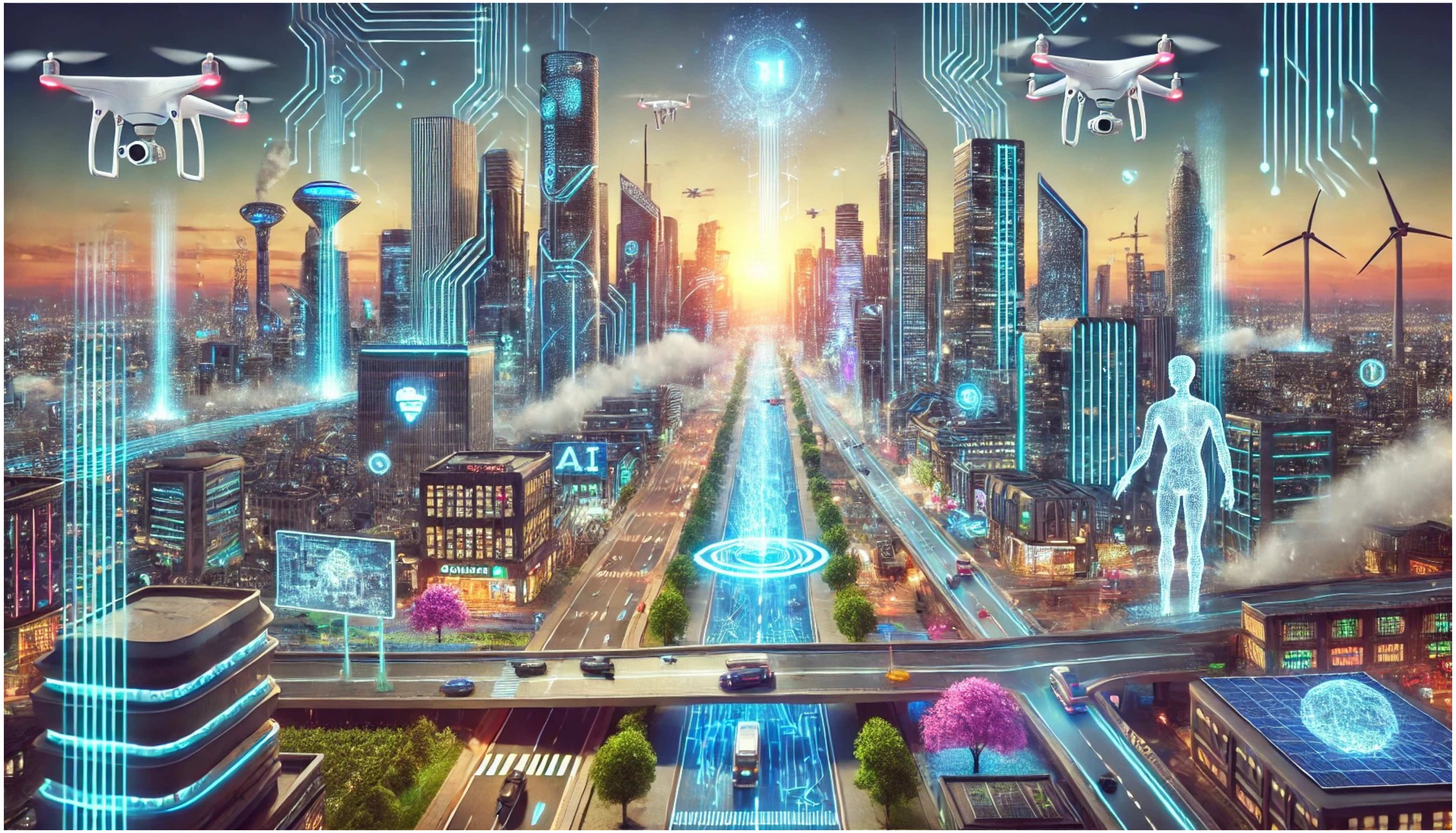
✗

Next-generation technologies



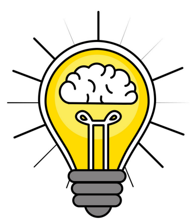
But..

To realize the full potential of AI..



Need of the hour

Fundamental understanding



Fundamental understanding

What do they learn?

How do they learn?

Principled frameworks and tools

What do they learn?

How do they learn?

Challenges

Challenges

Inherently complex

Mathematically intractable

Why complex?

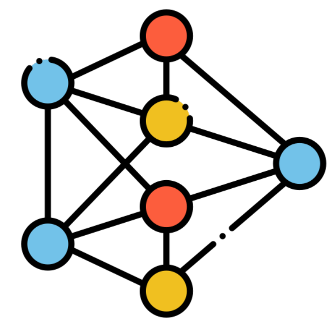
Why complex?

Input data

(source, size, ordering..)

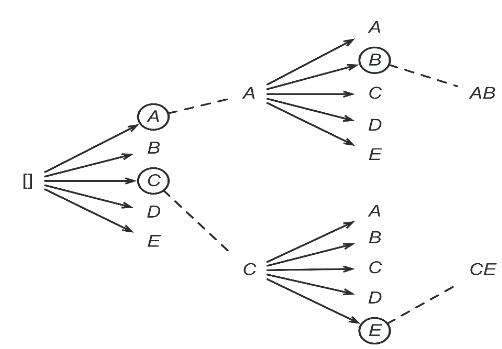
Optimization

(loss, optimiser, hyperparams..)



Model

(size, activation, normalization..)



Generation

(Sampling, order, strategy...)

Why complex?

Too many tuning knobs

$63M

Need a principled way to make progress

Need a principled way to make progress

- Useful abstractions: Sandboxes

Sandboxes

Simple enough to be mathematically **tractable** yet **powerful** enough to suggest practical interventions

How sandboxes help?

How sandboxes help?

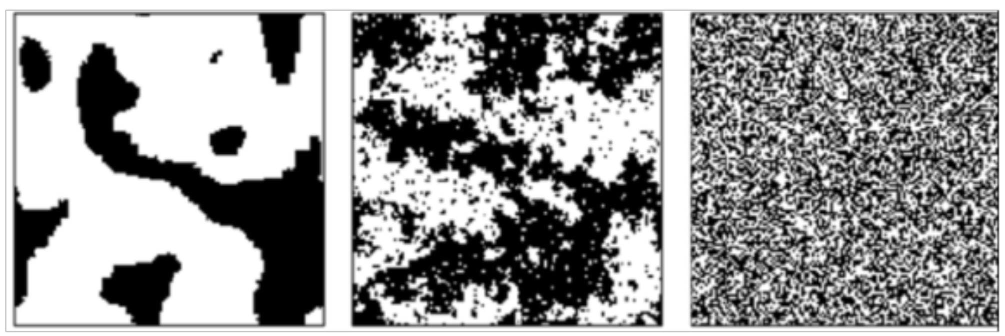
1. Clarity & scientific understanding.

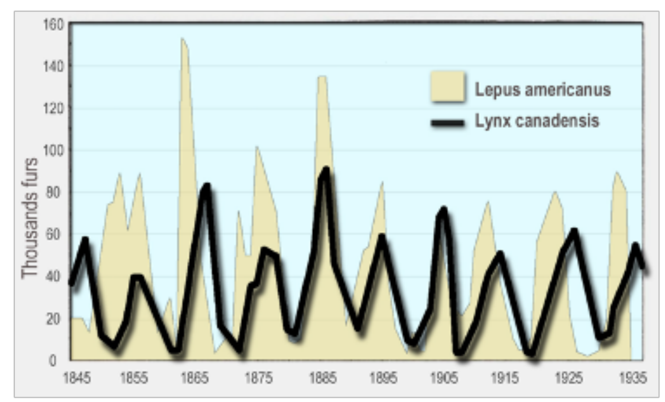
How sandboxes help?

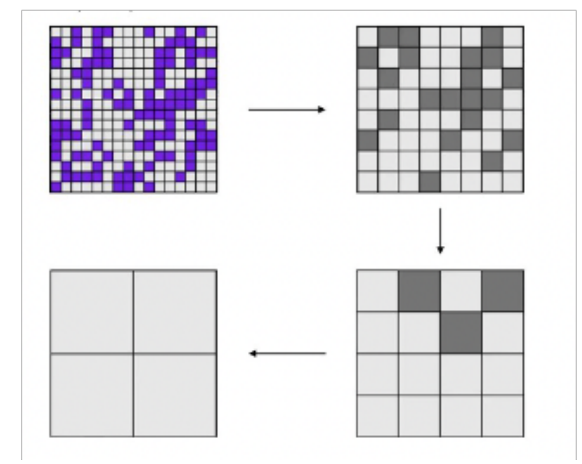
1. Clarity & scientific understanding.

Example: Many groundbreaking works in physics and biology: randomness in statistical physics,

Hardy-Weinberg equilibria (population genetics), Lotka-Volterra (predator-prey)





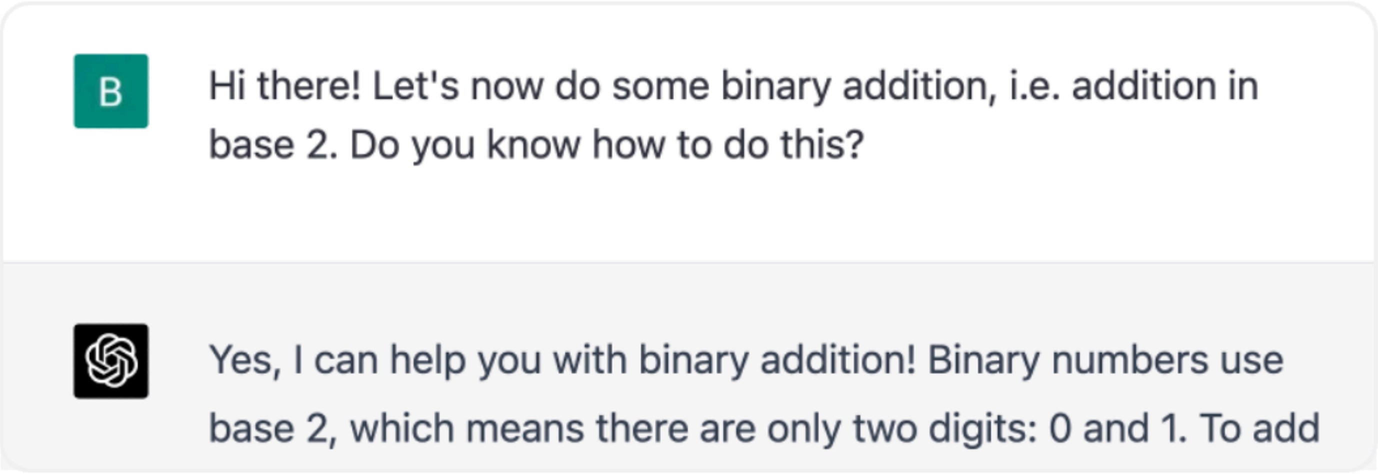


How sandboxes help?

1. Clarity & scientific understanding.
2. Diagnoses & stress tests.

How sandboxes help?

1. Clarity & scientific understanding.
2. Diagnoses & stress tests.



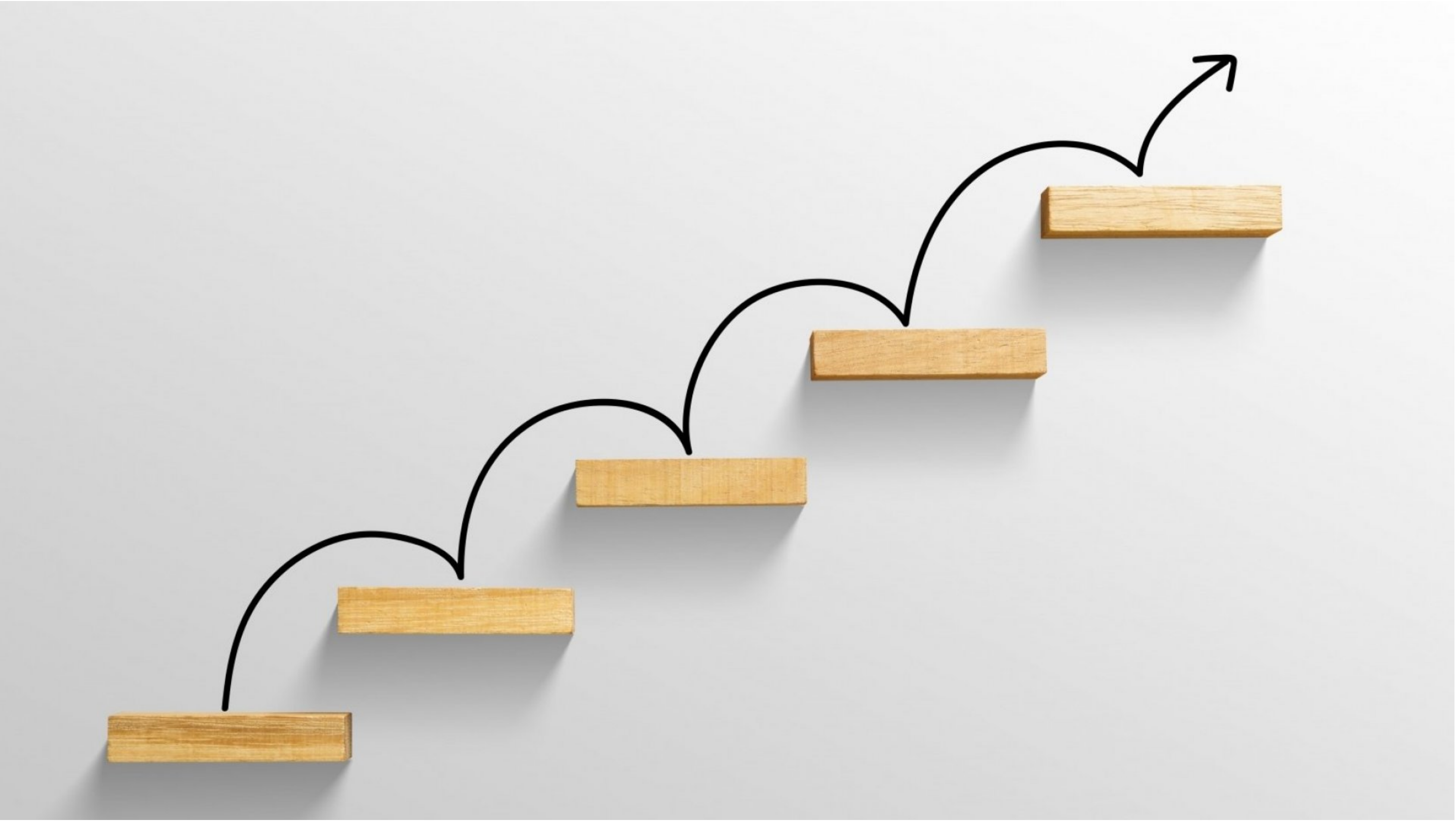

```
   11100111
 + 11011111
 ----------
 11 10001010
```

How they succeed or fail on even the simplest arithmetic tasks

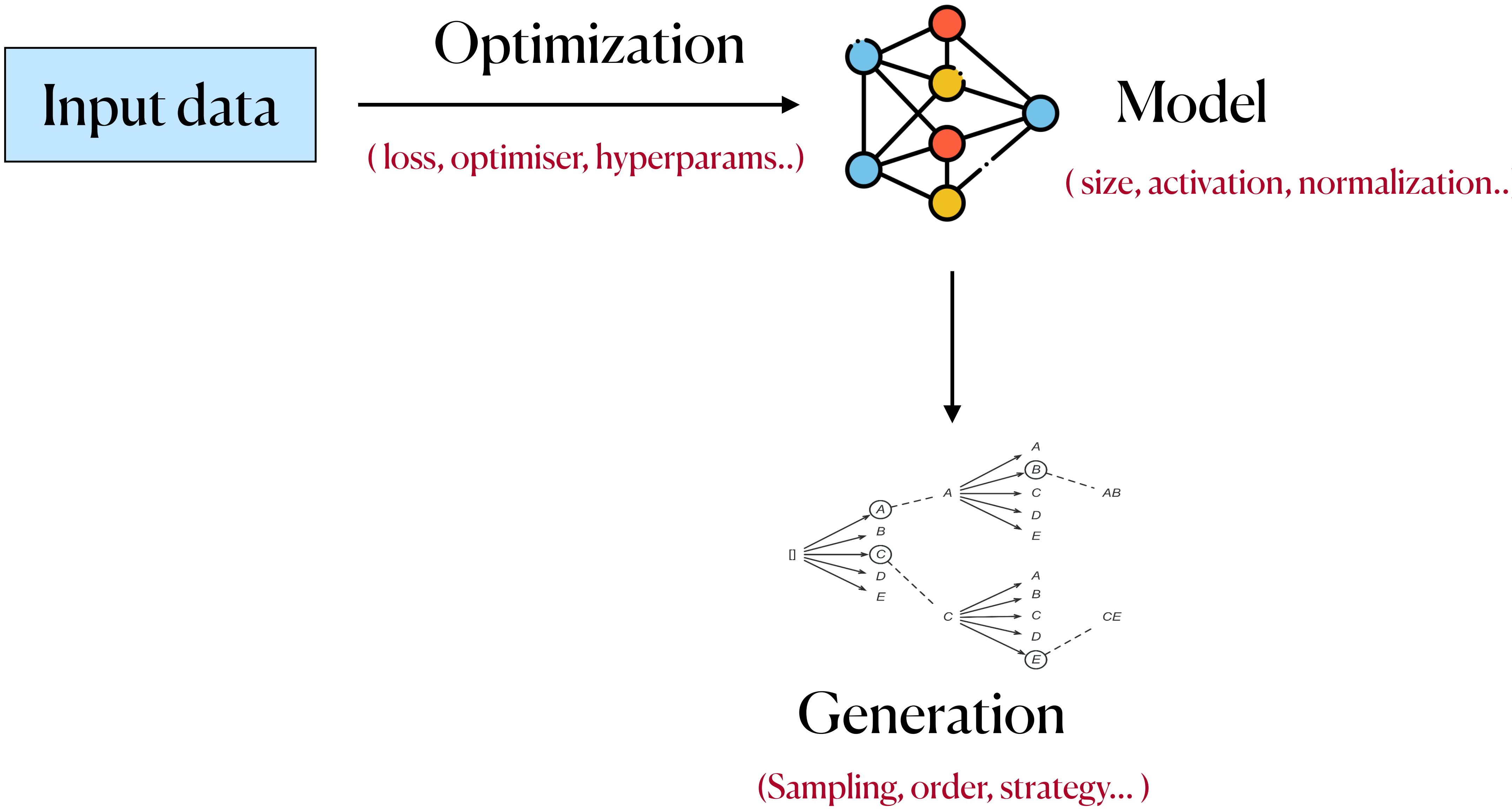
How sandboxes help?

1. Clarity & scientific understanding.
2. Diagnoses & stress tests.

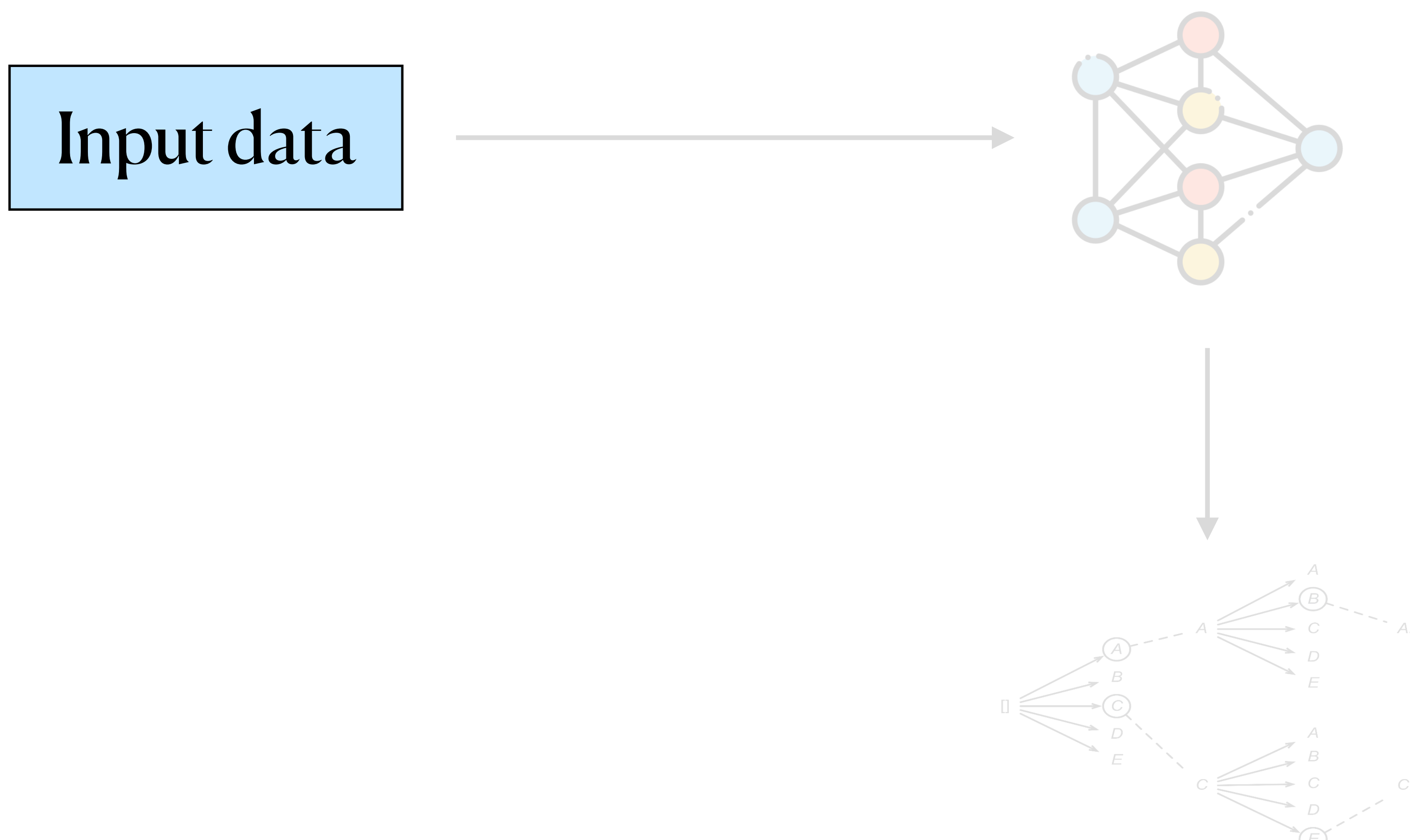
Algorithm design



Sandboxes


Input data
Optimization
(loss, optimiser, hyperparams..)
Model
(size, activation, normalization..)
Generation
(Sampling, order, strategy...)

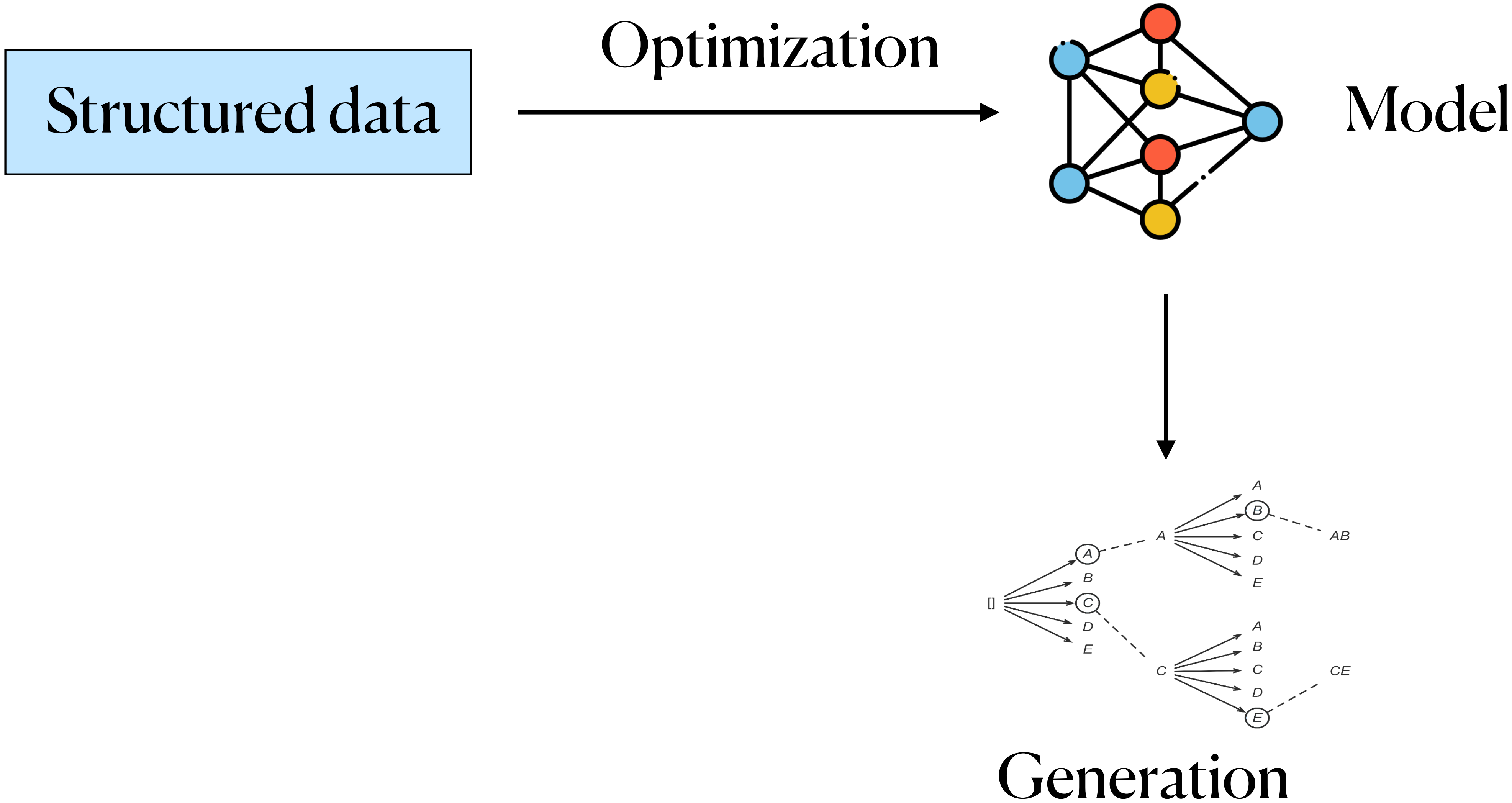
Sandboxes



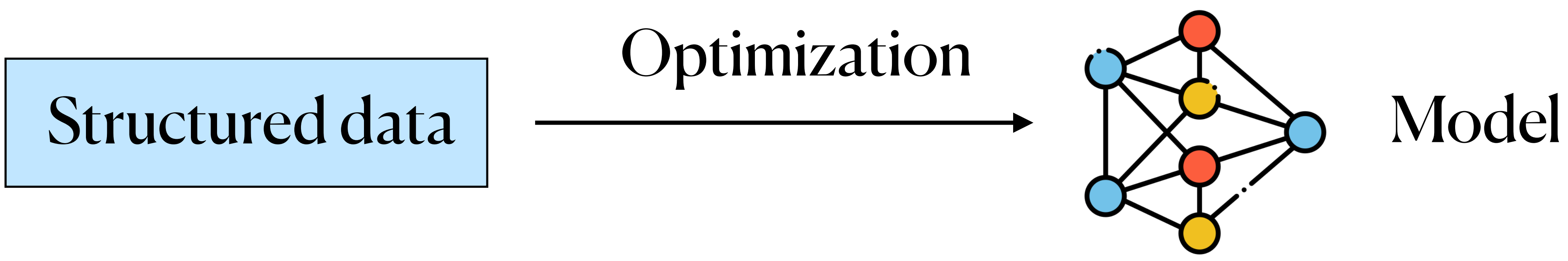
Sandboxes

Structured data

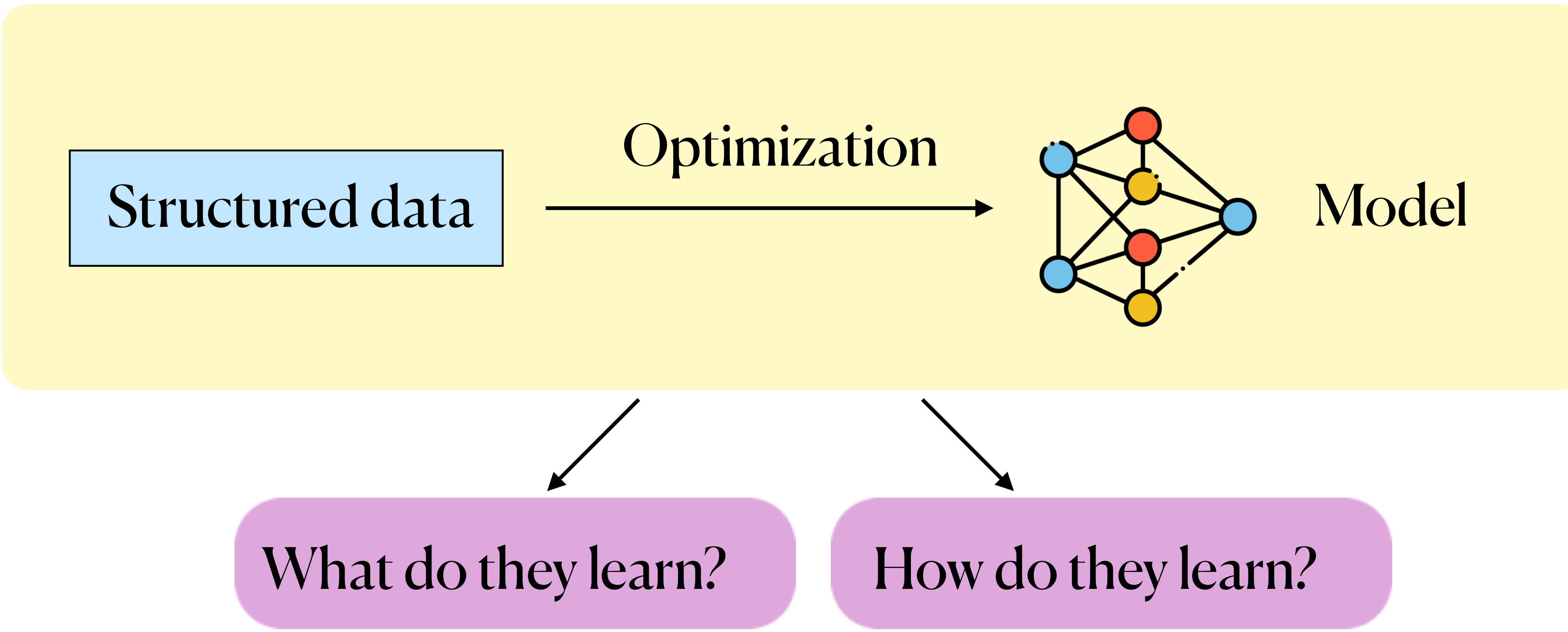
Sandboxes


Structured data
Optimization
Model
Generation
[]
A
B
C
D
E
A
AB
C
CE

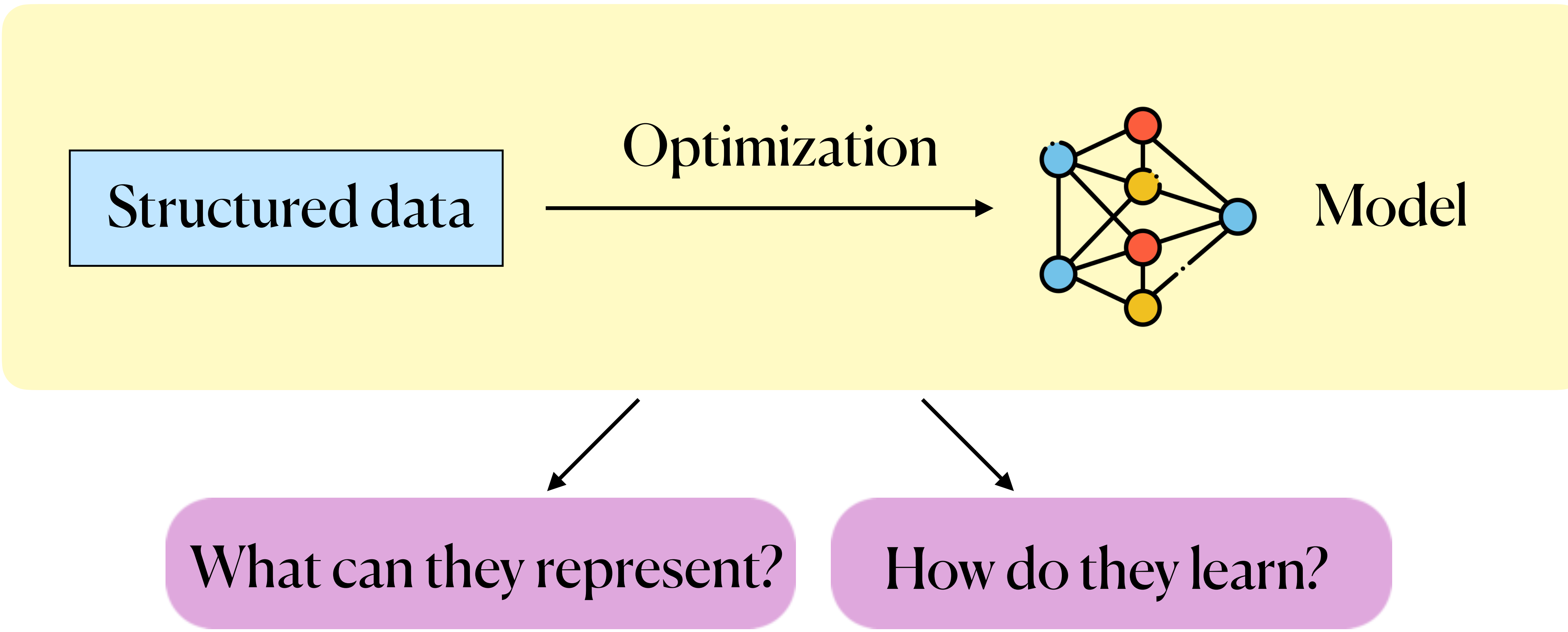
This tutorial



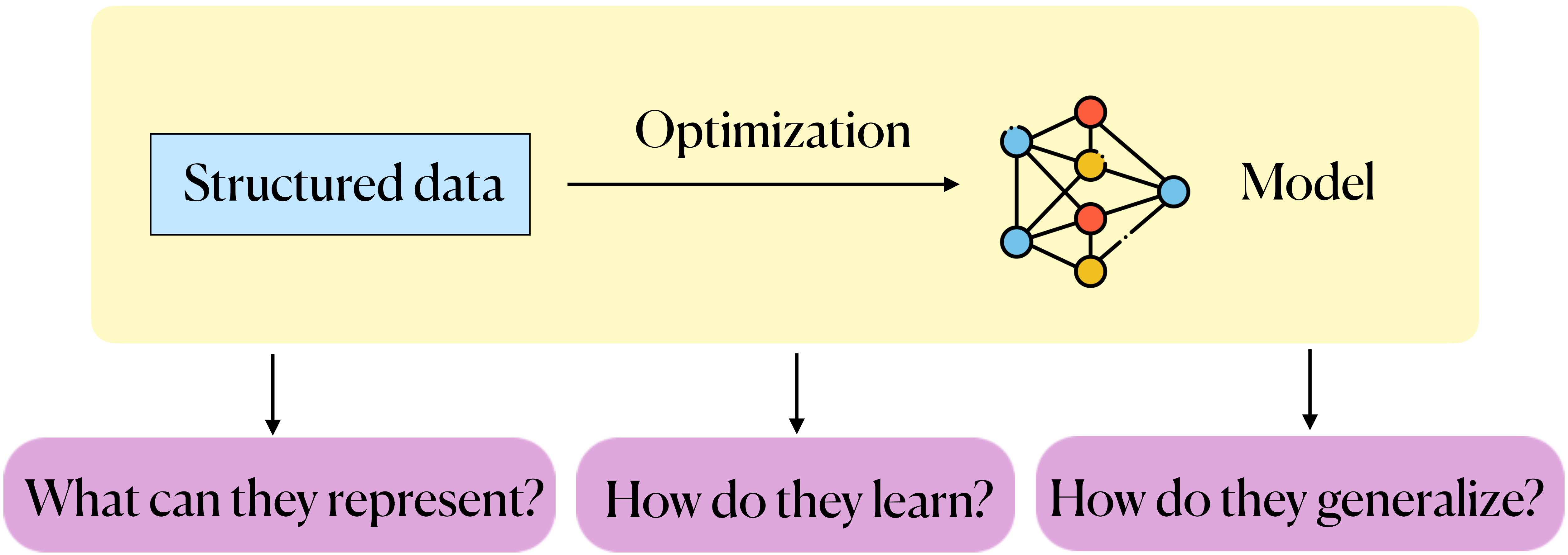
Goal of the tutorial



Goal of the tutorial



Goal of the tutorial



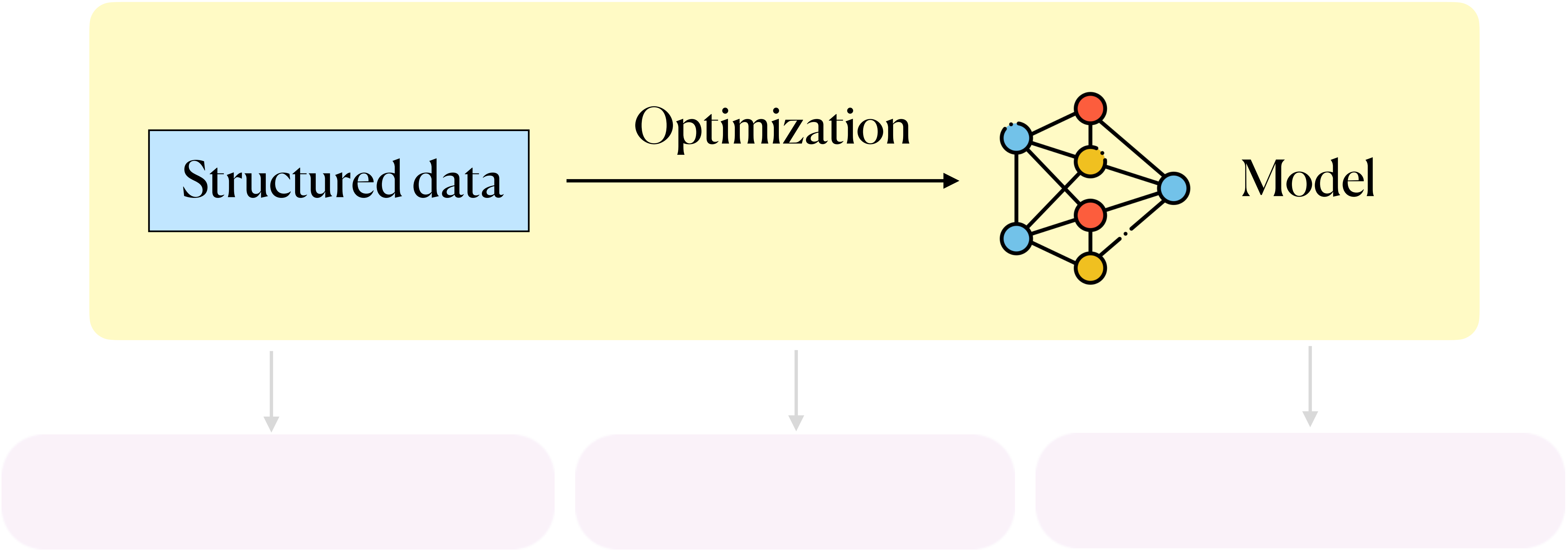
Outline of the tutorial

What can they represent? — **Part I**

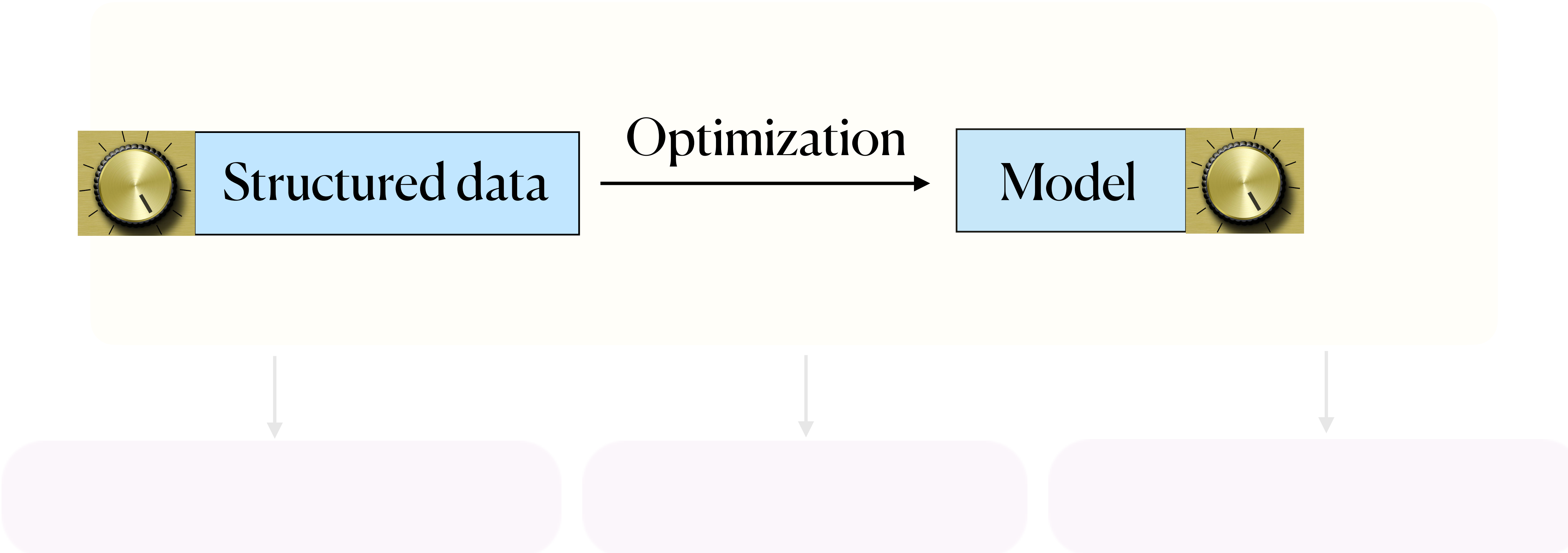
How do they learn? — **Part II**

How do they generalize? — **Part III**

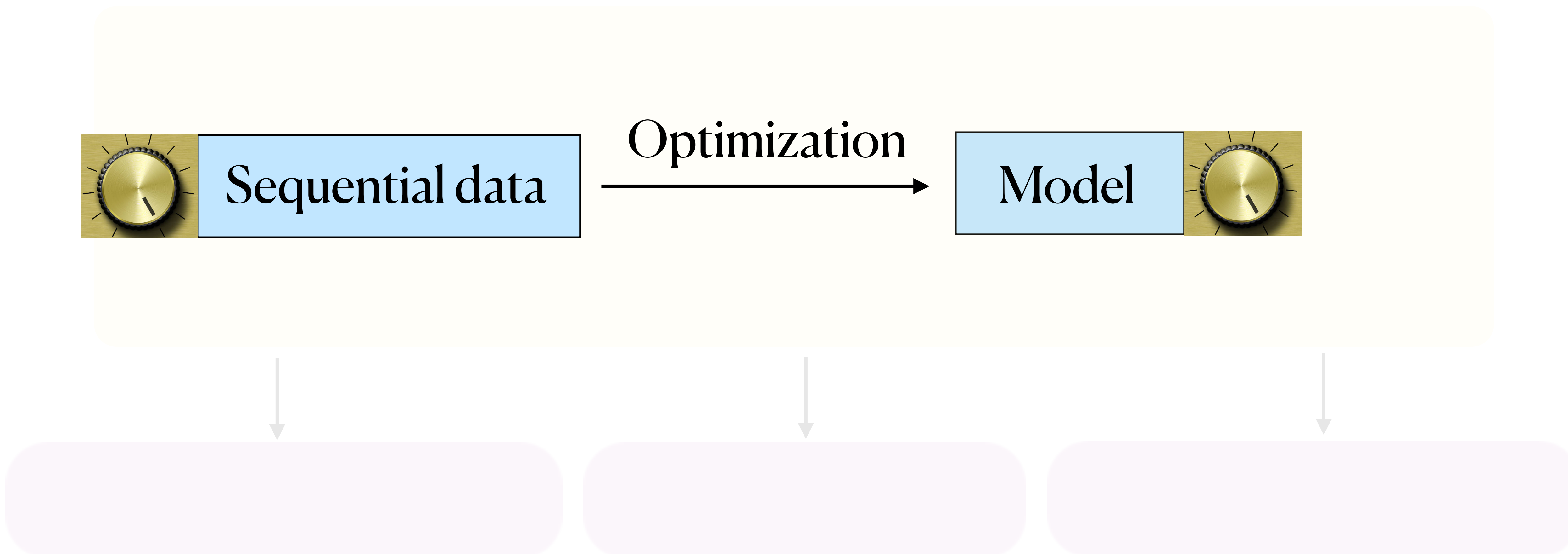
Outline of the tutorial



Outline of the tutorial



Outline of the tutorial



Outline of the tutorial

Markov/n-gram

Topic models

Formal languages

Outline of the tutorial

Markov/n-gram

Topic models

Formal languages

Transformers

RNNs/SSMs (brief)

Markov/n-gram

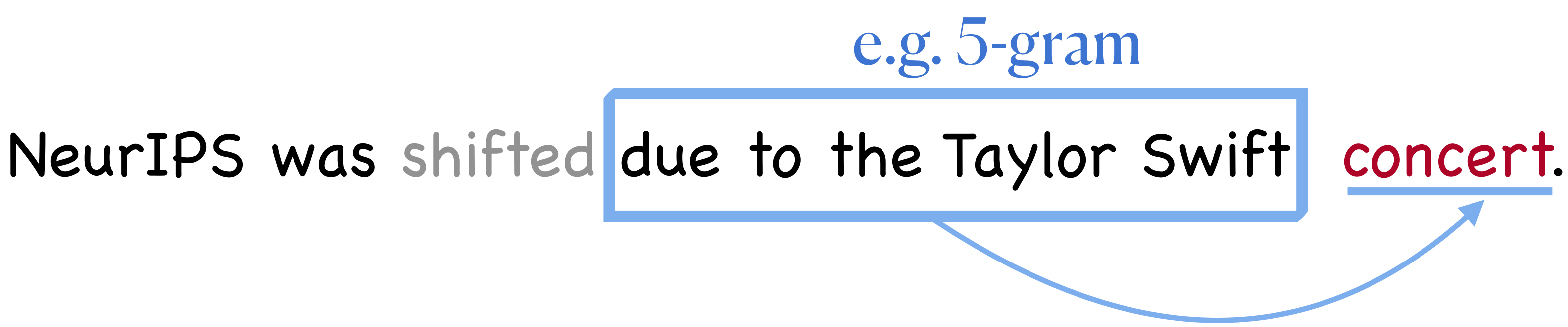
- $p(x_t \mid x_1, \cdots x_{t-1}) = p(x_t \mid x_{t-n}, x_{t-n+1}, \cdots, x_{t-1})$ *... dates back to [Shanon 1950].*

e.g. 5-gram

NeurIPS was shifted due to the Taylor Swift concert.

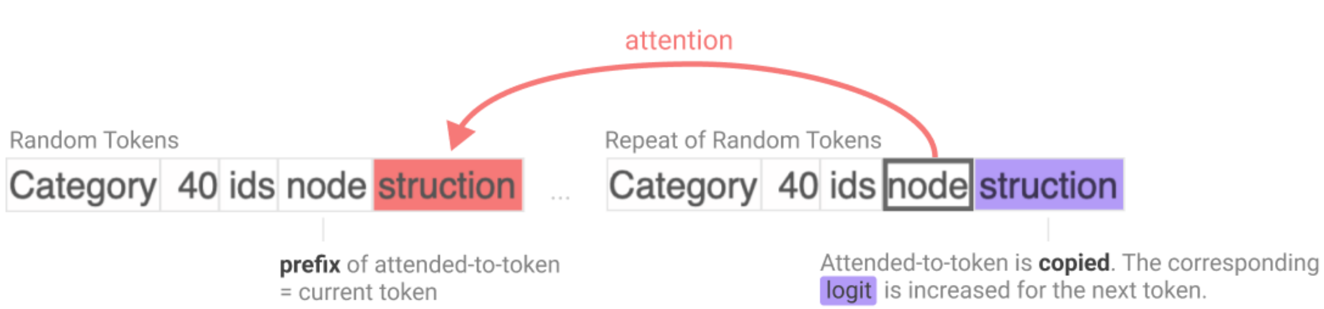
Markov/n-gram

- $p(x_t \,|\, x_1, \cdots x_{t-1}) = p(x_t \,|\, x_{t-n}, x_{t-n+1}, \cdots, x_{t-1})$ *... dates back to [Shanon 1950].*

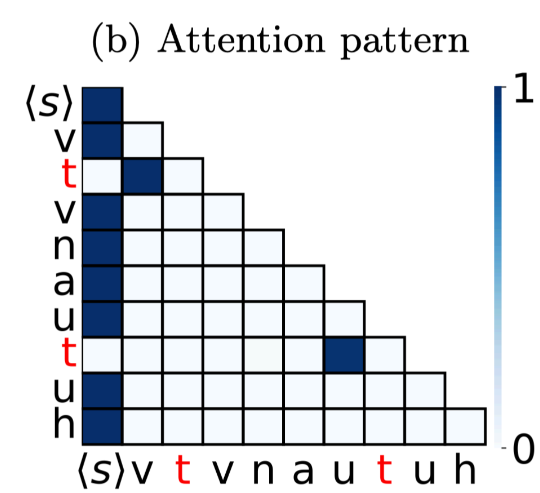


- Used for **mechanistic understanding** of language models.

induction heads
[Olsson et al. 22, Bietti et al. 23]



attention sink [Guo et al. 24]



Outline of the tutorial

Markov/n-gram ✓

Outline of the tutorial

Topic models

Topic models

- Each word sampled following a topic.

e.g. Latent Dirichlet Allocation (LDA; Blei et al. 2003).

To generate a document:

1. Sample a topic distribution $\theta \sim \text{Dir}(\alpha)$.
2. For each word,
    1. Sample a topic $z_i \sim \text{Multinomial}(\theta)$.
    2. Sample a word from the topic $w_i \sim \text{Multinomial}(z_i)$.

[Sontag & Roy, 2011; Awasthi & Risteski, 2015; Arora et al. 2016; Tosh et al. 2021; Luo et al., 2022, Li et al. 2023; Reuter et al. 2024]

Outline of the tutorial

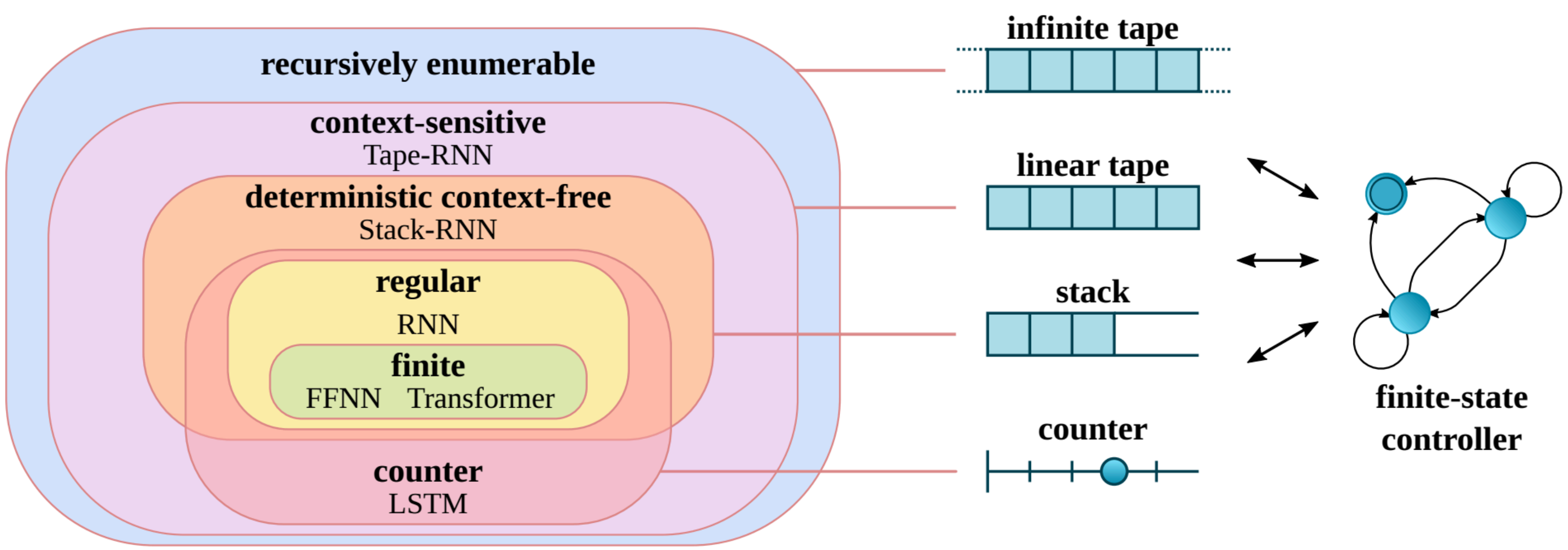
Topic models ✓

Outline of the tutorial

Formal languages

Formal languages

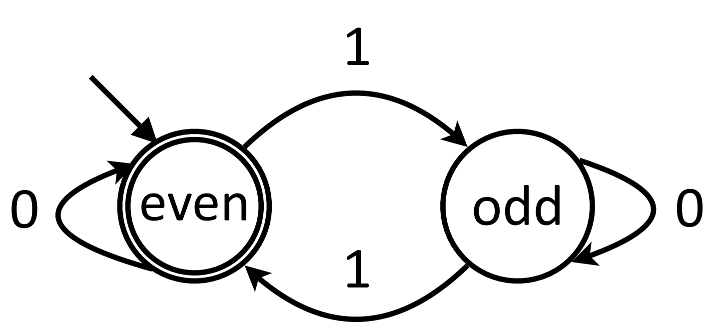
(Fig from Deletang et al. 2022)



Formal languages

Regular languages: e.g. parity.

(0* 1 0* 1 0*)*



An **on-off** switch is off.

(actions: **toggle** or **not**)

Now the switch is ?.

Formal languages


```
<div>
    <div>
        <div>
            <ul>
                <li></li>
                <li></li>
                <li></li>
                <li></li>
            </ul>
        </div>
    </div>
</div>
```

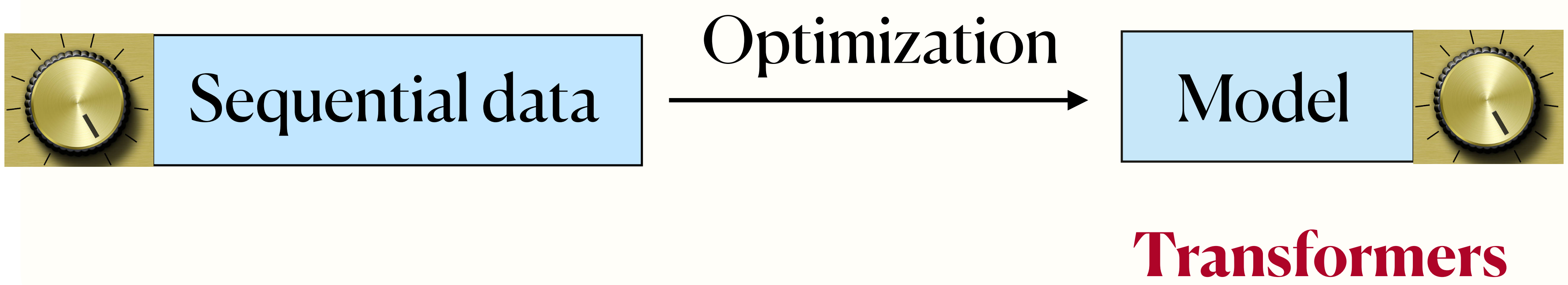
Context-free languages: e.g. Dyck.

$$S \mapsto \epsilon \mid [S] \mid (S)$$

Outline of the tutorial

Formal languages ✓

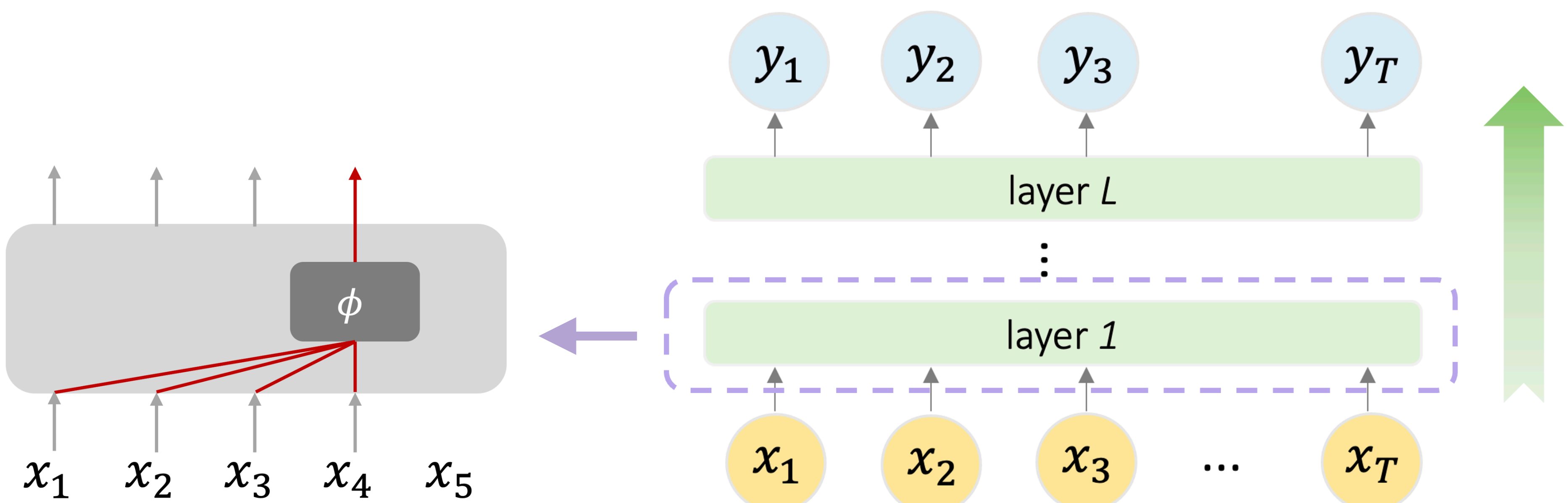
Outline of the tutorial



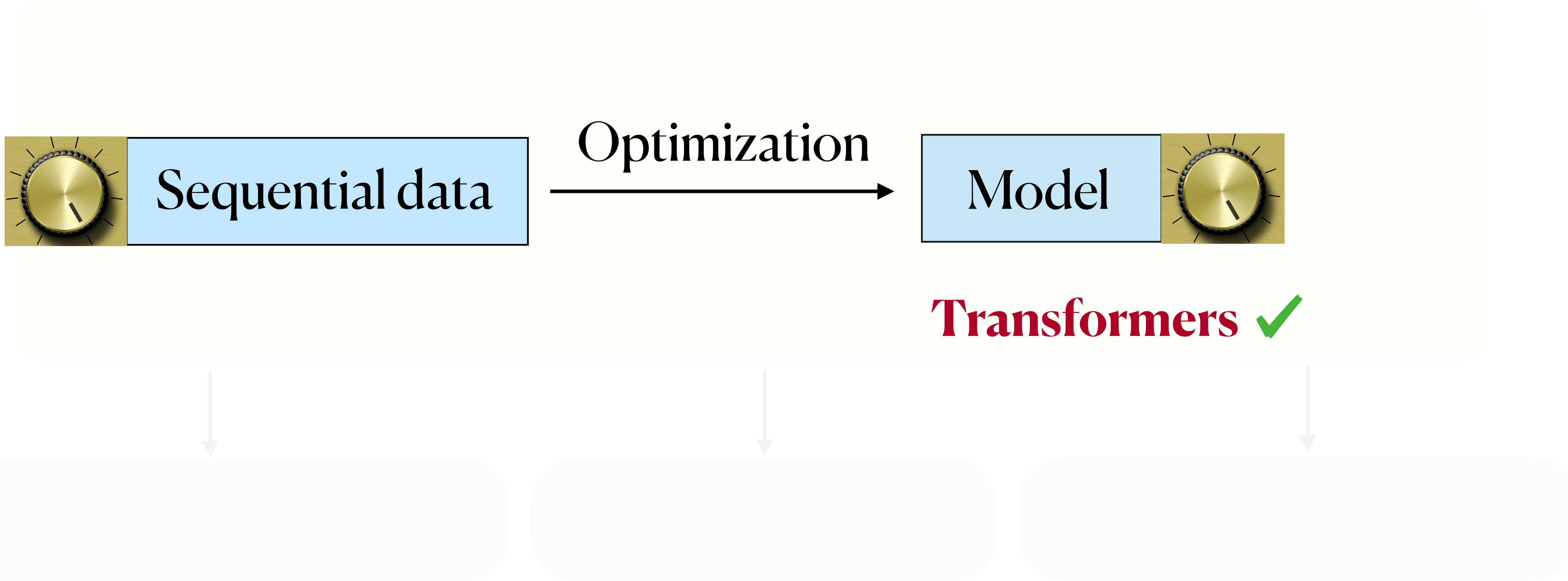
Transformers

In-parallel (across i) compute: $x_i^{(l)} = \phi(\sum_{j \leq i} \alpha_{i,j}^{(l-1)} x_j^{(l-1)})$.

**Decoder only; causal attention; omitting residual link / layer norm.*



Outline of the tutorial

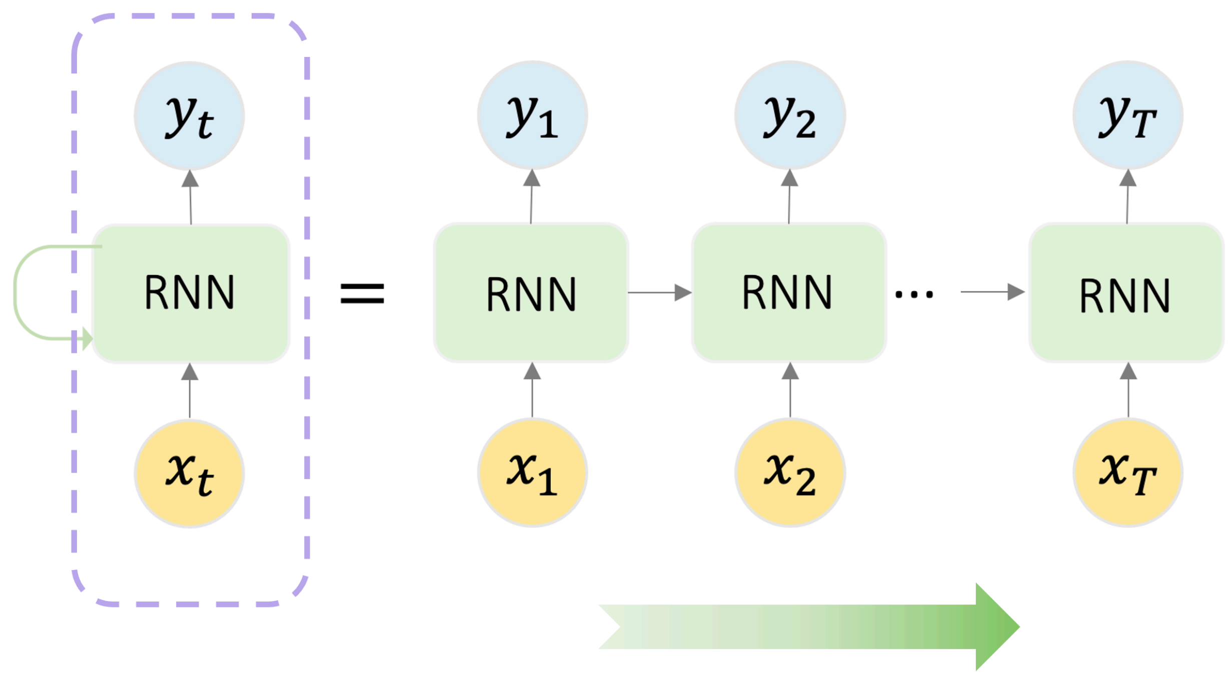


Outline of the tutorial

RNNs/SSMs

Recurrent Neural Nets (RNNs)

Sequentially compute: $h_t = f(x_t, h_{t-1}),\ y_t = \phi(h_t)$.



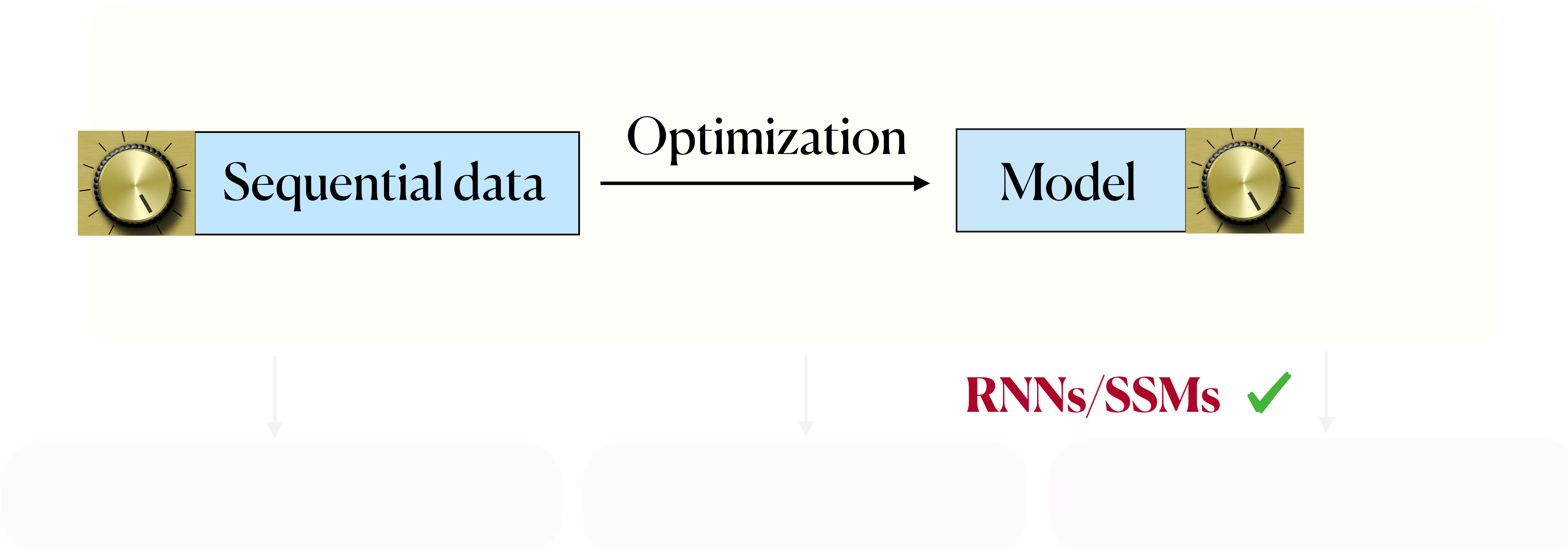
Nonlinear: $h_t = \sigma(W_1 x_t + W_2 h_{t-1})$

- the default; e.g. Elman RNN.

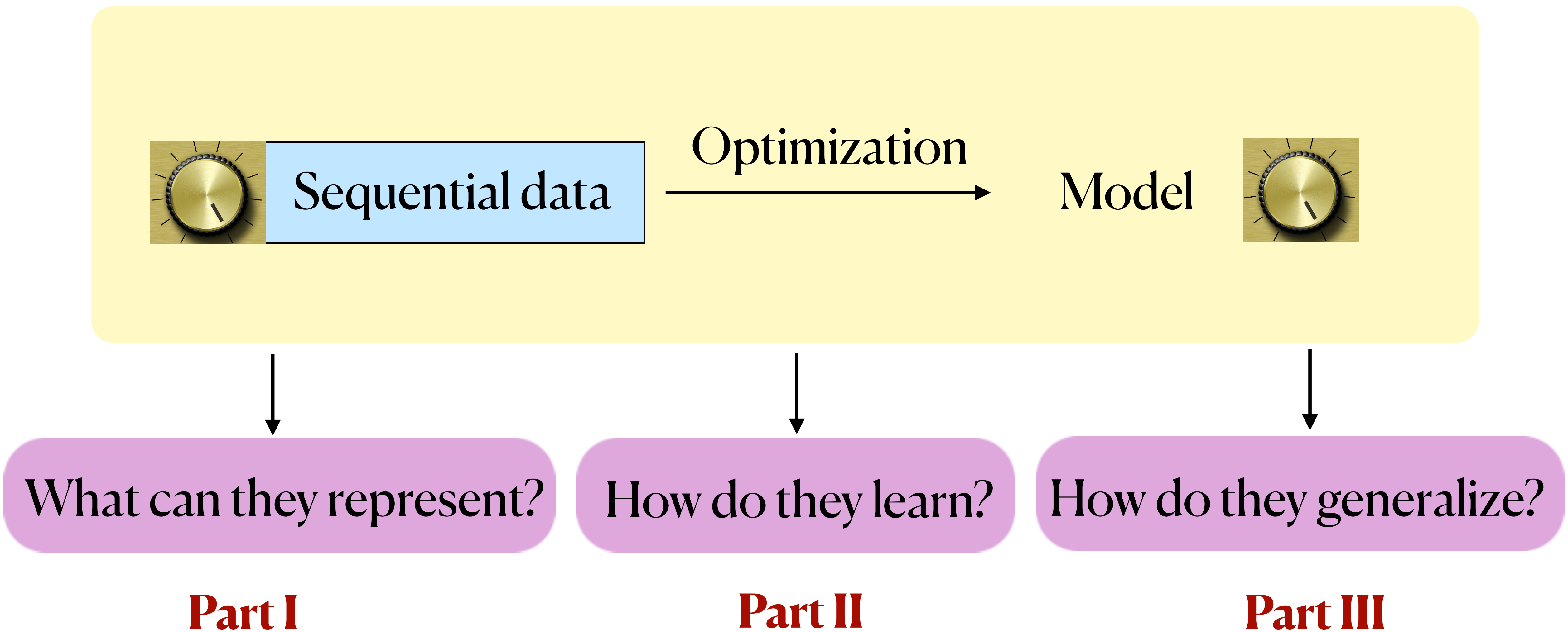
Linear: $h_t = W_1 x_t + W_2 h_{t-1}$

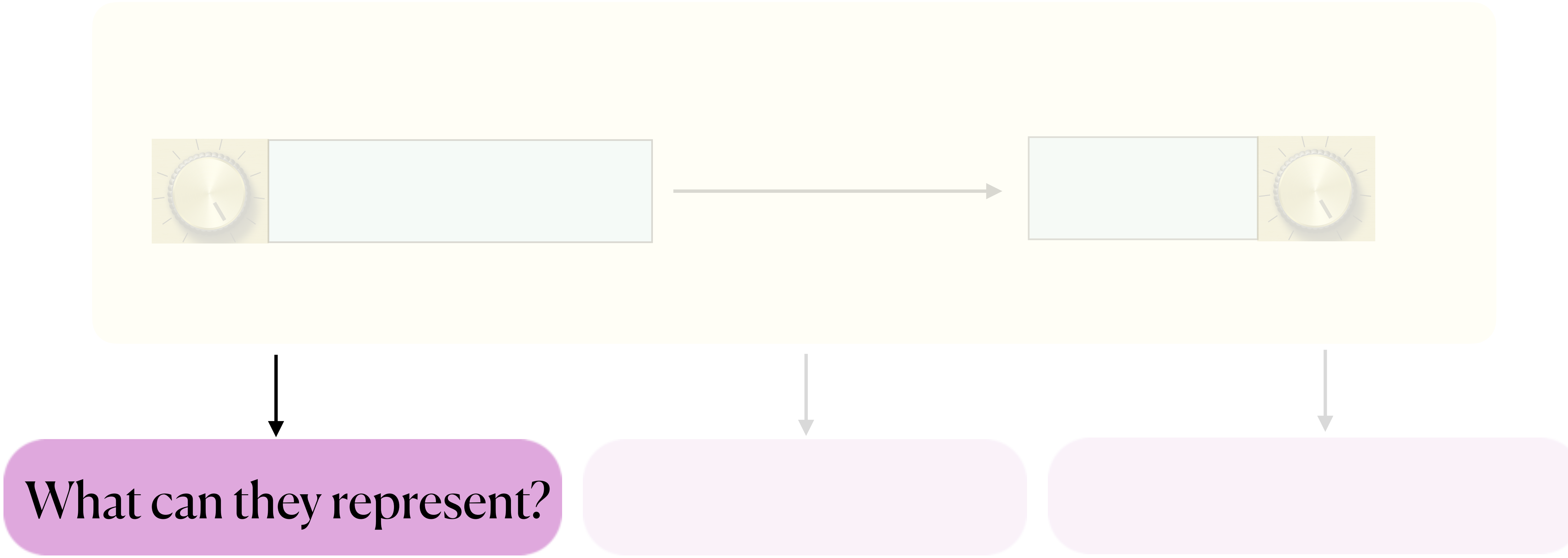
- e.g. state-space models (S4, Mamba, etc)

Outline of the tutorial



Outline of the tutorial





Part I

Part I: Representability

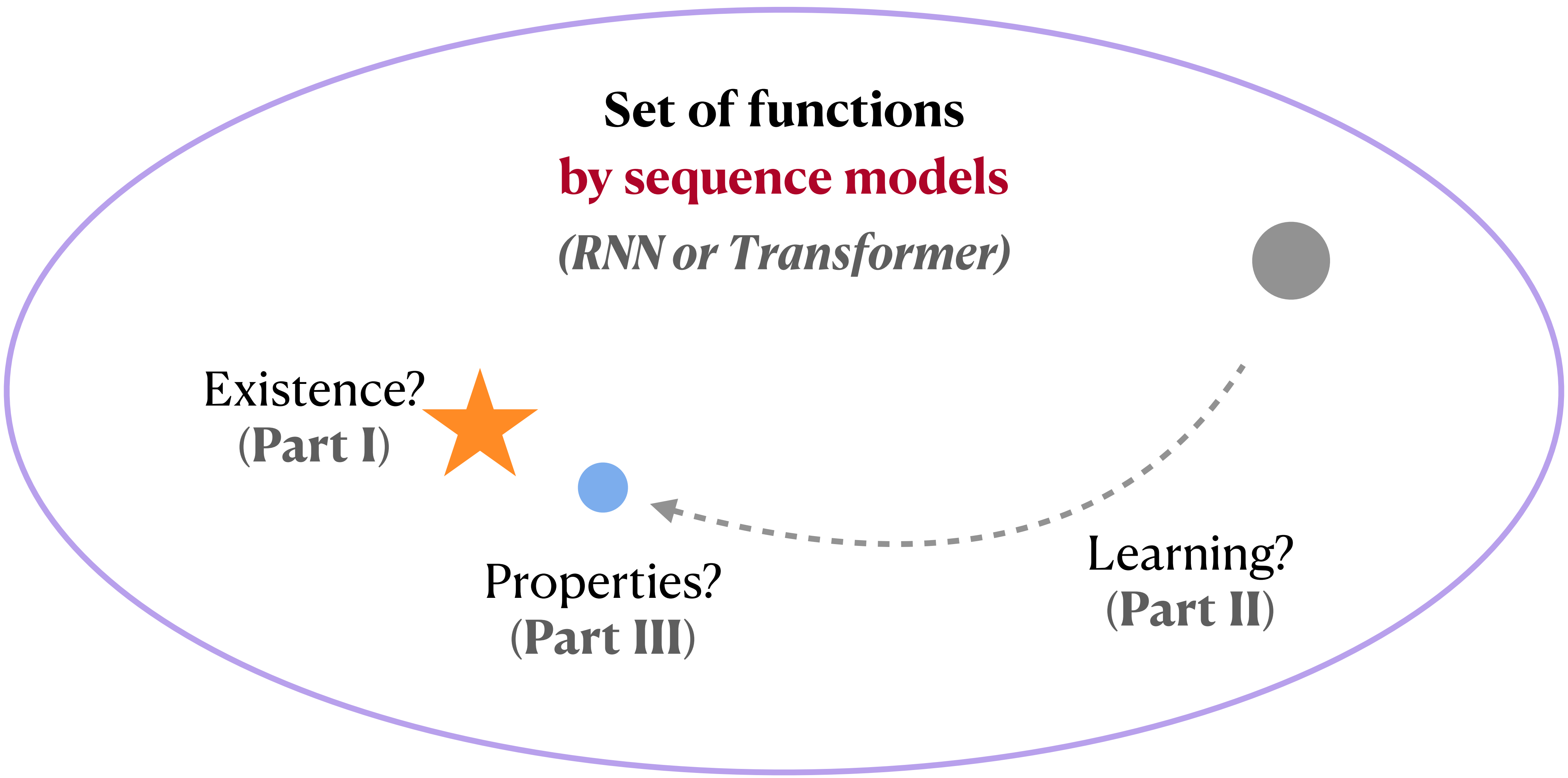
(aka. expressivity)

Part I: Representability

(aka. expressivity)

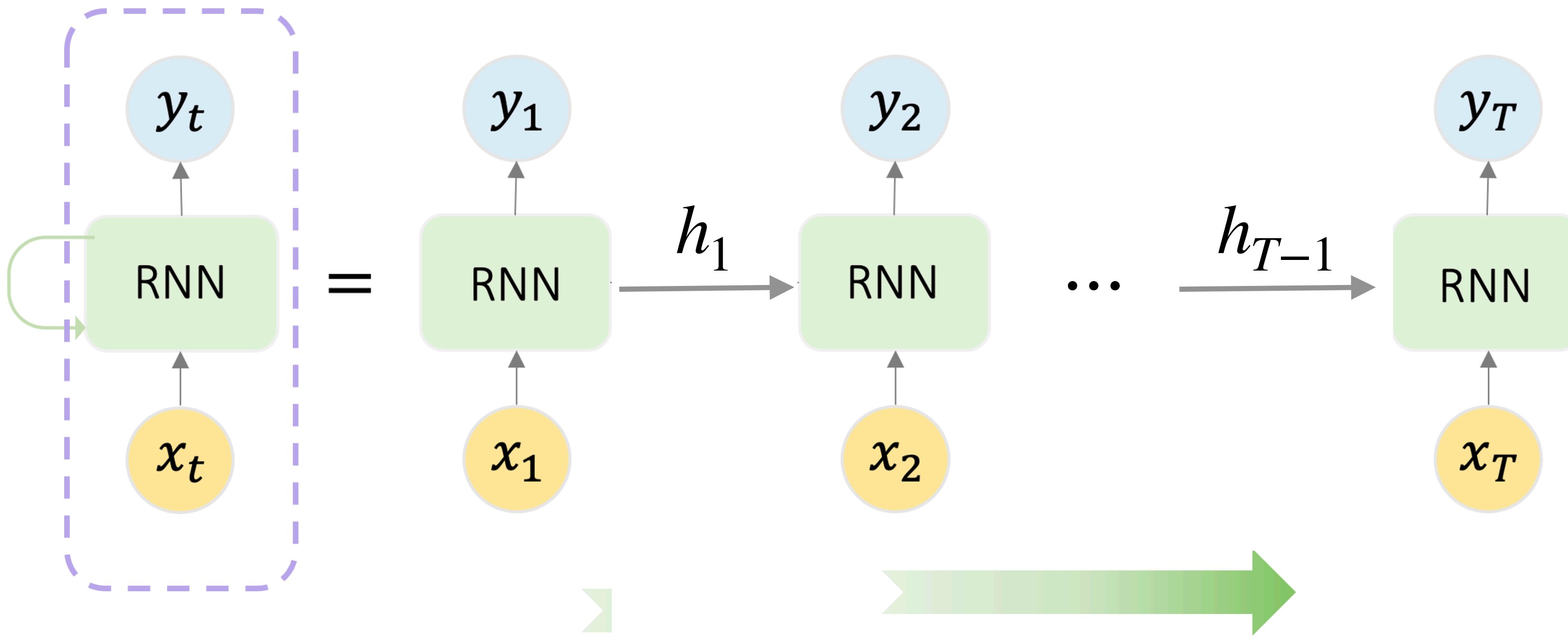
Part I - Representability

Main question: the existence of an (efficient) solution to a task.



Sequence models — Recurrent Neural Net (RNN)

Sequentially compute: $h_t = f(x_t, h_{t-1}),\ y_t = \phi(h_t)$.



Nonlinear: $h_t = \sigma(W_1 x_t + W_2 h_{t-1})$

- the default; e.g. Elman RNN.

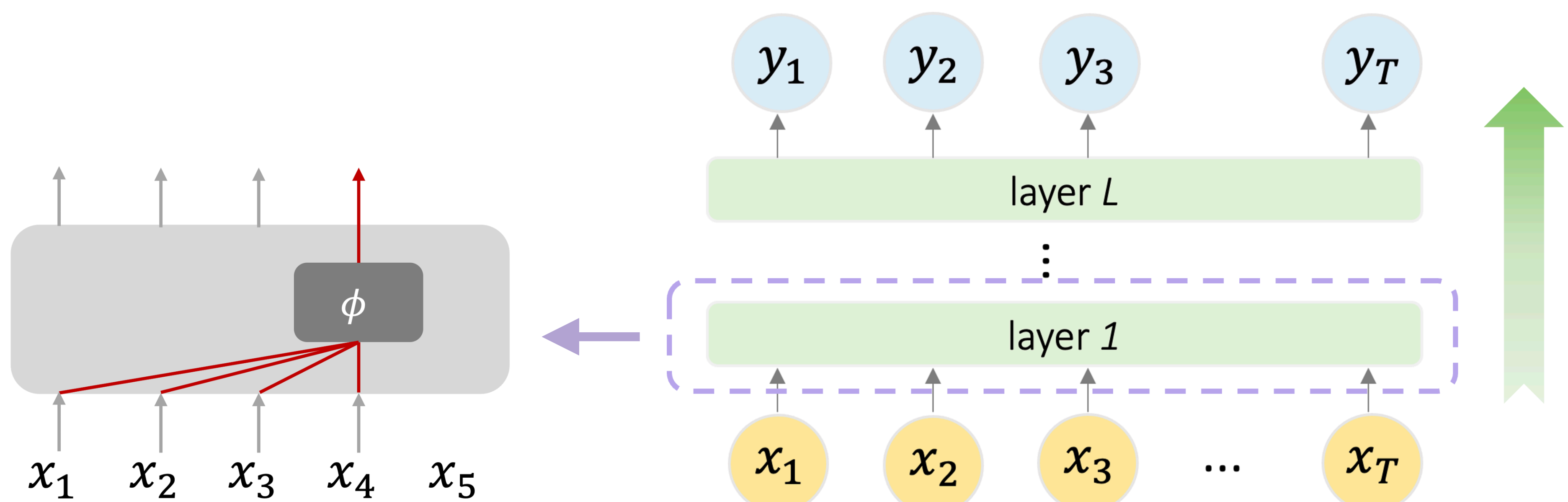
Linear: $h_t = W_1 x_t + W_2 h_{t-1}$

- e.g. state-space models (S4, Mamba, etc)

Sequence models — Transformer

In-parallel (across i) compute: $x_i^{(l)} = \phi(\sum_{j \leq i} \alpha_{i,j}^{(l-1)} x_j^{(l-1)})$.

**Decoder only; causal attention; omitting residual link / layer norm.*



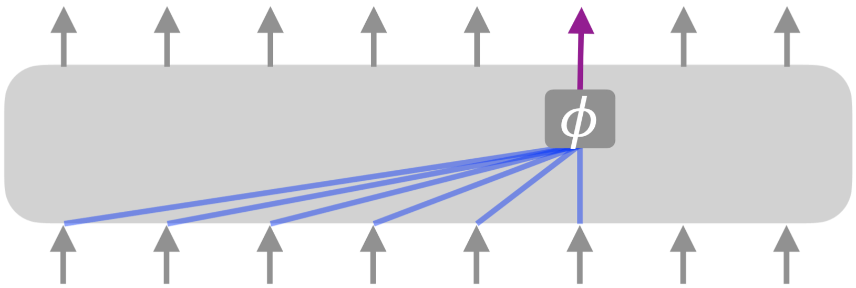
Sequence models — Transformer

In-parallel (across i) compute: $x_i^{(l)} = \phi(\sum_{j \le i} \alpha_{i,j}^{(l-1)} x_j^{(l-1)})$.

**Decoder only; causal attention; omitting residual link / layer norm.*

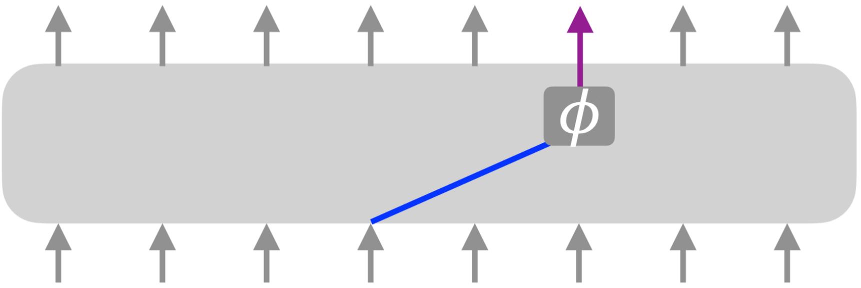
$$\alpha_{i,j} \propto \exp(\langle W_Q x_j, W_K x_i \rangle), \quad \sum_j \alpha_{i,j} = 1.$$

1. Uniform attention: $\vec{\alpha}_i = [\frac{1}{T}, \frac{1}{T}, \cdots, \frac{1}{T}]$.



e.g. average, sum

2. Sparse attention: $\vec{\alpha}_i = [0, \cdots, 0, 1, 0, \cdots]$.



e.g. selection

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

*a **prerequisite** to learnability and generalization.*

What type of results?

What type of representational results?

e.g. Transformer:

1. **Universal approximation** for seq2seq functions [Yun et al. 19].

[Yun et al. 19; informal] For any $f : \mathbb{R}^{d \times T} \to \mathbb{R}^{d \times T}$,
$\exists$ a Transformer $\mathcal{T}$ s.t. dist($f, \mathcal{T}$) is small.

- The sizes of each layer are independent of input dim d and length T.
- The # layers is exponential in d, T.

Under practical constraints?

2. **Turing completeness** [Perez et al. 2020].

- Idea: simulate each step of a Turing machine's execution.
- Assuming infinite precision, hence not practical [Dehghani et al. 18].

Yun et al. 19: Are Transformers universal approximators...?
Perez et al. 20: Attention is Turing complete

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

a* **prerequisite *to learnability and generalization.*

What type of results?

Insufficient: universal approximation, Turing completeness

Fine-grained characterization under practical constraints.

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

a* ***prerequisite* *to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

- **Upper bounds**: "*Construct* a solution for ..."
- **Lower bounds**: "Any solution needs to be *as large as* ..."

(b) **Implications** of representability:

1. Understanding design choices: depth-width tradeoff.
2. Comparing Transformers vs RNNs (SSMs).
3. Improving with Chain-of-Thought and hybrid models.

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*
    - Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
    - A class of tasks: finite-state automata [Liu et al. 23].
    - *Another perspective*: "Think like Transformers" — RASP (and variants) in Part III.
2. **Lower bounds**: *"Any solution needs to be as large as…"*
    - Depth lower bound — circuit complexity.
    - Width (& precision) lower bound — communication complexity.

Part I(a) — Tools

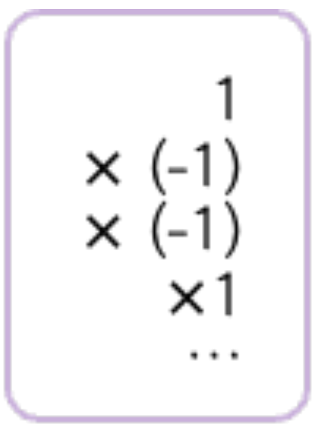
Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*
    - Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
    - A class of tasks: **finite-state automata** [Liu et al. 23].
    - *Another perspective*: "Think like Transformers" — RASP (and variants).
2. **Lower bounds**: *"Any solution needs to be as large as…"*
    - Depth lower bound — circuit complexity.
    - Width (& precision) lower bound — communication complexity.

Part I(a), upper bound — automata

[Liu et al. 2023a]

Sandbox for a class of sequential reasoning tasks

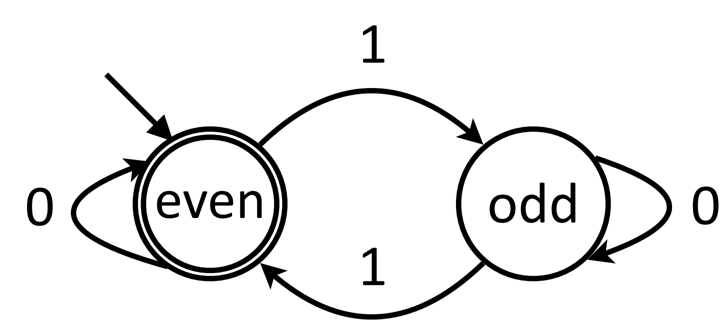


arithmetic

$$1 = (-1)^0$$

$$-1 = (-1)^1$$

Track the exponent with parity



Finite-state automata

Part I(a), upper bound — automata

[Liu et al. 2023a]

Finite-state automata as a sandbox for sequential reasoning.

$$\mathscr{A} := (Q,\ \Sigma,\ \delta)$$

states inputs transitions

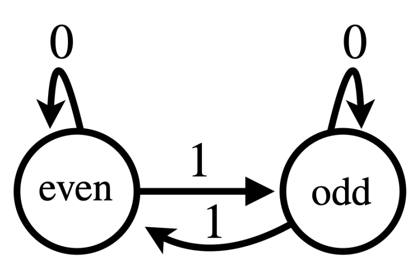
$$q_t = \delta(\sigma_t, q_{t-1})$$

(Q is finite)

parity counter

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

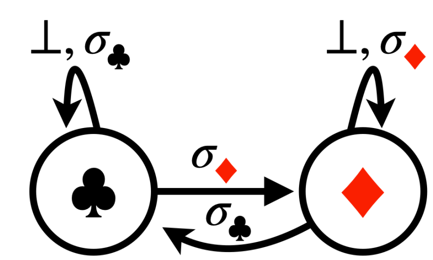
$\Sigma = \{0, 1\}$



1-bit memory unit

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

$\Sigma = \{\sigma_\clubsuit, \sigma_\diamondsuit, \perp\}$ (no-op)



Capturing a broad set of scenarios

e.g. regular languages; transitions in Markov models (Part II).

Liu et al. 23a: Transformers learn shortcuts to automata

Part I(a), upper bound — automata

[Liu et al. 2023a]

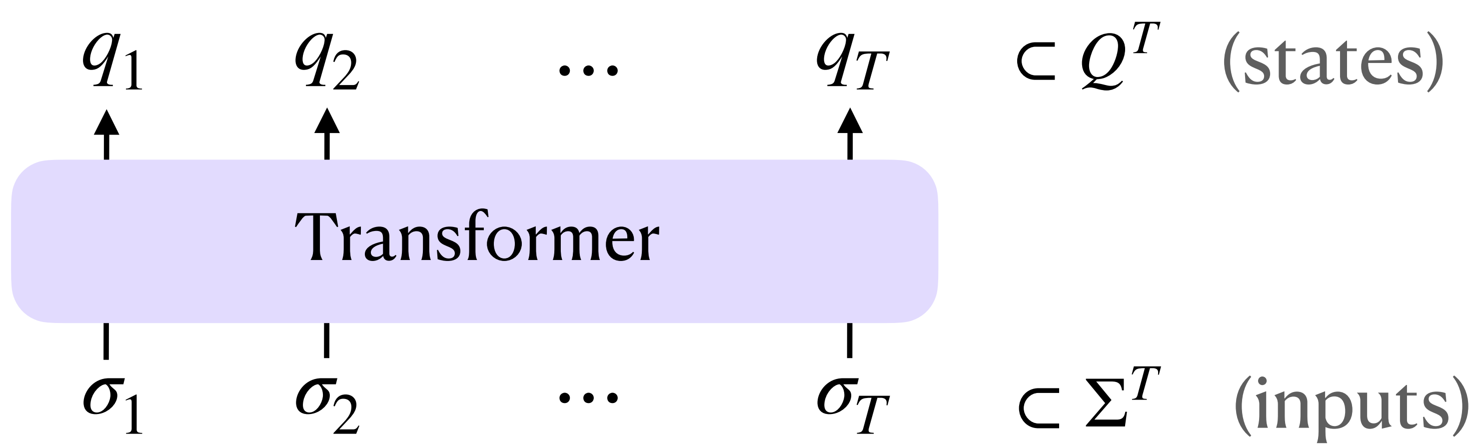
Finite-state automata as a sandbox for sequential reasoning.

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

states, inputs, transitions (Q is finite)

Task: simulating $\mathscr{A}$: learn a seq2seq function for sequence length T.

**easy for RNN*: compute δ.



Part I(a), upper bound — automata

[Liu et al. 2023a]

$\mathscr{A} := (Q, \Sigma, \delta)$

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

Simulating T steps of sequential transitions in $\mathscr{A}$.

1. $O(\log T)$ layers for *any* $\mathscr{A}$.

↓

(asymptotic notations)

- $O(f(T))$: "***No more than*** $f(T)$ (up to a ***constant***)."

Part I(a), upper bound — automata

[Liu et al. 2023a]

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

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

Simulating T steps of sequential transitions in $\mathcal{A}$.

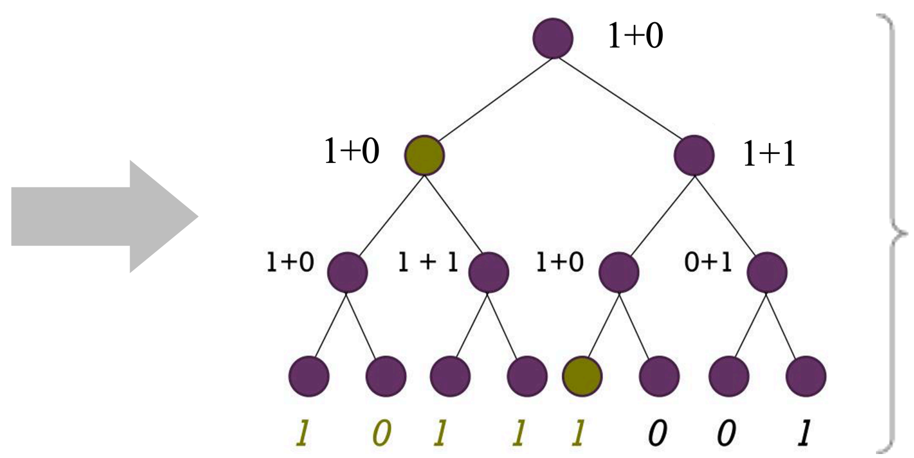
1. $O(\log T)$ layers for *any* $\mathcal{A}$: divide-and-conquer.

- Observation: simulation → function composition.

e.g. $T = 2$: $q_2 = \delta(\sigma_2, \delta(\sigma_1, q_0)) = (\ \delta(\sigma_2, \cdot\ ) \circ \delta(\sigma_1, \cdot\ )\ )(q_0)$.

associativity

$f_1 \circ f_2 \circ f_3 \circ f_4 = (f_1 \circ f_2) \circ (f_3 \circ f_4)$



Shorter?

$\log T$

- aka. associative scan [Blelloch 93] ... e.g. used in Mamba [Gu & Dao 23].

Part I(a), upper bound — automata

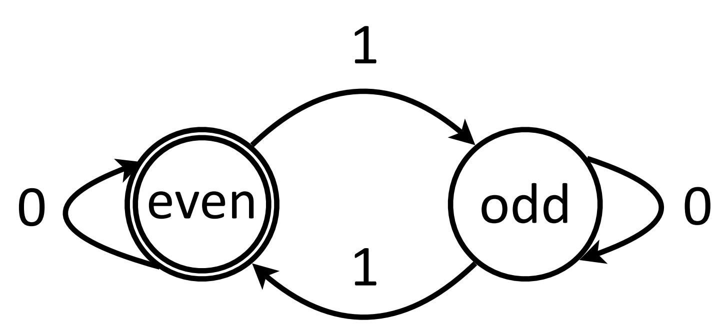
[Liu et al. 2023a]

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

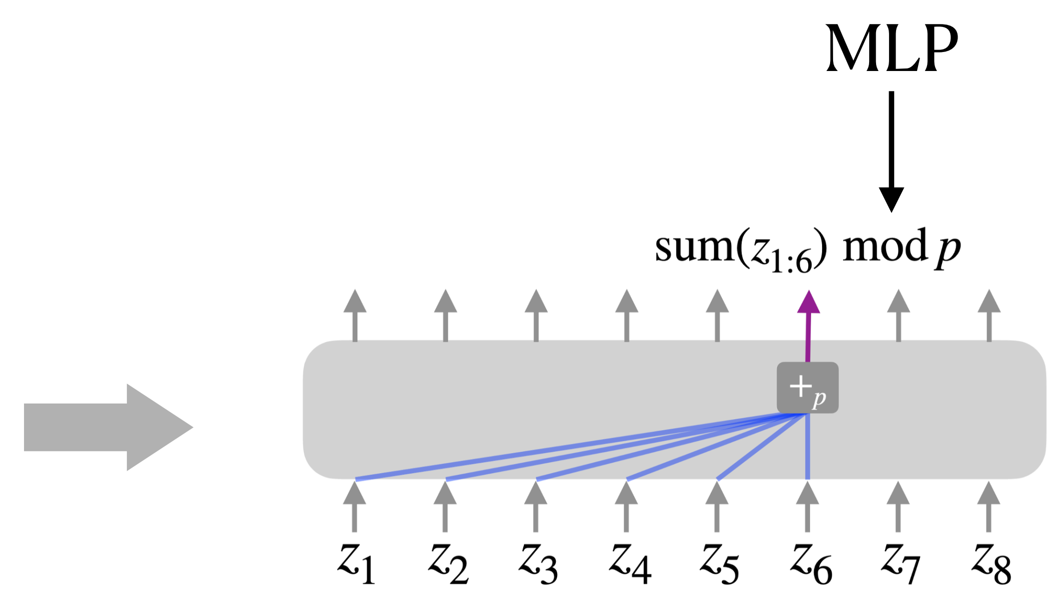
Simulating T steps of sequential transitions in $\mathcal{A}$.

2. $O(|Q|^2 \log |Q|)$ layers

e.g. parity:



$$q_T = \left(\sum_{t\in[T]} x_i\right) \mod 2$$



uniform attention

Part I(a), upper bound — automata

[Liu et al. 2023a]

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

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

Simulating T steps of sequential transitions in $\mathcal{A}$.

2. $O(|Q|^2 \log |Q|)$ layers for a *solvable* $\mathcal{A}$: decomposition.

- [**Krohn-Rhodes**] Any $\mathcal{A}$ can be decomposed.

Intuition: (rough) analogy to integer factorization:

42 = 2 x 3 x 7

depending on $|Q|, |\Sigma|$

(but not T)

factors

(2 types)

Part I(a), upper bound — automata

[Liu et al. 2023a]

$\mathscr{A} := (Q, \Sigma, \delta)$

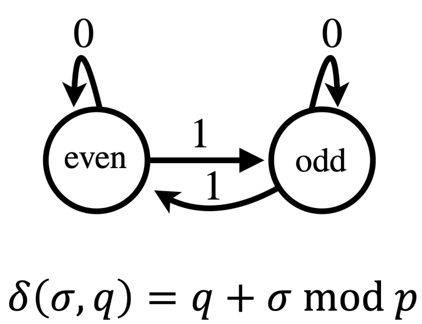
$q_t = \delta(\sigma_t, q_{t-1})$

Simulating T steps of sequential transitions in $\mathscr{A}$.

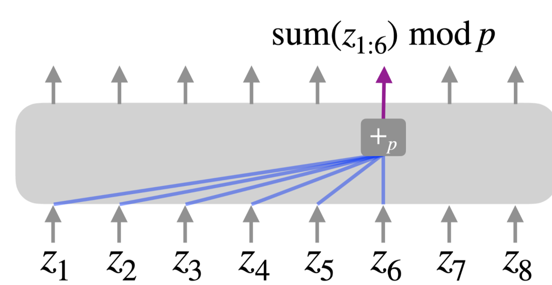
2. $O(|Q|^2 \log |Q|)$ layers for a *solvable* $\mathscr{A}$: decomposition.

- [**Krohn-Rhodes**] Any $\mathscr{A}$ can be decomposed into 2 types of "factors".

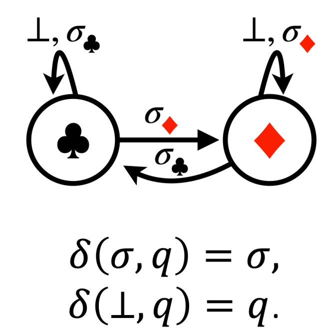
  which can each be represented by one Transformer layer.



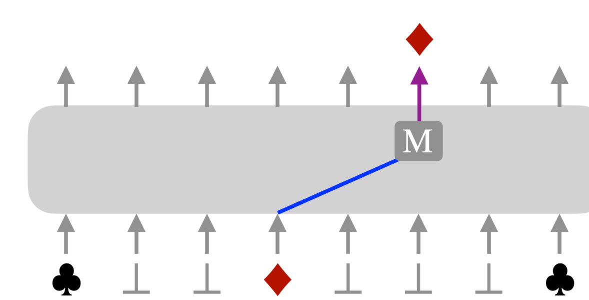
mod(-p) counter



uniform attention



memory unit



sparse attention

Part I(a), upper bound — automata

[Liu et al. 2023a]

$\mathscr{A} := (Q, \Sigma, \delta)$

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

Simulating T steps of sequential transitions in $\mathscr{A}$.

$O(\log T)$ layers for all $\mathscr{A}$; $O(|Q|^2 \log |Q|)$ layers for solvable $\mathscr{A}$.

Computational **shortcuts**: $o(T)$ layers for T steps.

(sublinear in T)

Part I(a), upper bound — automata

[Liu et al. 2023a]

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

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

Simulating T steps of sequential transitions in $\mathcal{A}$.

$O(\log T)$ layers for all $\mathcal{A}$; $O(|Q|^2 \log |Q|)$ layers for solvable $\mathcal{A}$.

Solutions with fewer layers?

Part I(a), upper bound — automata

[Liu et al. 2023a]

$\mathscr{A} := (Q, \Sigma, \delta)$

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

Simulating T steps of sequential transitions in $\mathscr{A}$.

$O(\log T)$ layers for all $\mathscr{A}$; $O(|Q|^2 \log |Q|)$ layers for solvable $\mathscr{A}$.

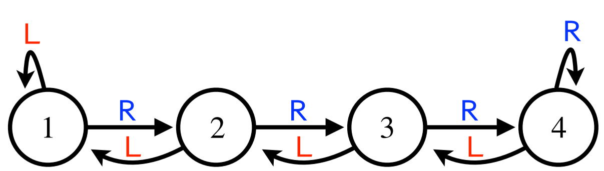
Solutions with fewer layers?

Yes, in special cases: $O(1)$ layers.

- Commutative (e.g. parity): counting suffices → uniform attention.

  $\delta(\delta(q, \sigma_1), \sigma_2) = \delta(\delta(q, \sigma_2), \sigma_1)$

- Non-commutative: a special case (i.e. gridworld).



Question: a hierarchy induced by Transformers? ... C-RASP (Part III)

Part I(a), upper bound — automata

[Liu et al. 2023a]

$\mathscr{A} := (Q, \Sigma, \delta)$

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

Simulating T steps of sequential transitions in $\mathscr{A}$.

$O(\log T)$ layers for all $\mathscr{A}$; $O(|Q|^2 \log |Q|)$ layers for solvable $\mathscr{A}$.

Solutions with fewer layers?

No, if we allow arbitrary automaton $\mathscr{A}$.

- *Non-solvable*: e.g. S_5 (permutation of 5 elements)

[*revisit in the lower bound part*]

... $O(\log T)$ is a *lower bound.*

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: "*Construct a solution to do some task…*" ✓

- Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
- A class of tasks: **finite-state automata**: $O(\log T)$ and $O_{|Q|}(1)$ layers.
- *Another perspective*: "Think like Transformers" — RASP(-L) in Part III.

2. **Lower bounds**: "*Any solution needs to be as large as…*"

- Depth lower bound — circuit complexity.
- Width (& precision) lower bound — communication complexity.

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*

- Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
- A class of tasks: **finite-state automata**: $O(\log T)$ and $O_{|Q|}(1)$ layers.
- *Another perspective*: "Think like Transformers" — RASP(-L) in Part III.

2. **Lower bounds**: *"Any solution needs to be as large as…"*

- Depth lower bound — circuit complexity.
- Width (& precision) lower bound — communication complexity.

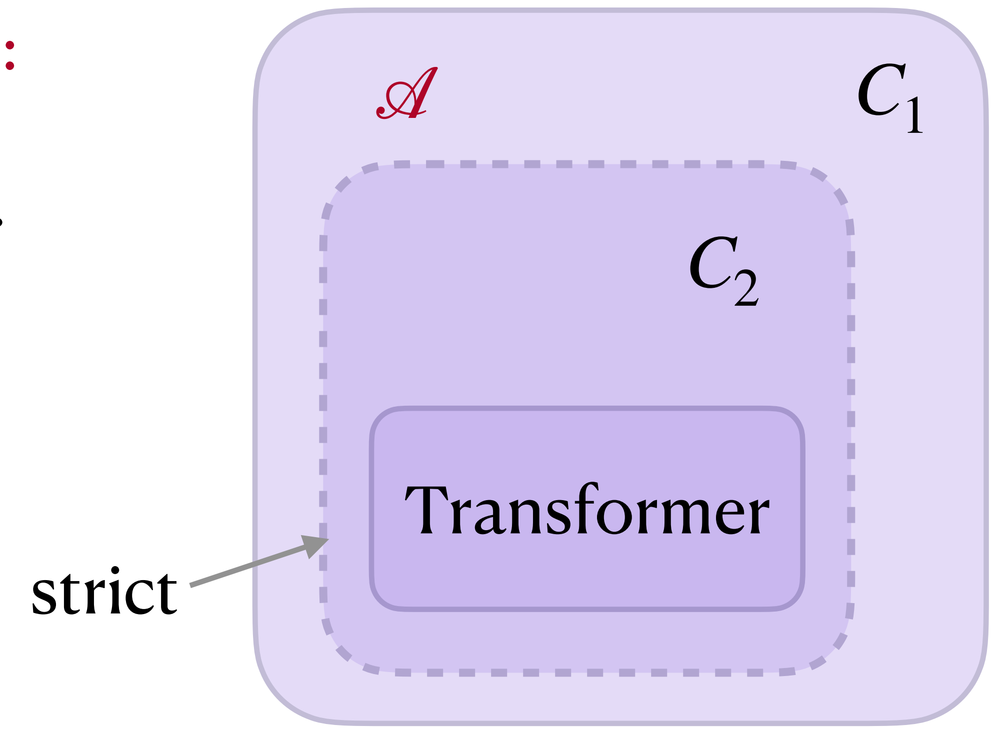
Part I(a), depth lower bound — circuit complexity

[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound:

- 2 classes $C_2 \subset C_1$, differing by **depth**.
- Some $\mathscr{A}$ is the hardest task in C_1.
- $O_{|Q|}(1)$-layer Transformer $\subset C_2$.
- Conjectured: $C_2 \subsetneq C_1$.



$\therefore$ $O_{|Q|}(1)$-layer Transformer (conjectured) cannot simulate $\mathscr{A}$.

Part I(a), depth lower bound — circuit complexity

[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound:

- Step 1: two classes $C_2 \subset C_1$, differing by **depth** (and gates).

Part I(a), depth lower bound — circuit complexity

[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound:

- Step 2: some $\mathscr{A}$ is the hardest task in NC^1.

Part I(a), depth lower bound — circuit complexity

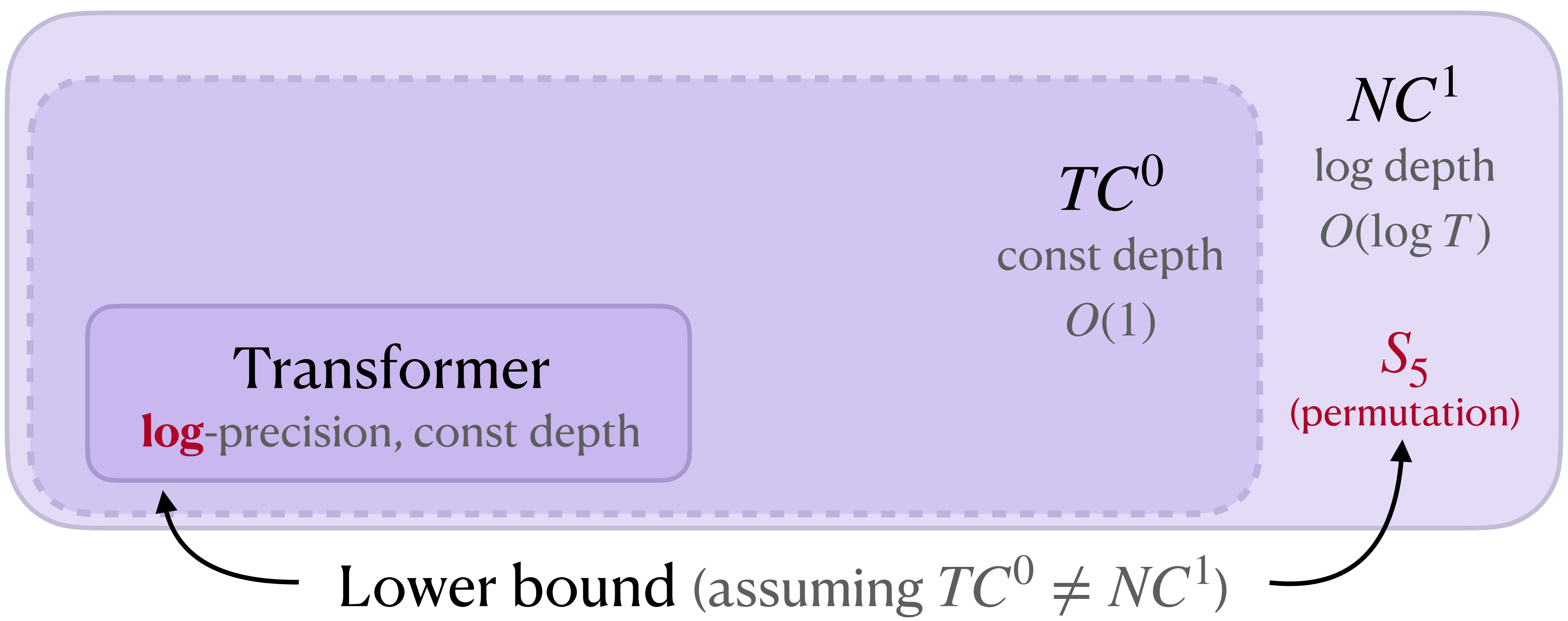
[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound:

- Step 3: $O(1)$-layer, log-precision Transformer $\subset TC^0$ [Merrill & Sabharwal 22].

  Intuition: the operations (e.g. add, multi) are in TC^0.



Merrill & Sabharwal 22: The Parallelism Tradeoff

Part I(a), depth lower bound — circuit complexity

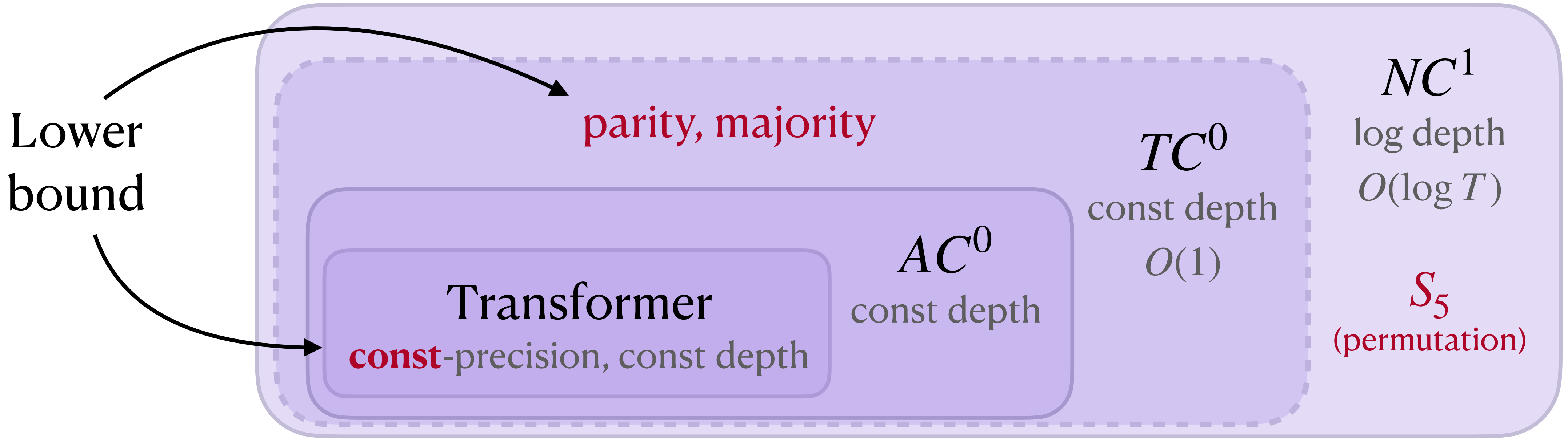
[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound:

- Step 3: $O(1)$-layer, log-precision Transformer $\subset TC^0$ [Merrill & Sabharwal 22].

  * ***constant***-precision: $\subset AC^0$ [Li et al. 24]



Li et al. 24: Chain of Thought Empowers Transformers

Part I(a), depth lower bound — circuit complexity

[Merrill & Sabharwal 22, Hao et al. 22, Li et al. 24, Strobl et al. 24]

Why can't we simulate every automaton with $O_{|Q|}(1)$ layers?

Idea: using a (conditional) lower bound.

TL;DR: $O(1)$-layer Transformer cannot simulate some automata.

- These (conditional) lower bounds are asymptotic (i.e. $T \to \infty$).
  i.e. "can't represent instances larger than some (unknown) threshold."
- It's possible that smaller instances can be represented.
  — *problems in practice?* Part III.

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*

- Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
- A class of tasks: **finite-state automata**: $O(\log T)$ and $O_{|Q|}(1)$ layers.
- *Another perspective*: "Think like Transformers" — RASP(and variants) in Part III.

2. **Lower bounds**: *"Any solution needs to be as large as…"*

- Depth lower bound — circuit complexity. ✓ (depth → separation)
- Width (& precision) lower bound — communication complexity.

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*
   - Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
   - A class of tasks: **finite-state automata**: $O(\log T)$ and $O_{|Q|}(1)$ layers.
   - *Another perspective*: "Think like Transformers" — RASP(and variants) in Part III.
2. **Lower bounds**: *"Any solution needs to be as large as…"*
   - Depth lower bound — circuit complexity.
   - Width (& precision) lower bound — communication complexity.

Part I(a), width lower bound — communication complexity

[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

Communication complexity: a common technique for lower bounds.

[Karchmer & Wigderson 88; Ben-David et al. 02; Martens et al. 13; Vardi et al. 21]

"How many bits need to be **communicated** *among the* **parties** *to solve a task?"*

Key ideas:

1. Turn a task into a known communication problem.
2. Make the width the communication bottleneck.

Part I(a), width lower bound — communication complexity

[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

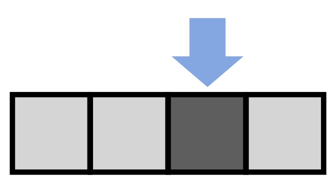
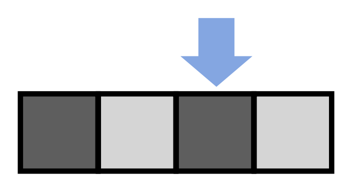
Communication complexity: a common technique for lower bounds.

[Karchmer & Wigderson 88; Ben-David et al. 02; Martens et al. 13; Vardi et al. 21]

"How many bits need to be **communicated** *among the* **parties** *to solve a task?"*

A known problem: e.g. set disjointness (2-party):

$$\mathrm{DISJ}^n(S, T) := \mathbf{1}[S \cap T = \varnothing], S, T \in \{0,1\}^n.$$

- $n = 4$: Alice: $S =$ , Bob: $T =$ , $\mathrm{DISJ}^n(S, T) = 0$.
- Known lower bound [Yao 1979]: $\mathrm{DISJ}^n(S, T)$ requires n bits.

Part I(a), width lower bound — communication complexity

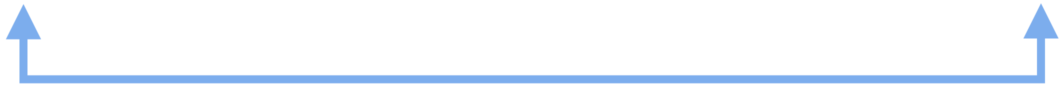
[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

Known lower bound (e.g. for DISJ^n) → Minimal size for a new task?

Idea: given a function f (e.g. RNN) that solves the new task, show:

1. A **reduction** ... e.g. solve DISJ^n by solving the new task.
   - *Recall*: Problem B **reduces to** Problem A ≈ *"B is no harder than A."*

2. An efficient **communication protocol** ... info to be sent for computing f.
   - *Make the* **size of interest** *the communication* **bottleneck**.

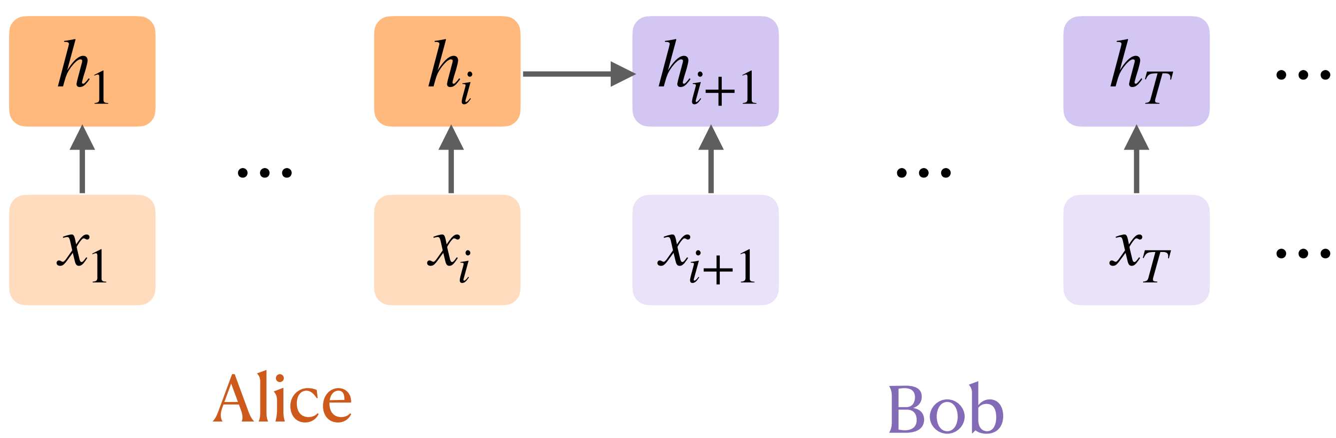


Part I(a), width lower bound — communication complexity

[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

Min size for a new task: reduction + communication protocol (bottleneck).

- **RNN**: the two parties = first vs second half of the positions.



$$h_{i+1} = f(h_i, x_{i+1})$$

→ Lower bound on the **size of** h_i, i.e. width · precision (e.g. T for DISJ^T).

Part I(a), width lower bound — communication complexity

[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

Min size for a new task: reduction + communication protocol (bottleneck).

- **Transformer**: the two parties = some positions vs the rest.

x_1 x_2 $\cdots$ x_i x_{i+1} $\cdots$ x_T $\cdots$

Alice Bob

$$\textit{Recall: } y_T = \phi\Big(\underbrace{\sum_{j \le T} \alpha_{T,j} \cdot V x_j}_{\textit{bottleneck}}\Big) = \phi\left(\sum_{j \le T} \frac{\exp(\langle Q(x_T), K(x_j)\rangle)}{\sum_i \exp(\langle Q(x_T), K(x_i)\rangle)} \cdot V x_j\right).$$

Part I(a), width lower bound — communication complexity

[Sanford et al. 23, 24, Peng et al. 24, Chen et al. 24, Arora et al. 24]

Min size for a new task: reduction + communication protocol (bottleneck).

- **Transformer**: the two parties = some positions vs the rest.

x_1 x_2 $\cdots$ x_i x_{i+1} $\cdots$ x_n $\cdots$

Alice Bob

$$\frac{\sum_{\text{Bob's } i} \exp(\langle Q(x_n), K(x_i)\rangle) \cdot V(x_i) + \sum_{\text{Alice's } j} \exp(\langle Q(x_n), K(x_j)\rangle) \cdot V(x_j) \in \mathbb{R}^{width}}{\sum_{\text{Bob's } i} \exp(\langle Q(x_n), K(x_i)\rangle) + \sum_{\text{Alice's } j} \exp(\langle Q(x_n), K(x_j)\rangle) \in \mathbb{R}}$$

→ Lower bound on the **size of** $Q(x_n), V(x_i)$, i.e. width · precision (e.g. T for DISJ^T).

Part I(a) — Tools

Model size needed to solve a task?

1. **Upper bounds**: *"Construct a solution to do some task…"*

- Case-by-case: parity, Dyck [Hahn 20, Yao et al. 21].
- A class of tasks: **finite-state automata**: $O(\log T)$ and $O_{|Q|}(1)$ layers.
- *Another perspective*: "Think like Transformers" — RASP(and variants) in Part III.

2. **Lower bounds**: *"Any solution needs to be as large as…"*

- Depth lower bound — circuit complexity.
- Width (& precision) lower bound — communication complexity. ✓

(size ↔ bottleneck)

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

**a prerequisite* *to learnability and generalization.*

(a) **Tools** for bounding the size of a solution. ✓

- **Upper bounds**: "*Construct* a solution for ..."
- **Lower bounds**: "Any solution needs to be *as large as* ..."

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models.

Part I - Representability

Main question: the existence of an (efficient) solution to a task.
a **prerequisite to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

- **Upper bounds**: "*Construct* a solution for …"
- **Lower bounds**: "Any solution needs to be *as large as* …"

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models.

Part I(b), implication: depth-width tradeoff

[Sanford et al. 24, Bietti et al. 23, Zhang et al. 23]

Induction head [Elhage et al. 21, Olsson et al. 22]: a conditional copying task.

... A B ... A → B

[Mr] [and] [Mrs] [**Durs**] [**ley**] [,] [of] [number] [four] [,] [Pri] [vet] [Drive] [,]
... [they] [just] [didn] ['t] [hold] [with] [such] [nonsense] [.] [Mr] [**Durs**] **[???]**.

- Ubiquitous and important: e.g. in-context learning.

Sanford et al. 24: 1-layer transformers fail to solve induction head

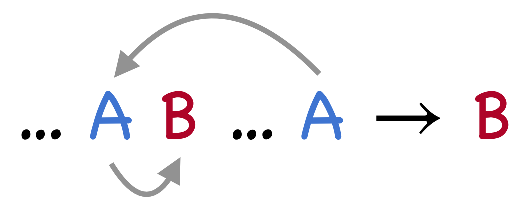
Bietti et al. 23: Birth of a Transformer

Zhang et al. 23: Unveiling transformers with lego

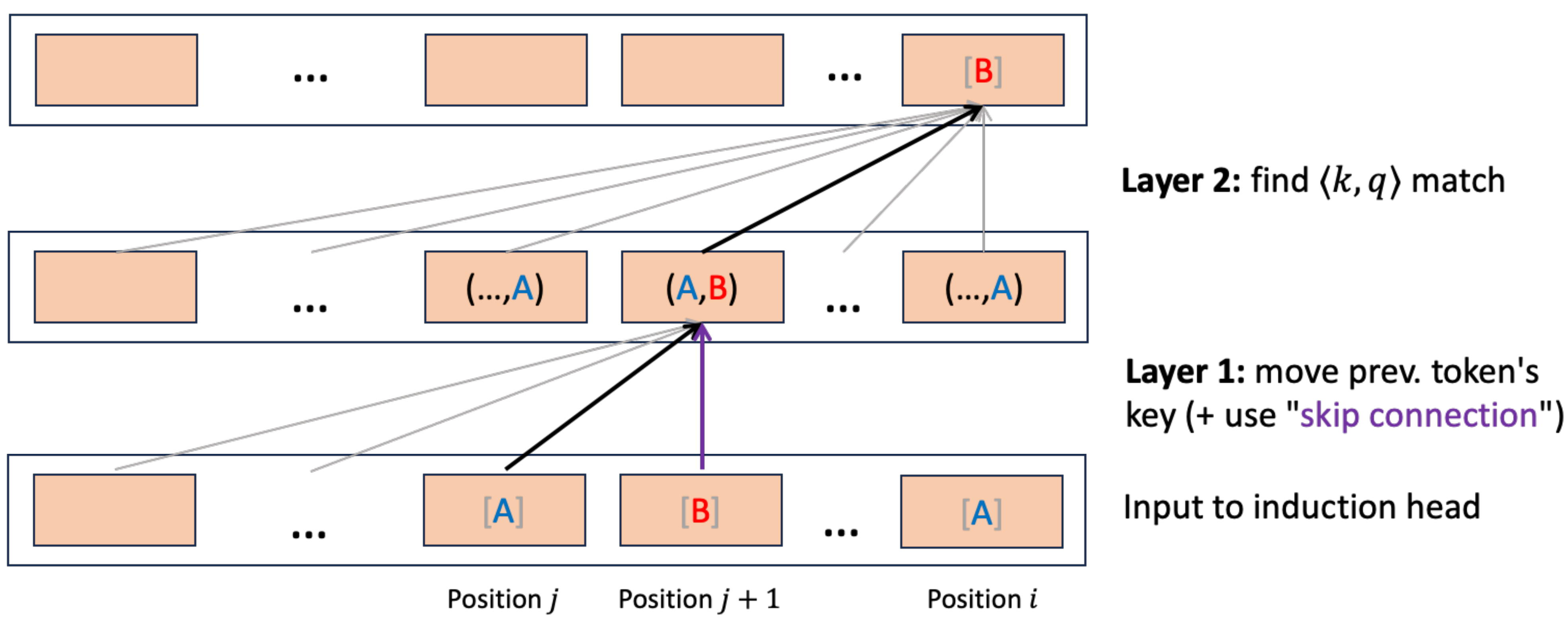
Part I(b), implication: depth-width tradeoff

[Sanford et al. 24, Bietti et al. 23, Zhang et al. 23]

Induction head [Elhage et al. 21, Olsson et al. 22]: a conditional copying task.



- **2 layers** suffice [Bietti et al. 23, Zhang et al. 23, Liu et al. 23b].
  - Length-T input: #heads $h = O(1)$, width $m = O(1)$, precision $p = O(\log T)$.



Part I(b), implication: depth-width tradeoff

[Sanford et al. 24, Bietti et al. 23, Zhang et al. 23]

Induction head [Elhage et al. 21, Olsson et al. 22]: a conditional copying task.

... A B ... A → B

- **2 layers** suffice [Bietti et al. 23, Zhang et al. 23, Liu et al. 23b].
  - Length-T input: #heads $h = O(1)$, width $m = O(1)$, precision $p = O(\log T)$.

*What about **1-layer**?* 1 fewer layer → more parameters: $Hmp = \Omega(T)$ [Sanford et al. 24].

Proof using **communication complexity**:

- Reduce a communication problem (e.g. INDEX) to induction head.
- Design the protocol: sending the key & value vectors.

Sanford et al. 24: 1-layer transformers fail to solve induction head

Part I(b), implication: depth-width tradeoff

Benefit of depth?

- Parameter efficiency (communication complexity; e.g. induction head)

- Representational power (circuit complexity; e.g. S_5).

  Also for *looped* Transformers [Dehghani et al. 18, Yang et al. 23, Merrill et al. 24].

Part I(b), implication: depth-width tradeoff

Benefit of depth?

- Parameter efficiency (communication complexity; e.g. induction head)

- Representational power (circuit complexity; e.g. S_5).

  Also for *looped* Transformers [Dehghani et al. 18, Yang et al. 23, Merrill et al. 24].

***Other complexity notions*?**

- separation rank [Levine et al. 20].
- formal logic [Chiang et al. 23, Barceló et al. 24].
  - ❖ C-RASP [Yang et al. 24]

Part I - Representability

Main question: the existence of an (efficient) solution to a task.
a **prerequisite to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

- **Upper bounds**: "*Construct* a solution for ..."
- **Lower bounds**: "Any solution needs to be *as large as* ..."

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff. ✓
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models.

Part I(b), implication: comparison

A > B: 1) a lower bound for B; 2) a (more efficient) construction of A.

Transformer > RNN: RNN is bottlenecked by the hidden state size.

- e.g. retrieval / copying / associative recall / (k-hop) induction head.
  [Arora et al. 24; Bhattamishra et al. 24; Jelassi et al. 24; Sanford et al. 24; Wen et al. 24]

RNN > (limited depth) Transformer: insufficient "effective depth".

- e.g. S_5, bounded Dyck [Merrill & Sabharwal 22, Liu et al. 23, Bhattamishra et al. 24]).

Efficient models?

e.g. subquadratic Transformers [Alman & Yu 24]; SSMs [Sarrof et al. 2024, Grazzi et al. 2024].

Part I(b), implication: comparison

A > **B**: 1) a lower bound for B; 2) a (more efficient) construction of A.

Transformer > **RNN**: RNN is bottlenecked by the hidden state size.

- e.g. retrieval / copying / associative recall / (k-hop) induction head.

  [Arora et al. 24; Bhattamishra et al. 24; Jelassi et al. 24; Sanford et al. 24; Wen et al. 24]

RNN > **(limited depth) Transformer**: insufficient "effective depth".

- e.g. S_5, bounded Dyck [Merrill & Sabharwal 22, Liu et al. 23, Bhattamishra et al. 24]).

Efficient models?

e.g. subquadratic Transformers [Alman & Yu 24]; SSMs [Sarrof et al. 2024, Grazzi et al. 2024].

Albert Gu's blog: https://goombalab.github.io/blog/2025/tradeoffs/

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

*a **prerequisite** to learnability and generalization.

(a) **Tools** for bounding the size of a solution.

- **Upper bounds**: "*Construct* a solution for ..."
- **Lower bounds**: "Any solution needs to be *as large as* ..."

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs). ✓
3. **Improving** with Chain-of-Thought and hybrid models.

Part I(b), implication: improvement

Transformers

- Representational advantages
  (param. efficiency over RNN/GNN)
- Representational limitations

→ **(1) Chain of Thought**
[Feng et al. 23; Malach 23, Merrill et al. 23, Li et al. 24]

- Quadratic cost
  (memory & compute)

RNNs/SSMs

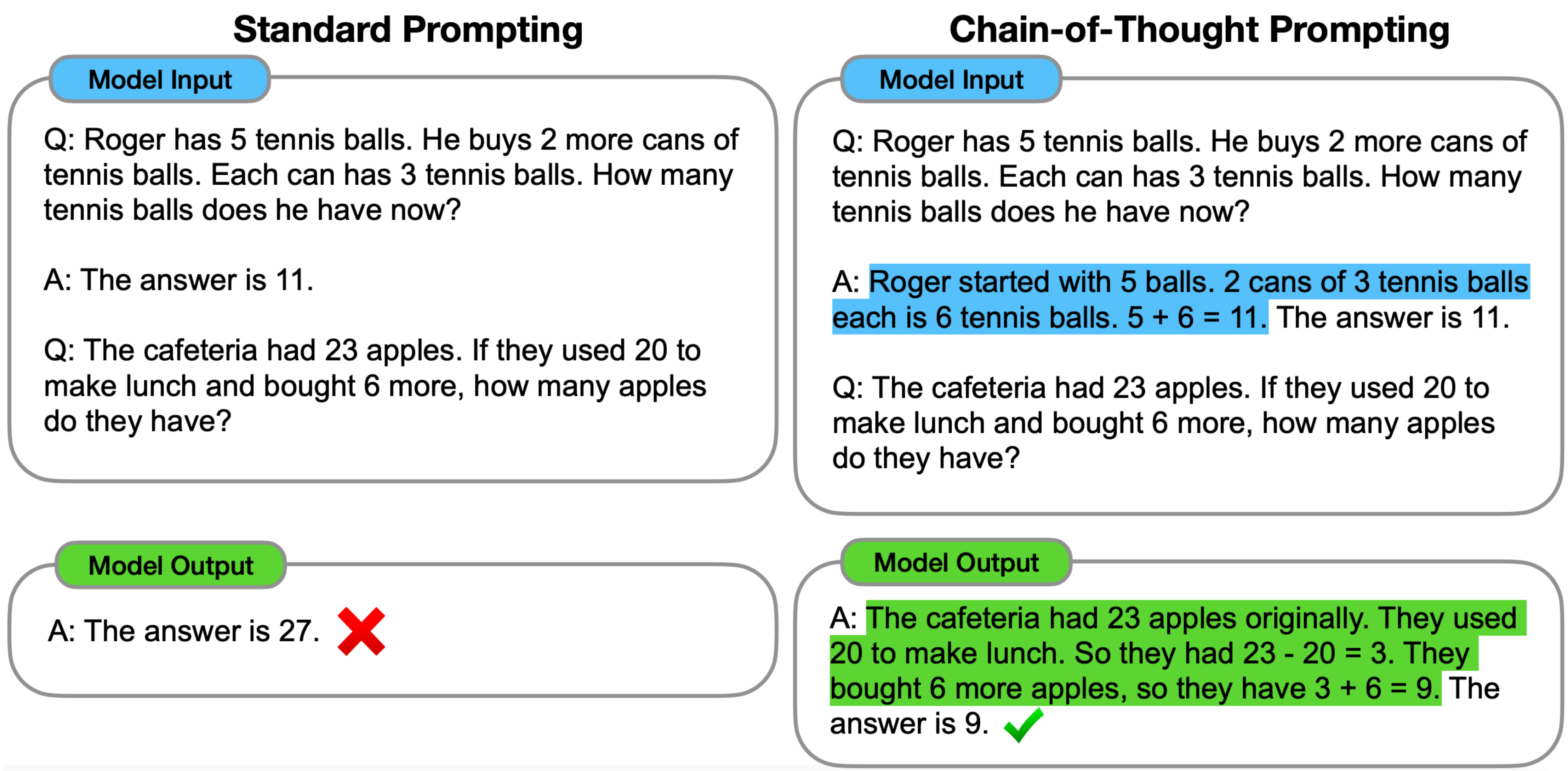
- Representational limitations.
  (e.g. copying; parity)
  [Jelassi et al. 24, Arora et al. 24, Sarrof et al. 24, Grazzi et al. 24]
- Linear cost
  (memory & compute)

→ **(2) Hybrid**

Combining RNN + Transformer layers [Wen et al. 24]

Part I(b), implication: improvement

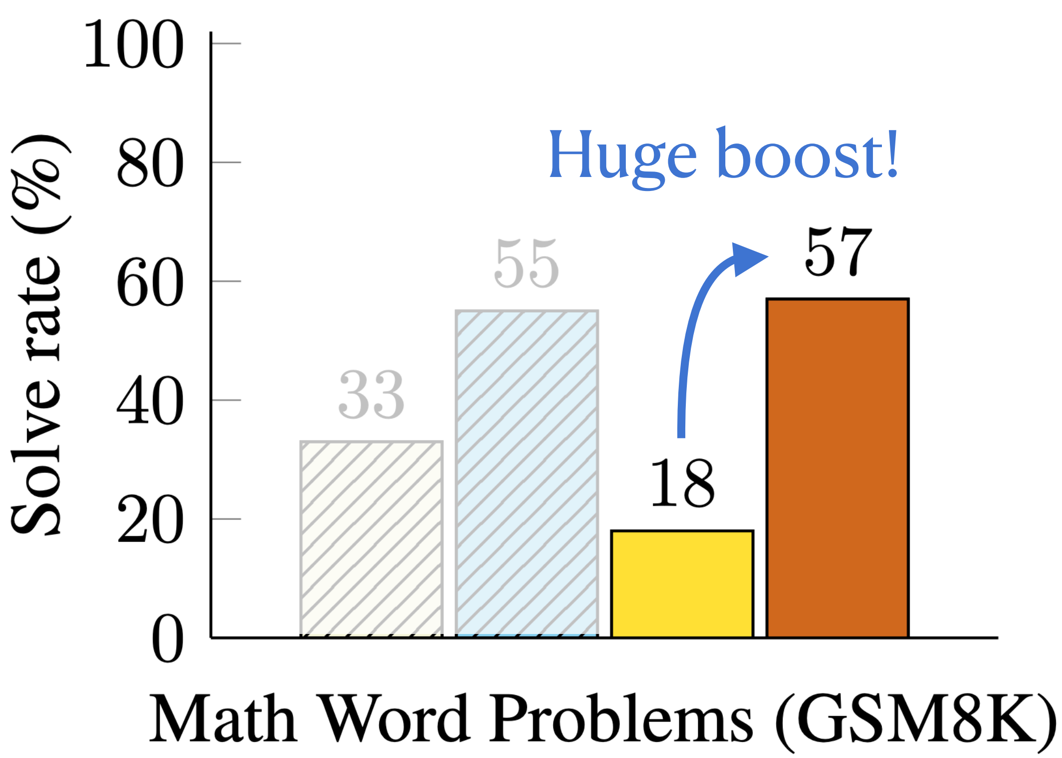
Chain of Thought (Wei et al. 22): solving a task step-by-step.



Part I(b), implication: improvement

Chain of Thought (Wei et al. 22): solving a task step-by-step.

Empirically



Theoretically

[Feng et al. 23; Malach 23; Merrill et al. 23; Li et al. 24]

(Informal) Circuits of M gates can be simulated by $O(M)$ steps of CoT.

Idea: simulating 1 gate with $O(1)$ steps:

- Attention to collect inputs.
- MLP to compute the gate.

Part I(b), implication: improvement

[Wei et al. 22; Feng et al. 23; Malach 23; Merrill et al. 23; Li et al. 24]

Increasing power via 1) more **depth** or 2) more **CoT** steps?

1. # **sequential steps**? ... depth "wins".

e.g. graph connectivity [Merrill & Sabharwal 24]:

$O(\log T)$ depth ✓ $O(\log T)$ CoT ×

2. "**Uniformity**" (≈ same construction for different T)? ... tied.

Implication for ***length generalization***: **yes** for both (log precision).

(more in Part III)

Merrill & Sabharwal et al. 24: A little depth goes a long way

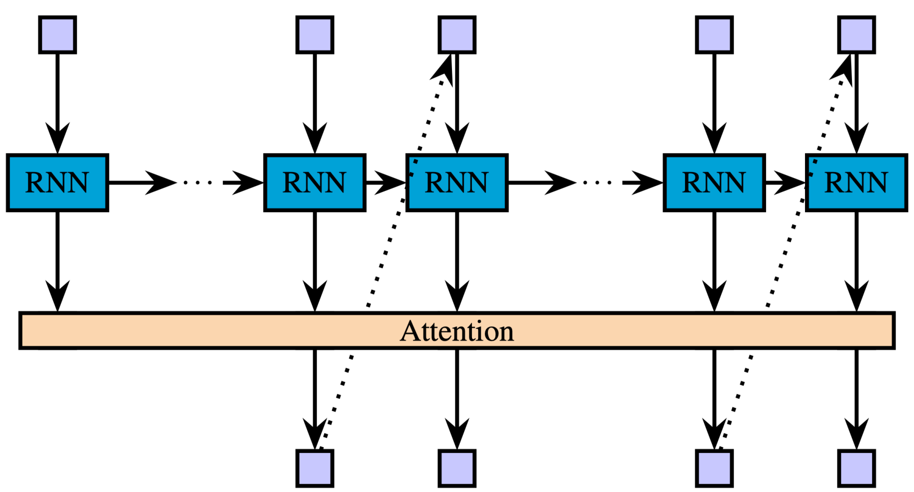
Part I(b), implication: improvement

[Wen et al. 2024]

How about improving RNNs?

- CoT is not sufficient: the memory constraint remains.

Hybrid: 1 attention layer suffices. → low memory footprint, high quality.



- e.g. **Jamba** [Lieber et al. 24]:
  Transformer:Mamba = **1:7**

Efficient alternatives to attention?

... quality & efficiency tradeoff

e.g. sliding window attn [Ren et al. 24]

- *conv layers in S4/Mamba?*

Part I - Representability

Main question: the existence of an (efficient) solution to a task.
a **prerequisite to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

- **Upper bounds**: "*Construct* a solution for …"
- **Lower bounds**: "Any solution needs to be *as large as* …"

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models. ✓
(or depth?) (efficiency + performance)

Part I - Representability

Main question: the existence of an (efficient) solution to a task.

a* **prerequisite *to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

- **Upper bounds** (“*Construct* a solution for …”): finite-state automata.
- **Lower bounds** (“Any solution needs to be …”): communication/circuit.

(b) **Implications** of representability:

1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models.

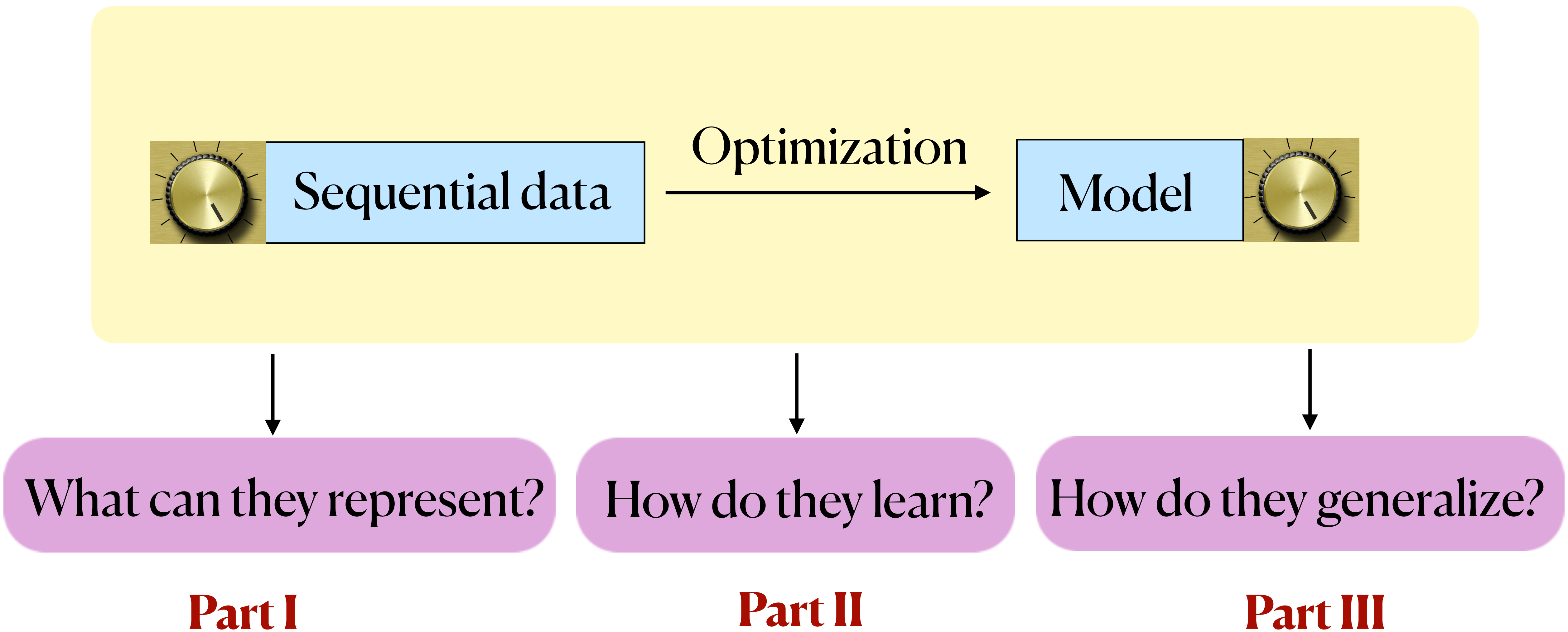
Sandbox for the Blackbox

How language models learn structured data

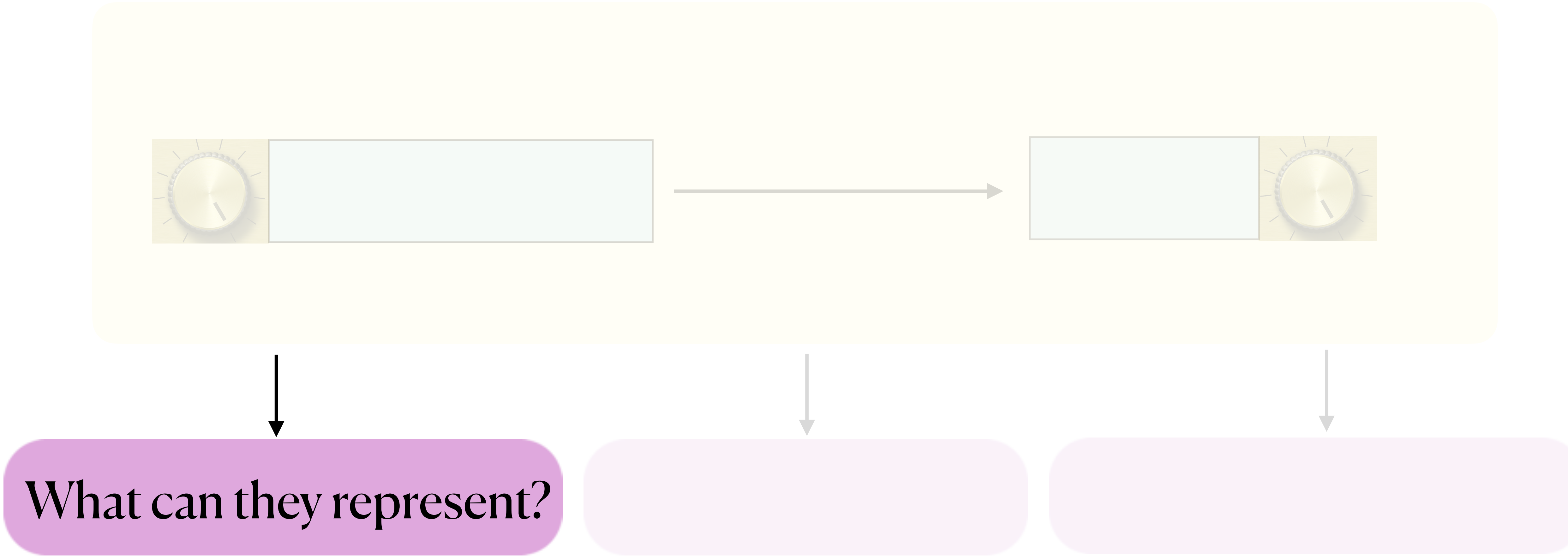
Ashok Vardhan Makkuva

(EPFL ⟶ Télécom Paris)

Recap


Sequential data
Optimization
Model
What can they represent?
How do they learn?
How do they generalize?
Part I
Part II
Part III

Recap



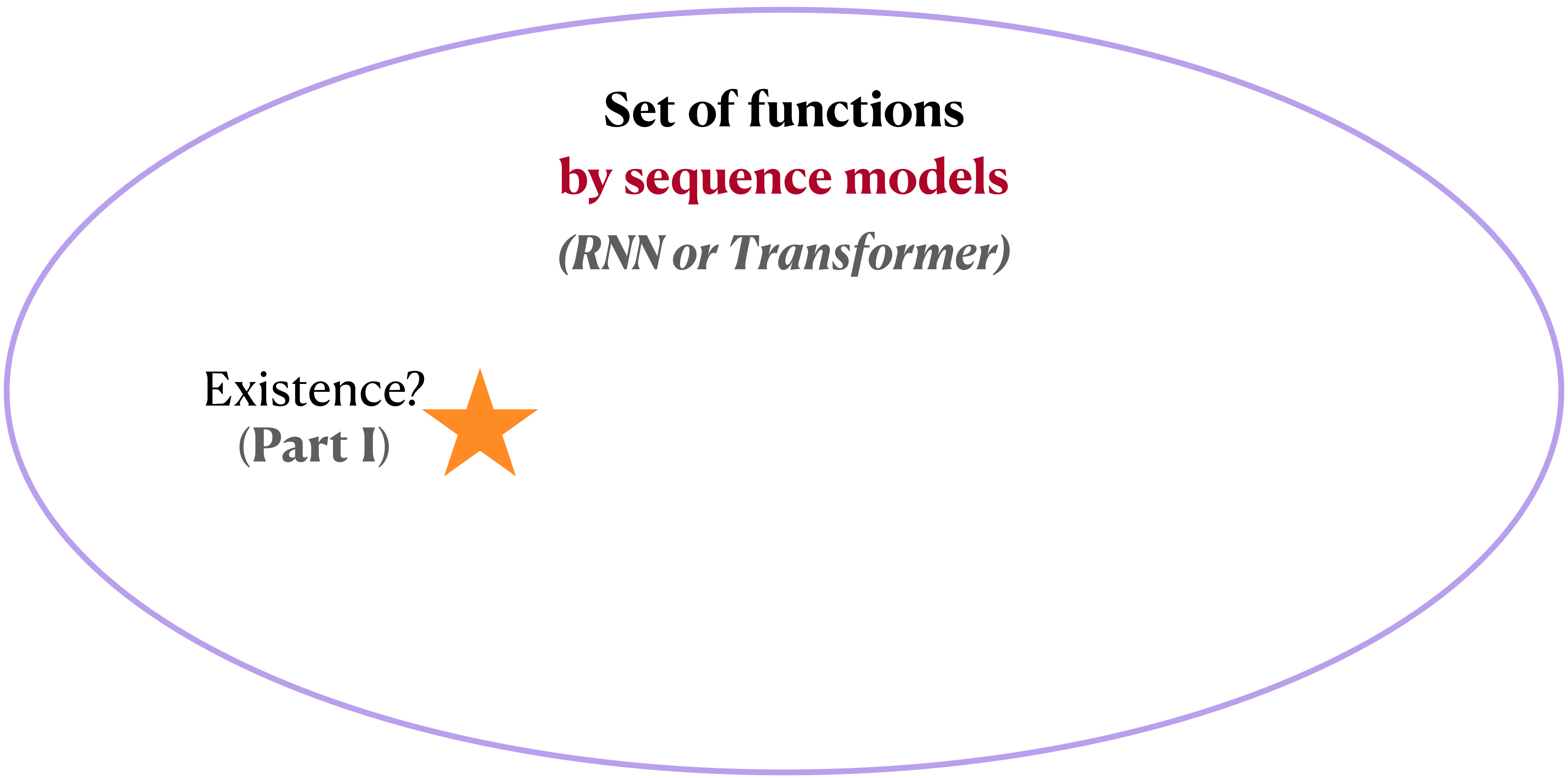
Part I

Part I: Representability

(aka. expressivity)

Part I - Recap

Main question: the existence of an (efficient) solution to a task.



Part I - Recap

Main question: the existence of an (efficient) solution to a task.

a* **prerequisite *to learnability and generalization.*

(a) **Tools** for bounding the size of a solution.

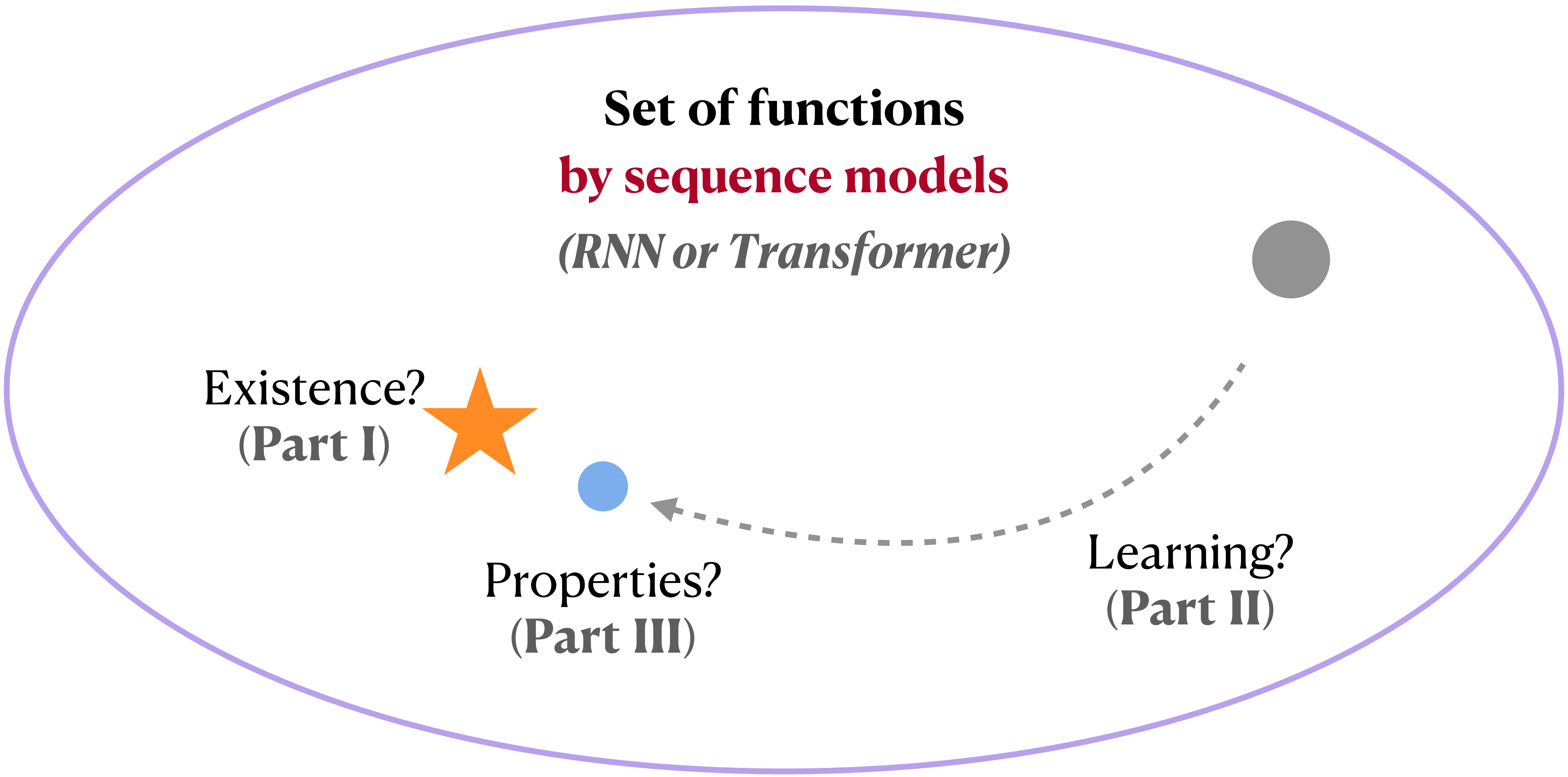
- **Upper bounds** ("*Construct* a solution for …"): finite-state automata.
- **Lower bounds** ("Any solution needs to be …"): communication/circuit.

(b) **Implications** of representability:

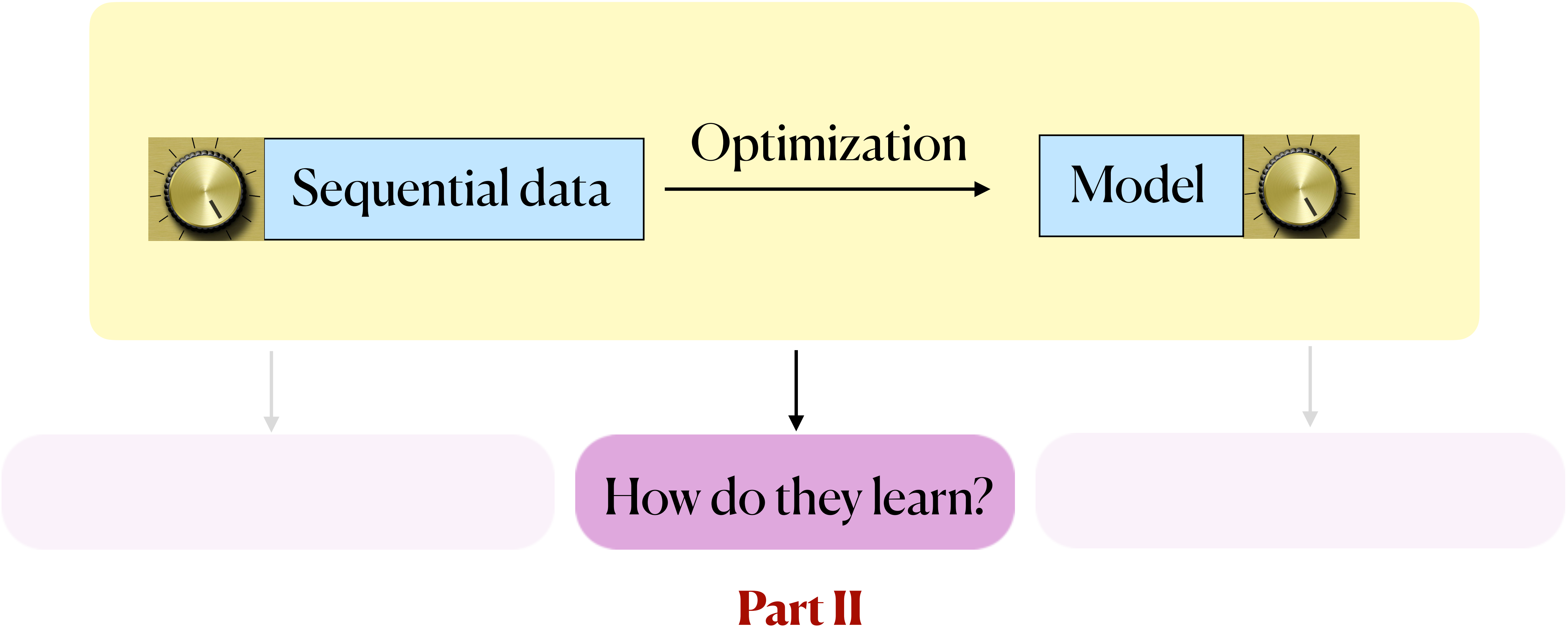
1. **Understanding design choices**: depth-width tradeoff.
2. **Comparing** Transformers vs RNNs (SSMs).
3. **Improving** with Chain-of-Thought and hybrid models.

Part I - Recap

Main question: the existence of an (efficient) solution to a task.



Today



Sequential data
Optimization
Model
How do they learn?
Part II

Markov/n-gram

Topic models

Factual recall

Transformers

How do they learn?

Markov/n-gram

Topic models

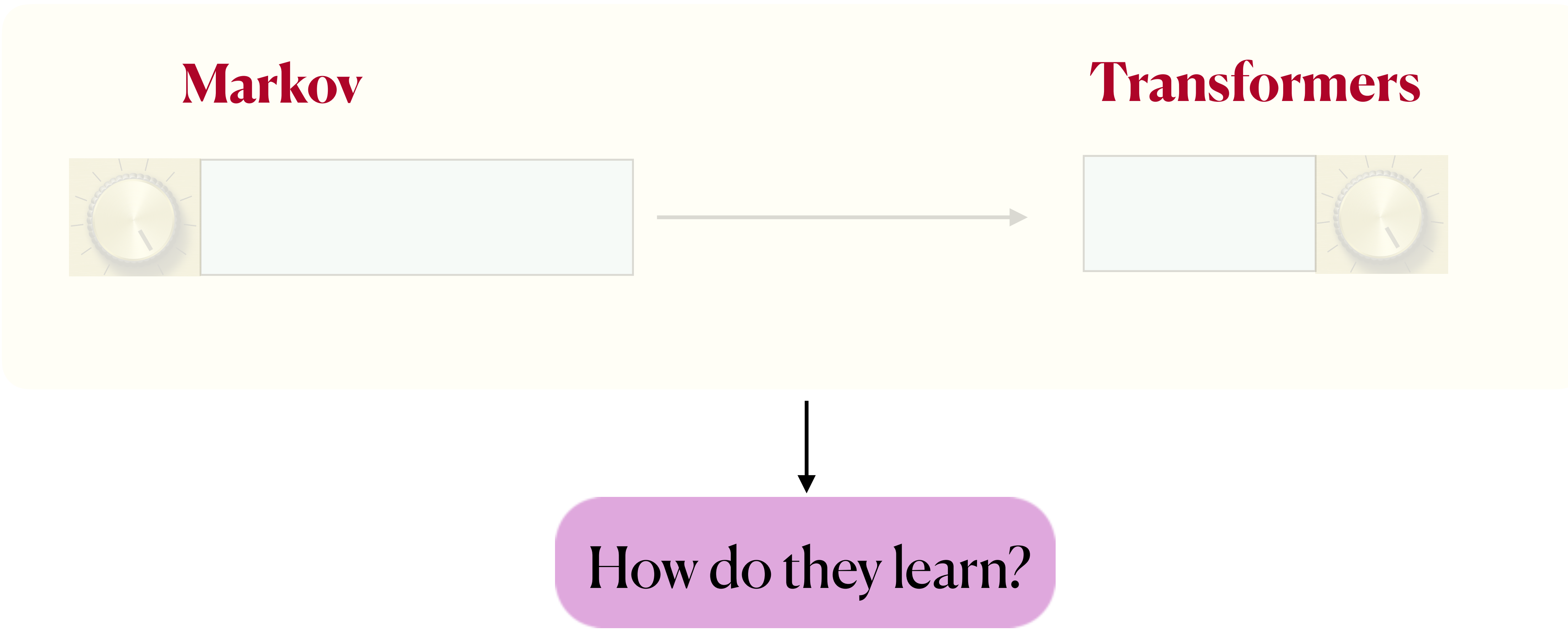
Factual recall

Transformers

How do they learn?

Optimization landscape

Learning dynamics



[**M**akkuva et al. 2024, **M**akkuva et al. 2024, Nichani et al. 2024, Bietti et al. 2023, Guo et al. 2024, Chen et al. 2024, Edelman et al. 2024, **R**ajaraman et al. 2024, **E**kbote et al. 2025]

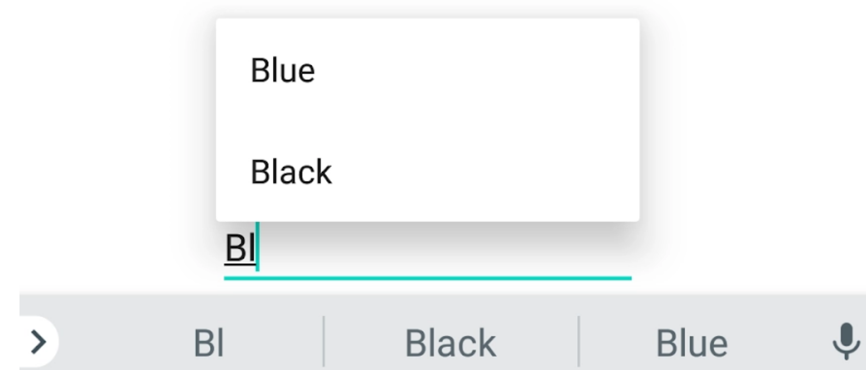
Why Markovian?

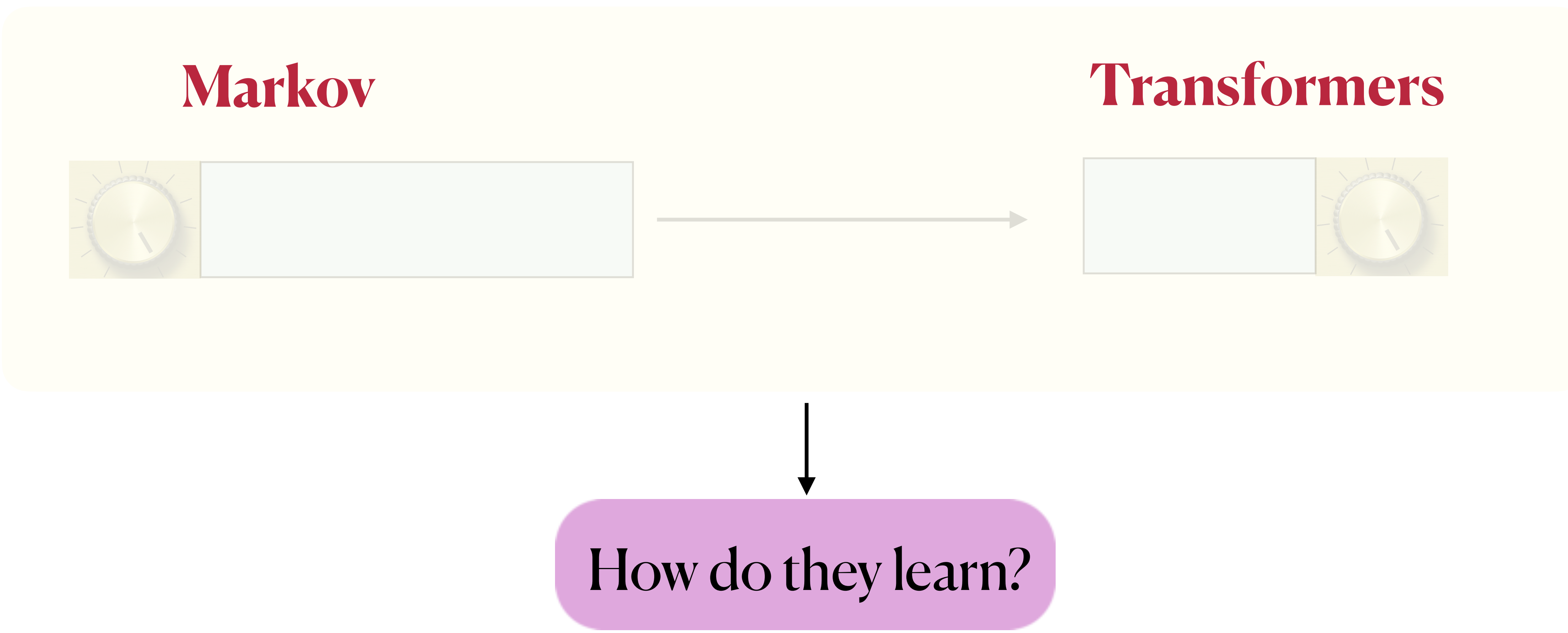
Shannon, 1948

Grammar

Syntax

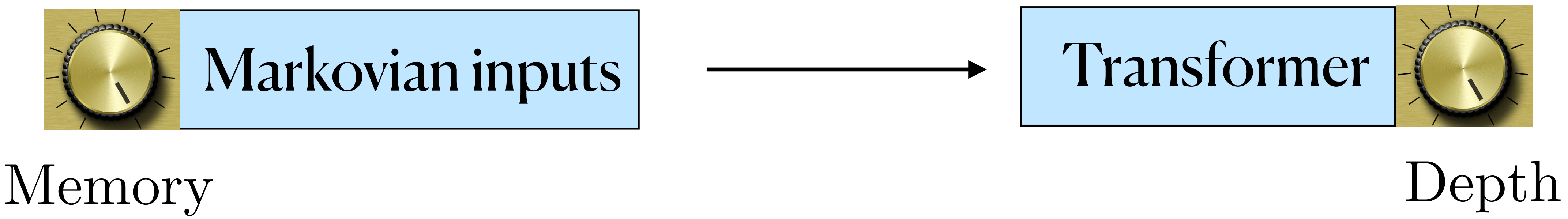
Markovian




Markov
Transformers
How do they learn?

Recipe

- Find the structure in the solutions learnt by gradient-based methods
- Reparametrize the transformer parameters using this structure
- Go with the flow! (or GD)



Markovian inputs
Transformer
Memory
Depth

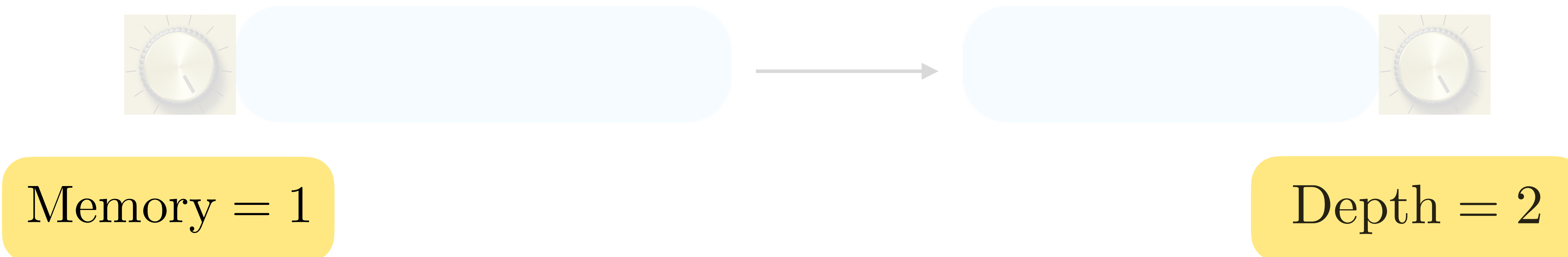
Outline

Memory $= 1$

Depth $= 1$

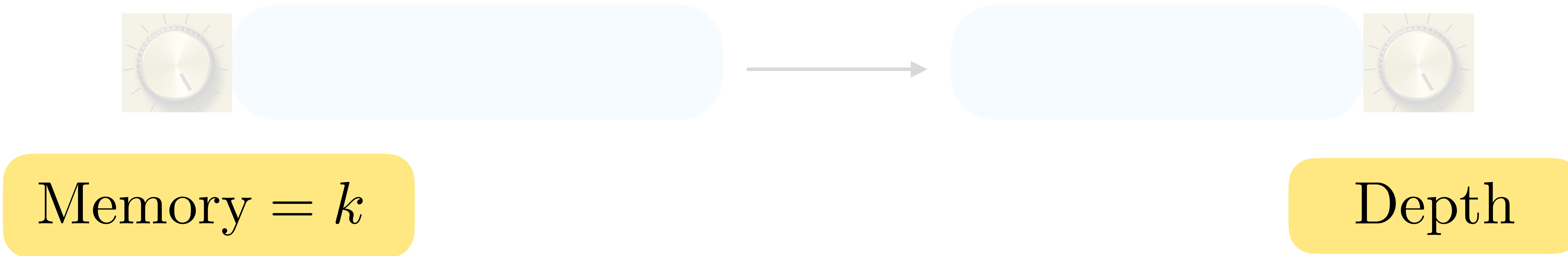
[**M**akkuva et al. 2024, **M**akkuva et al. 2024]

Outline

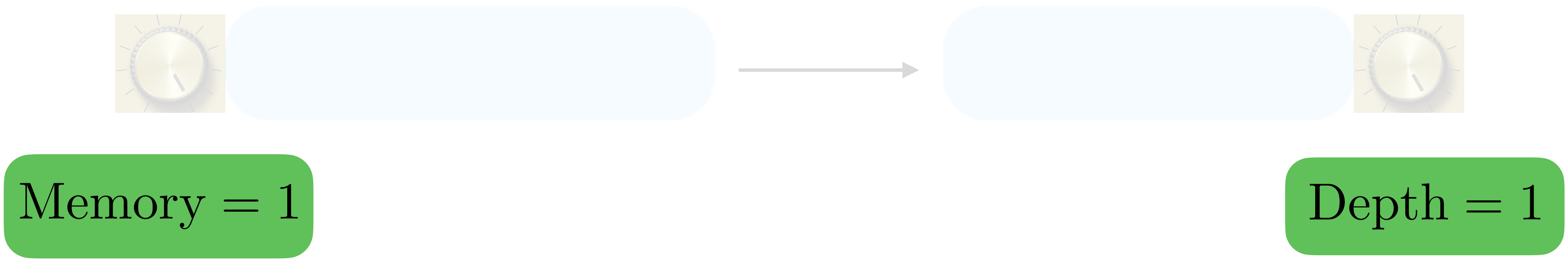


[Nichani et al. 2024, Bietti et al. 2023, Edelman et al. 2024]

Outline



[**R**ajaraman et al. 2024, Chen et al. 2024, **E**kbote et al. 2025]



[**M**akkuva et al. 2024, **M**akkuva et al. 2024]

Key Takeaways

Single-layer transformers sometimes fail to learn even first-order Markov chains!

Memory = 1

Depth = 1

Markovian switching and initialization play a key role in the learning dynamics

Markovian inputs

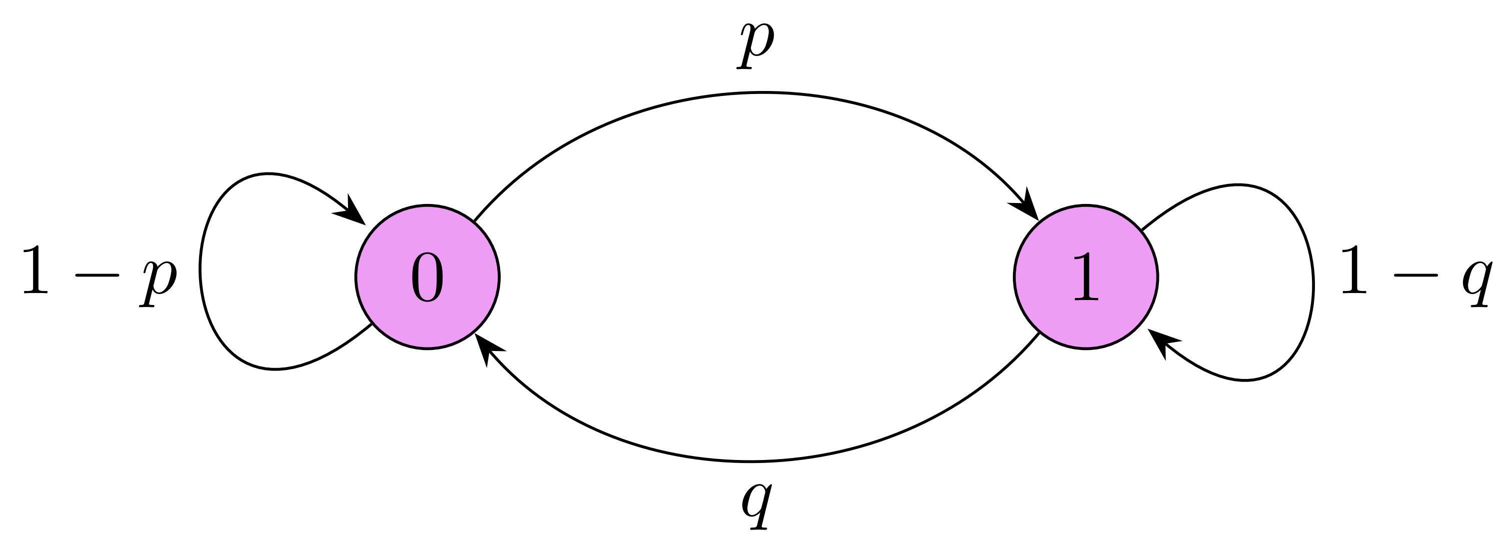
Memory = 1

Depth = 1

First-order Markov chain (Global)

First-order Markov chain

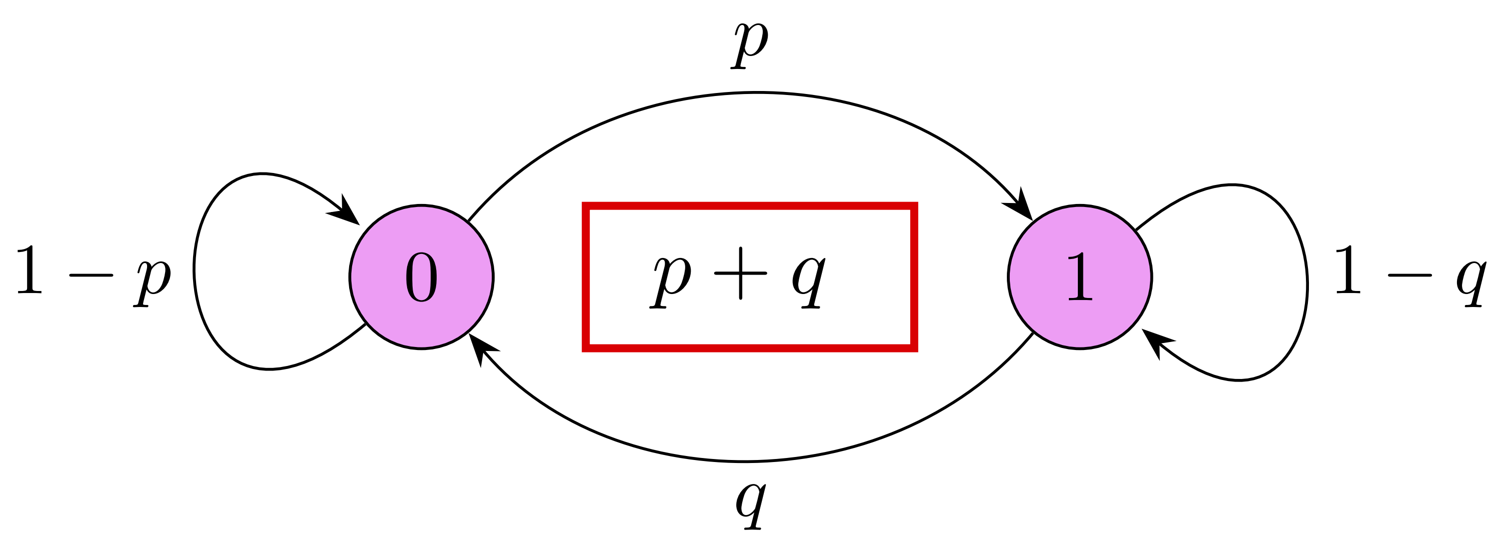
$$(x_n)_{n\geq 1} \sim (\boldsymbol{\pi}, \boldsymbol{P}) \quad \Leftrightarrow$$



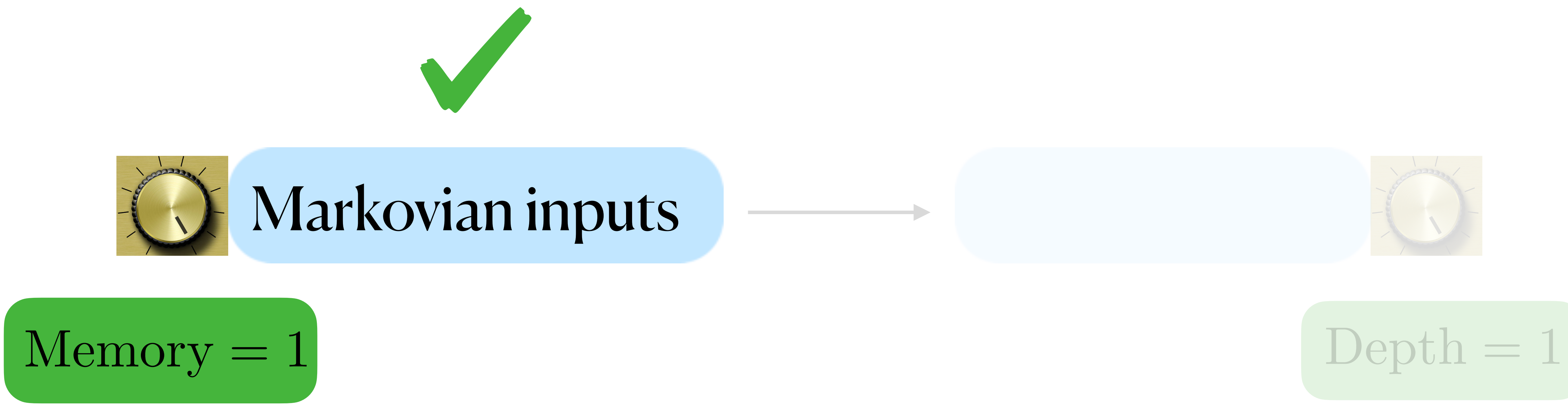
$$\boldsymbol{\pi} = (\pi_0, \pi_1) = \left(\frac{q}{p+q}, \frac{p}{p+q}\right), \quad \boldsymbol{P} = (\boldsymbol{P}_{ij}) = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}.$$

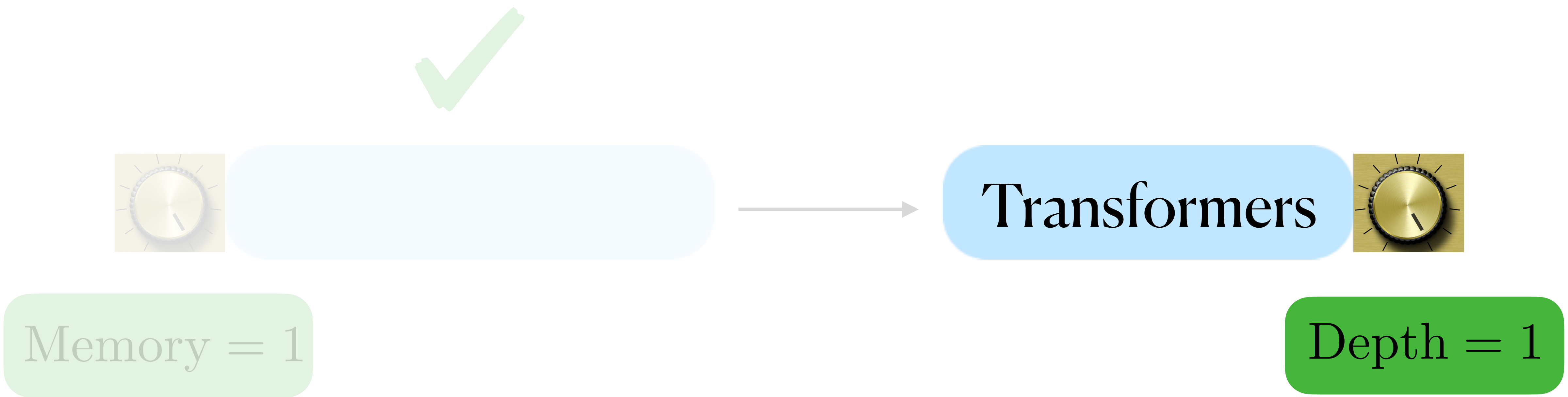
First-order Markov chain

$(x_n)_{n\geq 1} \sim (\boldsymbol{\pi}, \boldsymbol{P}) \quad \Leftrightarrow$



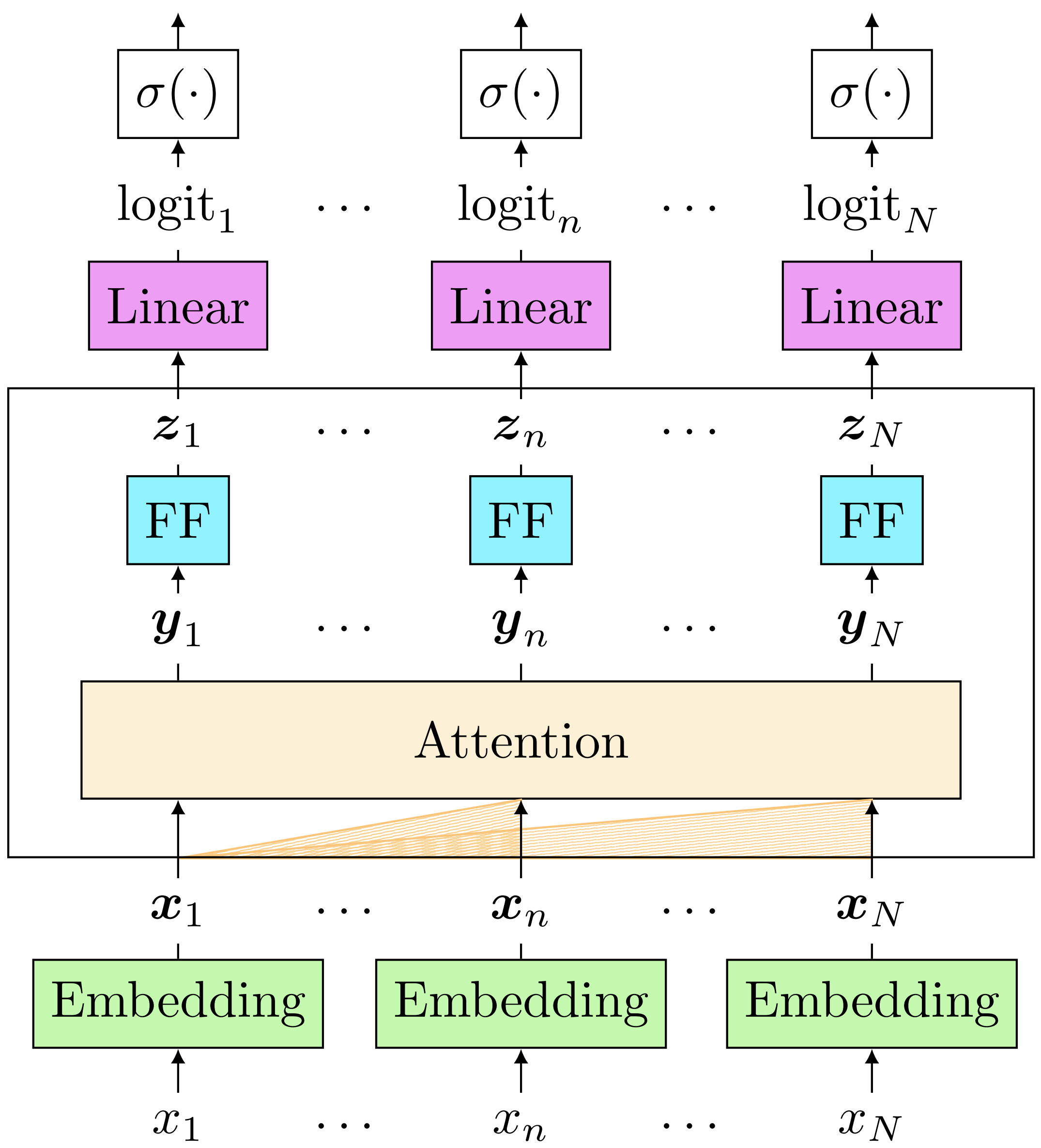
$$\boldsymbol{\pi} = (\pi_0, \pi_1) = \left(\frac{q}{p+q}, \frac{p}{p+q}\right), \quad \boldsymbol{P} = (\boldsymbol{P}_{ij}) = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}.$$


Markovian inputs
Memory = 1
Depth = 1


Transformers
Memory = 1
Depth = 1

Transformers

Single-layer Transformer



[Vaswani et. al., 2017]

Single-layer Transformer

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\mathrm{logit}_n)$$

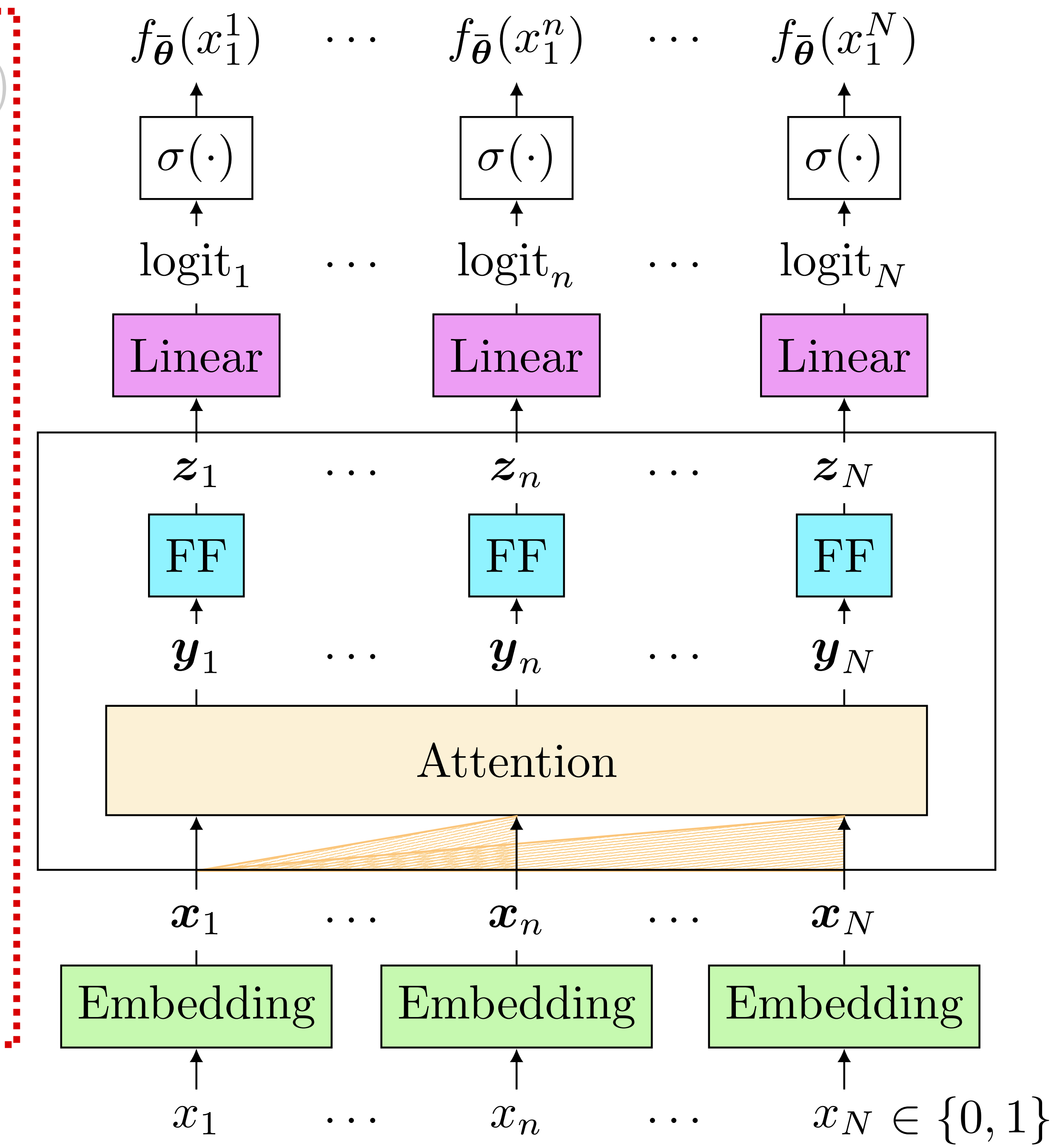
$$\mathrm{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \,\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \mathrm{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

$\boldsymbol{\theta}$



Single-layer Transformer

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\mathrm{logit}_n)$$

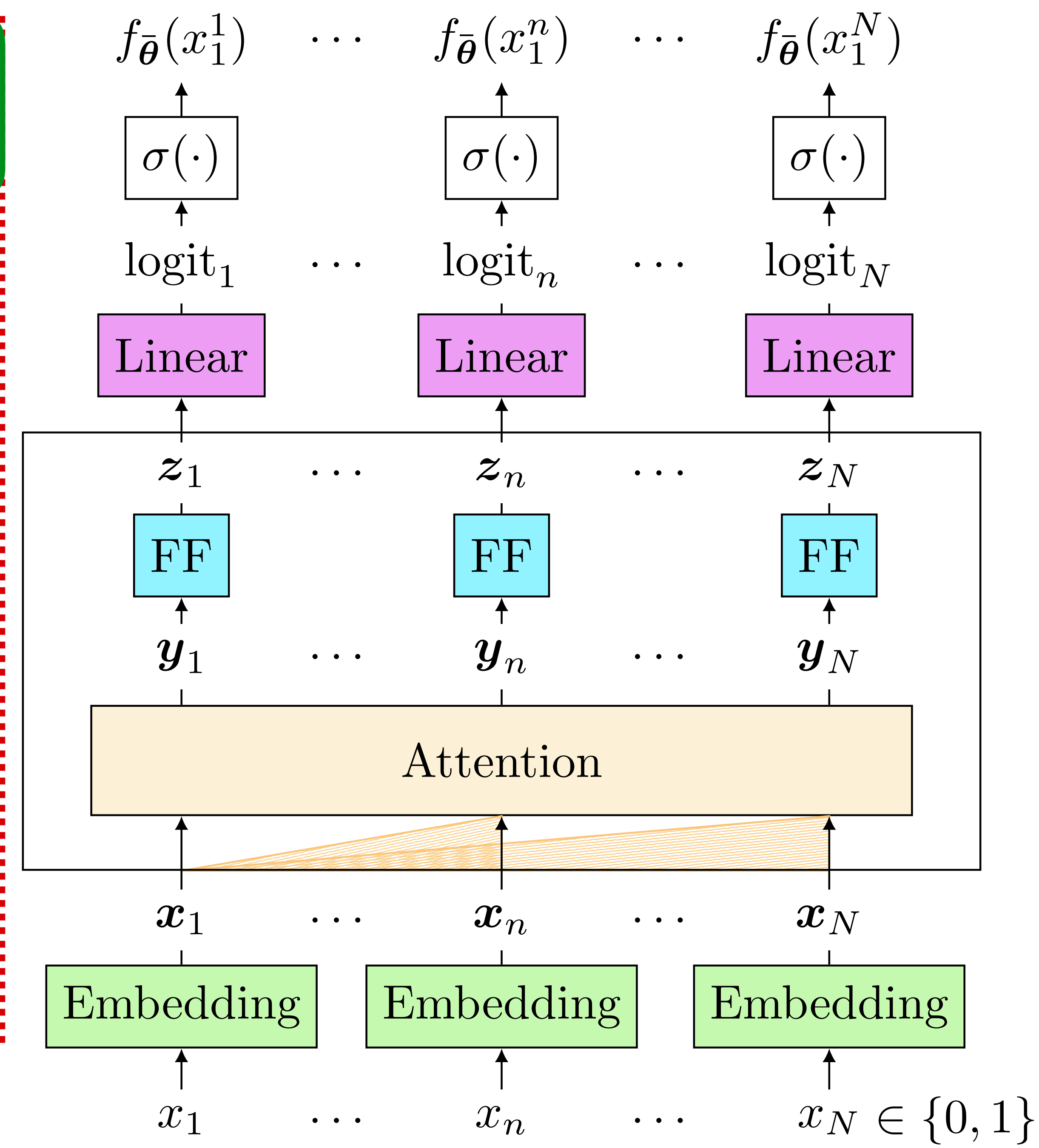
$$\mathrm{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \,\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \mathrm{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

$\boldsymbol{\theta}$



Single-layer Transformer

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\mathrm{logit}_n)$$

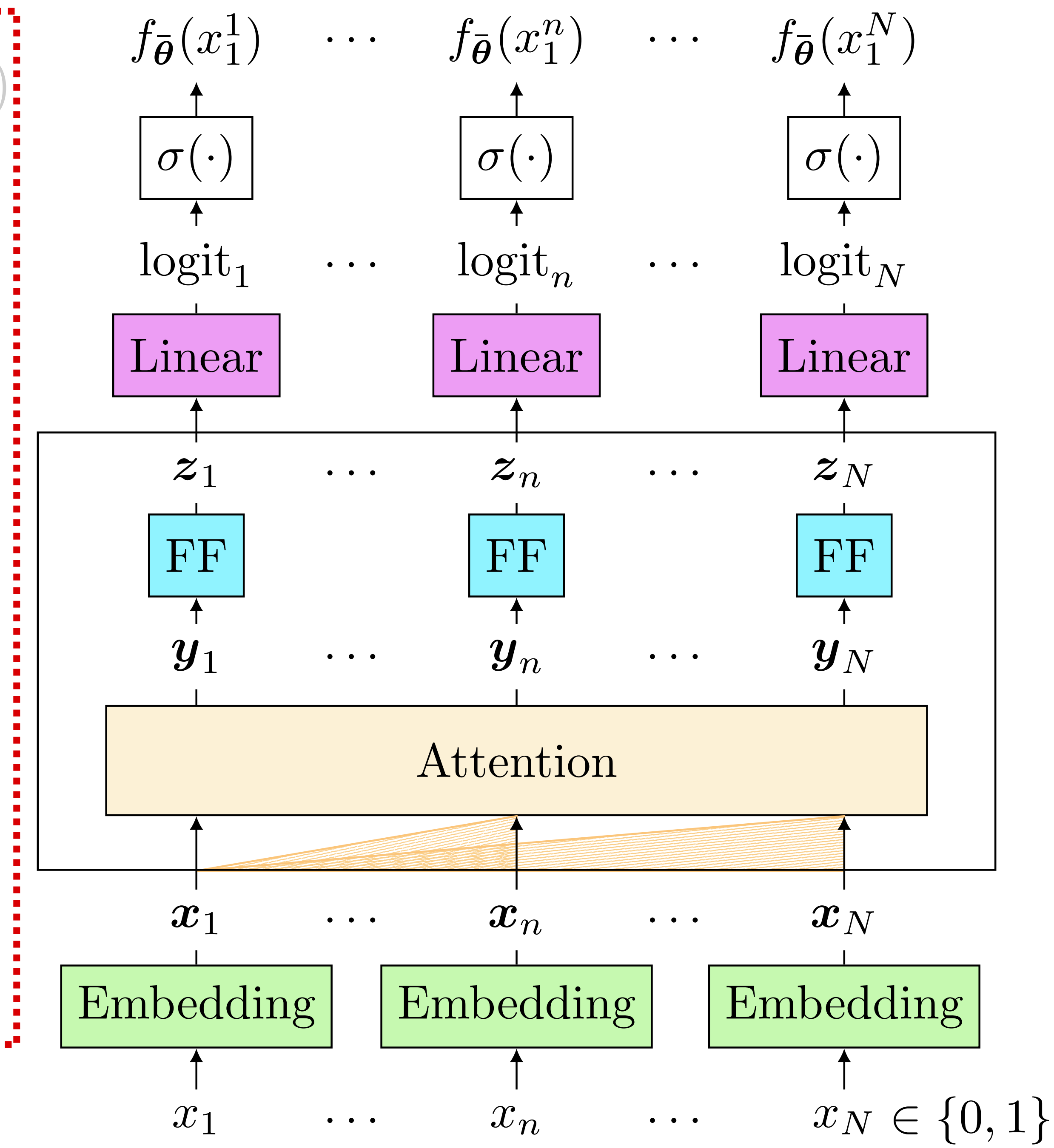
$$\mathrm{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \,\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

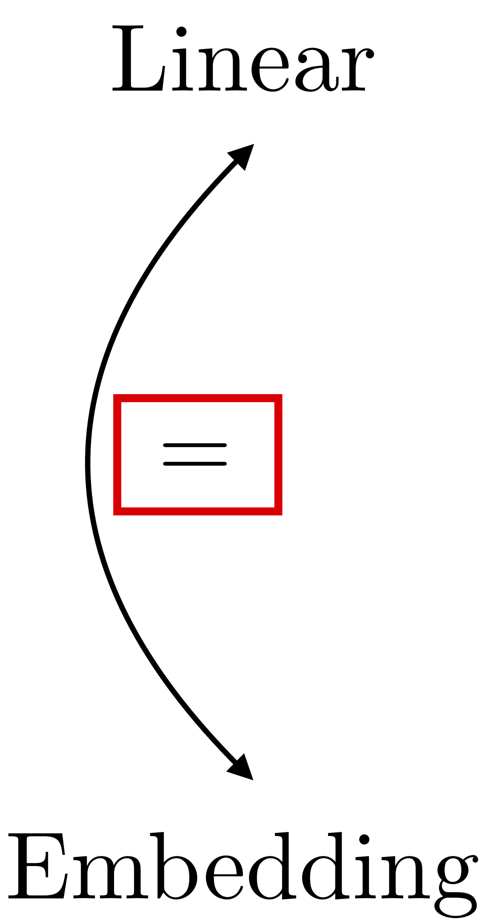
$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \mathrm{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

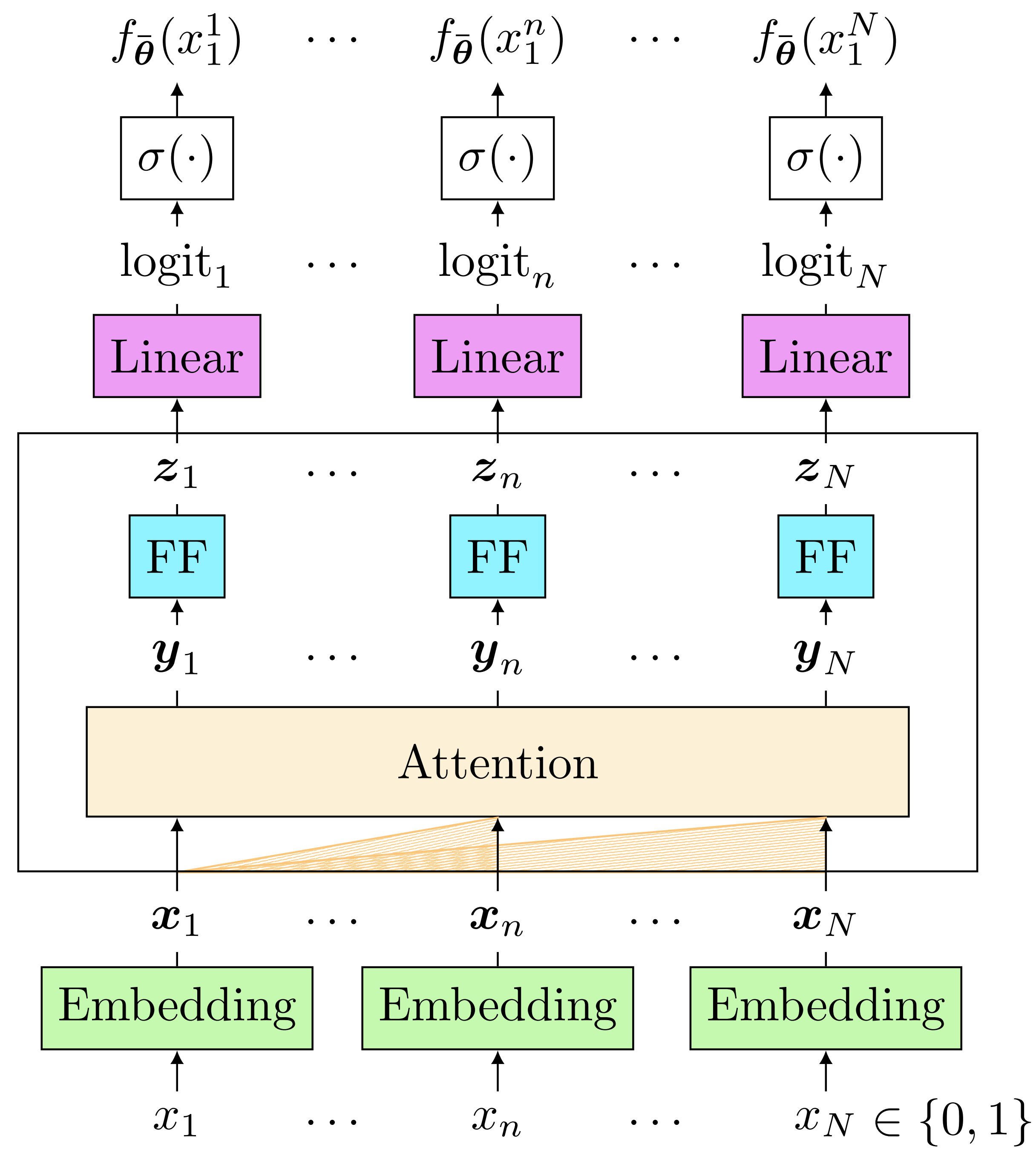
$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

$\boldsymbol{\theta}$



Weight tying

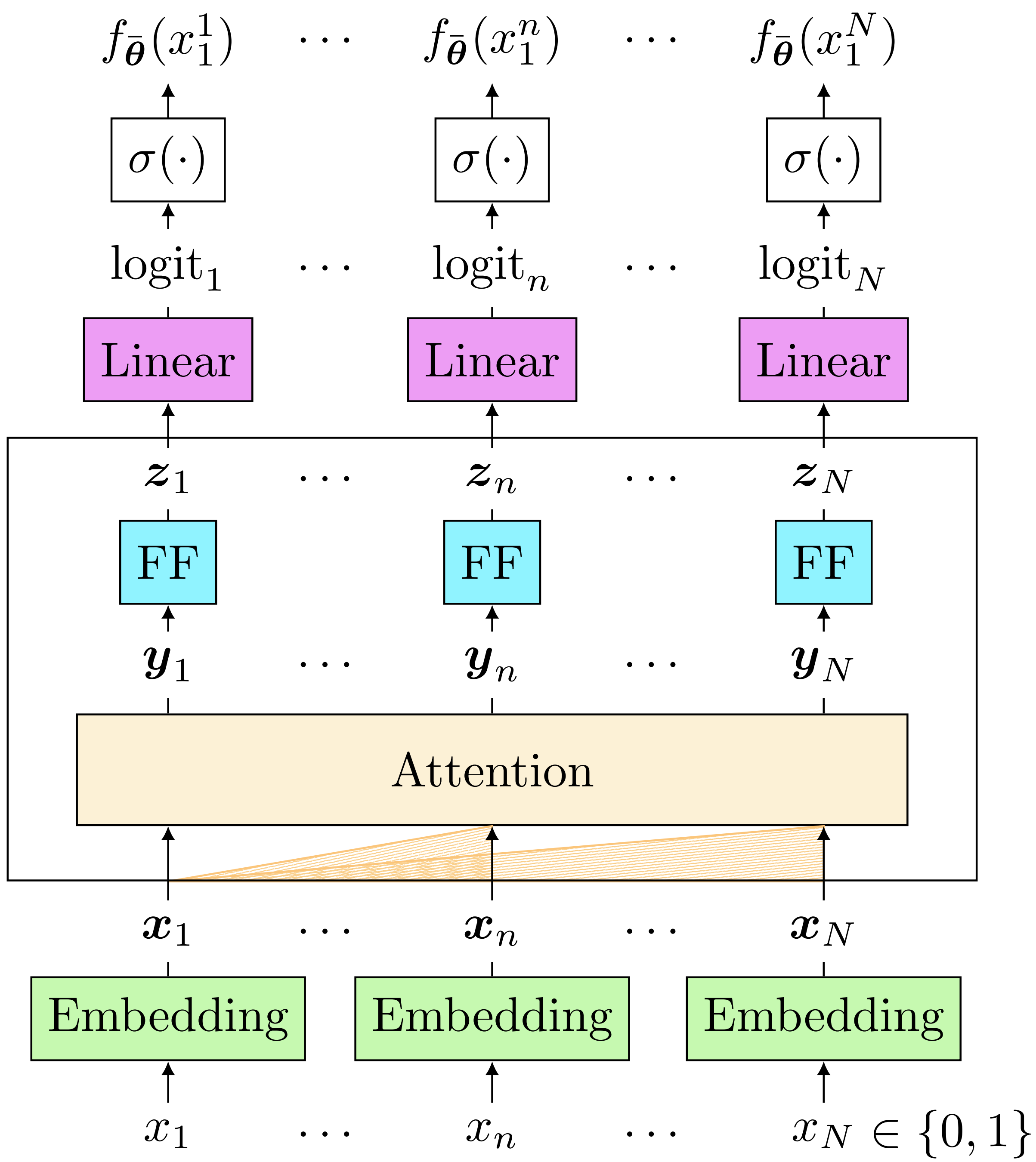




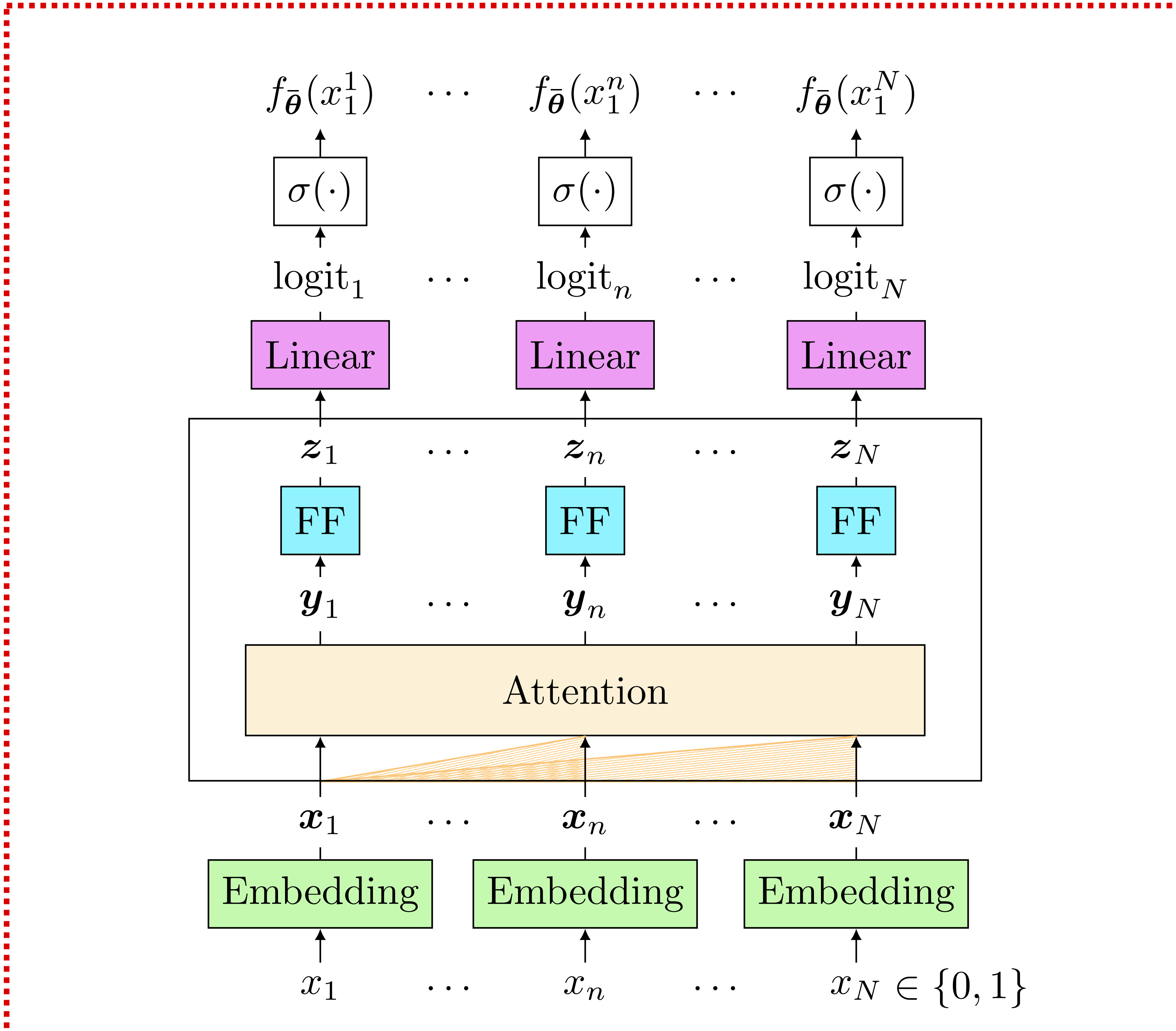
No Weight tying

Linear

Embedding

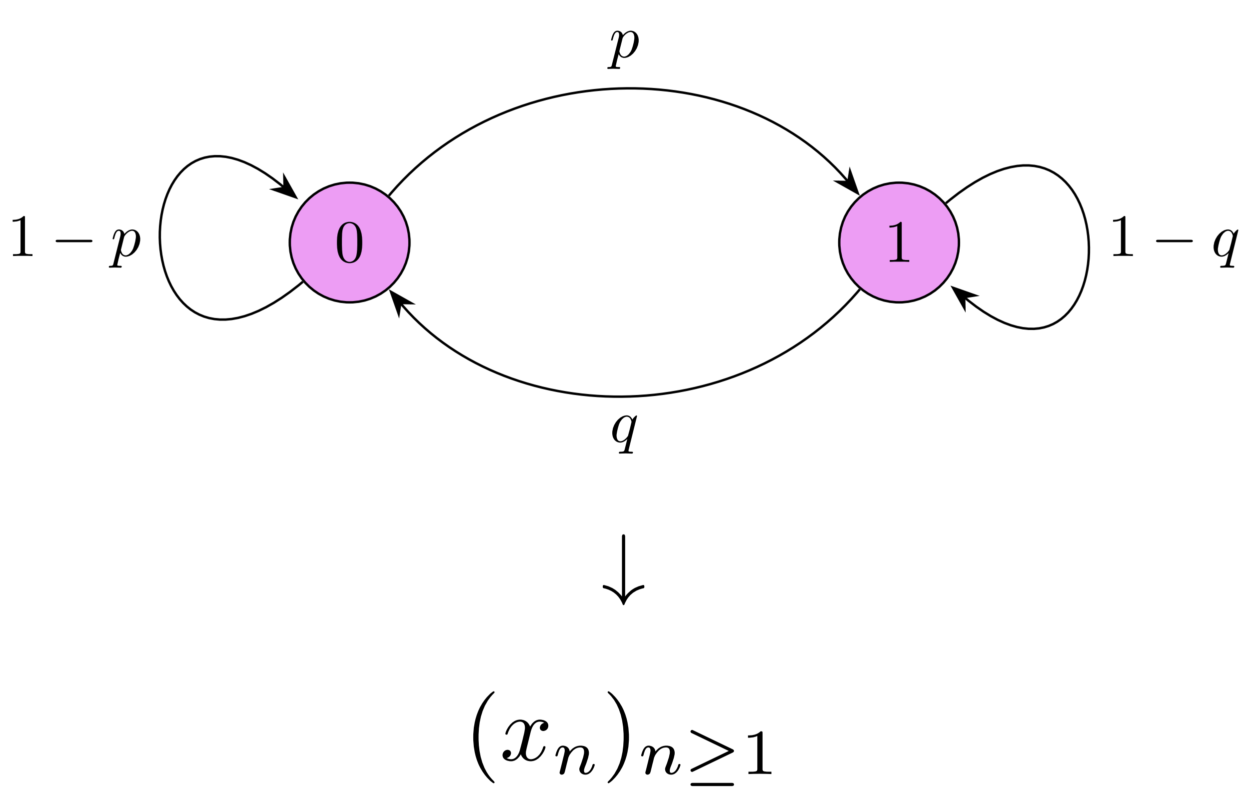


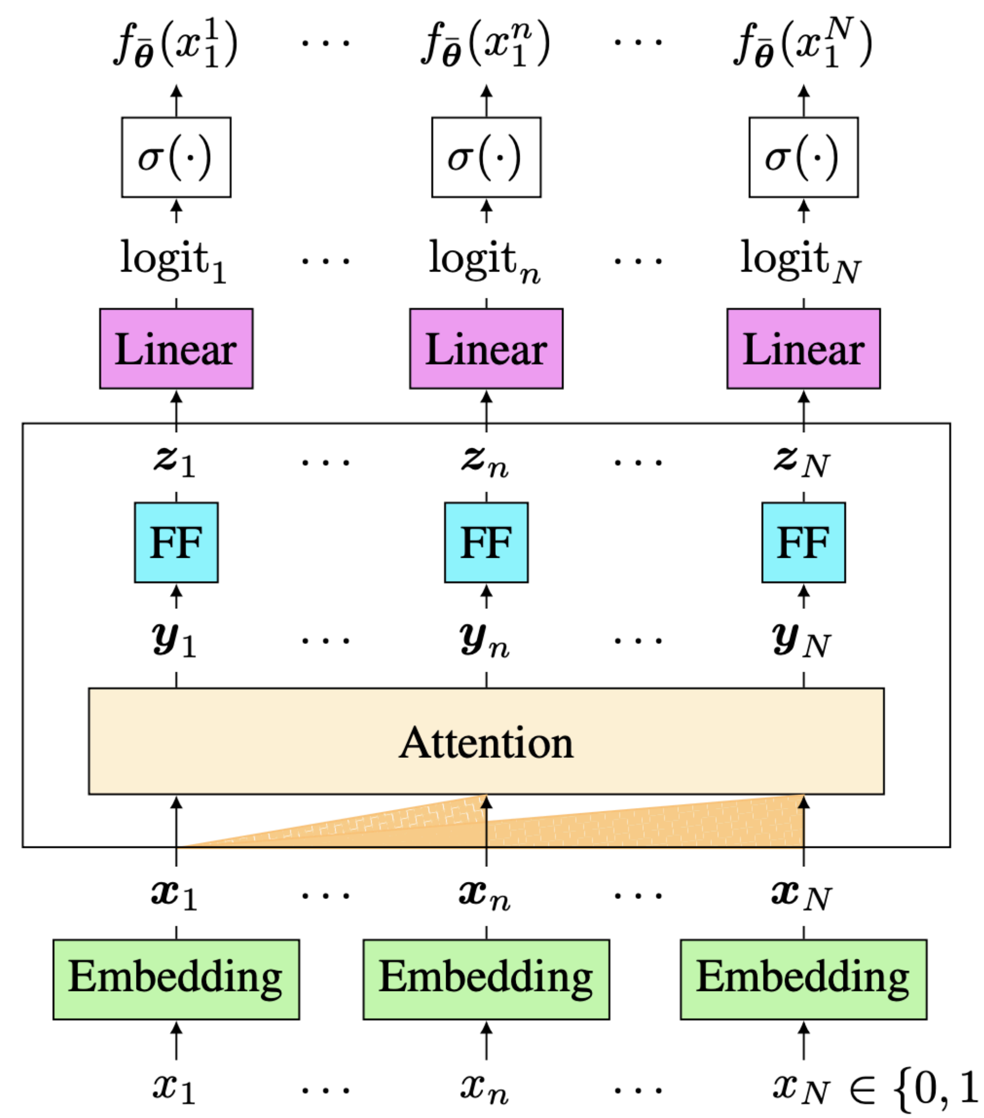
Single-layer Transformer



$\boldsymbol{\theta}$

First-order Markov chain + Single-layer transformer





Input: First-order Markov chain

Model: Depth = 1

Next-token prediction loss

Next-token Prediction Loss

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{x_1^{n+1}} \left[x_{n+1} \cdot \log f_{\boldsymbol{\theta}}(x_1^n) + (1 - x_{n+1}) \cdot \log(1 - f_{\boldsymbol{\theta}}(x_1^n)) \right]$$

Cross-entropy loss between x_{n+1} and prediction probability $f_{\boldsymbol{\theta}}(x_1^n)$

Ideally…

Prediction probability

1st-order Markov kernel

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{x_1^{n+1}}\left[x_{n+1}\cdot\log f_{\boldsymbol{\theta}}(x_1^n) + (1-x_{n+1})\cdot\log(1-f_{\boldsymbol{\theta}}(x_1^n))\right]$$

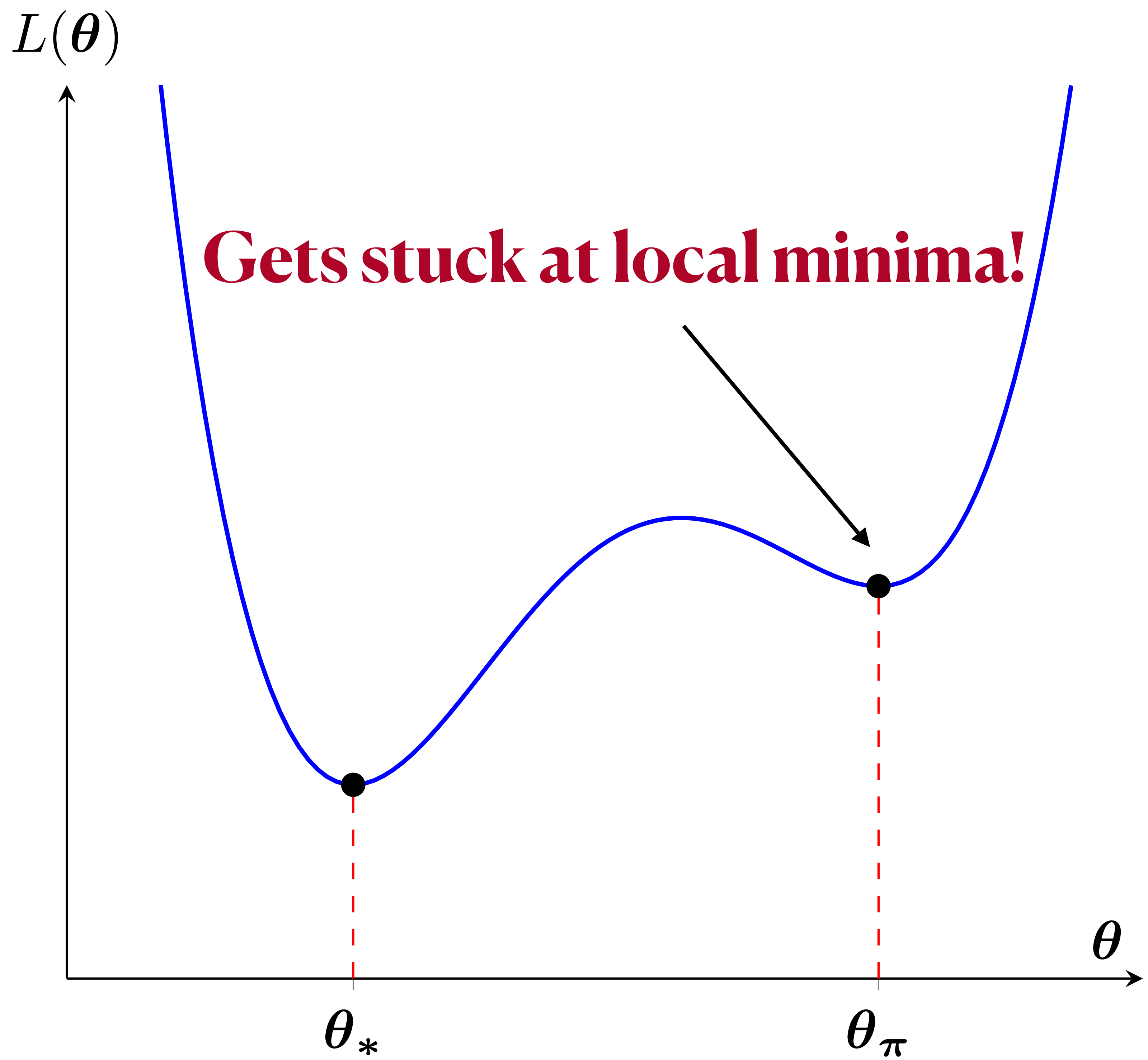
$$f_{\boldsymbol{\theta}}(x_1^n) \approx \mathbb{P}(x_{n+1} = 1 \mid x_n)$$

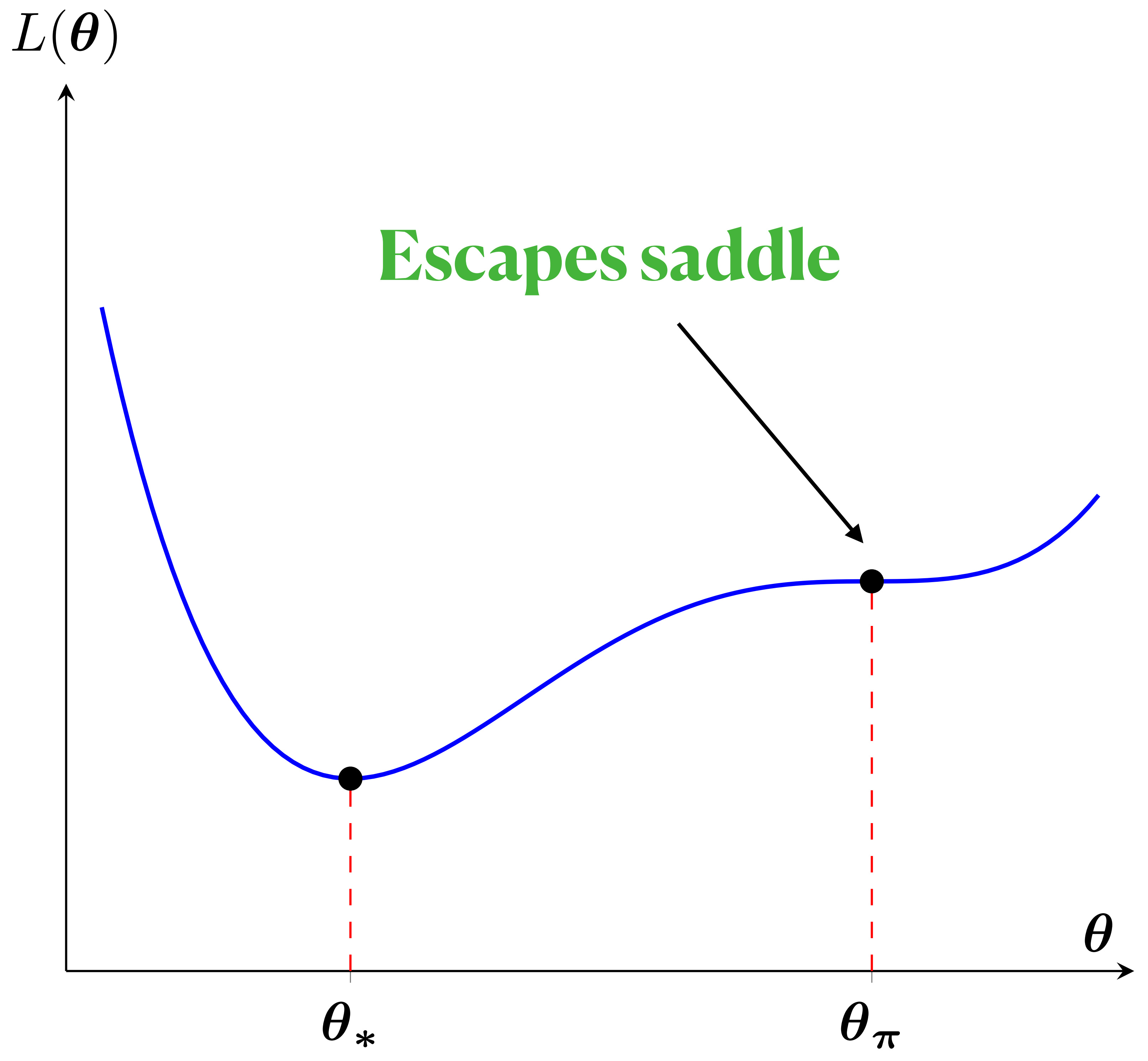
x_{n+1} $f_{\boldsymbol{\theta}}(x_1^n)$

But...

But…

Single-layer transformers sometimes fail to learn even first-order Markov chains! *

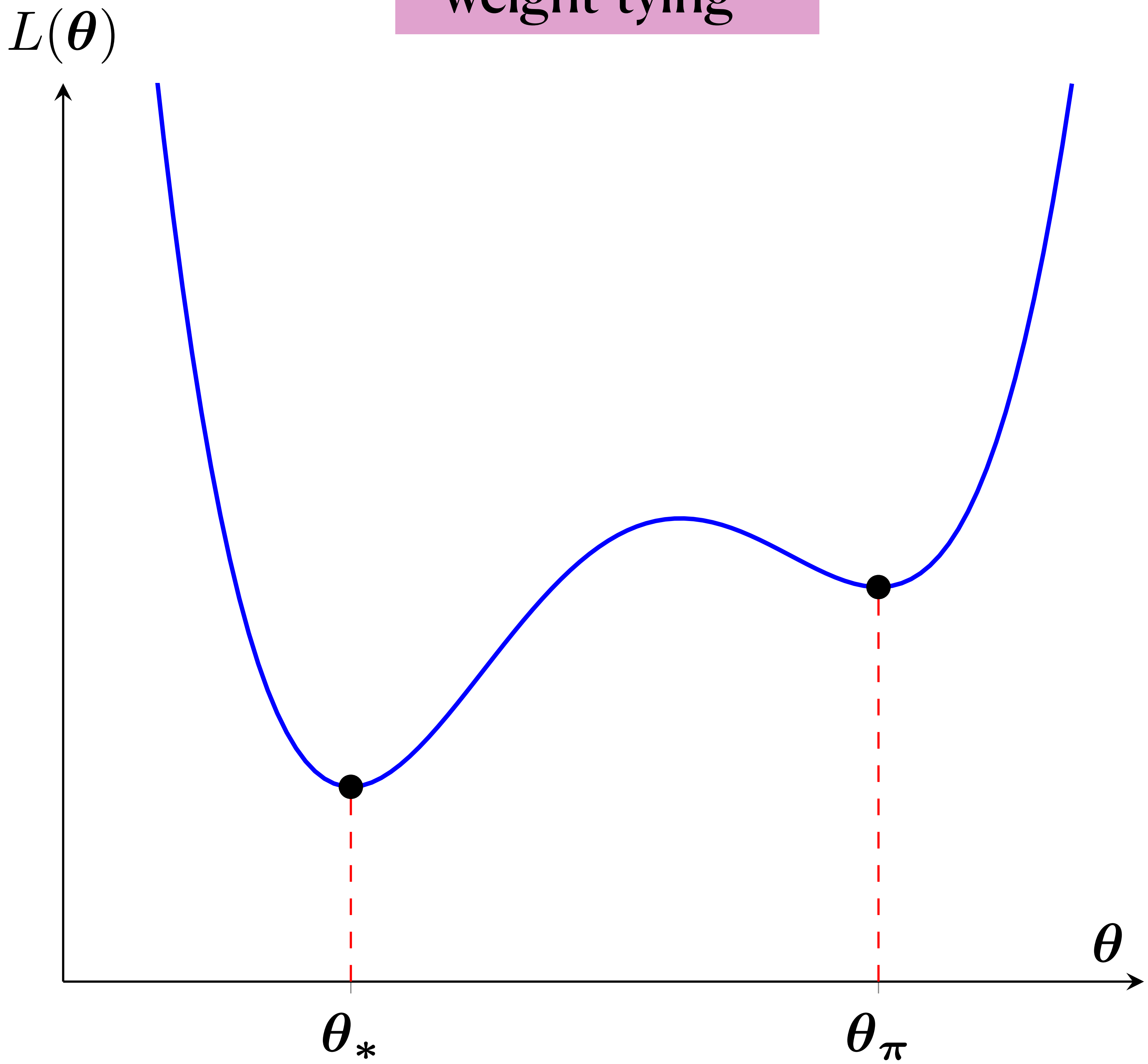

$L(\boldsymbol{\theta})$
Gets stuck at local minima!
$\boldsymbol{\theta}$
$\boldsymbol{\theta}_*$
$\boldsymbol{\theta}_\pi$

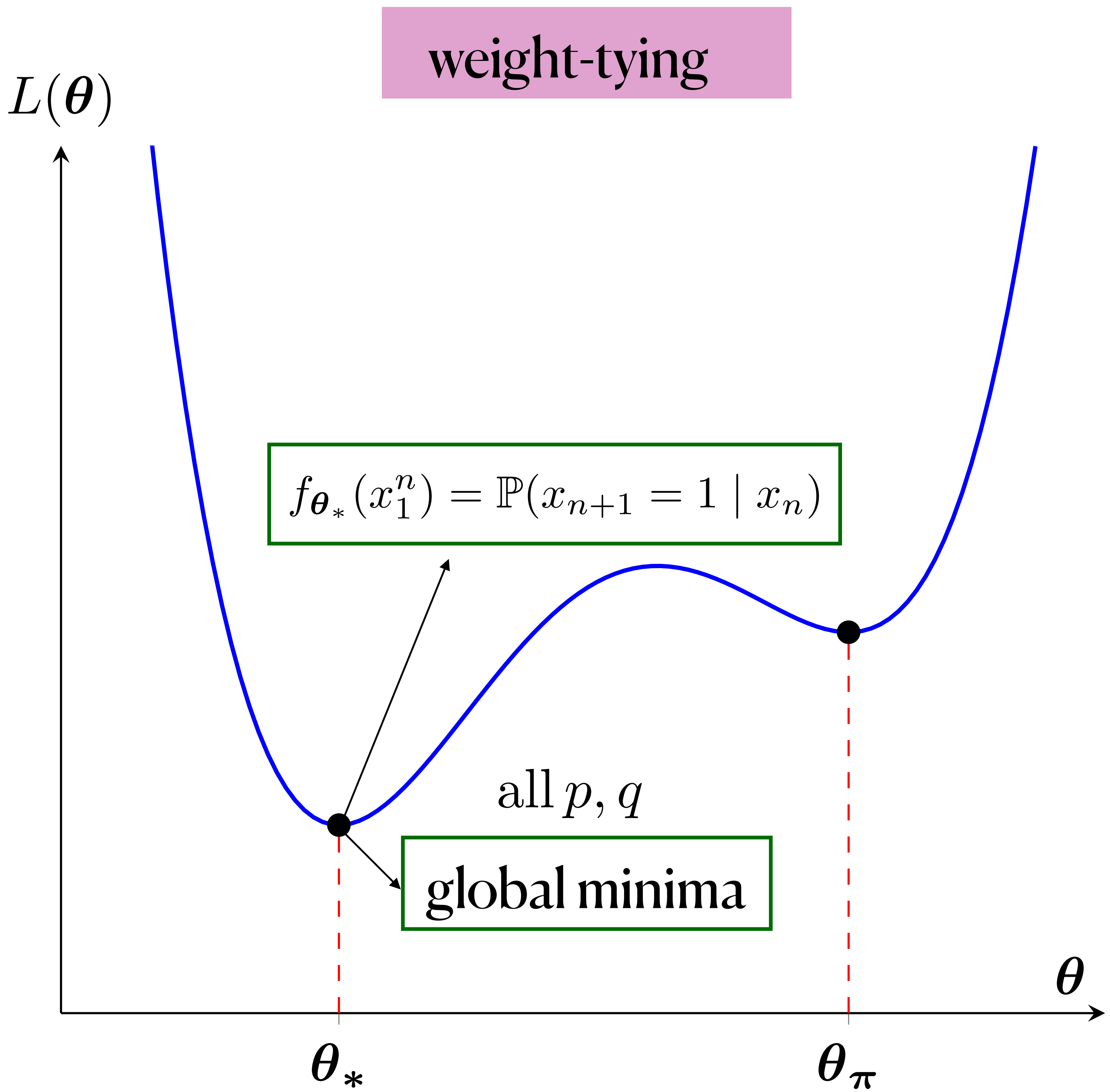


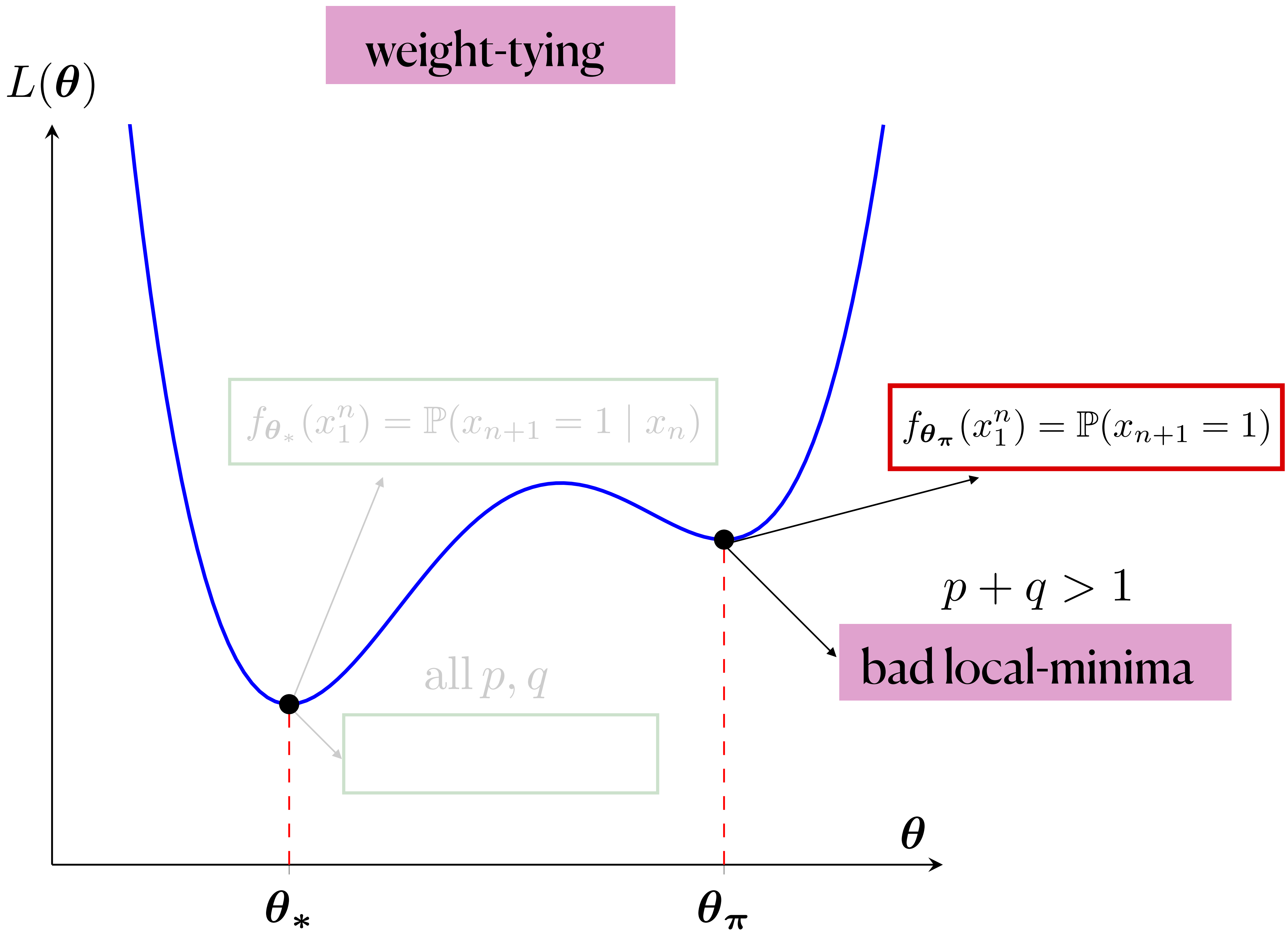
$L(\boldsymbol{\theta})$
Escapes saddle
$\boldsymbol{\theta}$
$\boldsymbol{\theta}_*$
$\boldsymbol{\theta}_{\boldsymbol{\pi}}$

weight-tying

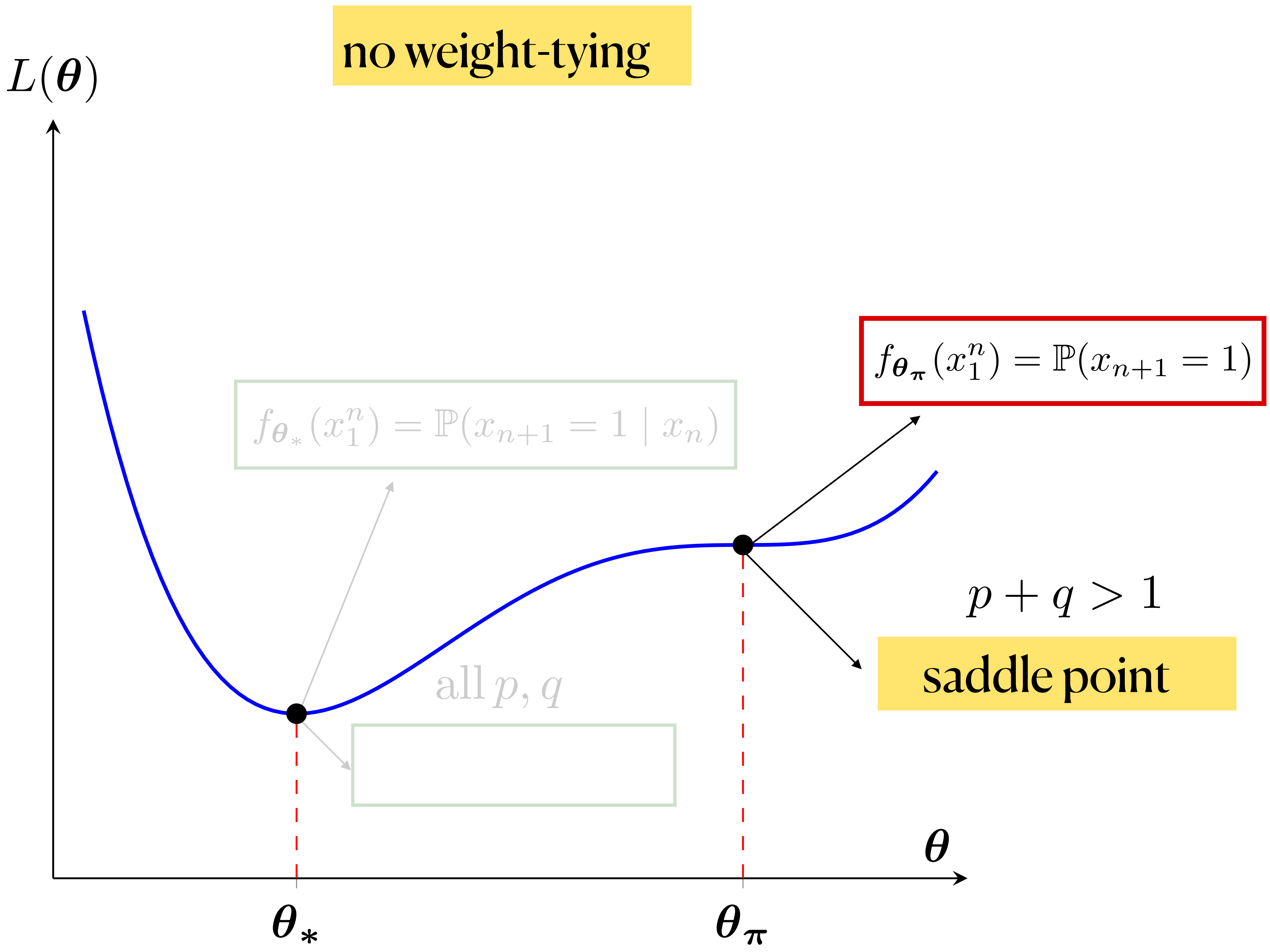
weight-tying




weight-tying
$L(\boldsymbol{\theta})$
$f_{\boldsymbol{\theta}_*}(x_1^n) = \mathbb{P}(x_{n+1} = 1 \mid x_n)$
all p, q
global minima
$\boldsymbol{\theta}$
$\boldsymbol{\theta_*}$
$\boldsymbol{\theta_\pi}$


weight-tying
$L(\boldsymbol{\theta})$
$f_{\boldsymbol{\theta}_*}(x_1^n) = \mathbb{P}(x_{n+1} = 1 \mid x_n)$
all p, q
$f_{\boldsymbol{\theta}_{\boldsymbol{\pi}}}(x_1^n) = \mathbb{P}(x_{n+1} = 1)$
$p + q > 1$
bad local-minima
$\boldsymbol{\theta}$
$\boldsymbol{\theta}_*$
$\boldsymbol{\theta}_{\boldsymbol{\pi}}$

Interestingly...



Theoretical results

Theoretical Results

Bad local minima (weight tying)

If $p + q > 1$ and the weights are tied, there exists a $\boldsymbol{\theta_\pi}$ with explicit construction such that:

(i) $\boldsymbol{\theta_\pi}$ is a bad local minima for $L(\cdot)$ with $L(\boldsymbol{\theta_\pi}) > L(\boldsymbol{\theta}_*)$

(ii) $f_{\boldsymbol{\theta_\pi}}(x_1^n) = \mathbb{P}(x_{n+1} = 1)$, the marginal distribution

(iii) $L(\boldsymbol{\theta_\pi}) = H(\boldsymbol{\pi})$, the entropy of the stationary distribution

(iv) $\nabla L(\boldsymbol{\theta_\pi}) = 0$, i.e. $\boldsymbol{\theta_\pi}$ is a stationary point

Theoretical Results

If $p + q > 1$ and the weights are tied, there exists a $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ with explicit construction such that:

(i) $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ is a bad local minima for $L(\cdot)$ with $L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) > L(\boldsymbol{\theta}_*)$

(ii) $f_{\boldsymbol{\theta}_{\boldsymbol{\pi}}}(x_1^n) = \mathbb{P}(x_{n+1} = 1)$, the marginal distribution

(iii) $L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) = H(\boldsymbol{\pi})$, the entropy of the stationary distribution

(iv) $\nabla L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) = 0$, i.e. $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ is a stationary point

Saddle point (no weight tying)

Under the same setting as above with the weights not tied, $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ becomes a saddle point. It satisfies the same properties.

Theoretical Results

Bad local minima (weight tying)

If $p + q > 1$ and the weights are tied, there exists a $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ with explicit construction such that:

(i) $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ is a bad local minima for $L(\cdot)$ with $L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) > L(\boldsymbol{\theta}_*)$

(ii) $f_{\boldsymbol{\theta}_{\boldsymbol{\pi}}}(x_1^n) = \mathbb{P}(x_{n+1} = 1)$, the marginal distribution

(iii) $L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) = H(\boldsymbol{\pi})$, the entropy of the stationary distribution

(iv) $\nabla L(\boldsymbol{\theta}_{\boldsymbol{\pi}}) = 0$, i.e. $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ is a stationary point

Saddle point (no weight tying)

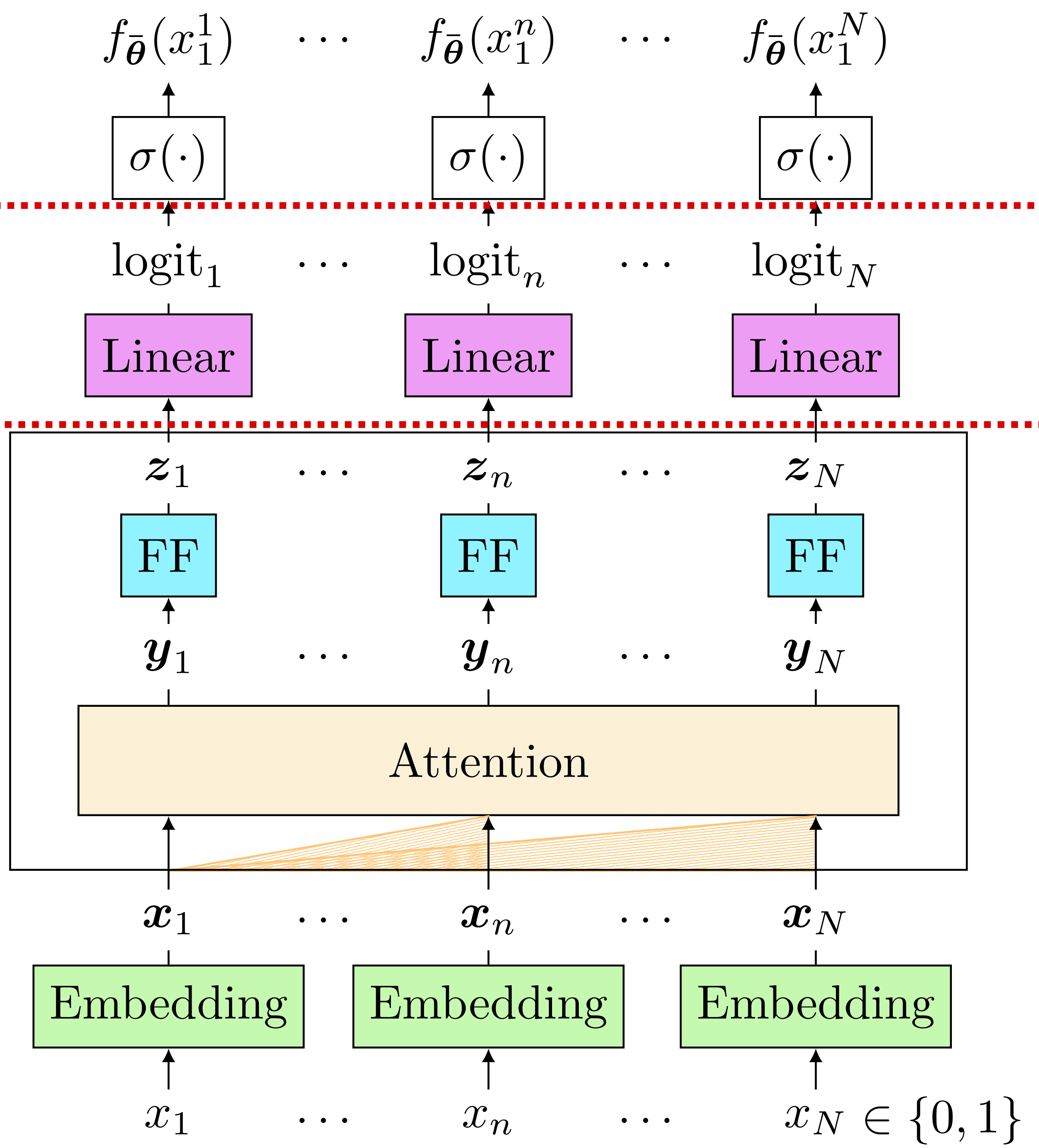
Under the same setting as above with the weights not tied, $\boldsymbol{\theta}_{\boldsymbol{\pi}}$ becomes a saddle point. It satisfies the same properties.

Intuition

$$f_{\boldsymbol{\theta}}(x_1^n) = \sigma(b) = \pi_1$$

$\boldsymbol{a} = 0$

$$\text{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$



Intuition

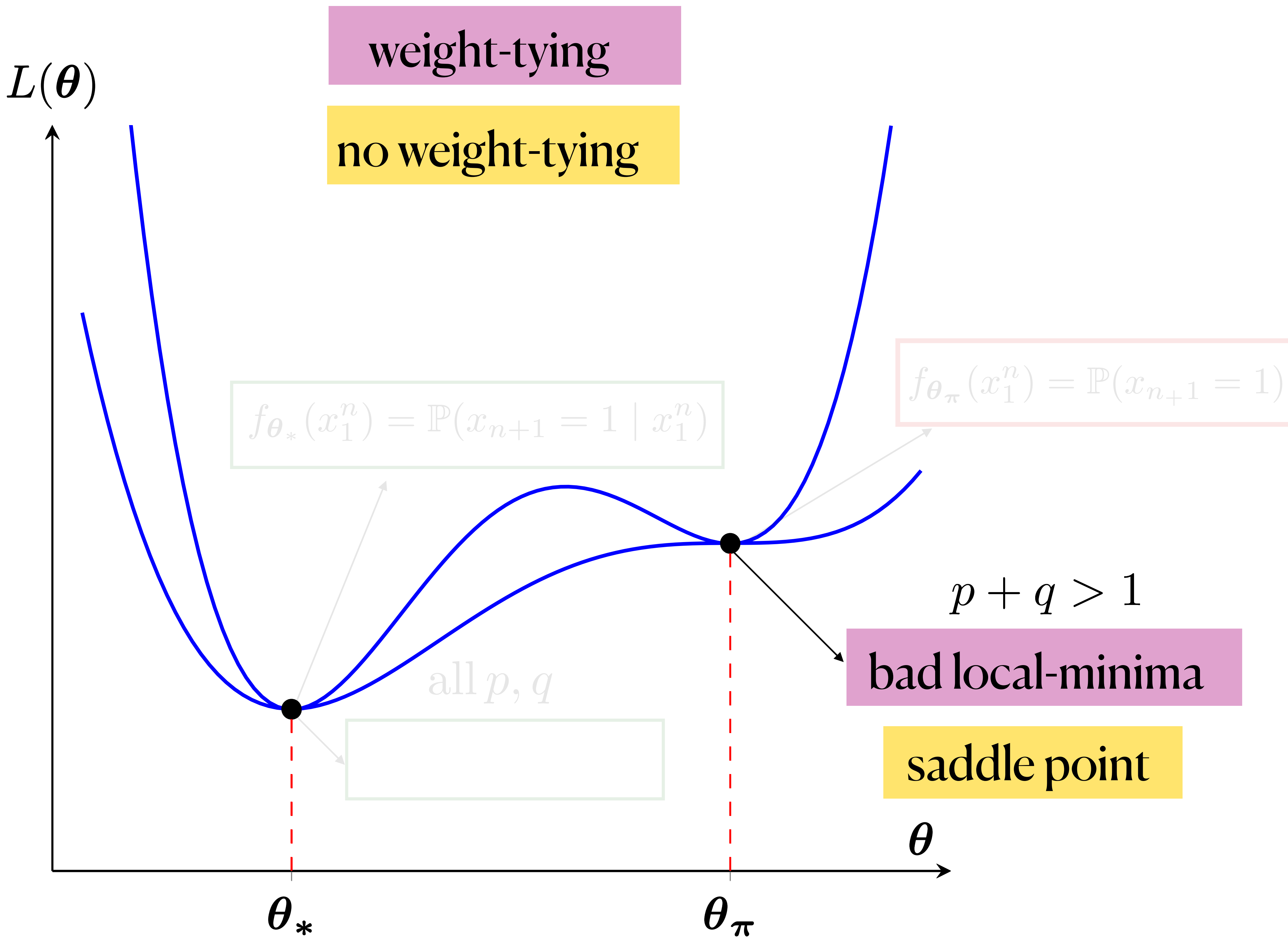
weight-tying

Hessian is (almost) positive-definite

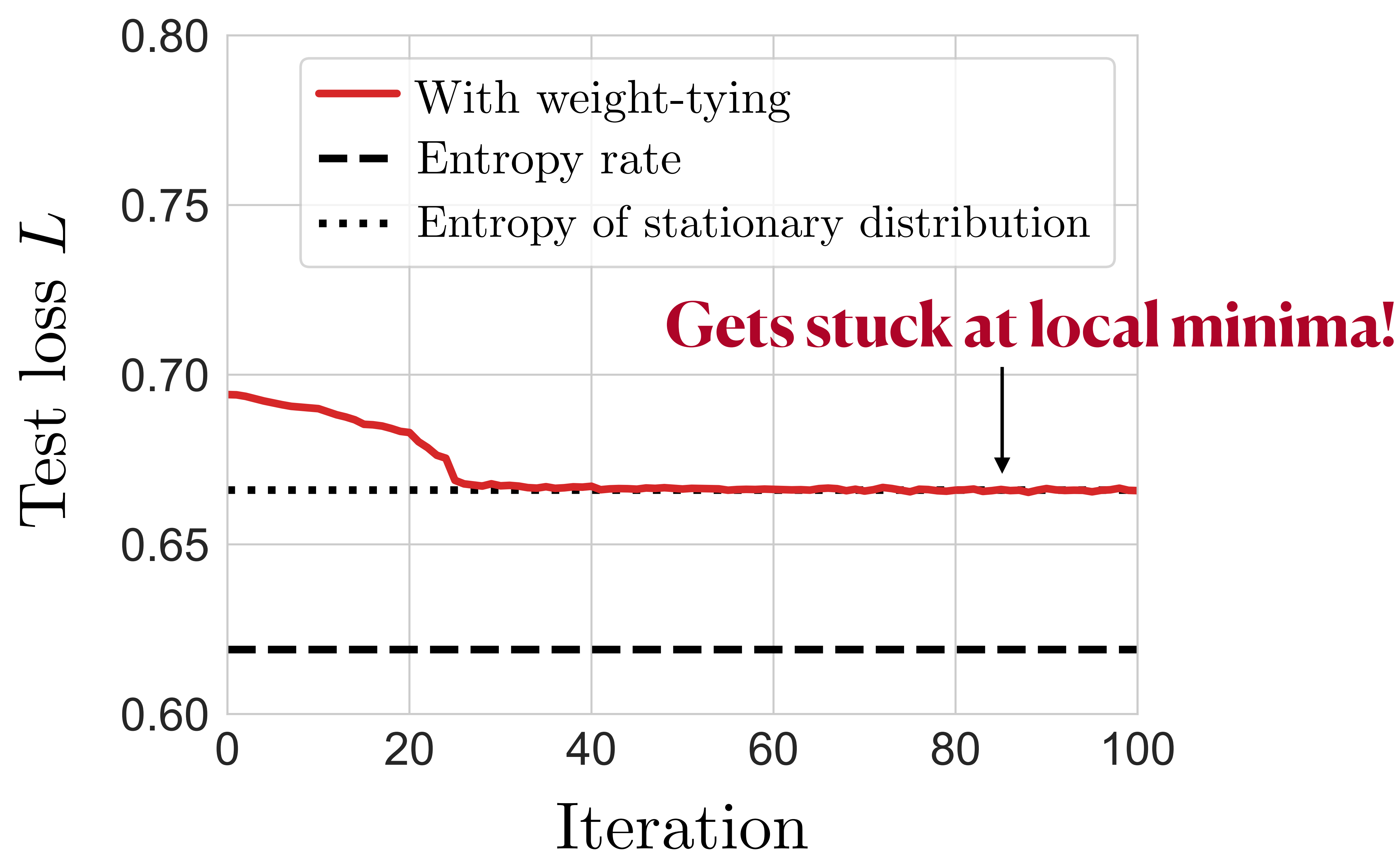
no weight-tying

Hessian is indefinite

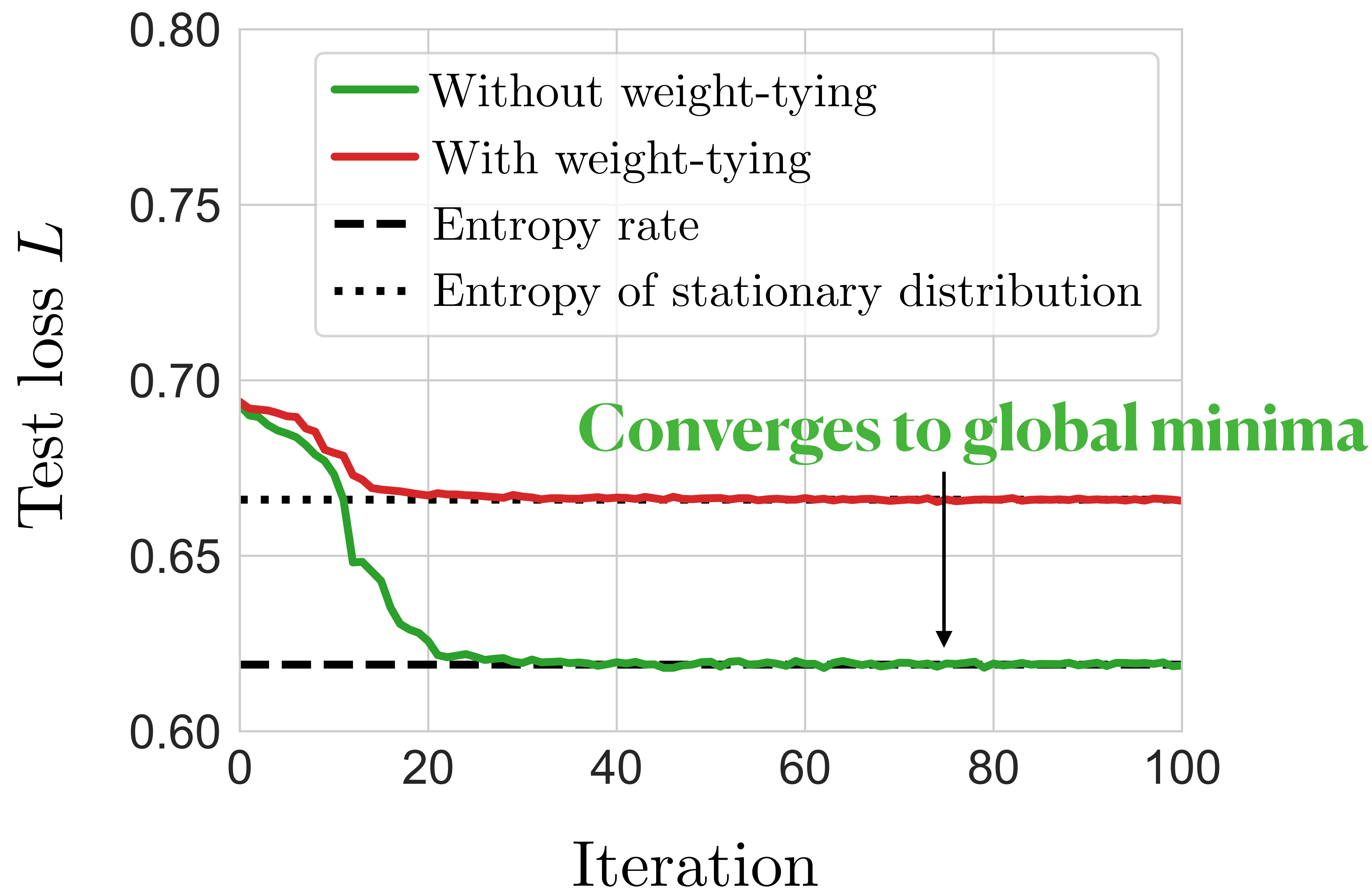
Main Results

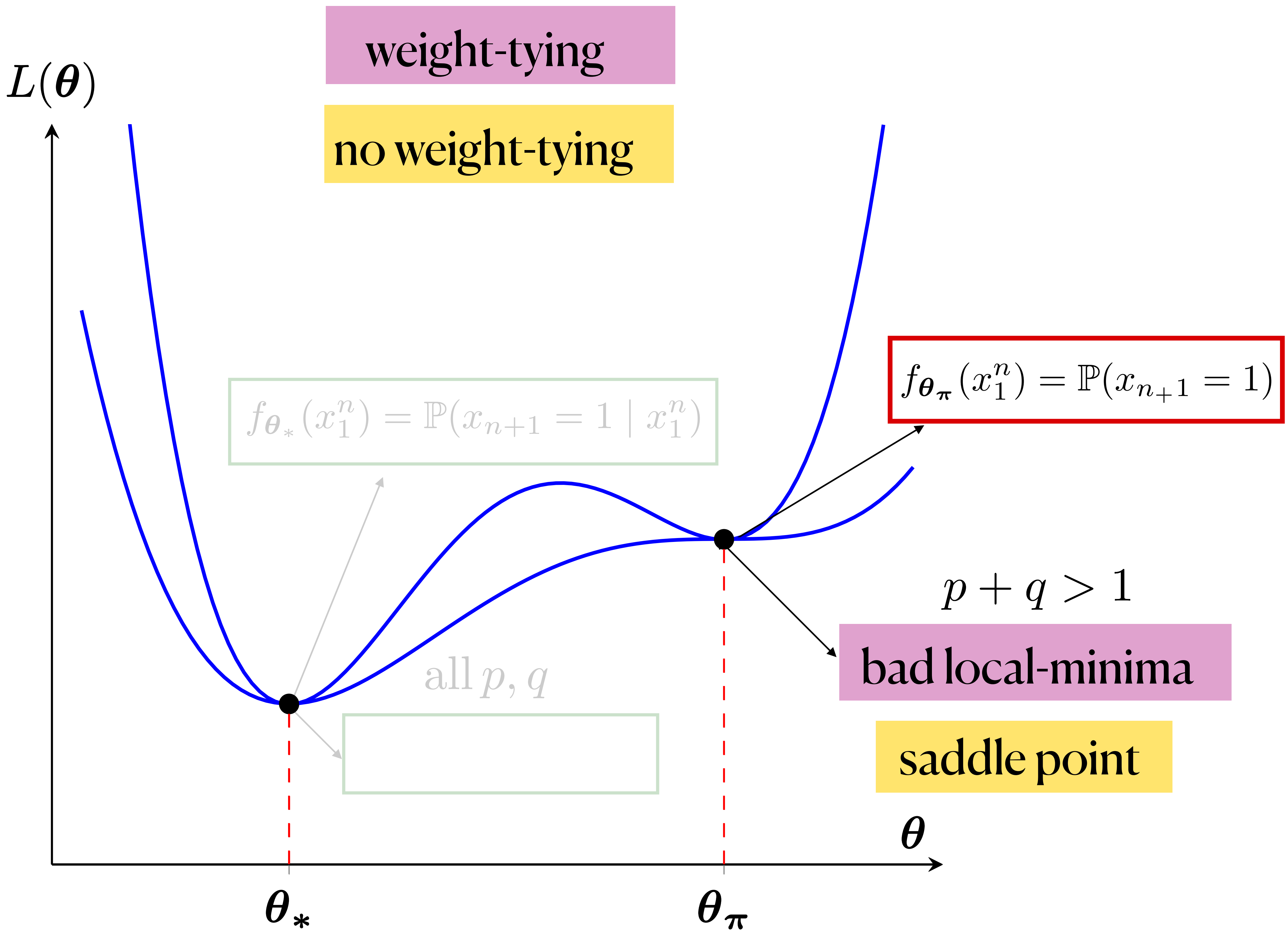


Weight tying

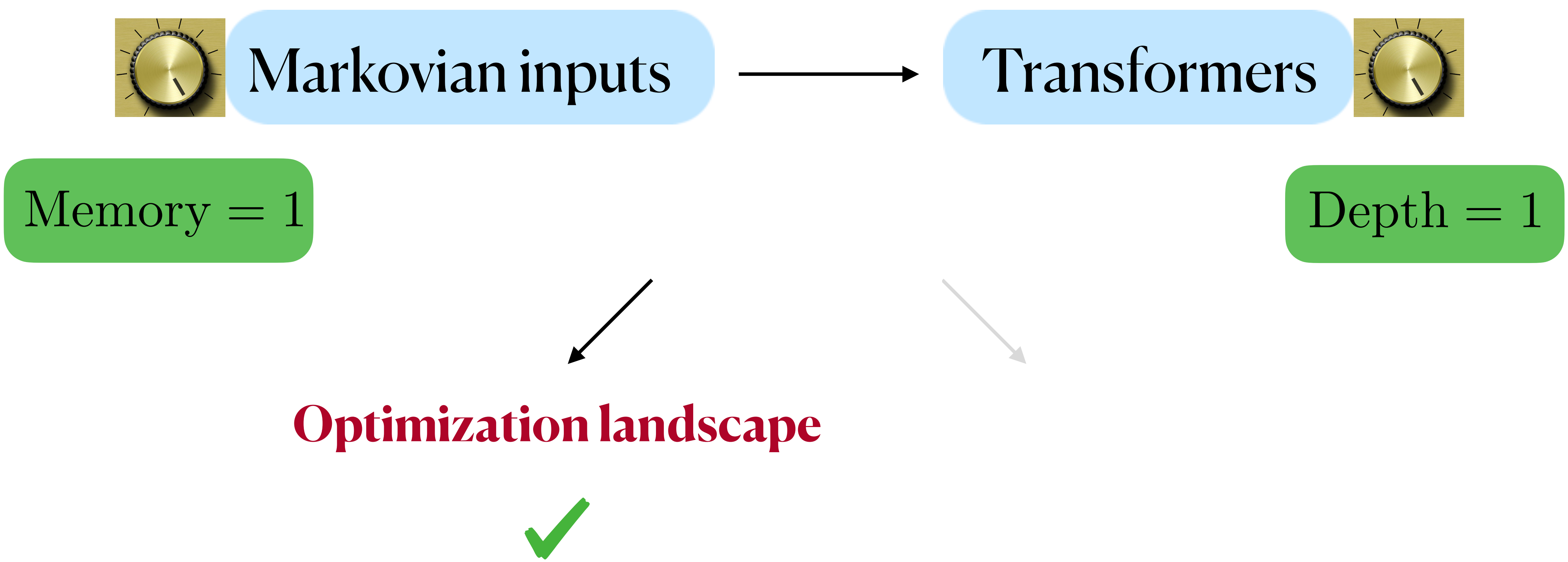


Without Weight tying

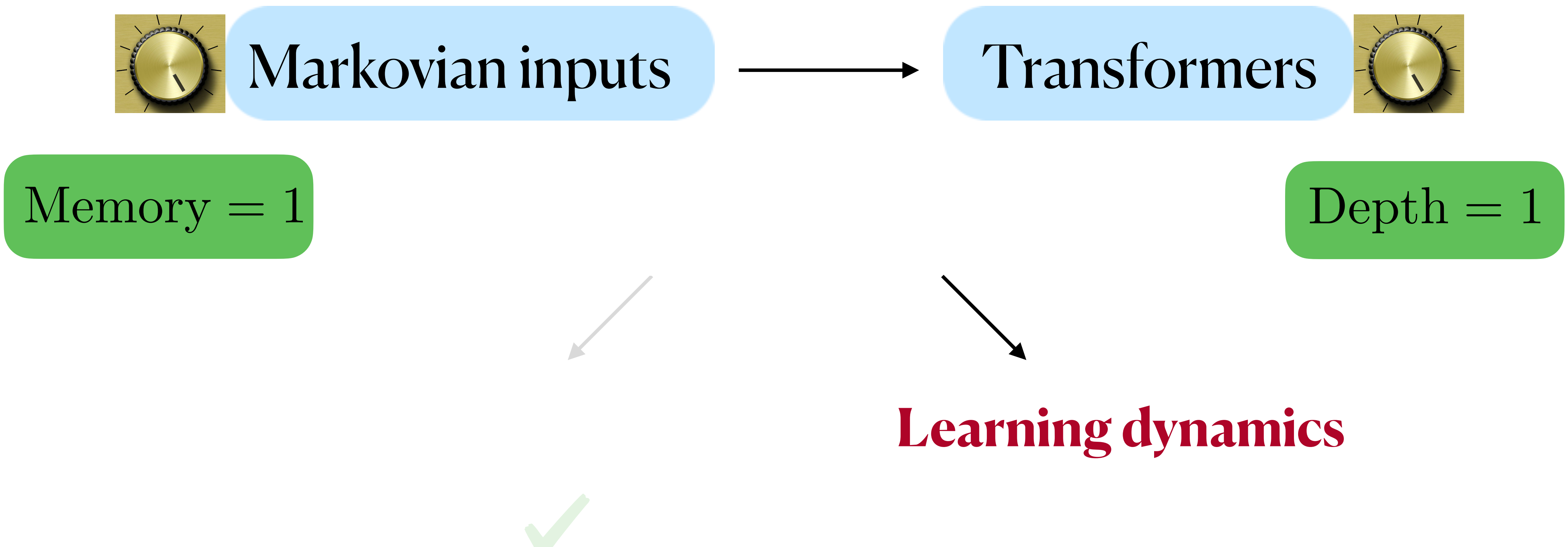


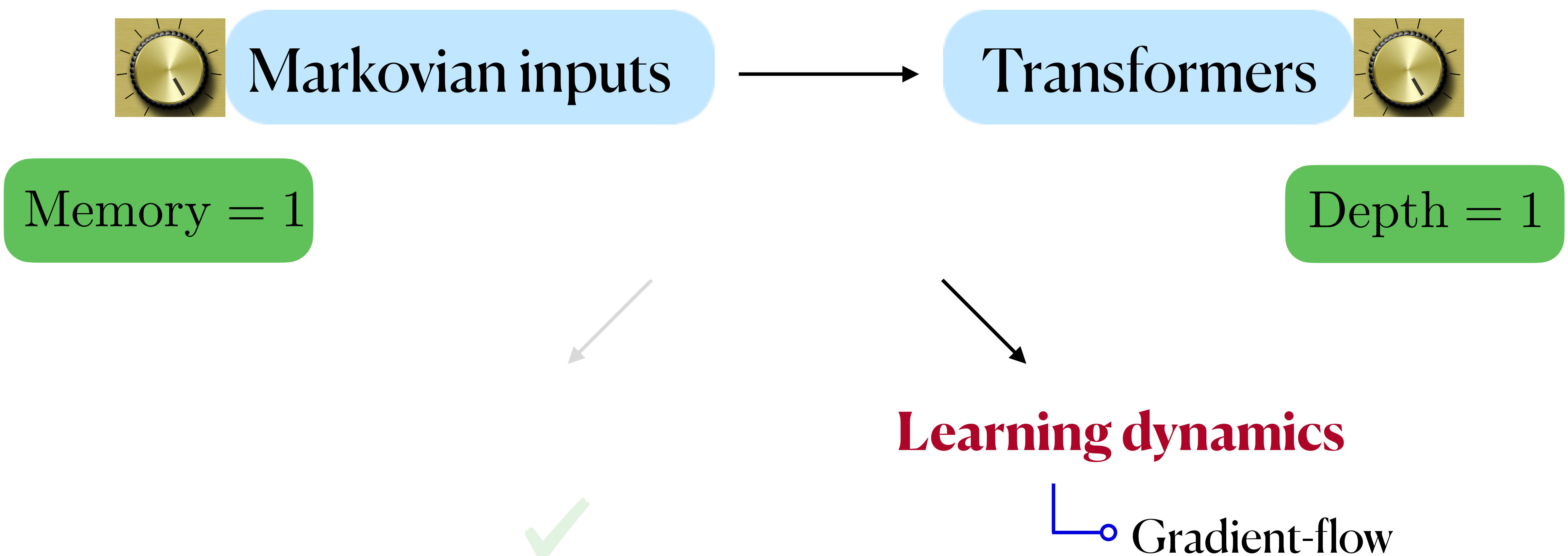


$L(\boldsymbol{\theta})$
weight-tying
no weight-tying
$f_{\boldsymbol{\theta}_*}(x_1^n) = \mathbb{P}(x_{n+1} = 1 \mid x_1^n)$
$f_{\boldsymbol{\theta}_{\boldsymbol{\pi}}}(x_1^n) = \mathbb{P}(x_{n+1} = 1)$
$p + q > 1$
all p, q
bad local-minima
saddle point
$\boldsymbol{\theta}$
$\boldsymbol{\theta}_*$
$\boldsymbol{\theta}_{\boldsymbol{\pi}}$



Markovian inputs
Transformers
Memory = 1
Depth = 1
Optimization landscape


Markovian inputs
Transformers
Memory = 1
Depth = 1
Learning dynamics



Markovian inputs
Transformers
Memory = 1
Depth = 1
Learning dynamics
Gradient-flow

Gradient Flow

Gradient Flow

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\nabla L(\boldsymbol{\theta}_t)$$

Gradient Flow

Transformer parameters

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\nabla L(\boldsymbol{\theta}_t)$$

Next-token prediction loss

Recall

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\text{logit}_n)$$

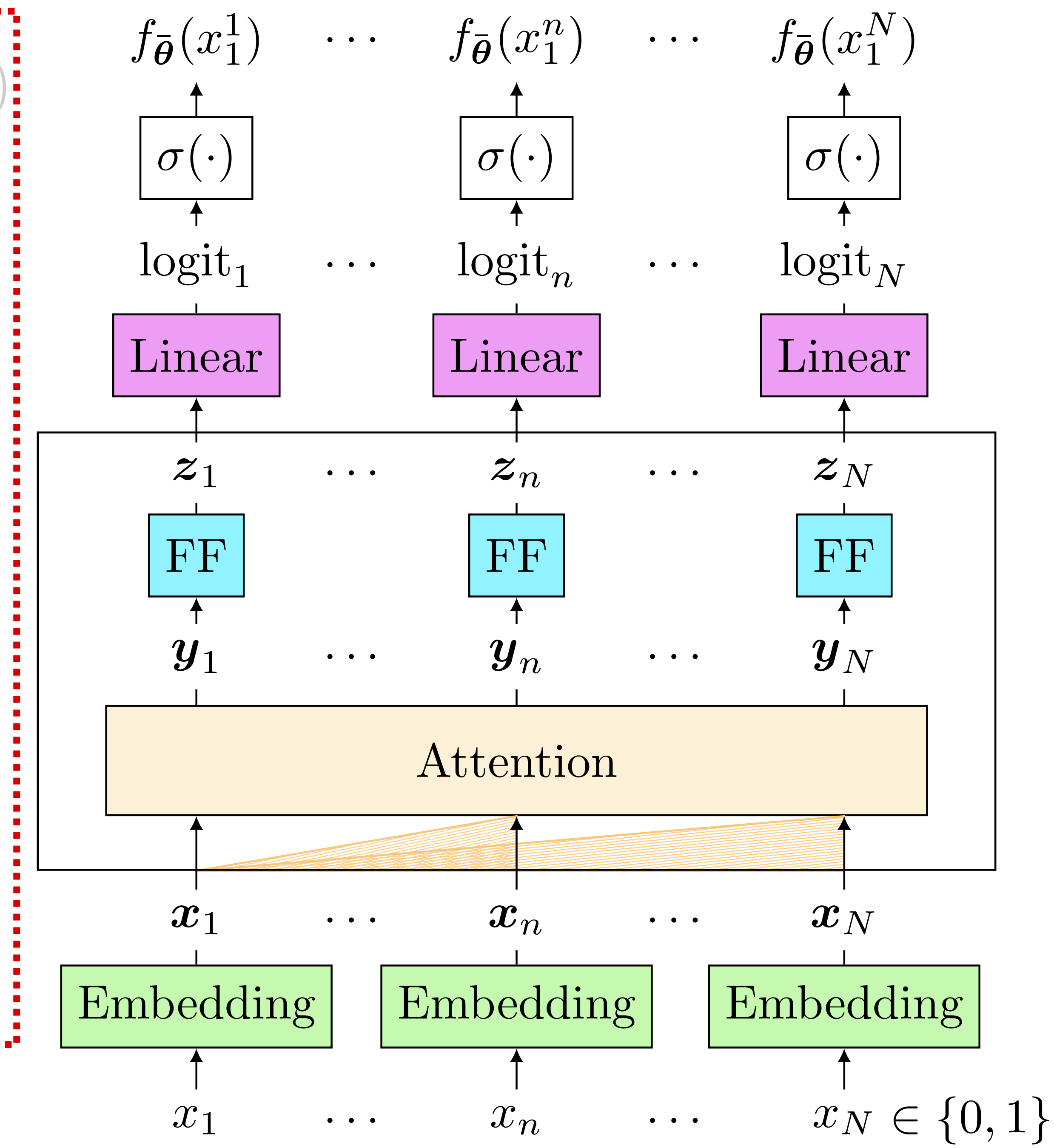
$$\text{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \, \text{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \text{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

$\boldsymbol{\theta}$



Low-rank Structure

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\text{logit}_n)$$

$$\text{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \, \text{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \text{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

Reparametrization

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\mathrm{logit}_n)$$

$$\mathrm{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

w

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \,\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \mathrm{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

a

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

e

Reparametrization

$$f_{\boldsymbol{\theta}}(x_1^n) = \mathbb{P}_{\boldsymbol{\theta}}(x_{n+1} = 1 \mid x_1^n) = \sigma(\mathrm{logit}_n)$$

$$\mathrm{logit}_n = \langle \boldsymbol{a}, \boldsymbol{z}_n \rangle + b \in \mathbb{R}$$

w

$$\boldsymbol{z}_n = \boldsymbol{y}_n + \boldsymbol{W}_2 \,\mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{y}_n)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n + \boldsymbol{W}_O \sum_{i=1}^{n} \mathrm{att}_{n,i} \cdot \boldsymbol{W}_V \boldsymbol{x}_i$$

a

$$\boldsymbol{x}_n = x_n \cdot \boldsymbol{e} + \boldsymbol{p}_n$$

e

$$\boldsymbol{\theta} = (e, w, a) \in \mathbb{R}^3$$

Gradient Flow

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\nabla L(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_t = (e_t, w_t, a_t) \in \mathbb{R}^3$$

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\nabla L(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_t = (e_t, w_t, a_t) \in \mathbb{R}^3$$

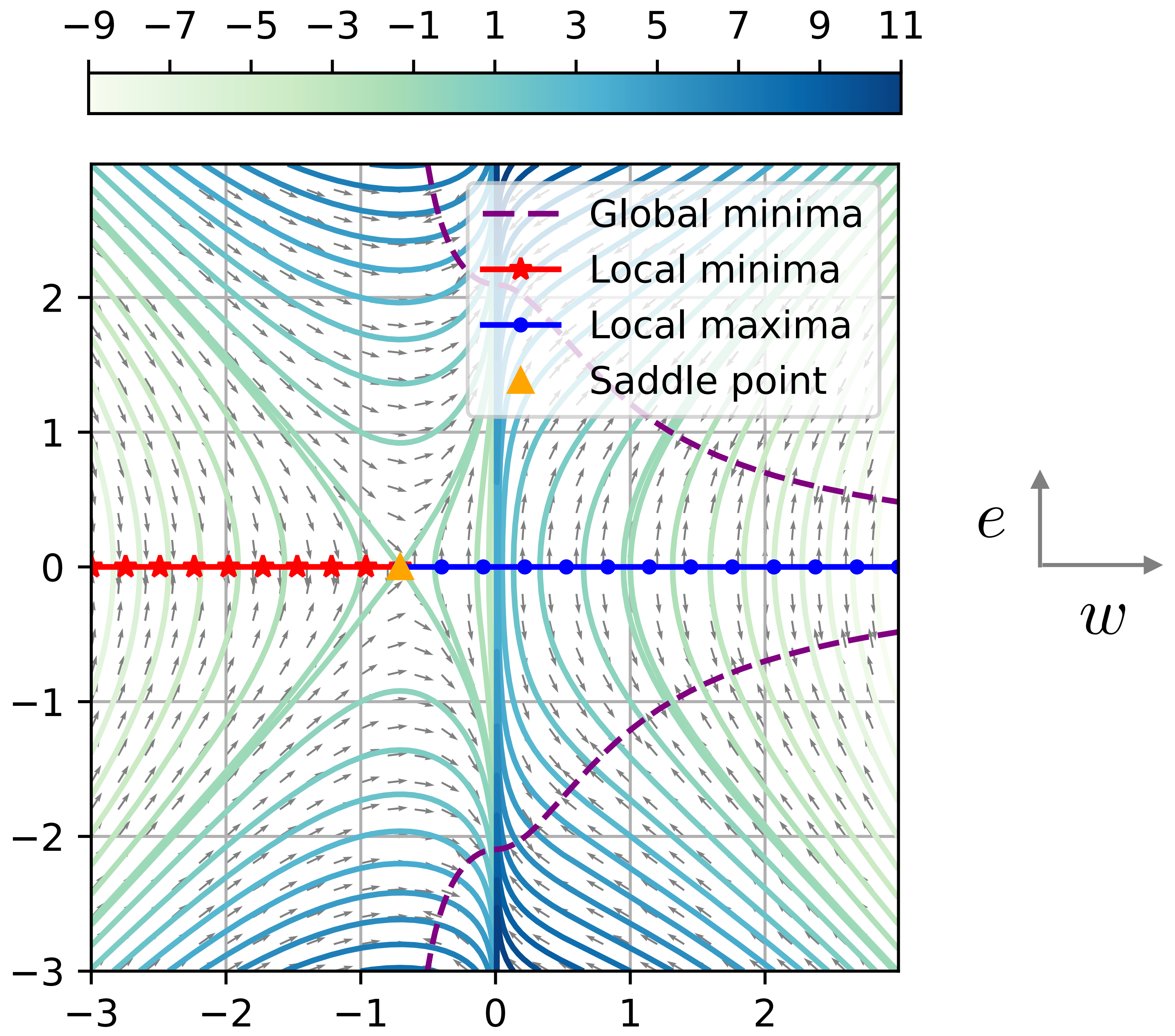
How does the flow look like?

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = -\nabla L(\boldsymbol{\theta}_t), \quad \boldsymbol{\theta}_t = (e_t, w_t, a_t) \in \mathbb{R}^3$$

How does the flow look like?

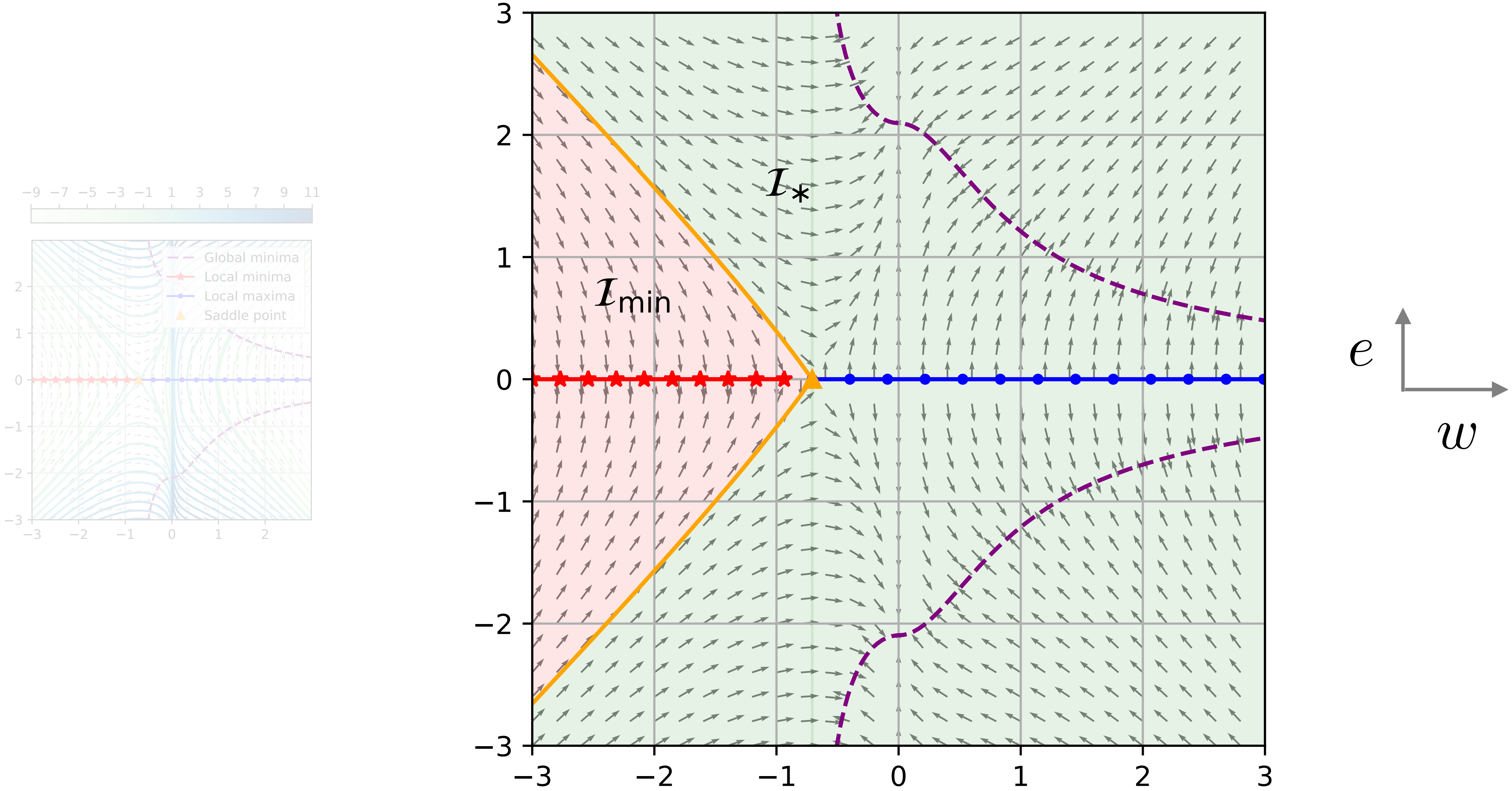
- $a = 0 \rightarrow \boldsymbol{\theta} = (e, w)$

p+q < 1



−9
−7
−5
−3
−1
1
3
5
7
9
11
Global minima
Local minima
Local maxima
Saddle point
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
e
w

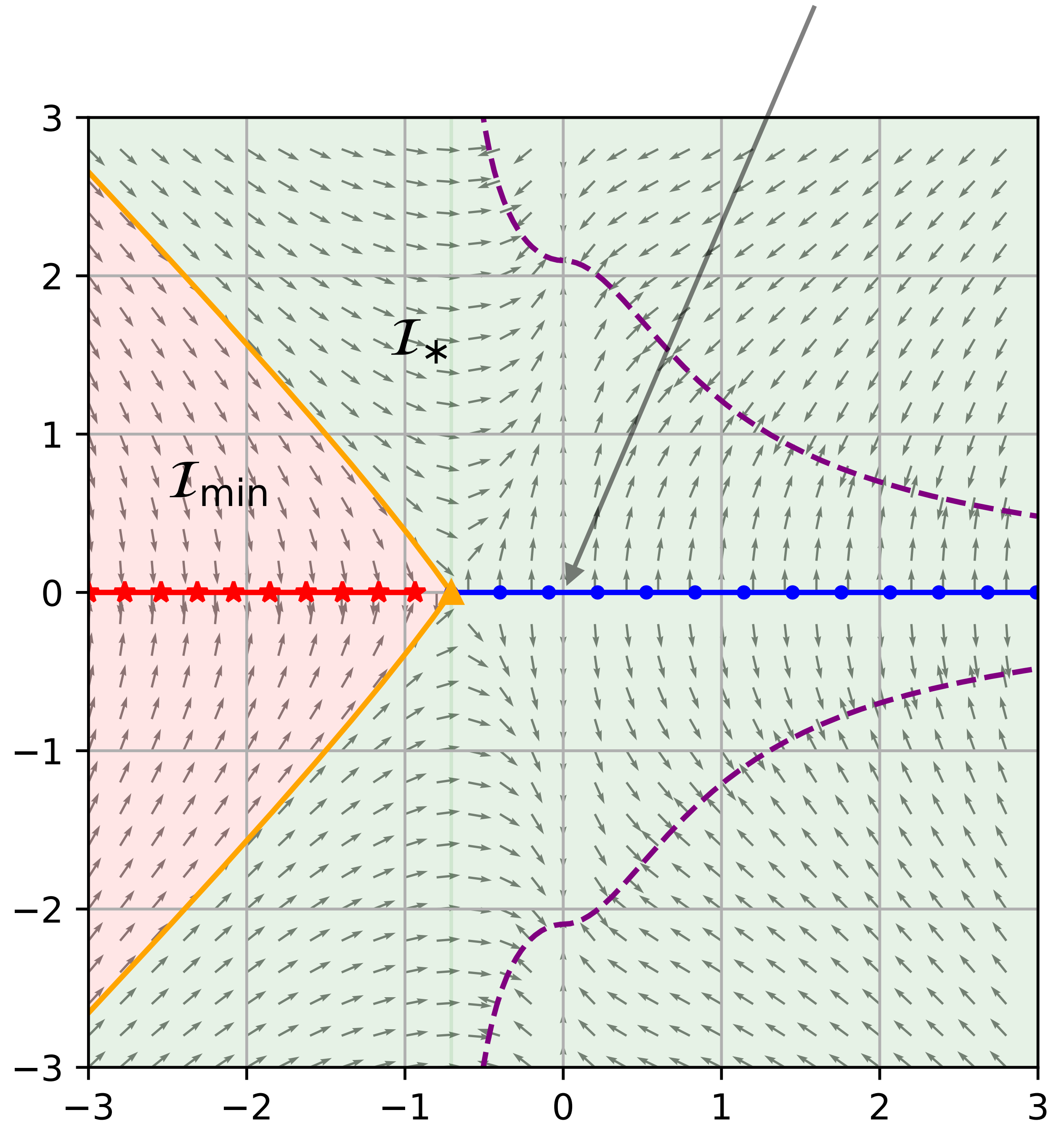
$\mathbf{p+q<1}$



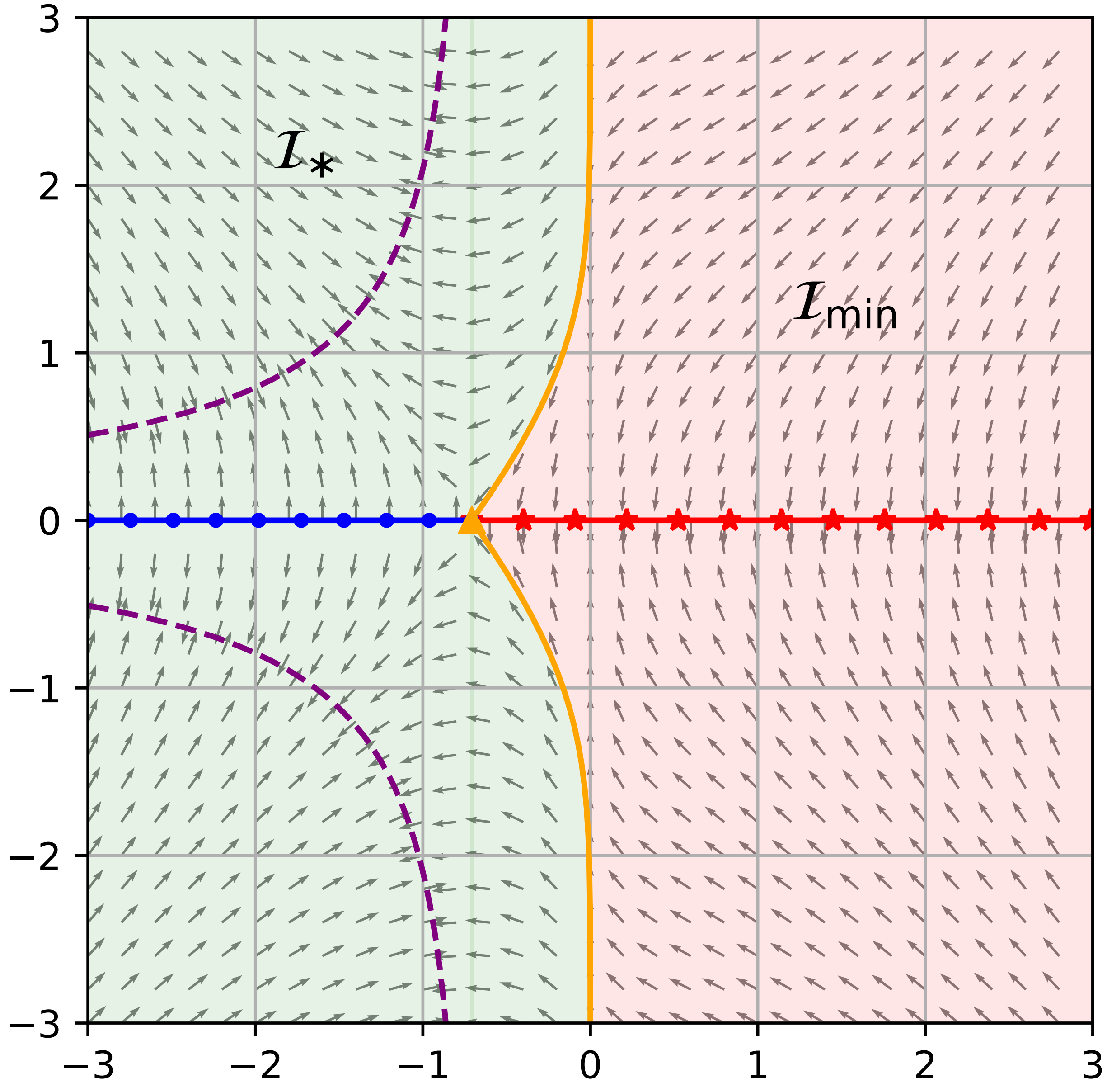
−9 −7 −5 −3 −1 1 3 5 7 9 11
Global minima
Local minima
Local maxima
Saddle point
$\mathcal{I}_*$
$\mathcal{I}_{\text{min}}$
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
e
w

$p+q<1$

Gaussian init. converges to global minima

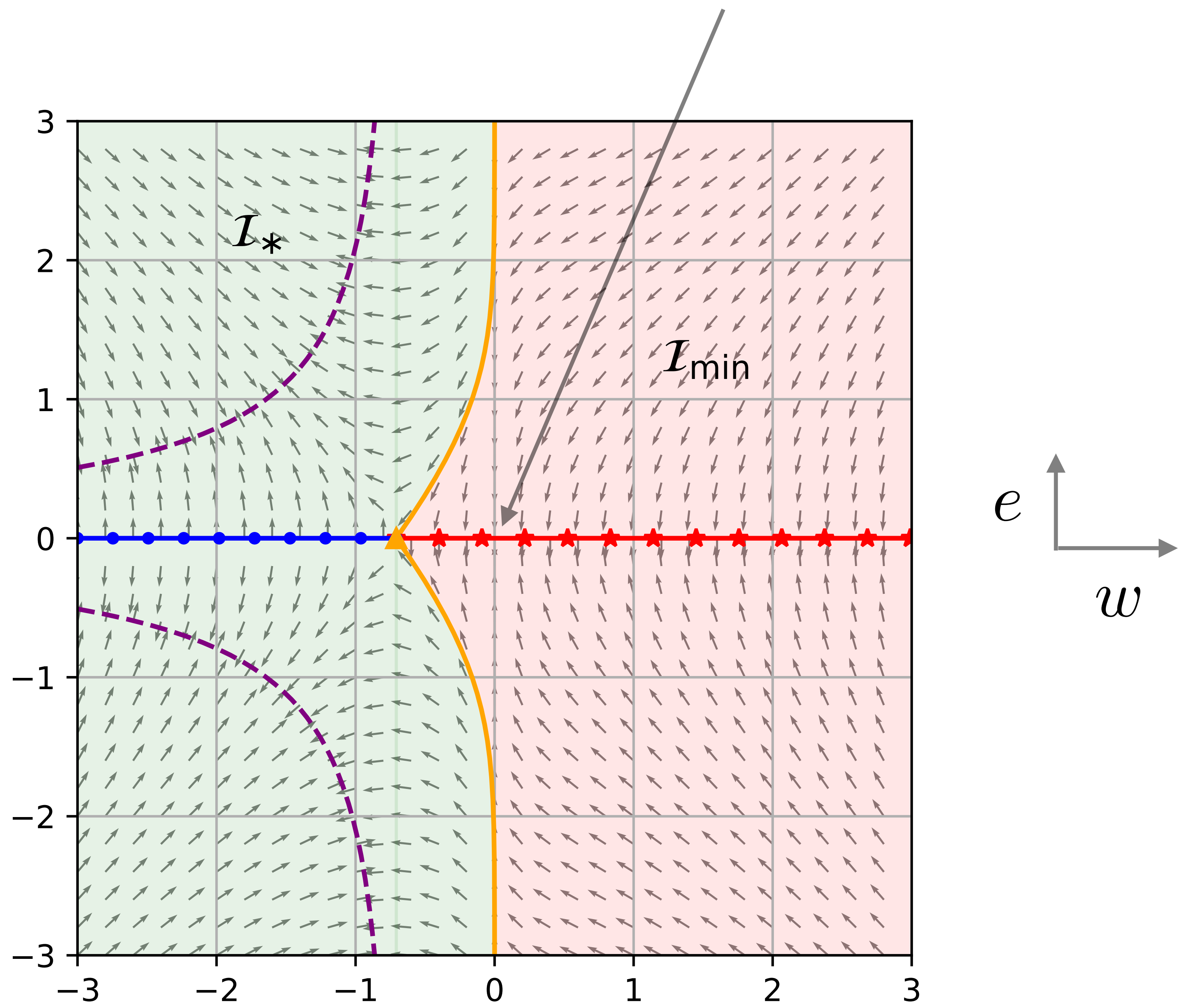


p+q > 1



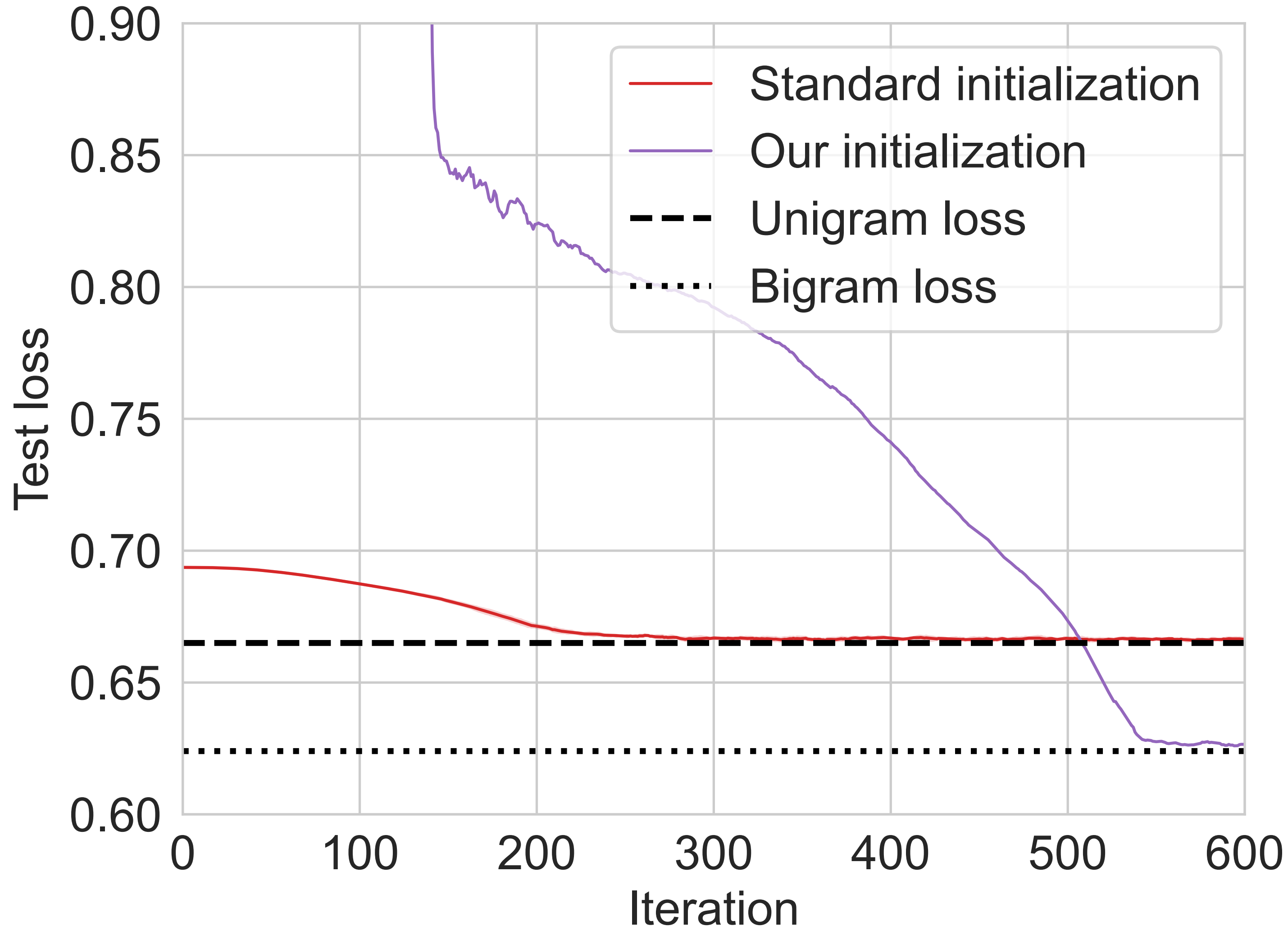
$p+q > 1$

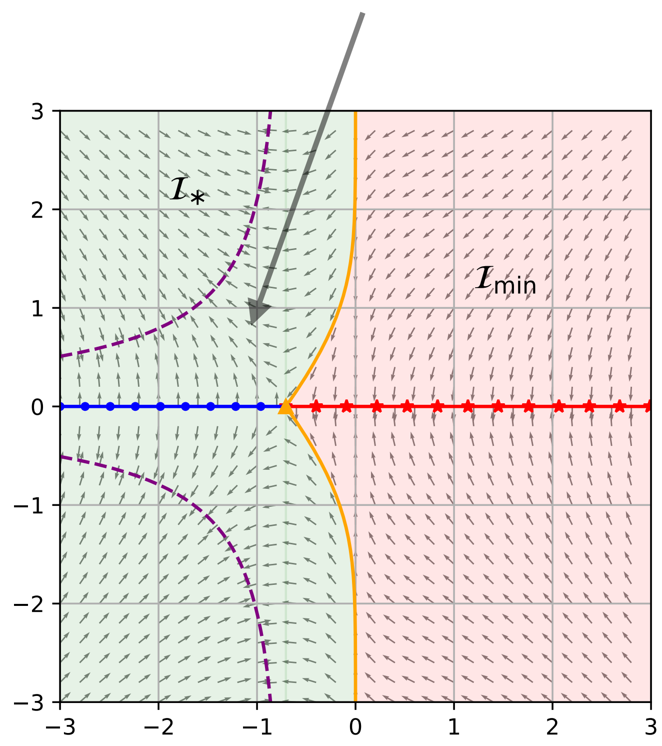
Gets stuck at local minima!



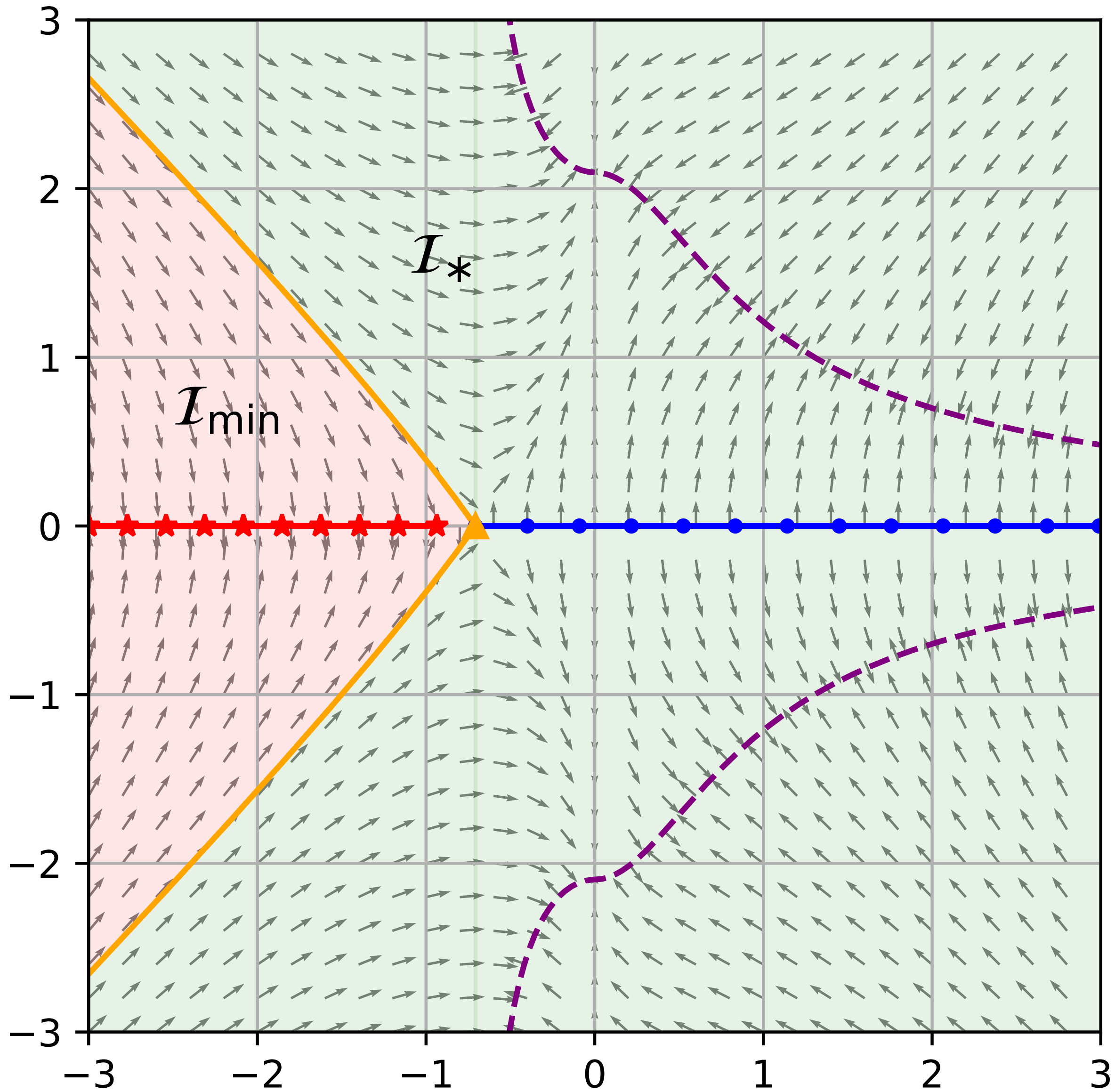
Can we escape it?

Initialize here



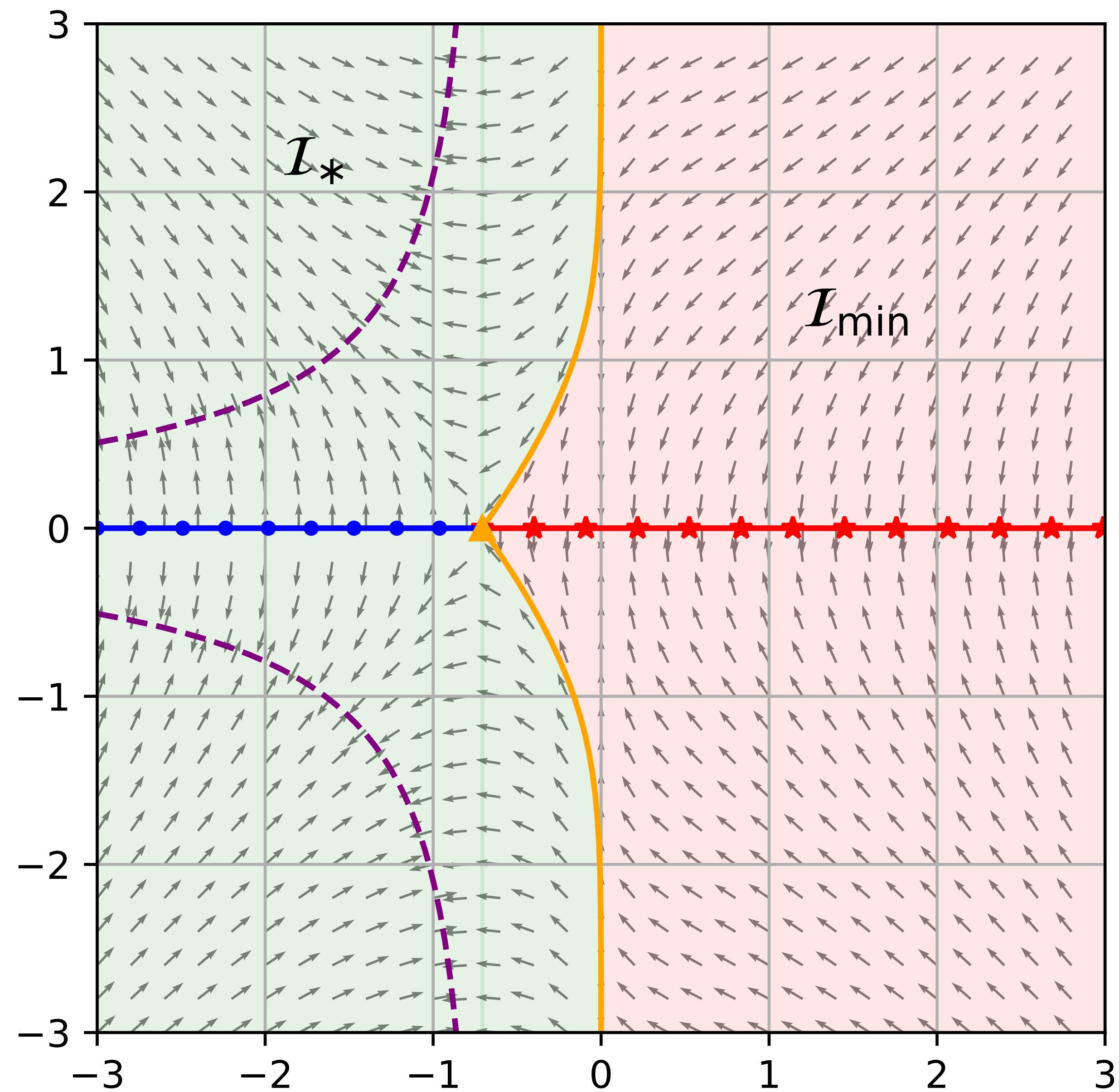


Converges to global minima

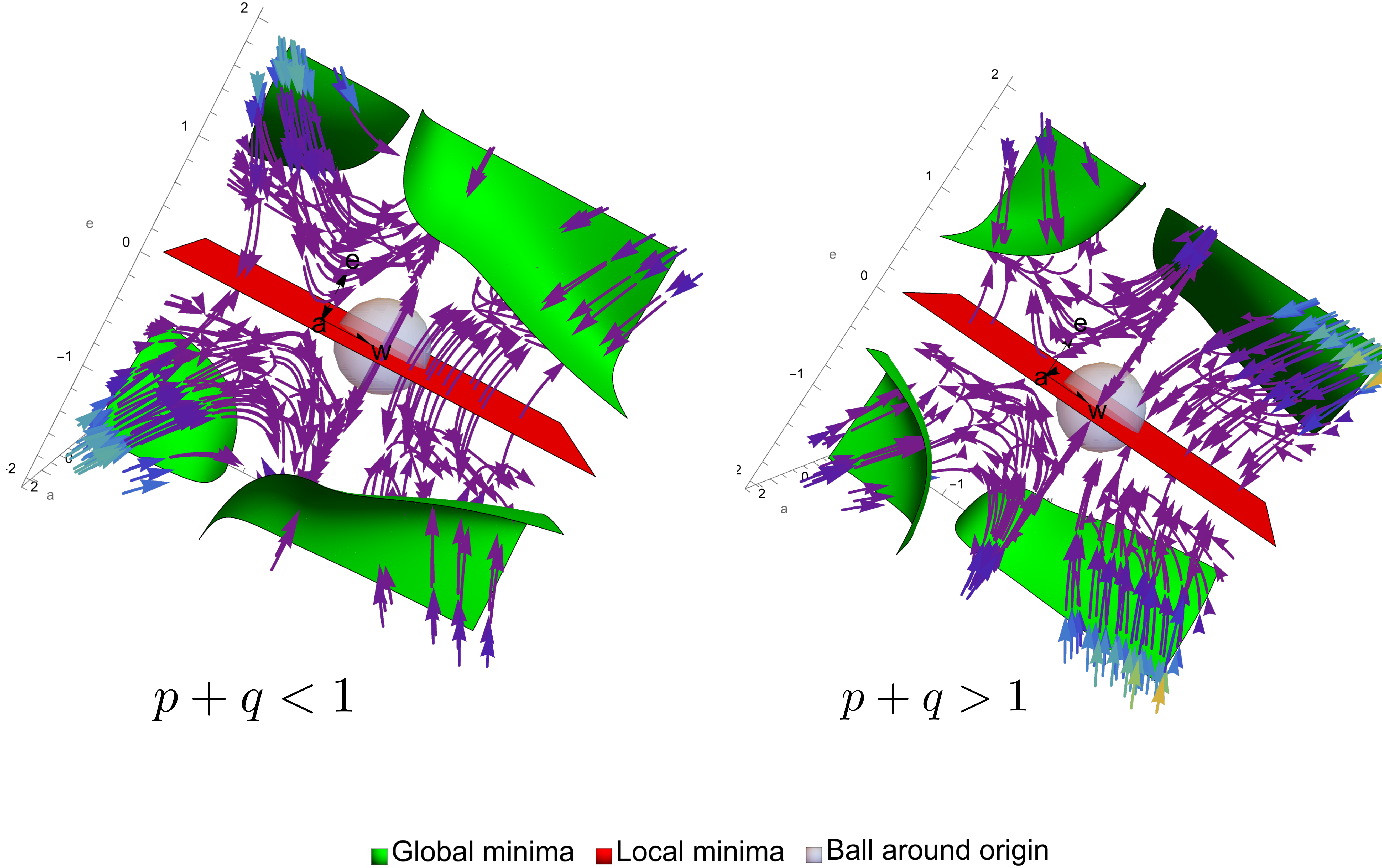


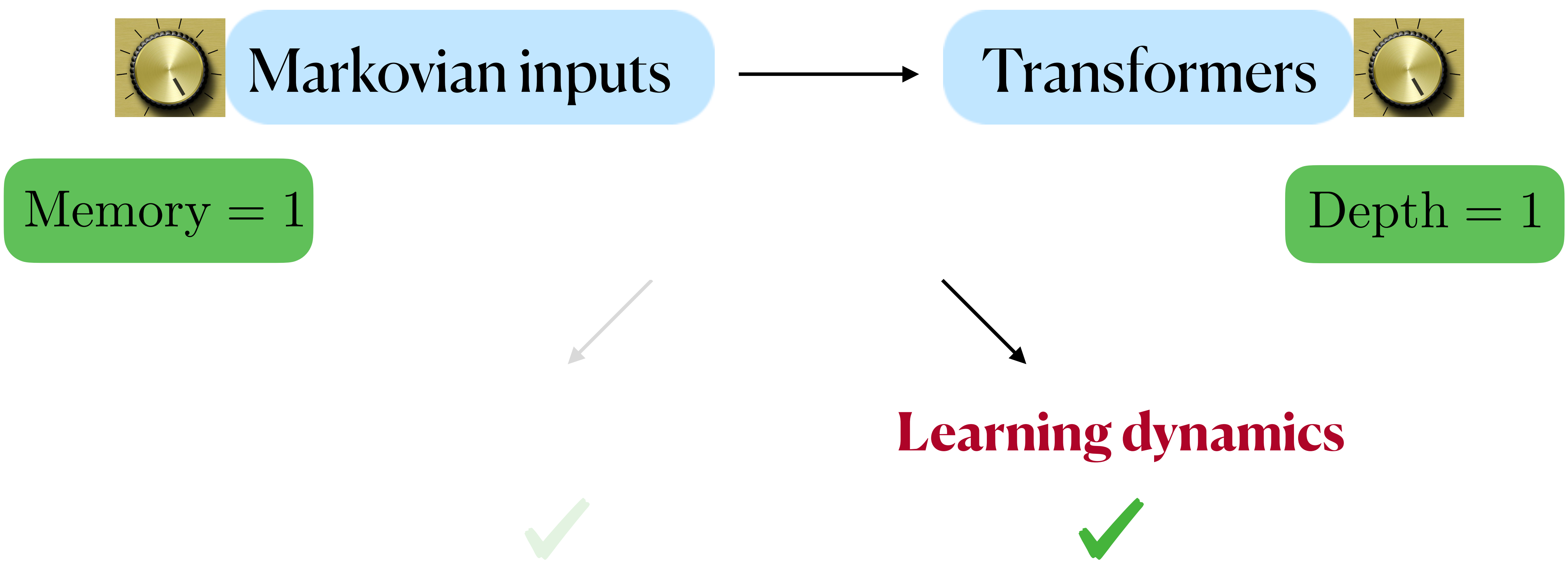
$p + q < 1$

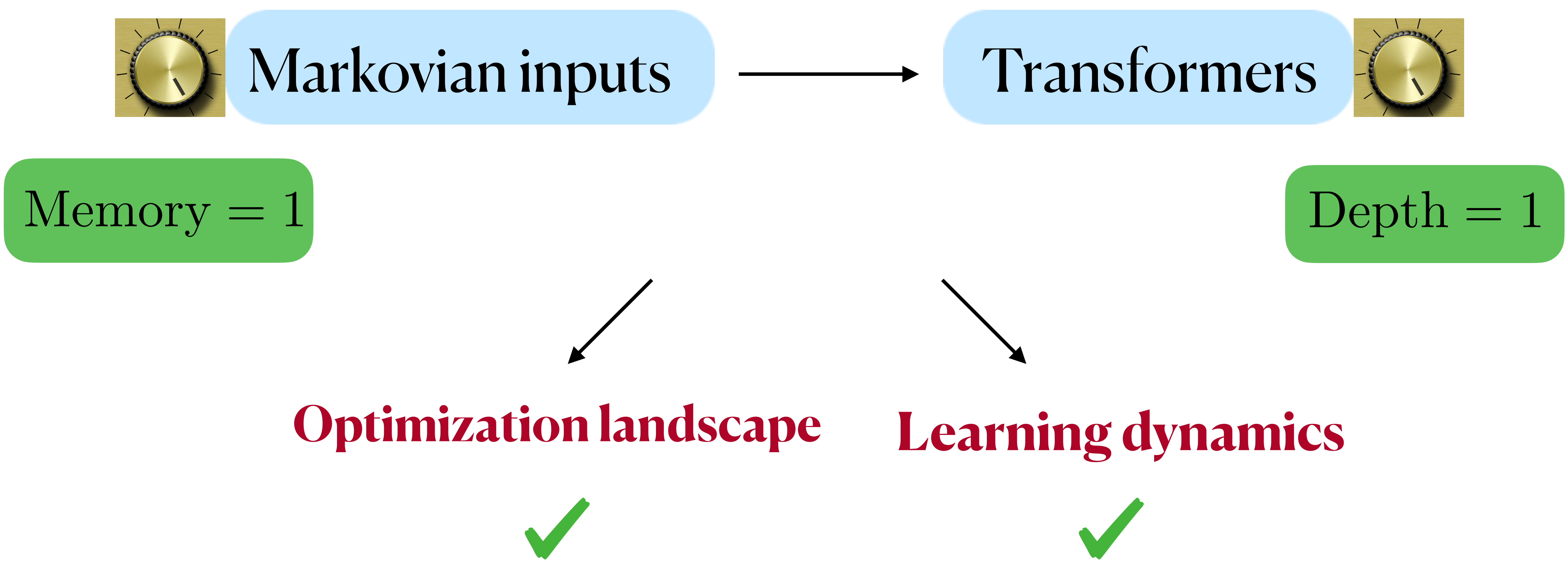
Gets stuck at local minima!



$p + q > 1$


2
1
e
0
−1
-2
2
a
0
e
a
w
$p + q < 1$
2
1
e
0
−1
2
2
a
0
−1
e
a
w
$p + q > 1$
Global minima
Local minima
Ball around origin

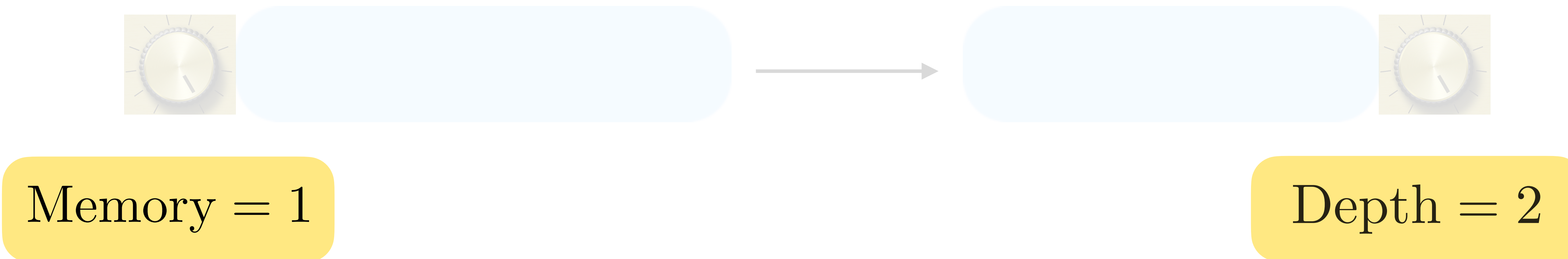

Markovian inputs
Transformers
Memory = 1
Depth = 1
Learning dynamics


Markovian inputs
Transformers
Memory = 1
Depth = 1
Optimization landscape
Learning dynamics

Key Takeaways

Single-layer transformers sometimes fail to learn even first-order Markov chains!

Markovian switching and initialization play a key role in the learning dynamics



[Nichani et al. 2024, Bietti et al. 2023, Chen et al. 2024]

Key Takeaways

In-context learning (ICL) emerges!

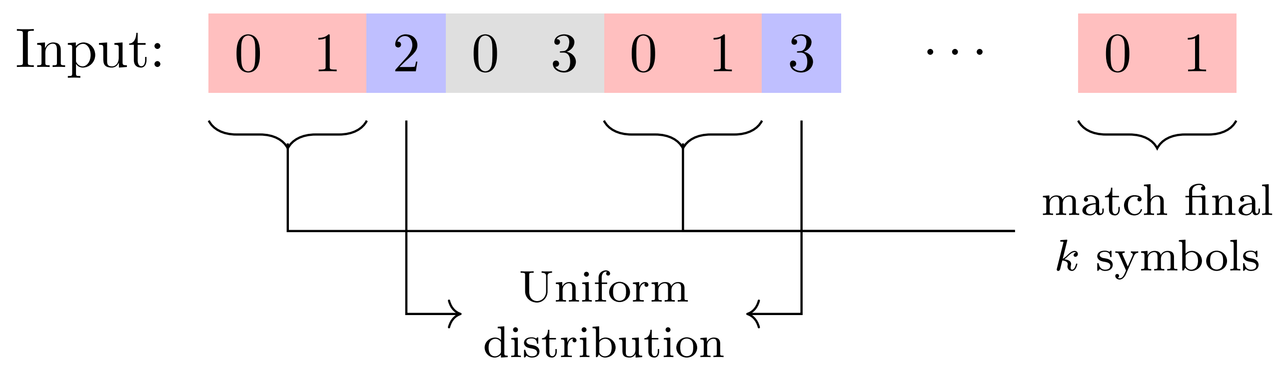
Depth plays a critical role in transformer functionality

ICL — Induction Heads

Mr and Mrs Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Mr Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large moustache. Mrs Durs___

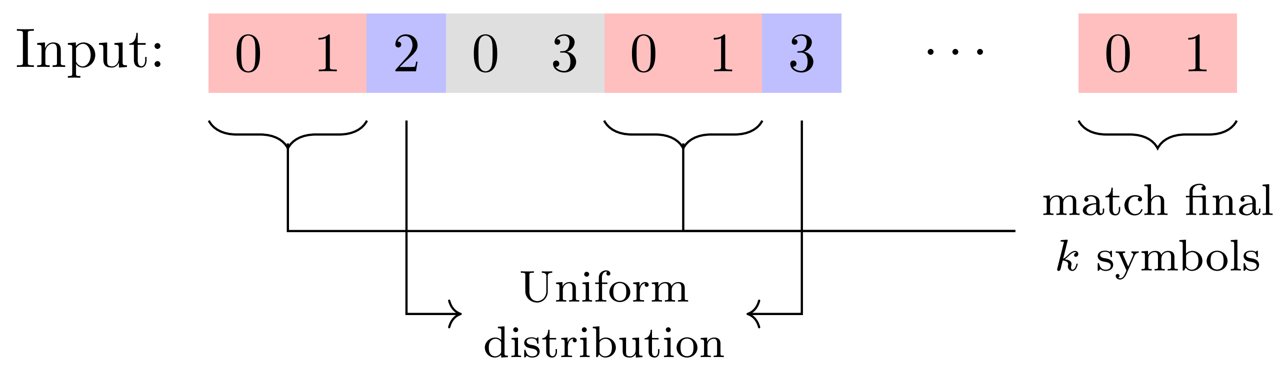
[Olsson et al. 2022]

Induction Head for Markov inputs



$(k = 2)$

Induction Head for kth-order Markov



$(k = 2)$

Main idea: Use historical tokens of same context as x_t to predict x_{t+1}

Induction Head for kth-order Markov

In-context estimator:

$$\widehat{\Pr}_k(x \mid x_1, \cdots, x_t) \triangleq \frac{\sum_{i=k+1}^{t} \mathbb{I}(x_i = x, x_{i-1} = x_t, \ldots, x_{i-k} = x_{t-k+1})}{\sum_{i=k+1}^{t} \mathbb{I}(x_{i-1} = x_n, \ldots, x_{i-k} = x_{t-k+1})}$$

Induction Head for kth-order Markov

In-context estimator:

Context-matching

$$\widehat{\Pr}_k(x \mid x_1, \cdots, x_t) \triangleq \frac{\sum_{i=k+1}^{t} \mathbb{I}(x_i = x, x_{i-1} = x_t, \ldots, x_{i-k} = x_{t-k+1})}{\sum_{i=k+1}^{t} \mathbb{I}(x_{i-1} = x_n, \ldots, x_{i-k} = x_{t-k+1})}$$

How do Transformers implement Induction Heads?

How do Transformers implement Induction Heads?

Markovian inputs → Transformers

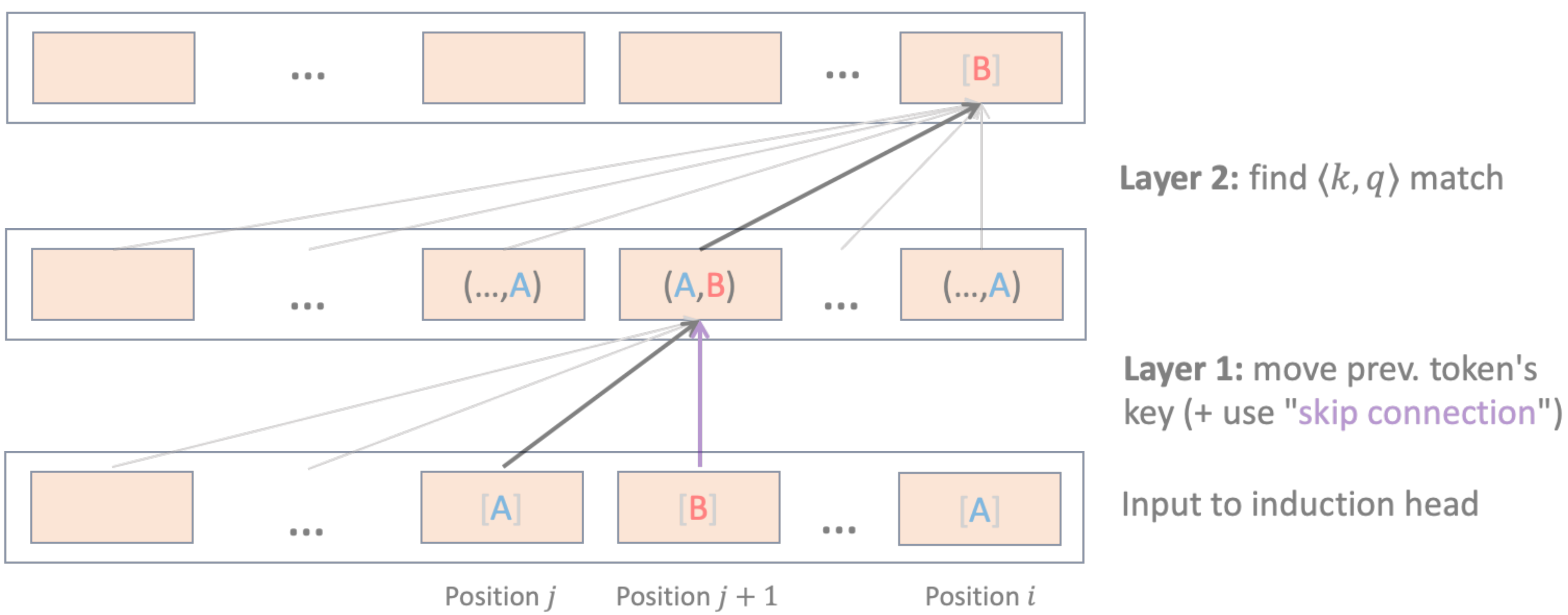
Memory $= 1$

Depth $= 2$

[Nichani et al. 2024, Bietti et al. 2023, Chen et al. 2024]

Induction Heads via Two-layer Transformers

Second-layer does pattern matching and prediction



First-layer copies the previous token

How do Transformers learn Induction Heads?

Markovian inputs → Transformers

Memory $= 1$

Depth $= 2$

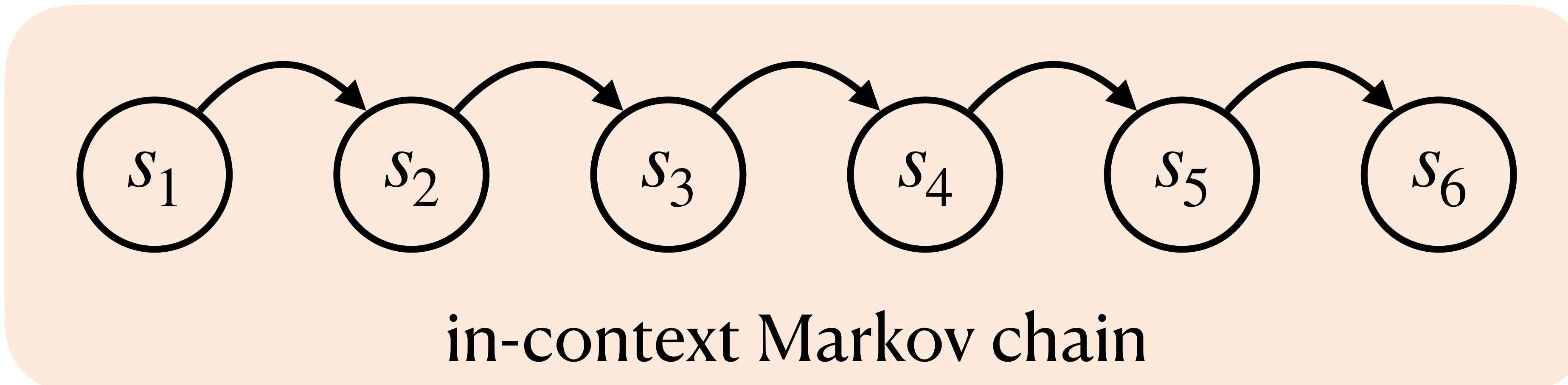
[Nichani et al. 2024, Bietti et al. 2023, Chen et al. 2024]

In-context Markov chains

$\text{Memory} = 1$

$\text{Depth} = 2$

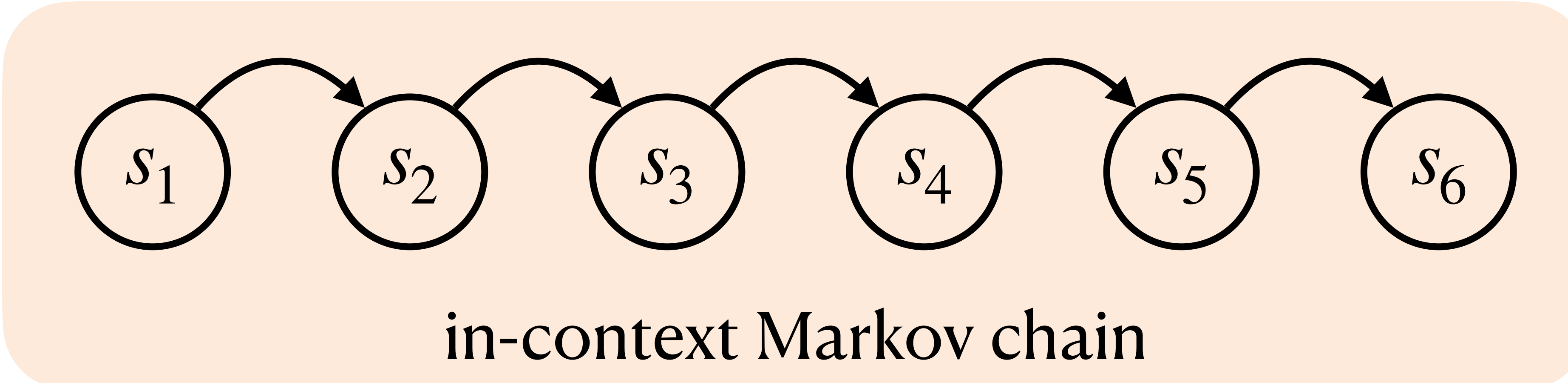
In-context Markov chains



To generate each sequence:

- Sample a probability transition matrix or kernel π from some prior (e.g. Dirichlet)
- Sample s_1 from its stationary measure
- For $i = 1,\ldots,T-1$: sample $s_{i+1} \sim \pi(\,\cdot \mid s_i)$

In-context Markov chains



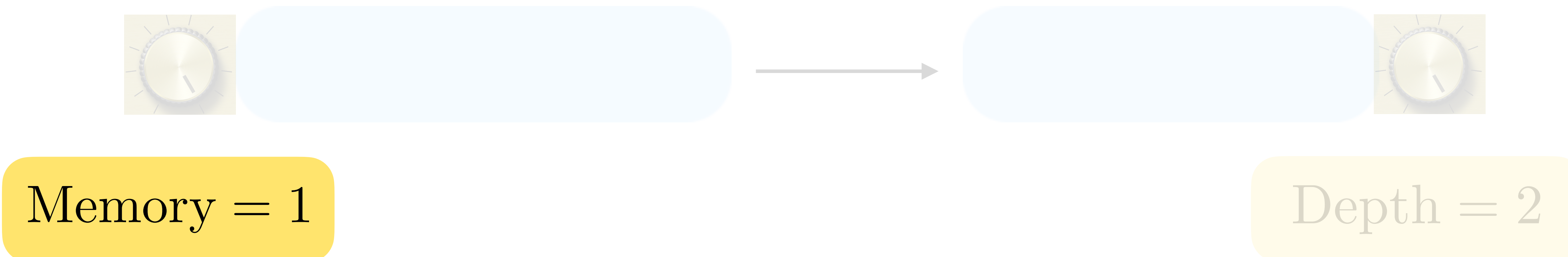
To generate each sequence:

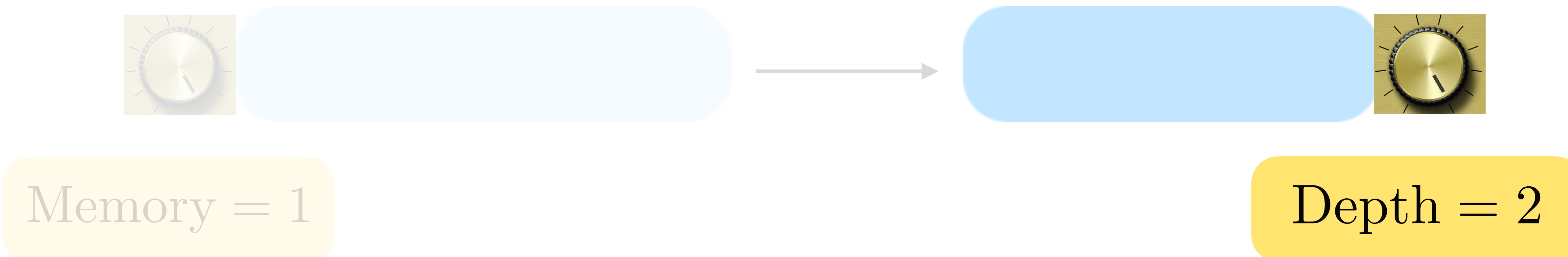
- Sample a probability transition matrix or transition kernel π from some prior (e.g. Dirichlet)
- Sample s_1 from its stationary measure
- For $i = 1, \ldots, T-1$: sample $s_{i+1} \sim \pi(\cdot \,|\, s_i)$

Natural Estimator: compute the empirical transition counts in-context

$$\hat{p}(s' \,|\, s) = \frac{\# s \to s' \text{ transitions in the sequence}}{\# s \text{ in the sequence}}$$

In-context Markov chains




Memory = 1
Depth = 2

The Disentangled Transformer

The Disentangled Transformer

1. Use one-hot token+positional embeddings
2. Replace linear projections with concatenation

[similar model considered in Friedman et al. 2023, Learning Transformer Programs]

The Disentangled Transformer

1. Use one-hot token+positional embeddings
2. Replace linear projections with concatenation

$$x_i = [\underbrace{\text{onehot}(s_i)}_{\text{token}} \,|\, \underbrace{\text{onehot}(i)}_{\text{position}}]$$

For $i = 1, \ldots, L$:

$$x_i \leftarrow [x_i, \text{attn}(X)_i]$$

$$x_i \leftarrow [x_i, \text{mlp}(X)_i]$$

Return $W_O x_T$

Theorem:

Transformers with H heads and L layers have the same expressive power as disentangled transformers with H heads and L

The Disentangled Transformer

1. Use one-hot token+positional embeddings
2. Replace linear projections with concatenation

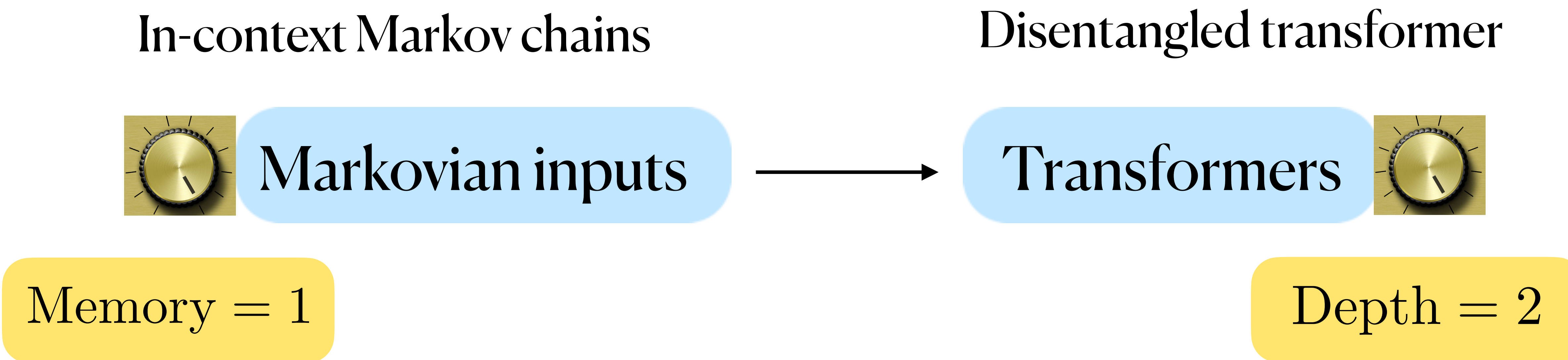
✗

- Completely impractical: the embedding dimension **doubles** at every step

✓

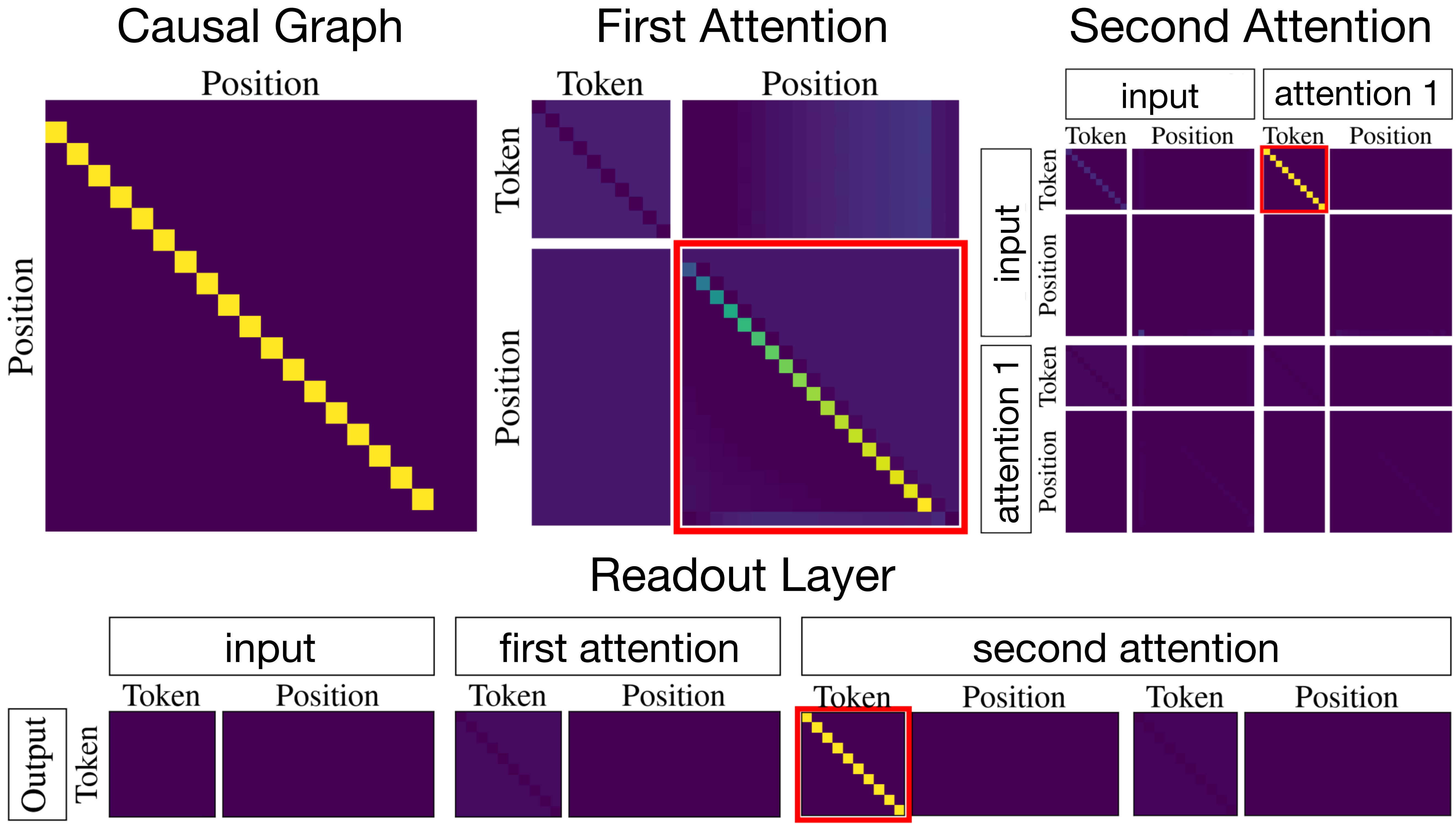
- weights are directly interpretable
- easier to reason about the flow of information through the model
- useful tool for theory and mechanistic interpretability

Disentangled transformer ✓

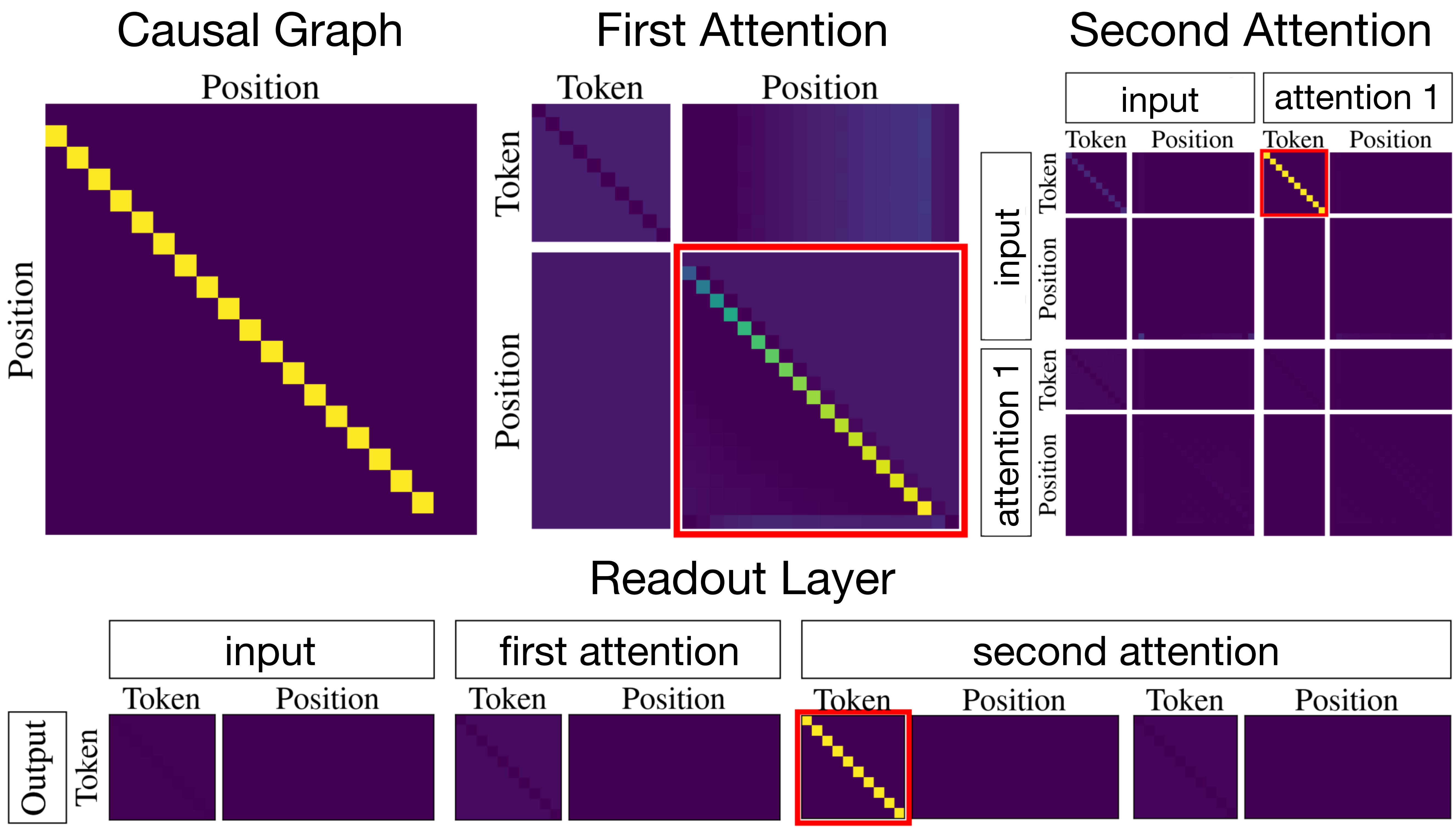

In-context Markov chains
Disentangled transformer
Markovian inputs
Transformers
Memory = 1
Depth = 2

How do Transformers solve this task?

How do Transformers solve this task?

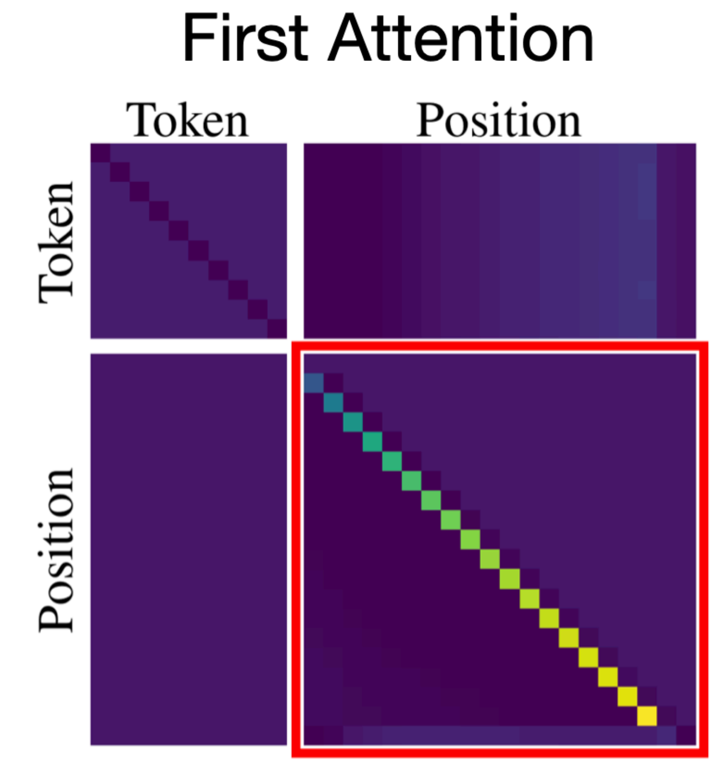


How do Transformers solve this task?



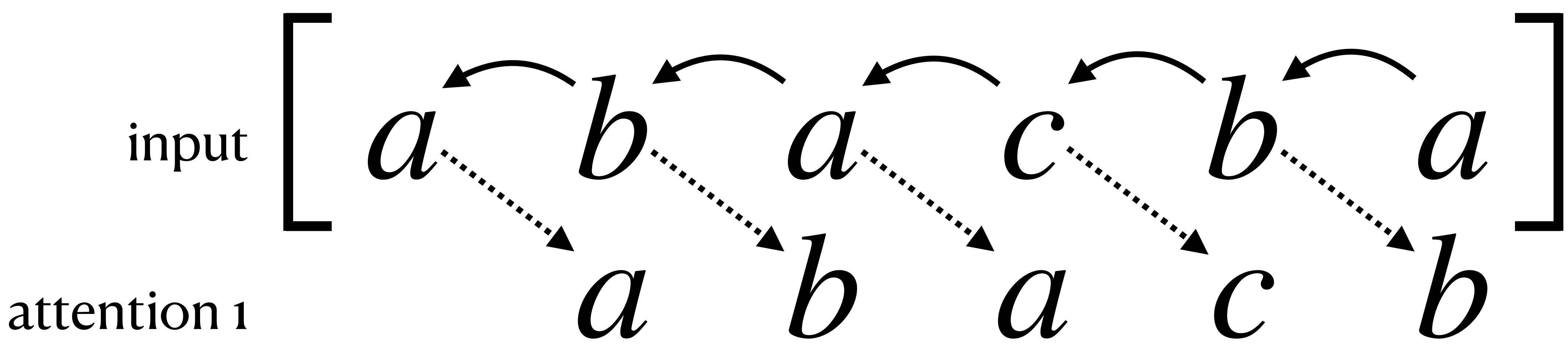
The first attention matrix is the adjacency matrix for the causal graph!

How Transformers Solve This Task

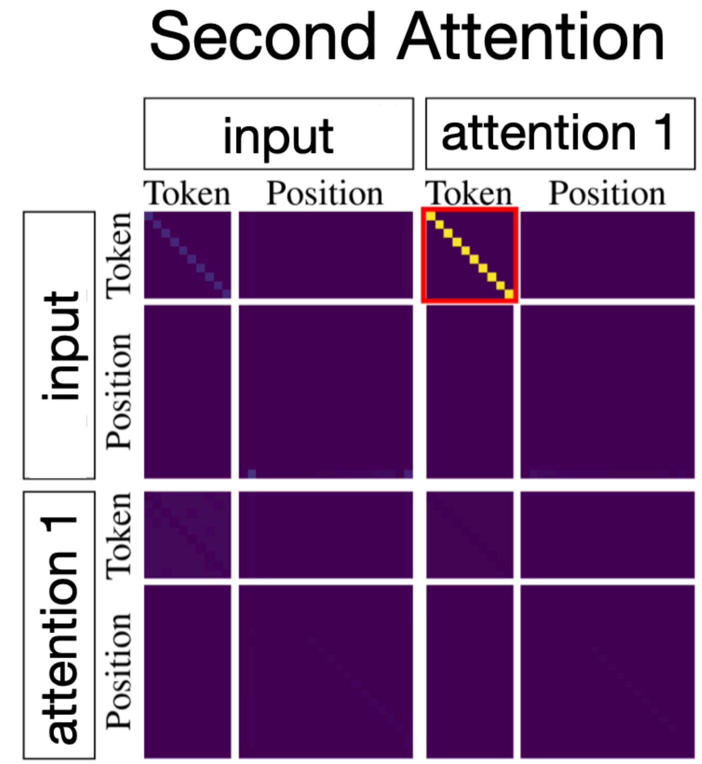


First Attention:

copy each parent

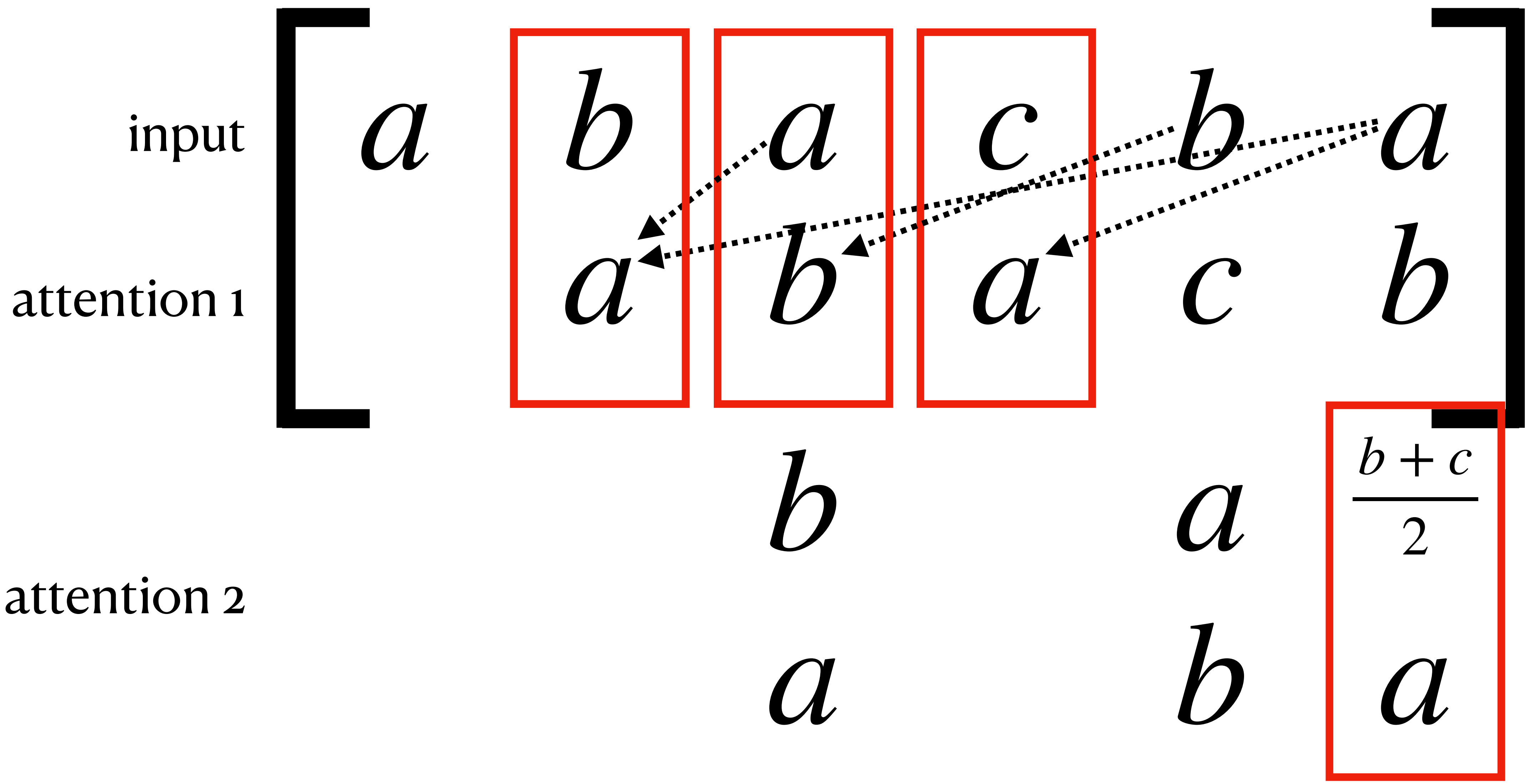


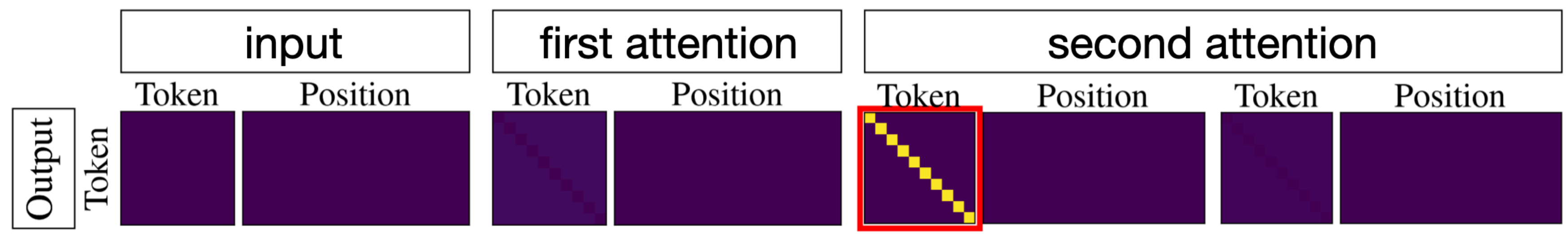
How Transformers Solve This Task



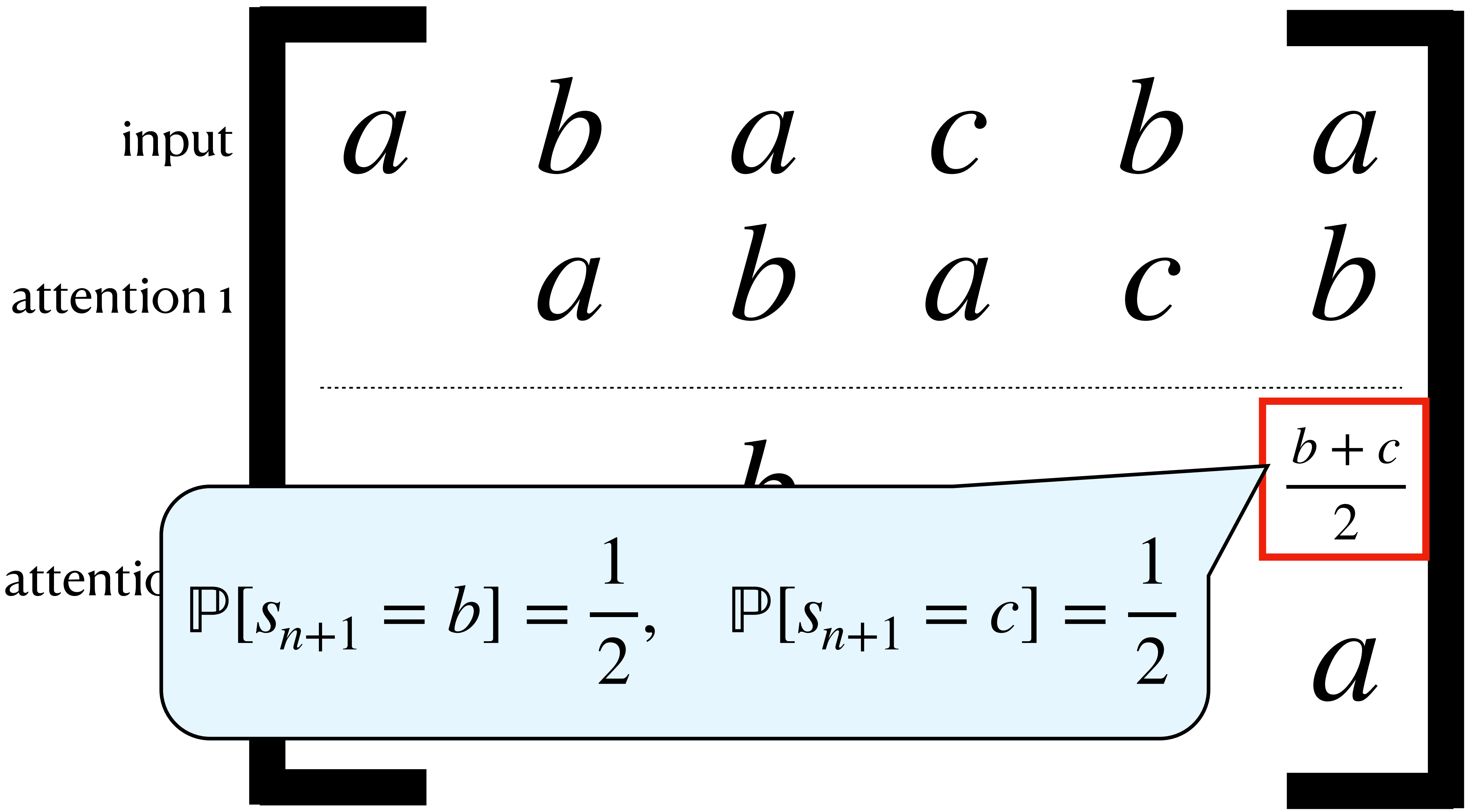
Second Attention:

compare to each parent

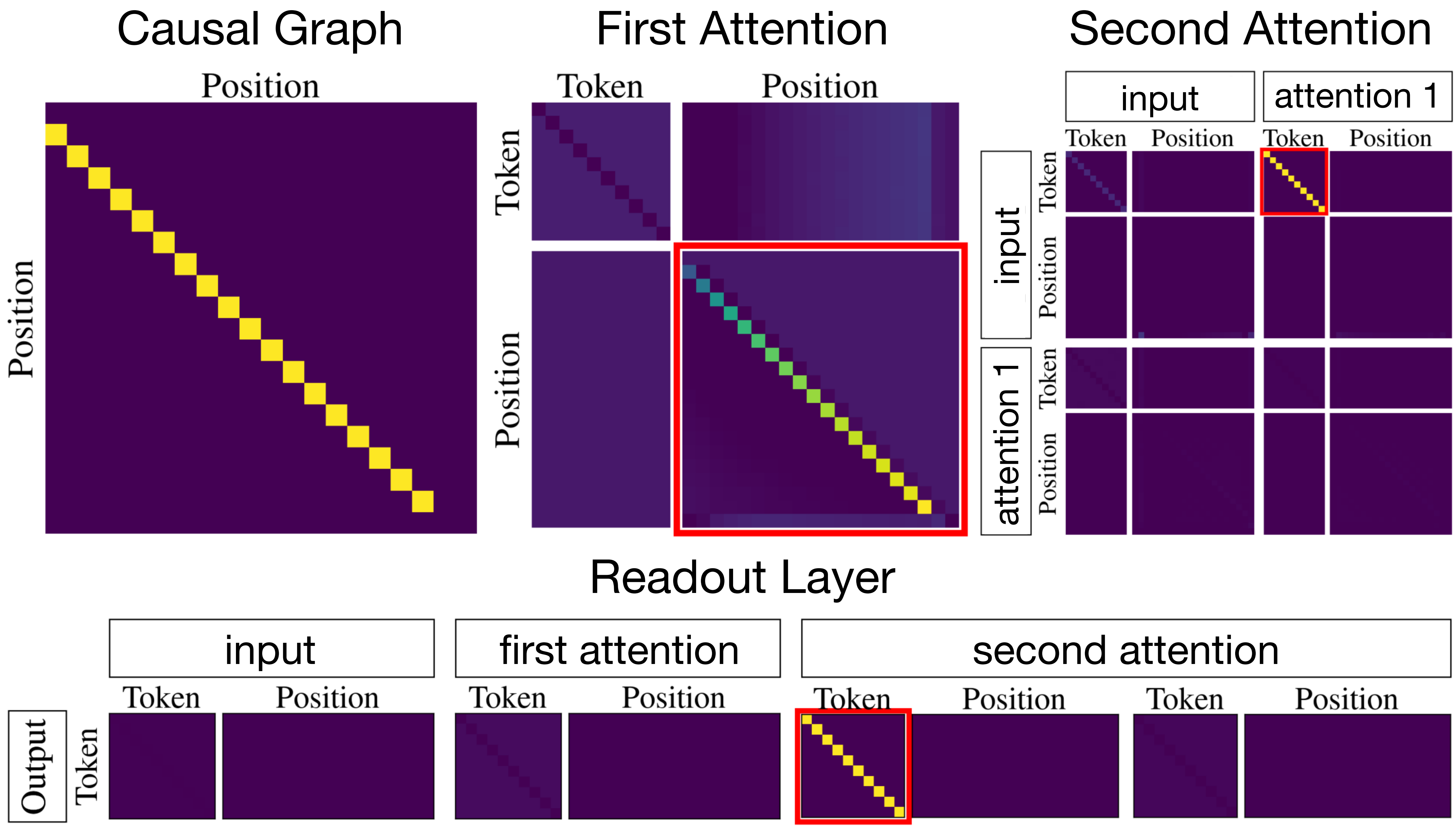




Readout Layer: output empirical counts



Gradient Descent Dynamics



Main Result

Loss: cross-entropy

$$L(\theta) = -\mathbb{E}_{\pi, s_{1:T}}\left[\sum_{s' \in [S]} \pi(s' \mid s_T) \log(f_\theta(s_{1:T})_{s'})\right]$$

Theorem (informal): If $\min_{s,s'} \pi(s \mid s') \geq \gamma / S$ almost surely over the prior P_π,

(1) There exists $c > 0$ such that GD returns θ satisfying:

$$L(\theta) - \mathrm{OPT} \lesssim \frac{1}{T^{c\gamma}}$$

(2) For any input sequence, the first attention pattern $A \in \mathbb{R}^{T \times T}$ satisfies:

$$\|A - G\|_\infty \lesssim \frac{1}{T},$$

where G is the adjacency matrix of the causal graph.

Corollary: Transformers trained on in-context Markov chains learn an induction head

OOD Generalization

Mechanistic understanding leads to provable OOD generalization:

Corollary:

Let $\tilde{\pi}$ satisfy $\min_{s,s'} \tilde{\pi}(s' \mid s) \geq \gamma / S$. Then with high probability over draw of $s_{1:T}$:

$$\left\| f_{\hat{\theta}}(s_{1:T}) - \tilde{\pi}(\cdot \mid s_T) \right\|_\infty \lesssim \frac{1}{T^{c\gamma}}$$

Note that $\tilde{\pi}$ does not need to be in the support of P_π.

Even if you learn an induction head on a very restricted class of sequences, this circuit automatically generalizes out of distribution to arbitrary sequences

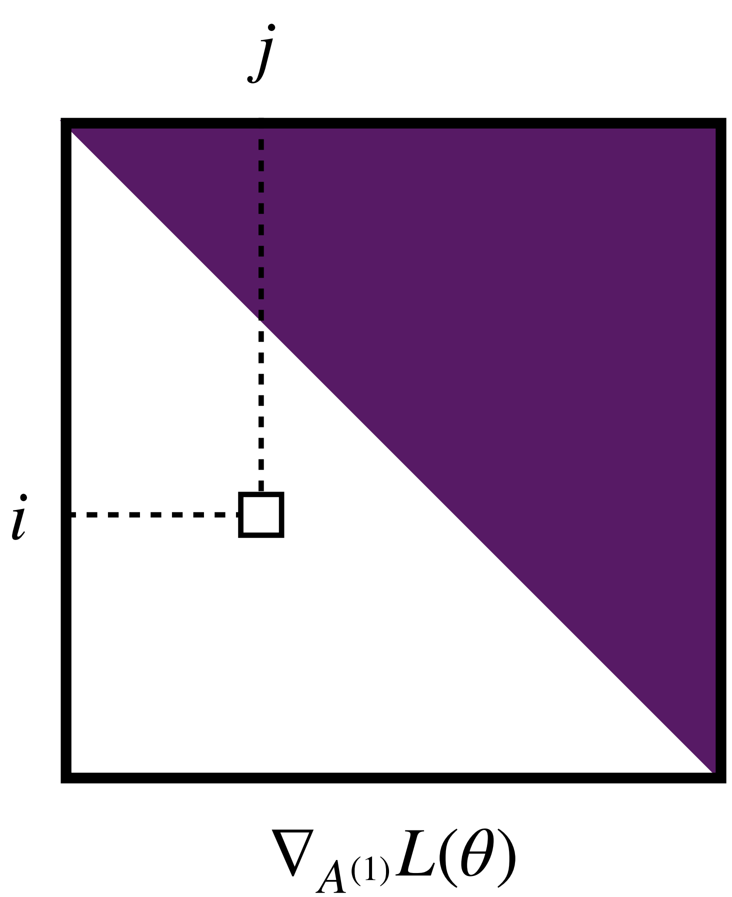
Proof Sketch

Key Lemma: For $j < i$, the gradient of the first attention layer is approximately

$$\nabla_{A_{ij}^{(1)}} L(\theta) \approx -I_{\chi}^{2}(s_i; s_j \mid \pi) \quad \text{where} \quad I_{\chi^2}(s_i; s_j \mid \pi) := \mathbb{E}_{\pi}\left[\sum_{s_i, s_j} \frac{\mathbb{P}[s_i, s_j]^2}{\mathbb{P}[s_i]\mathbb{P}[s_j]} - 1\right].$$

how much token i attends to token j

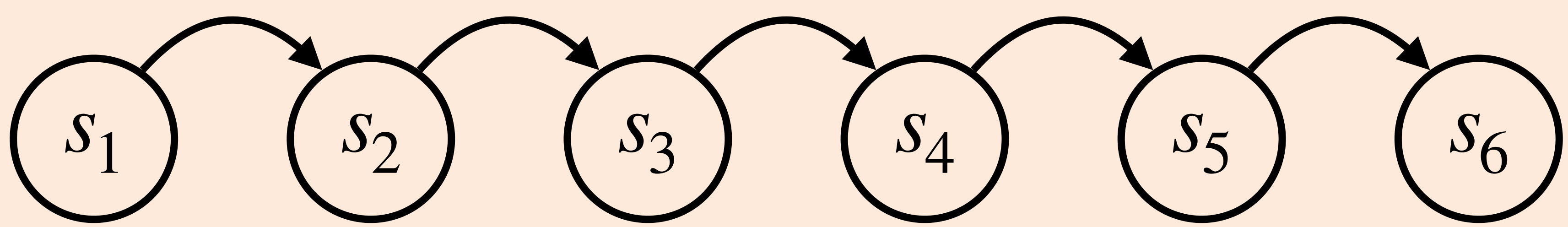
χ^2 mutual information between the token at position i and the token at position j



Corollary: Each position i will eventually attend to the position $j < i$ that maximizes the χ^2 mutual information

Proof Sketch

Corollary: Each position i will eventually attend to the position $j < i$ that maximizes the χ^2 mutual information between s_i and s_j



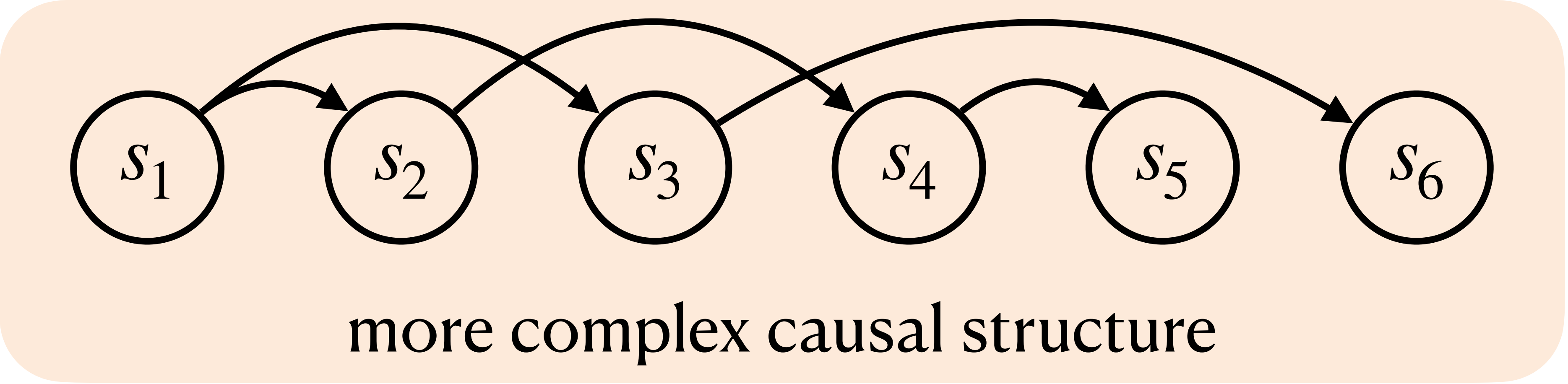
Data Processing Inequality:

Passing through a channel can only decrease mutual information:

$$\ldots < I_\chi^2(s_6; s_3) < I_\chi^2(s_6; s_4) < I_\chi^2(s_6; s_5)$$

- Each token will attend to the token immediately before it
- **The transformer learns an induction head!**

Corollary: Each position i will eventually attend to the position $j < i$ that maximizes the χ^2 mutual information between s_i and s_j



Data Processing Inequality:

Passing through a channel can only decrease mutual information:

$$I_{\chi^2}(s_i, s_j) \text{ is maximized at } j = p(i), \text{ the parent of } i$$

- The first attention layer learns the causal graph
- Special case of the well-known Chow-Liu algorithm (Chow & Liu, 1968) for learning tree-structured graphical models!

How do Transformers learn Induction Heads?

Markovian inputs → Transformers

Memory $= 1$

Depth $= 2$

How do Transformers Implement Induction Heads?

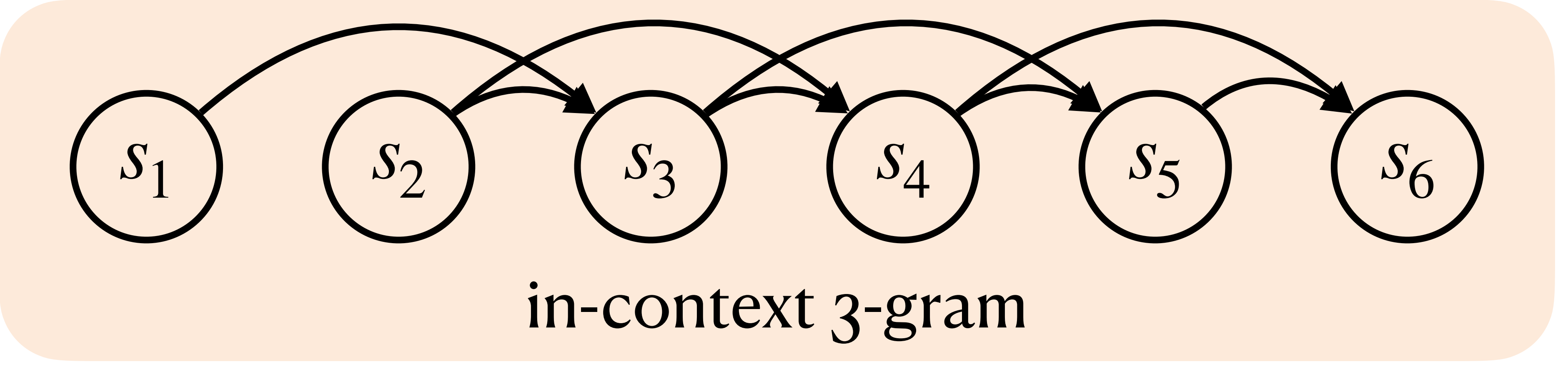
Markovian inputs → Transformers

Memory $= k$

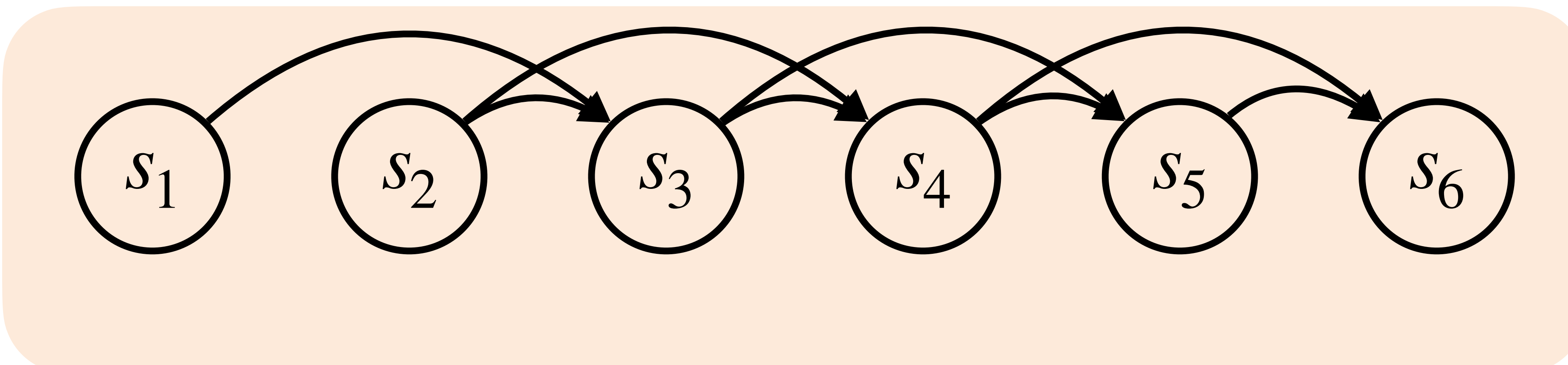
Depth/Heads

Multiple Parents & k-grams

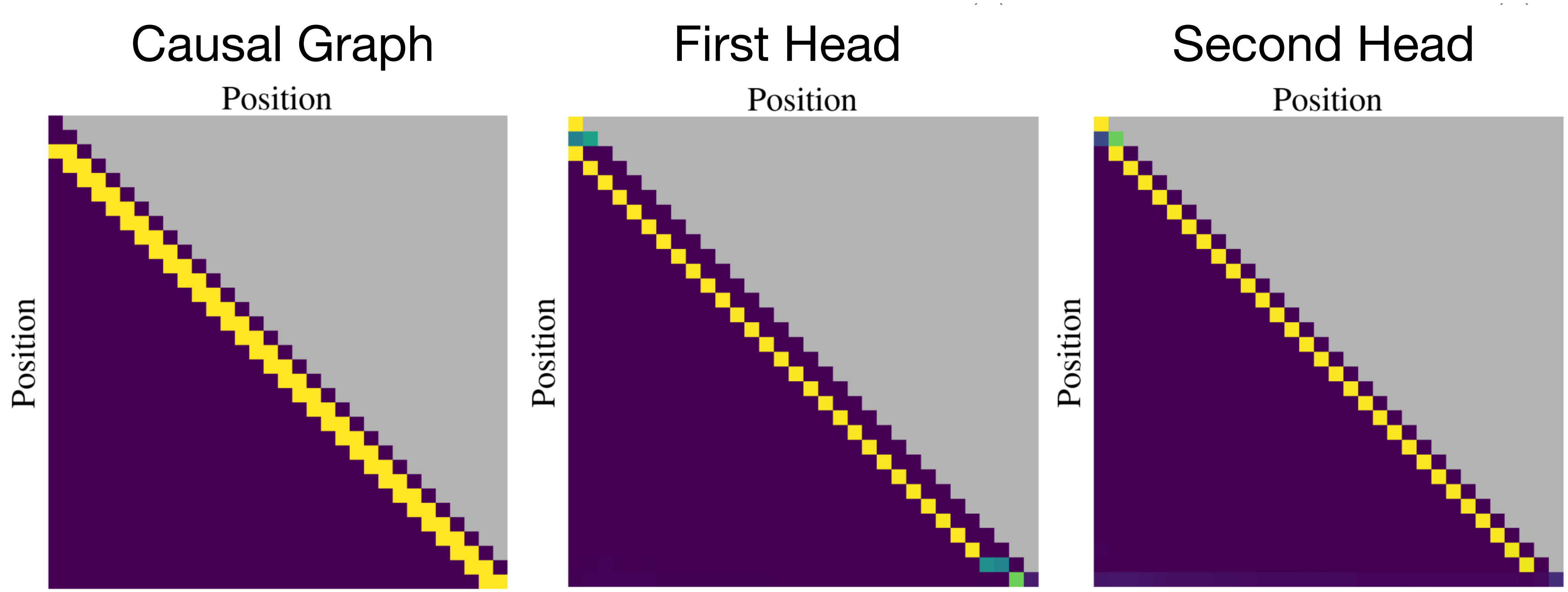
- Each node can have multiple parents in the causal graph
- Example: k-gram language models



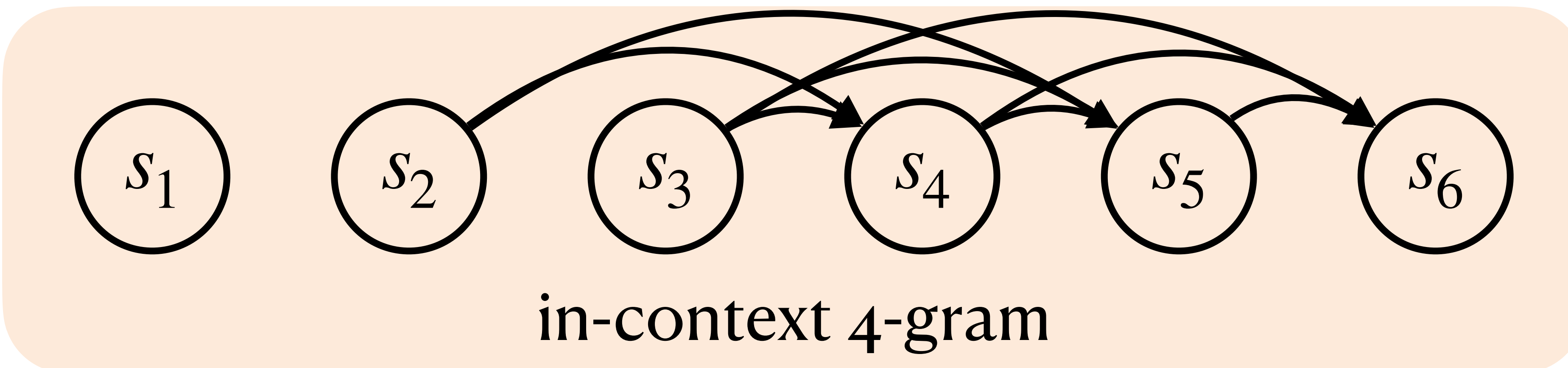
Multiple Heads



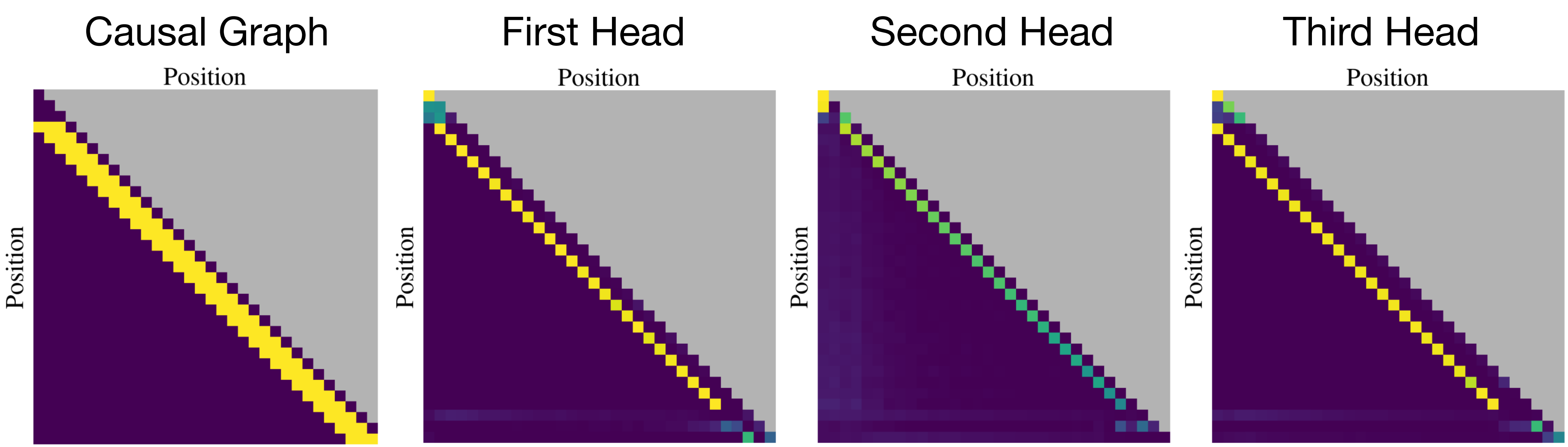
Construction & Experiments: Each head attends to a different parent



Multiple Heads

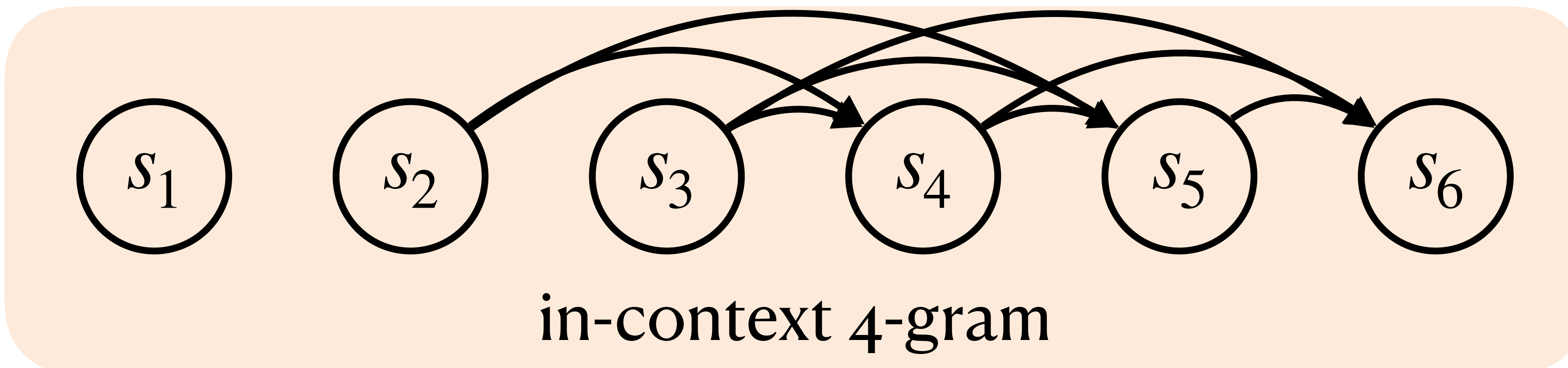


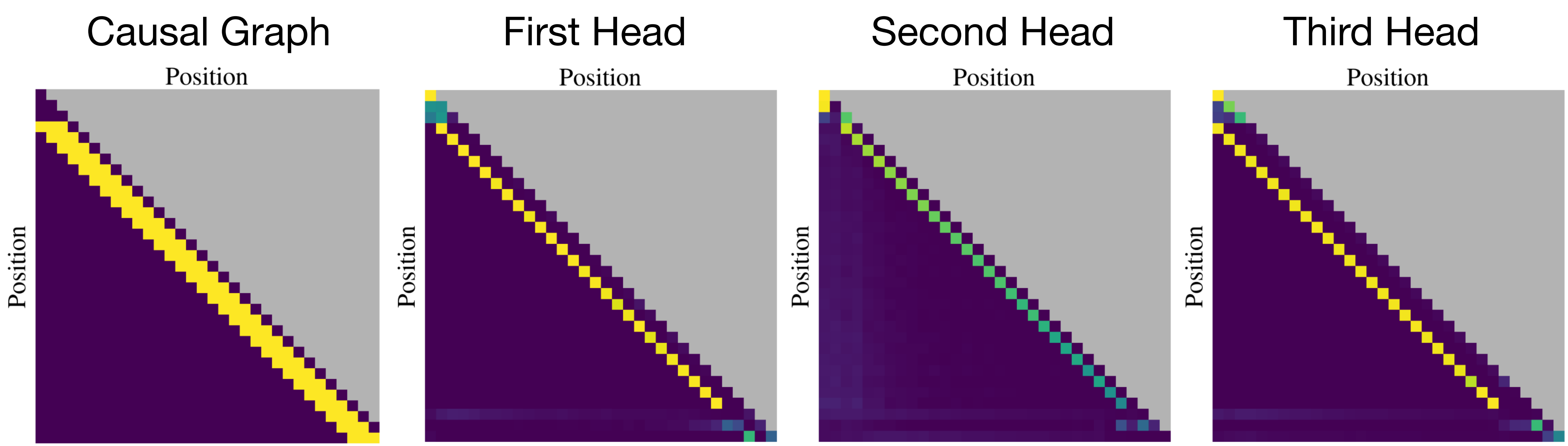
Construction & Experiments: Each head attends to a different parent



Multiple Head: Learning Dynamics

[Chen et al. 2024]



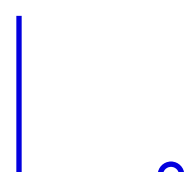


Learning Dynamics

How gradient-descent learns induction heads

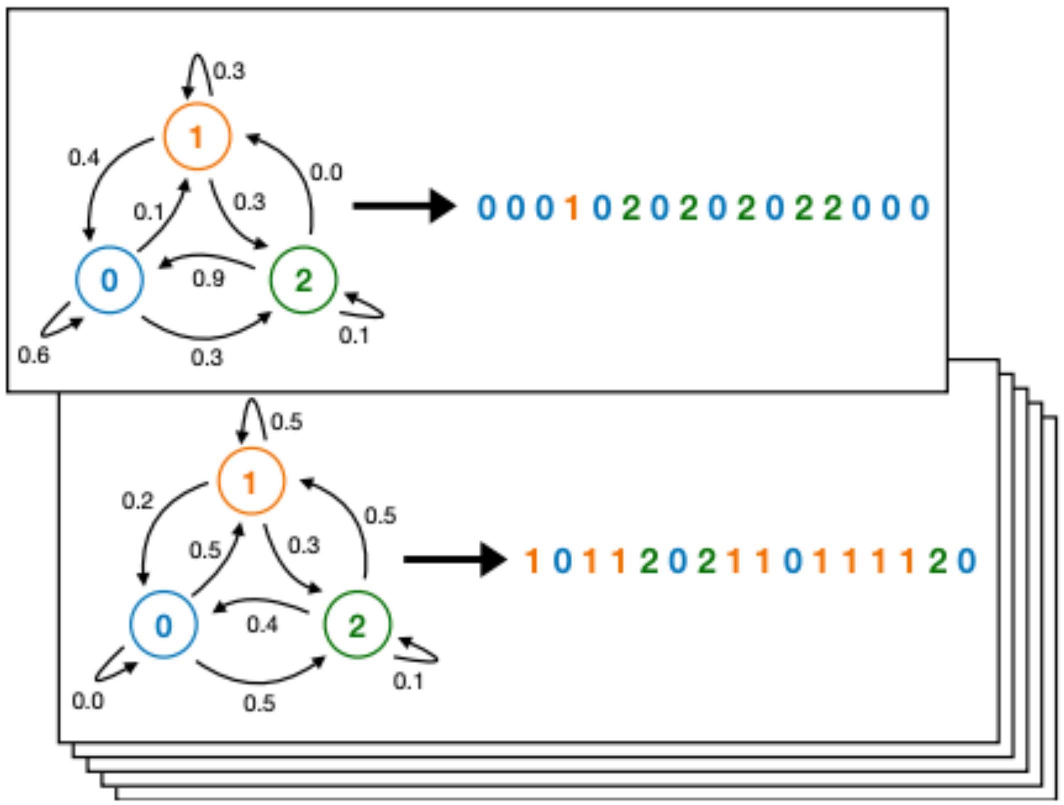
Learning Dynamics

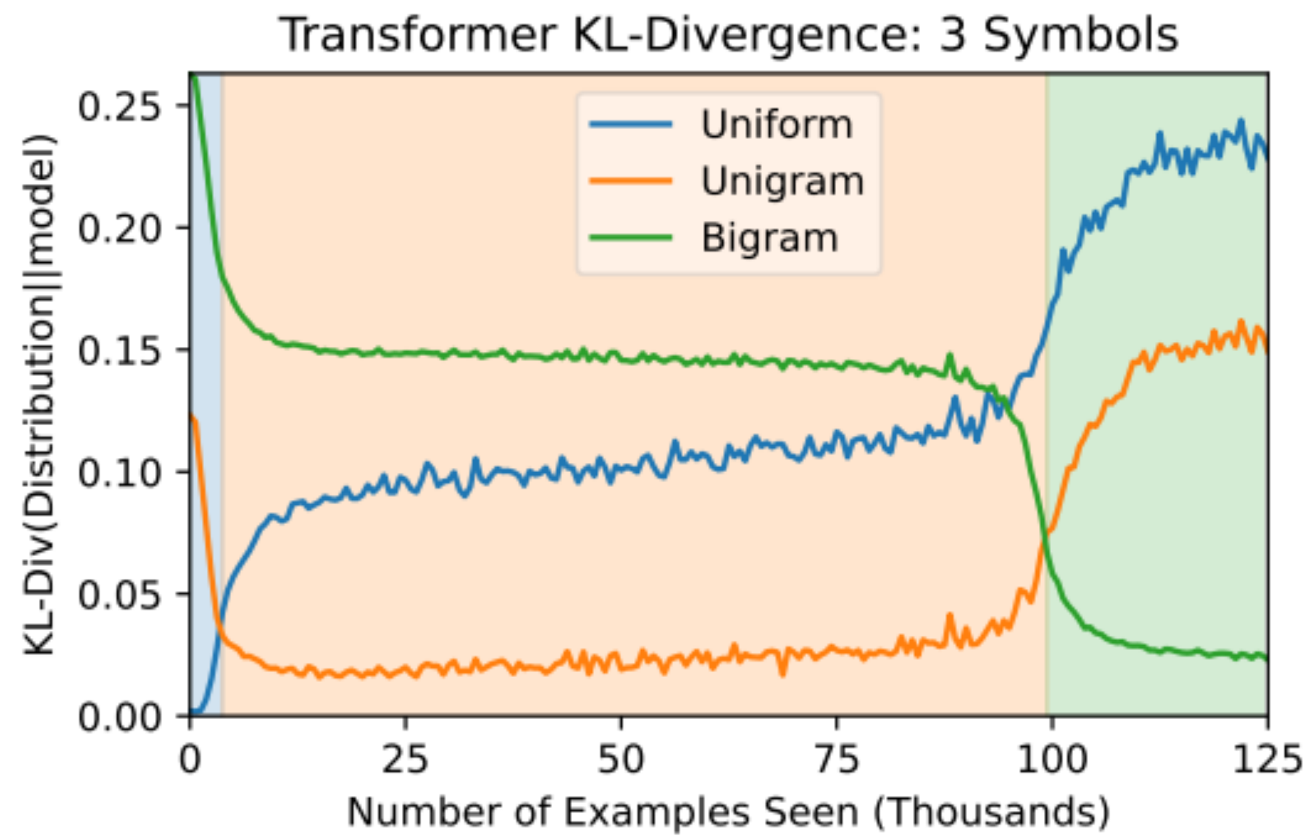
How gradient-descent learns induction heads



Stage-wise learning dynamics

Stage-wise Learning Dynamics



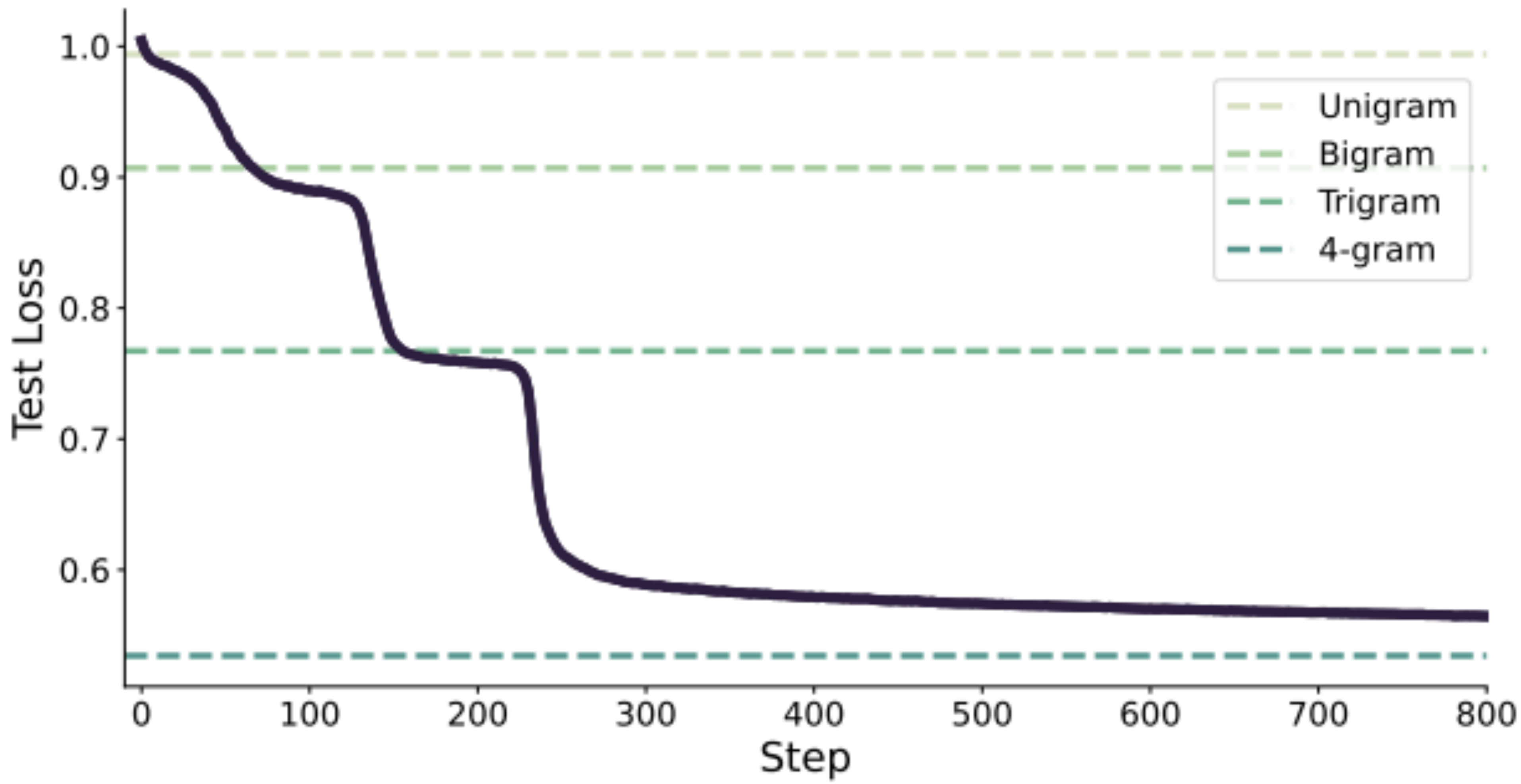


Memory = 1

Depth = 2

[Edelman et al. 2024]

Stage-wise Learning Dynamics

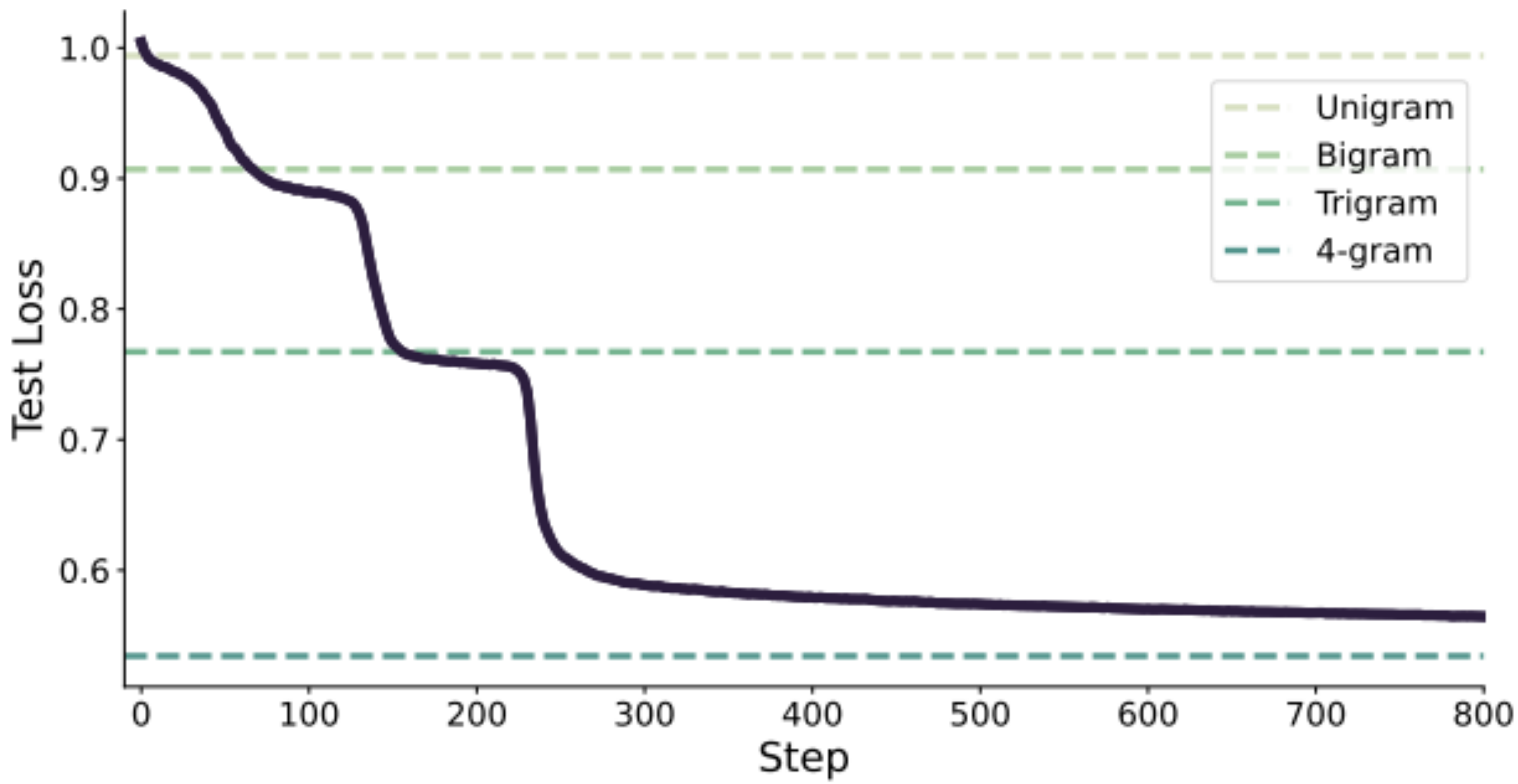


$\text{Memory} = n - 1$ n-gram

$\text{Depth} = 2, \text{Heads} = n$ Two-layer disentangled Transformer with n heads

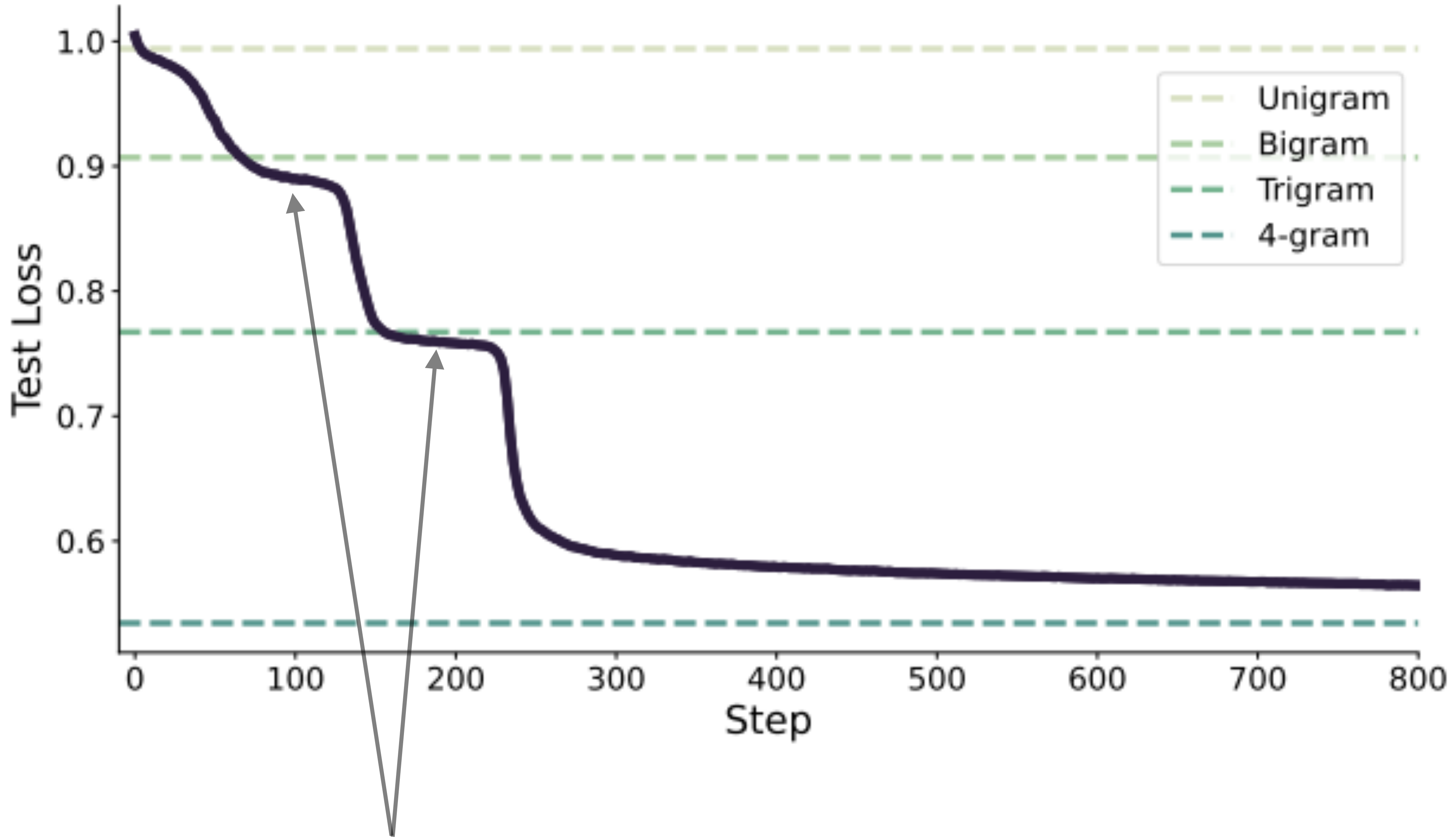
[Varre et al. 2025]

Why does training linger at plateaus?



[Varre et al. 2025]

Plateaus correspond to sub n-grams

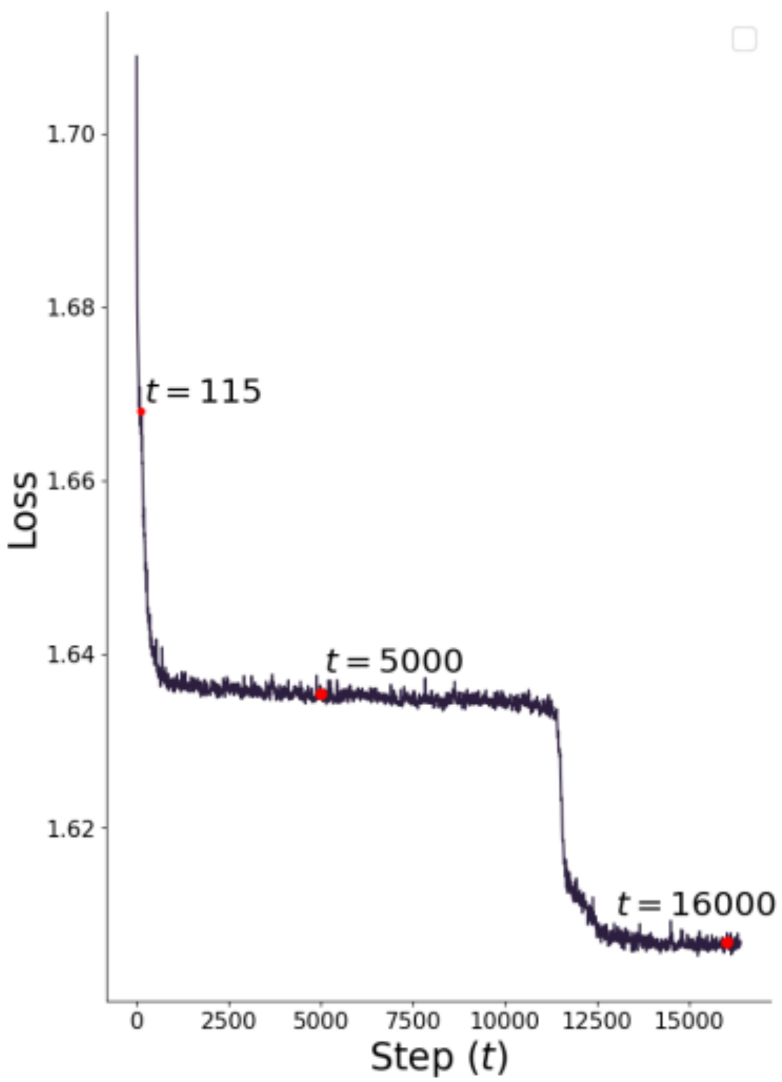


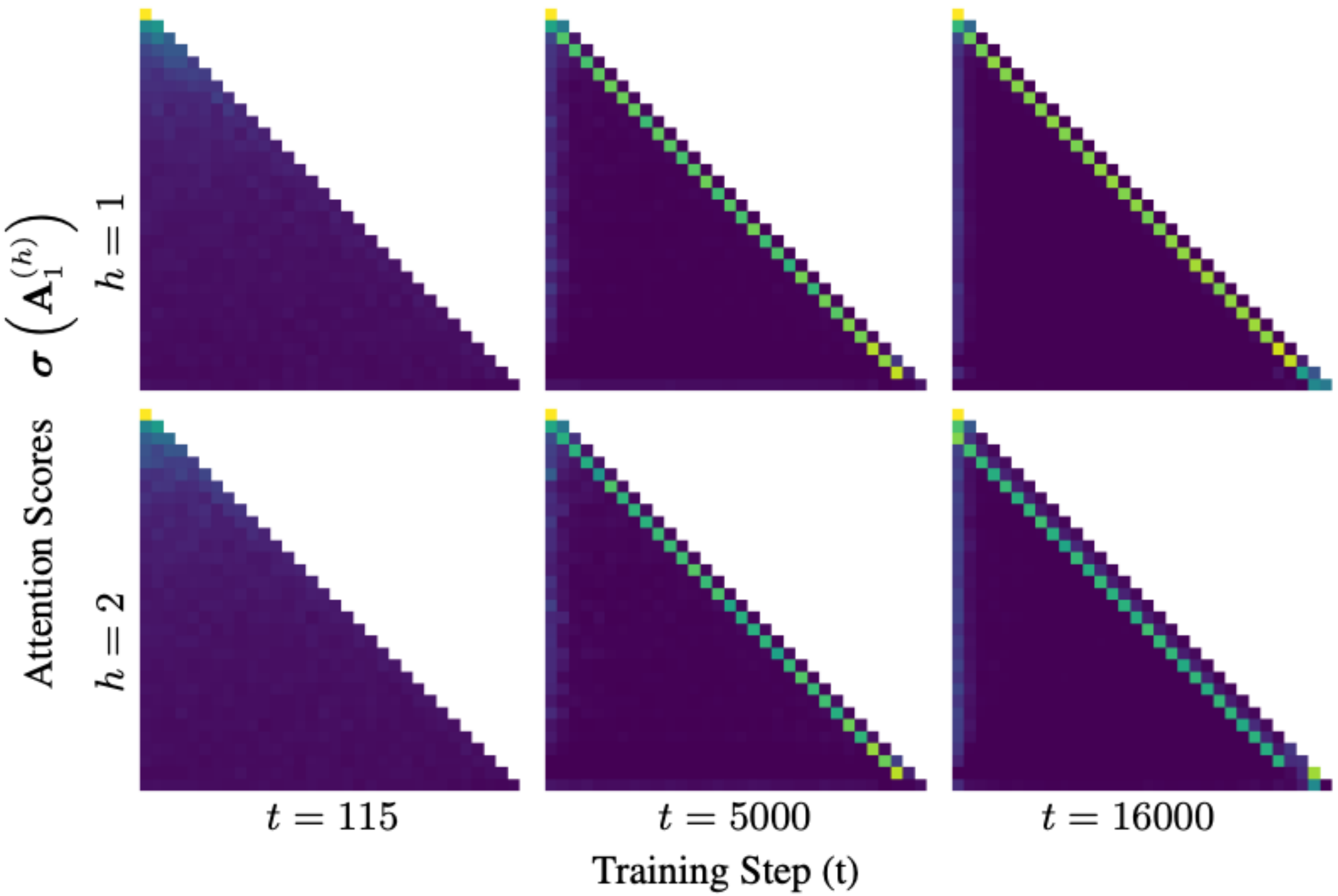
Main result (Plateaus correspond to sub n-grams)

$\exists\, \boldsymbol{\theta}_k^*$ such that $\boldsymbol{\theta}_k^*$ represents a k-gram for $k < n$

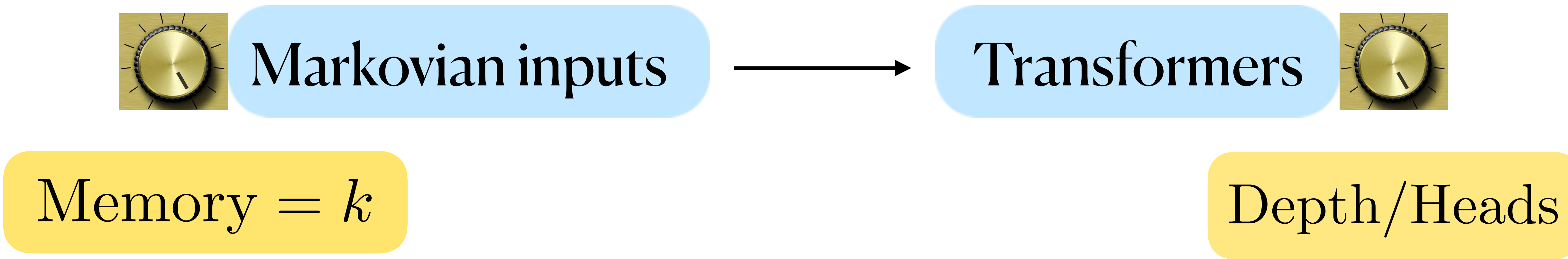
$\boldsymbol{\theta}_k^*$ is a (near) stationary point in the limit (length, param norm) $\to \infty$

Stage-wise Learning Dynamics





What we know so far



What we know so far

 Markovian inputs → Transformers 

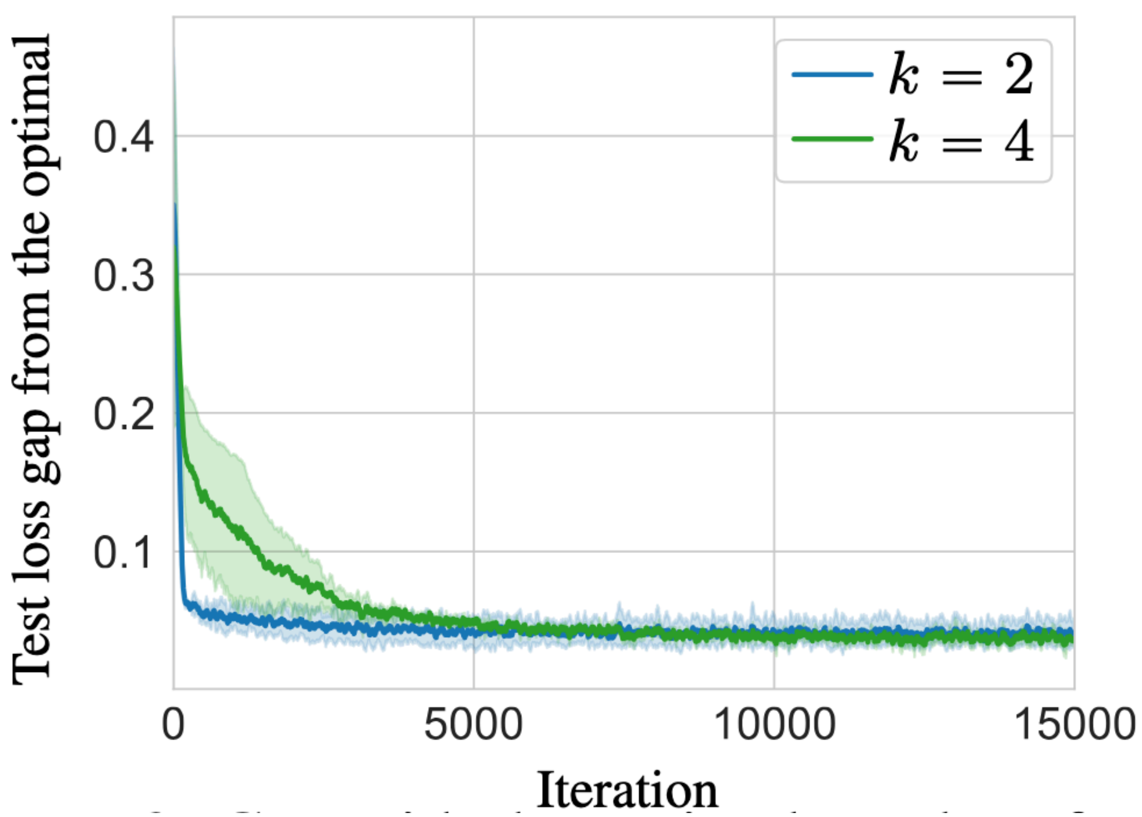
Memory $= k$

Depth/Heads

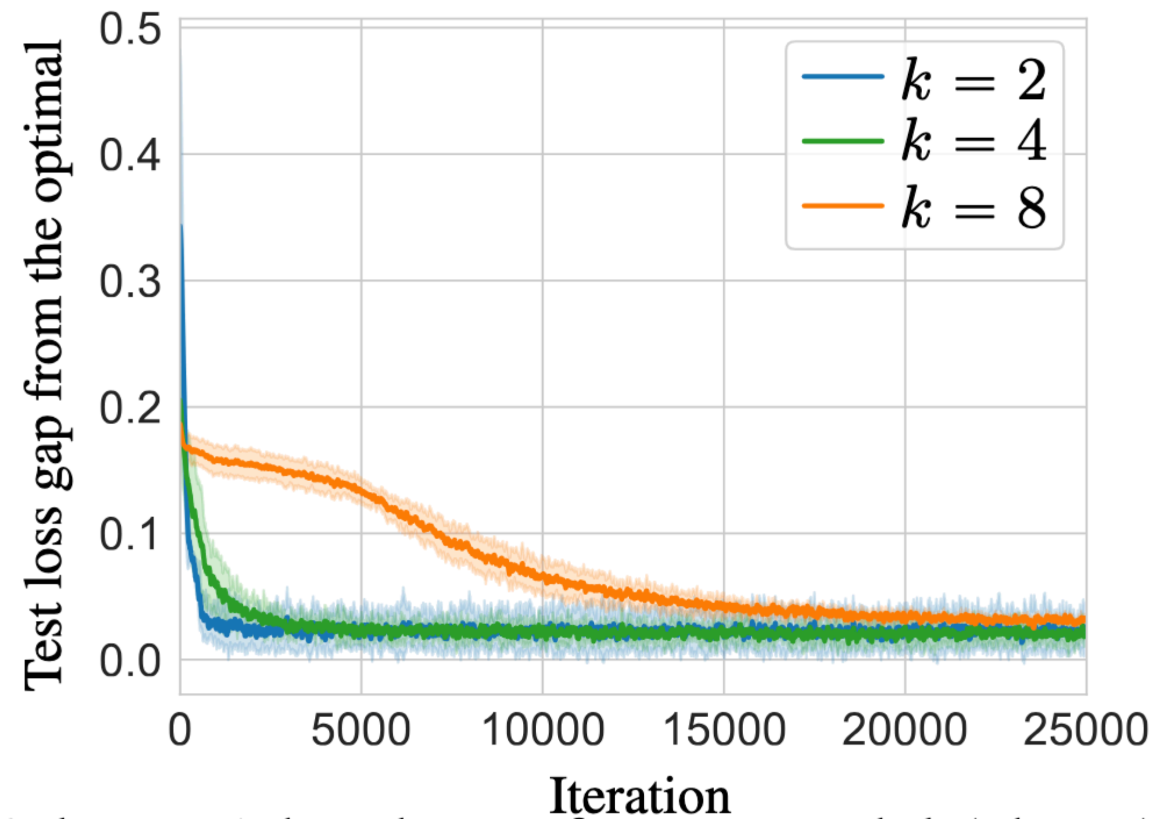
- Number of heads/layers should scale with k

[Edelman et al. 2024, Nichani et al. 2024, Chen et al. 2024]

But...



2-layer transformer



3-layer transformer

What's happening?

Representation result

Main result (Constant depth suffices)

Any order k in-context estimator can be represented by a transformer with 3 layers, 1 head per layer, relative positional encodings and layer norm

[**R**ajaraman et al. 2024]

Representation result

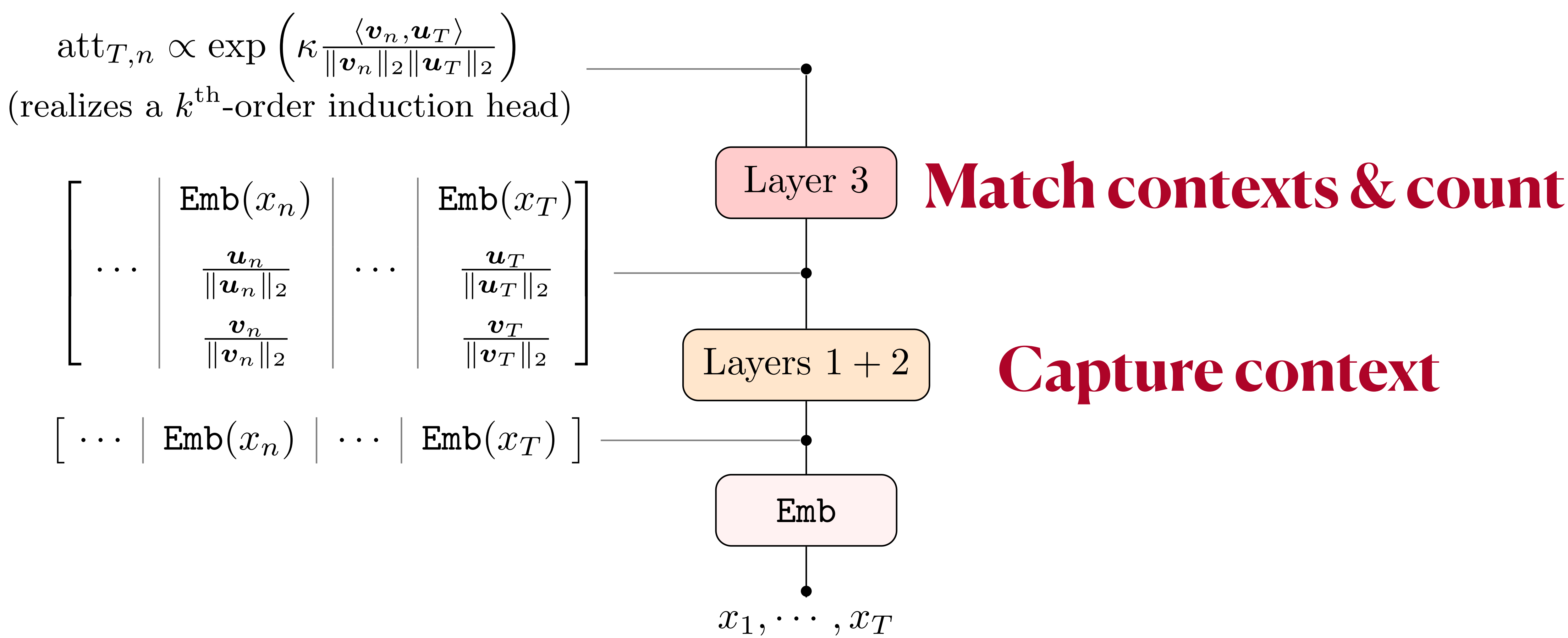
Main result (Constant depth suffices)

Any order k in-context estimator can be represented by a transformer with 3 layers, 1 head per layer, relative positional encodings and layer norm

Without non-linearities, you need logarithmic depth

[**R**ajaraman et al. 2024]

Intuition



$\mathrm{att}_{T,n} \propto \exp\left(\kappa \frac{\langle \boldsymbol{v}_n, \boldsymbol{u}_T \rangle}{\|\boldsymbol{v}_n\|_2 \|\boldsymbol{u}_T\|_2}\right)$
(realizes a k^{th}-order induction head)
$\left[\cdots \middle| \begin{matrix} \mathtt{Emb}(x_n) \\ \frac{\boldsymbol{u}_n}{\|\boldsymbol{u}_n\|_2} \\ \frac{\boldsymbol{v}_n}{\|\boldsymbol{v}_n\|_2} \end{matrix} \middle| \cdots \middle| \begin{matrix} \mathtt{Emb}(x_T) \\ \frac{\boldsymbol{u}_T}{\|\boldsymbol{u}_T\|_2} \\ \frac{\boldsymbol{v}_T}{\|\boldsymbol{v}_T\|_2} \end{matrix} \right]$
$[\, \cdots \mid \mathtt{Emb}(x_n) \mid \cdots \mid \mathtt{Emb}(x_T) \,]$
Layer 3
Match contexts & count
Layers 1 + 2
Capture context
Emb
$x_1, \cdots, x_T$

Intuition

2 layers suffice!

[Ekbote et al. 2025]

How Transformers exhibit In-Context Learning?

Markovian inputs → Transformers

Memory $= k$

Depth $= 2, 3$

✓

Key Takeaways

Shallow depth suffices!

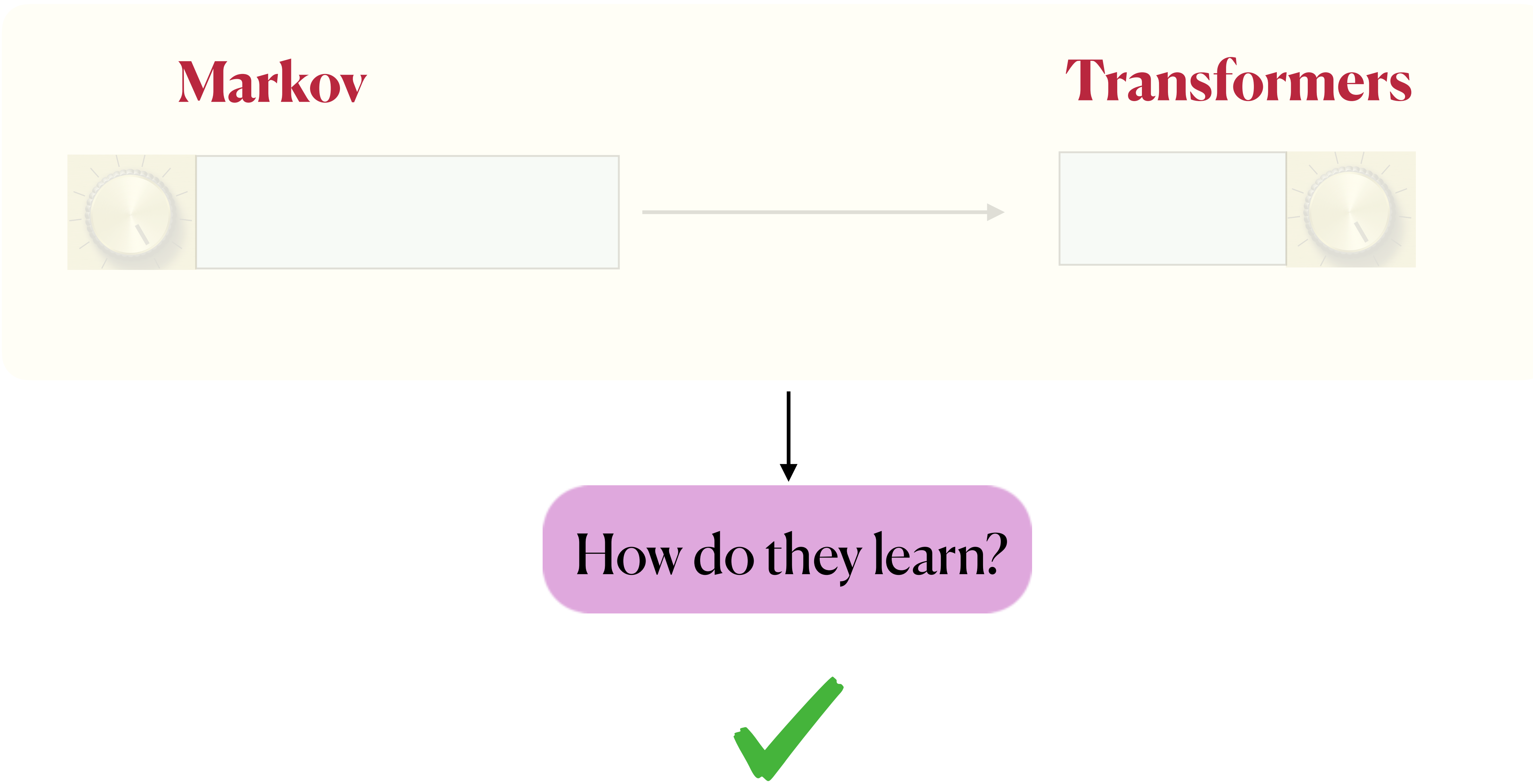
Learning dynamics?

Markovian inputs → Transformers

Memory $= k$

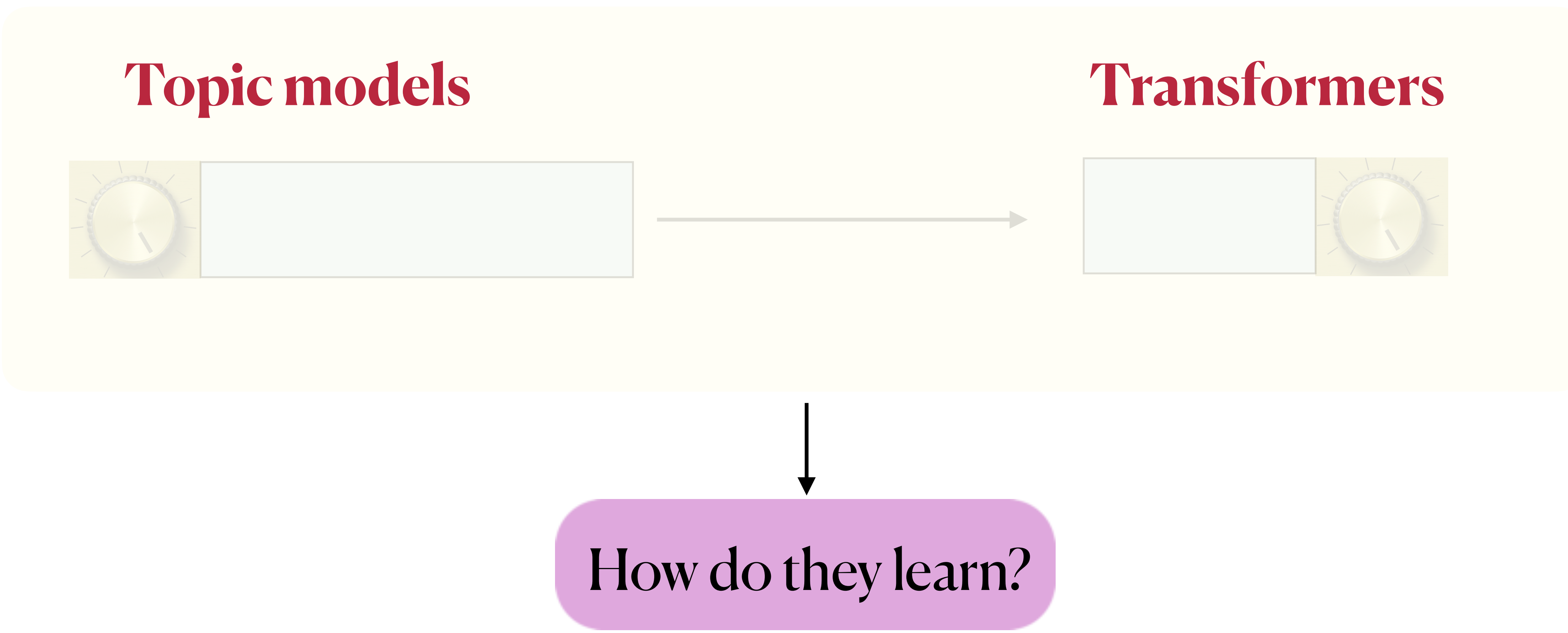
Depth $= 2, 3$

Open....


Markov
Transformers
How do they learn?

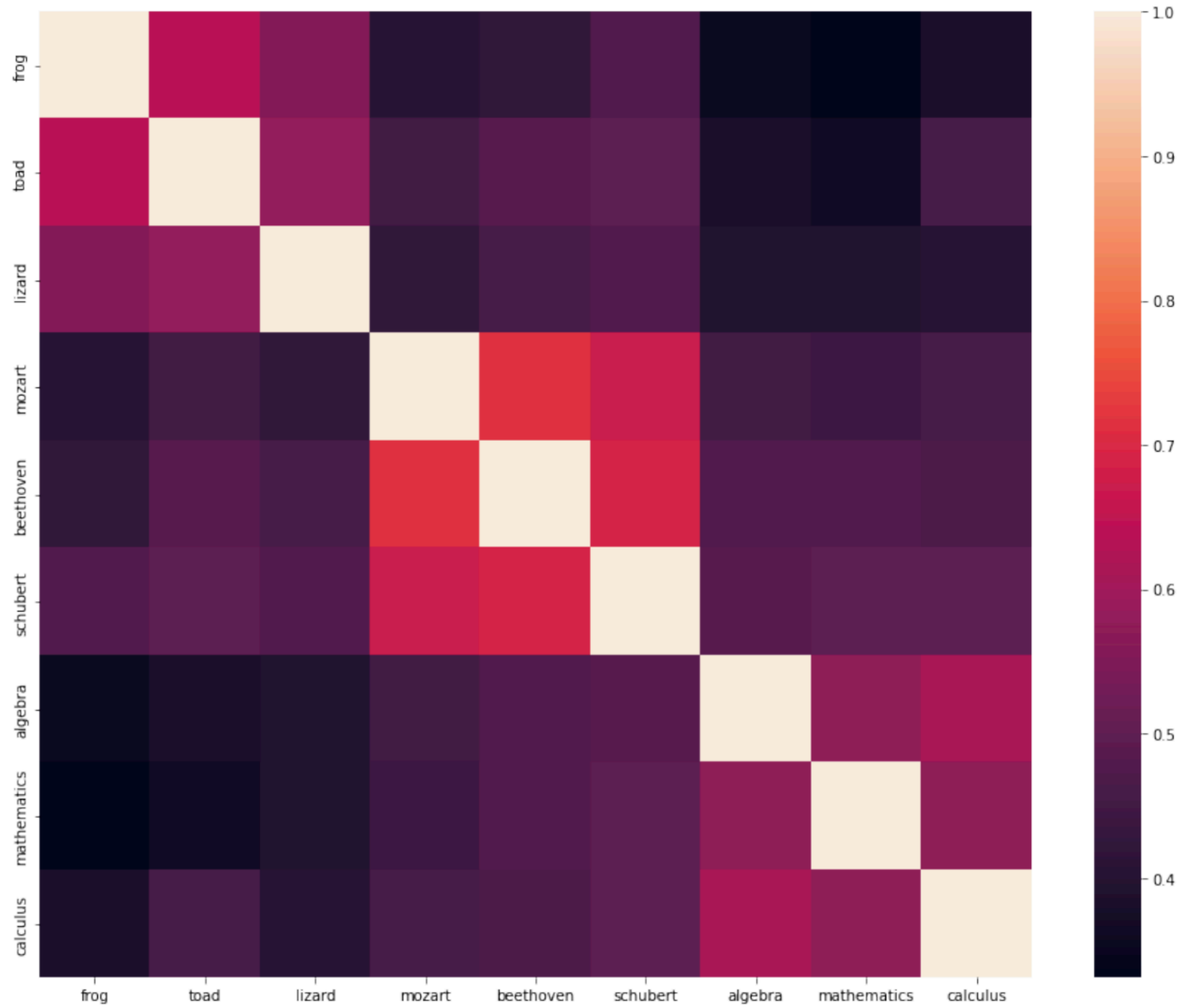
More...

- Attention sinks [Guo et al. 2024]
- Interleaved Markov chains [D'Angelo et al. 2025]


Topic models
Transformers
How do they learn?

Why topic models

Topical structure in BERT



Topic models

Latent Dirichlet Allocation (LDA) [Blei et al. 2003].

Topic models

Latent Dirichlet Allocation (LDA) [Blei et al. 2003].

[Sontag & Roy, 2011; Awasthi & Risteski, 2015; Arora et al. 2016; Tosh et al. 2021; Luo et al., 2022, Li et al. 2023; Reuter et al. 2024]

Input data: LDA

To generate a document:

1. Randomly sample a set of τ distinct topics from $[T]$.
2. For each word,
   1. Randomly sample a topic.
   2. Sample a word from the vocabulary of the topic .

[Li et al. 2023, Lu et al. 2023]

Input data - assumption

Assumption: Each word belongs to exactly one topic.

To generate a document X:

1. Randomly sample a set of τ distinct topics from $[T]$.
2. For each word,
    1. Randomly sample a topic.
    2. Sample a word from the vocabulary of the topic .

[Li et al. 2023, Lu et al. 2023]

Input data - masking

Randomly mask the tokens in the document.

Input data - masking

Randomly mask the tokens in the document.

Bangalore's traffic is like Reviewer #2. Always pleasant to deal with.

Input data - masking

Randomly mask the tokens in the document.

Bangalore's traffic is like Reviewer #2. Always pleasant to deal with.

Input data - masking

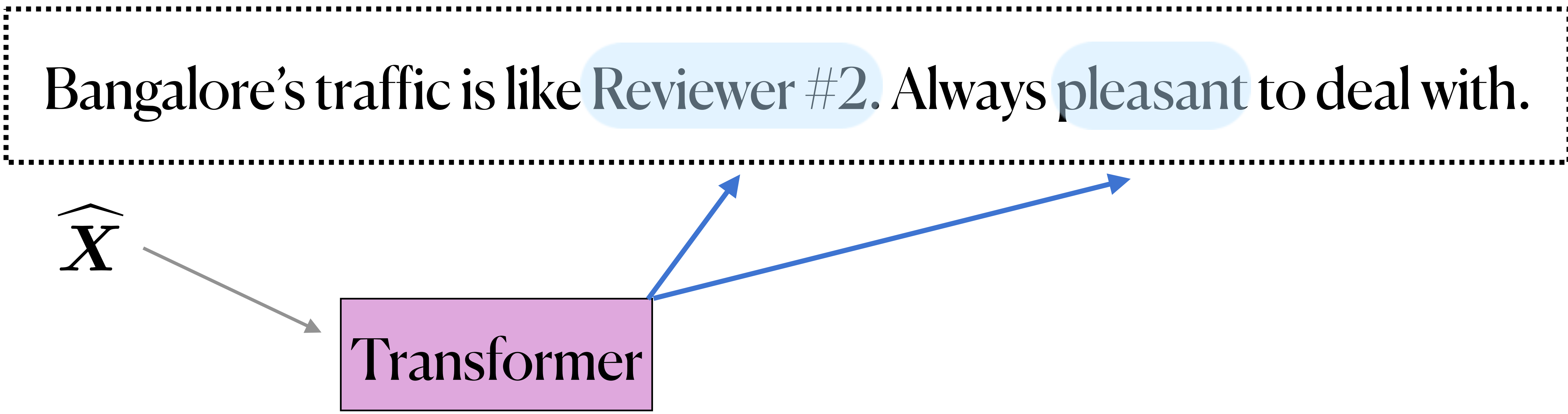
Randomly mask the tokens in the document

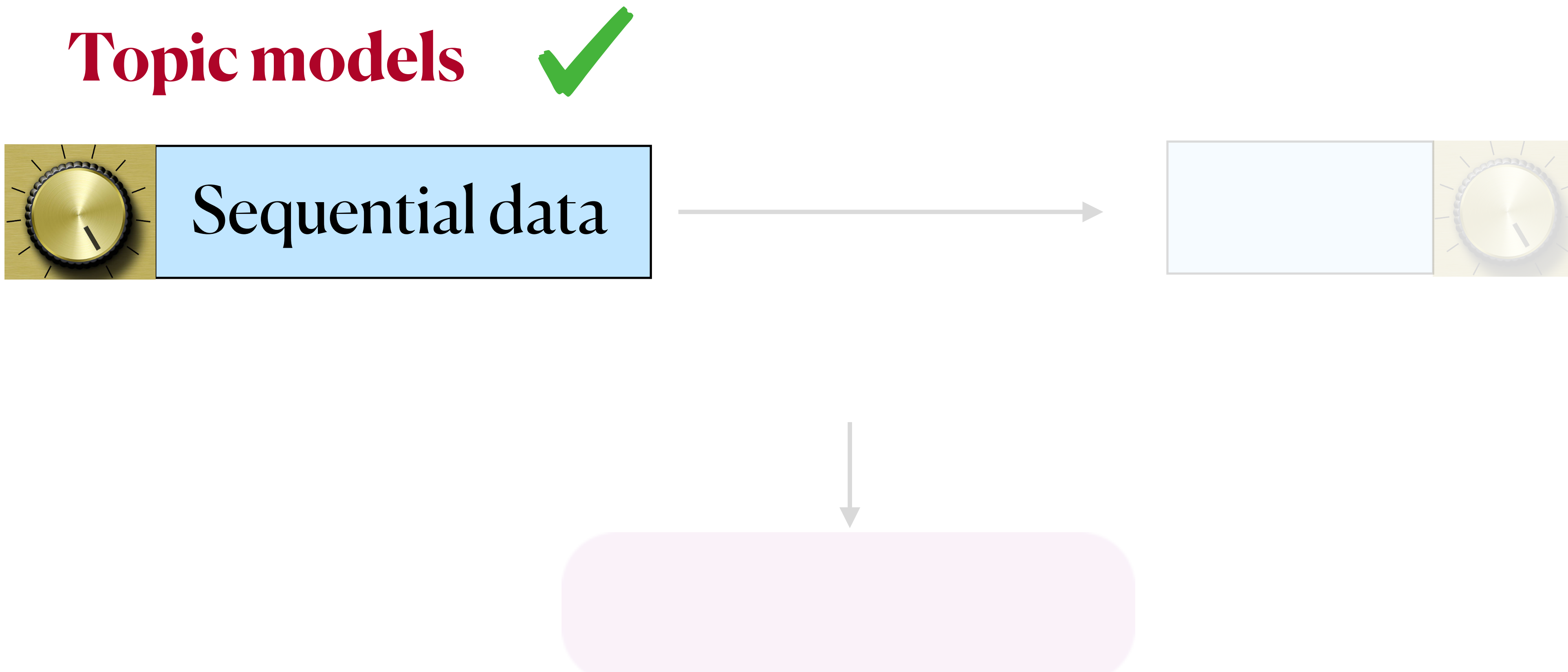
Bangalore's traffic is like Reviewer #2. Always pleasant to deal with.

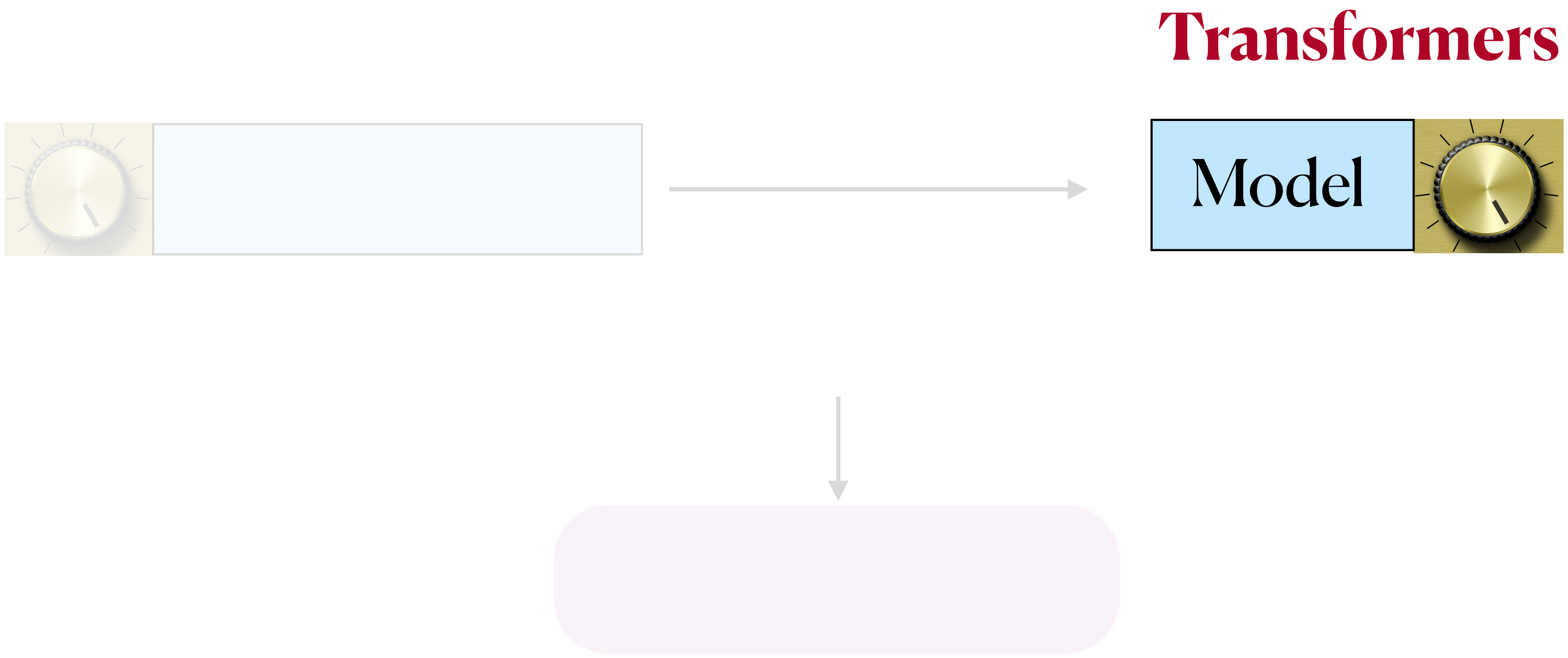
$\widehat{X}$

Masked language modeling

Predict the masked tokens using the unmasked ones




Topic models
Sequential data


Transformers
Model

Single-layer transformer

Encoder model. Embedding and attention layer without MLP.

Single-layer transformer

$$f(\boldsymbol{Z}) = \boldsymbol{W}_O\,(\boldsymbol{W}_V\boldsymbol{Z})\,\mathsf{SoftMax}\left((\boldsymbol{W}_K\boldsymbol{Z})^\top \boldsymbol{W}_Q\boldsymbol{Z}\right) + \boldsymbol{b}$$

Attention layer

$$\boldsymbol{Z} = \boldsymbol{W}_E\widehat{\boldsymbol{X}}$$

Embedding layer

Single-layer transformer

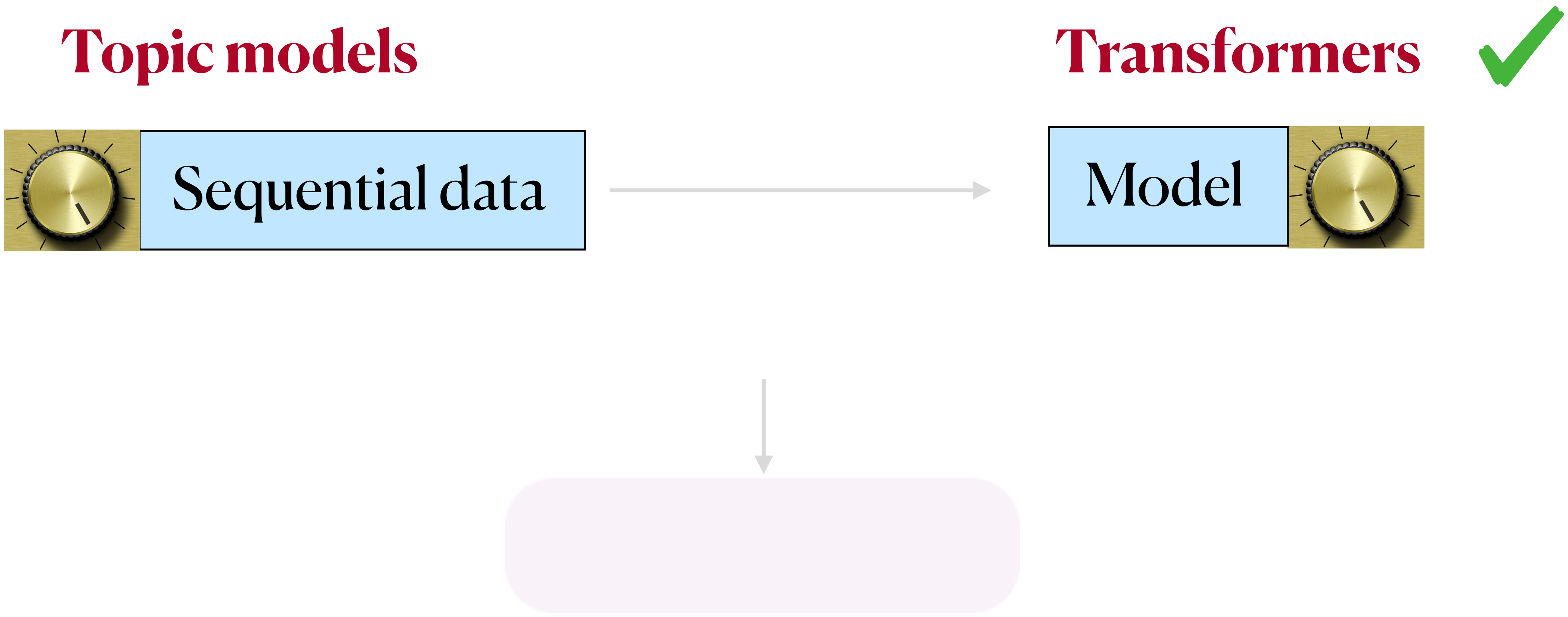
$$f(\boldsymbol{Z}) = \boldsymbol{W}_O (\boldsymbol{W}_V \boldsymbol{Z}) \, \mathsf{SoftMax} \left((\boldsymbol{W}_K \boldsymbol{Z})^\top \boldsymbol{W}_Q \boldsymbol{Z} \right) + \boldsymbol{b}$$

$$\boldsymbol{Z} = \boldsymbol{W}_E \widehat{\boldsymbol{X}}$$

Key parameters

$\boldsymbol{W}_E$

$\boldsymbol{W}_V, (\boldsymbol{W}_K, \boldsymbol{W}_Q)$



Topic models
Transformers
Sequential data
Model

Analysis

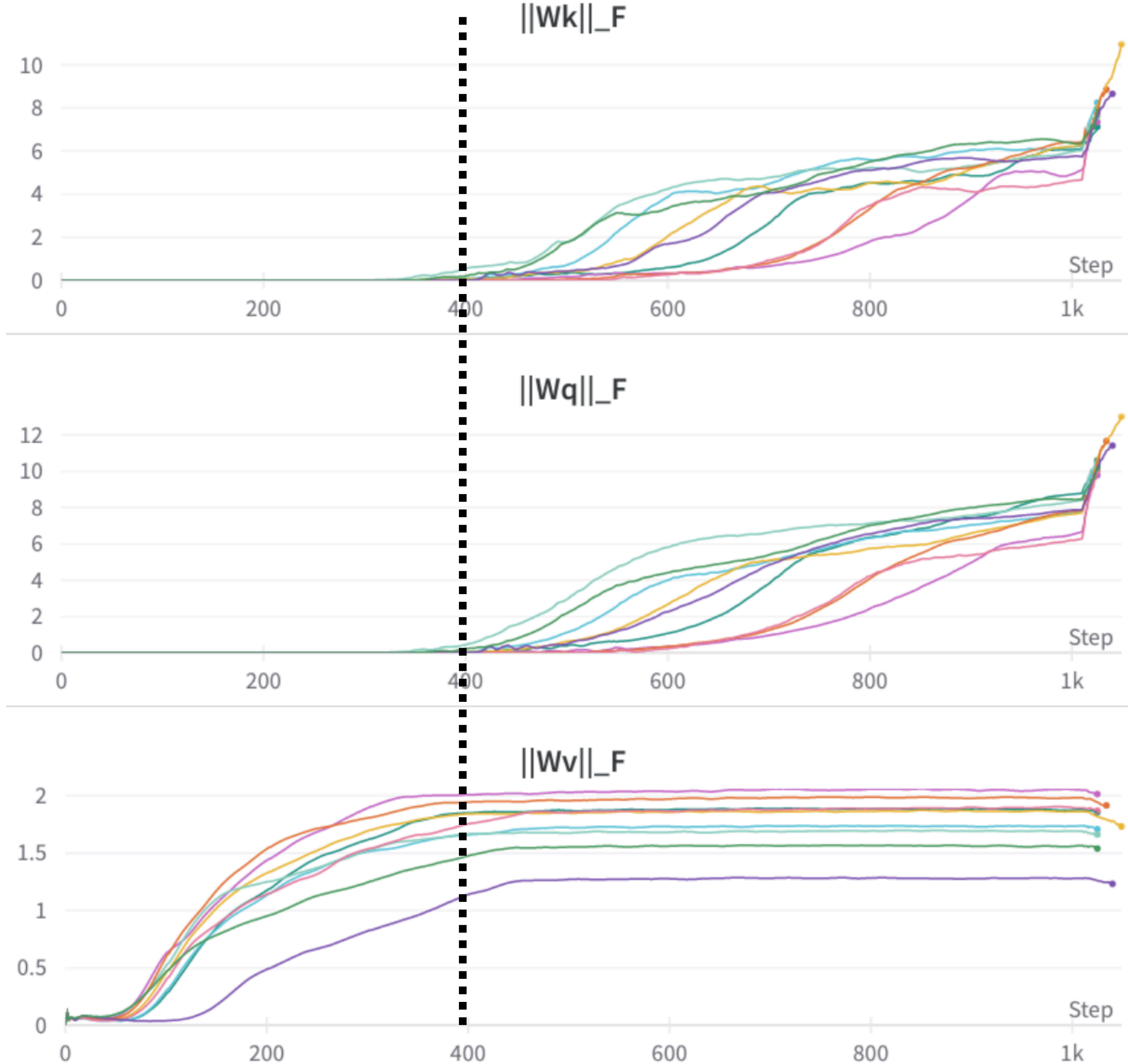
Freeze the embedding weights to one-hot encodings.

Analysis

Two-stage dynamics of value and (key, query) matrices

Stage 1:

- Value matrix grows.
- (Key, Query) near zero.



Stage 2:

- Value matrix stays constant.
- (Key, Query) grow.

Main result - Stage 1

Value vectors encode the topic structure (informal)

Assuming the attention-weights to be frozen to uniform and token embeddings to one-hot, training only the value matrix $\boldsymbol{W}_V$ is a convex problem. The optimal solution satisfies that

$$(\boldsymbol{W}_V^*)_{\text{same-topic}} = (\boldsymbol{W}_V^*)_{\text{diff-topic}} + c, \quad c > 0.$$

Main result - Stage 1

Value vectors encode the topic structure (informal)

Assuming the attention-weights to be frozen to uniform and token embeddings to one-hot, training only the value matrix $\boldsymbol{W}_V$ is a convex problem. The optimal solution satisfies that

$$(\boldsymbol{W}_V^*)_{\text{same-topic}} = (\boldsymbol{W}_V^*)_{\text{diff-topic}} + c, \quad c > 0.$$

$$\Downarrow$$

While predicting masked token, unmasked tokens of similar topic contribute more

Main result - Stage 2

Attention weights encode the topic structure (informal)

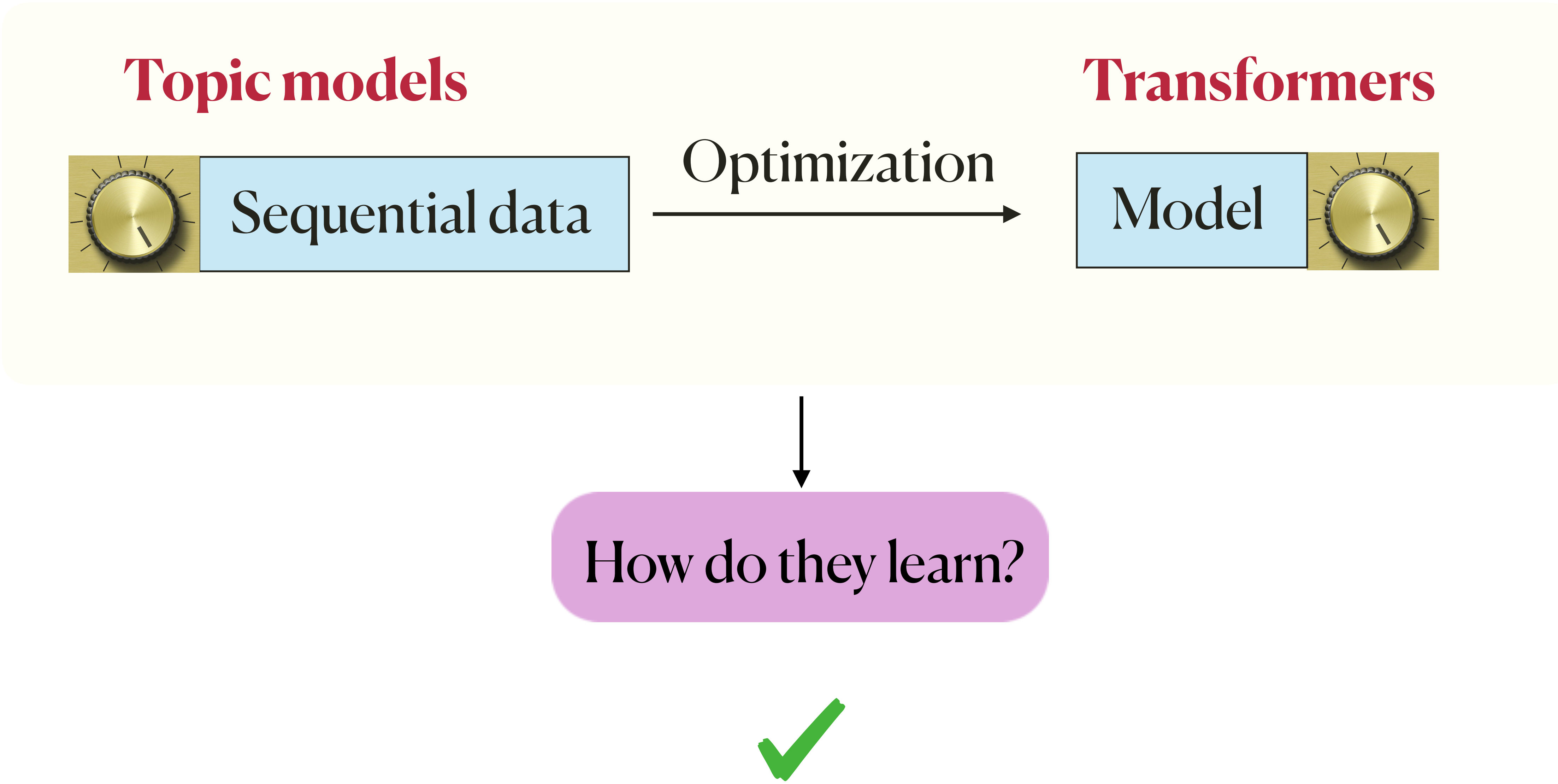
Assuming value matrix to be frozen at the optimum from Stage 1 and token embeddings to one-hot, training the (key, query) matrices yield attention matrices that have higher attention across words of same topic than those from different ones.

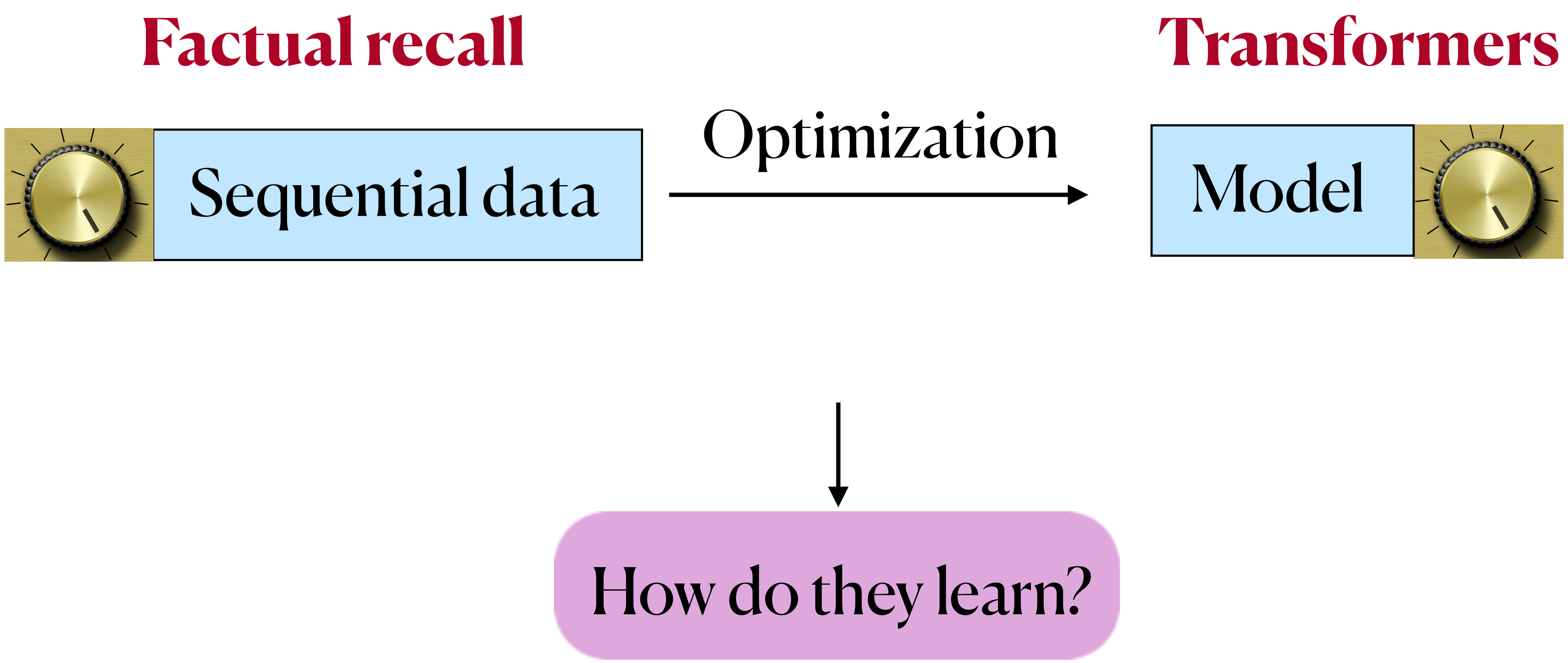
Key takeaways

Topic structure can be encoded in either token embeddings and self-attention

Attention layer encodes this structure in a two stage process

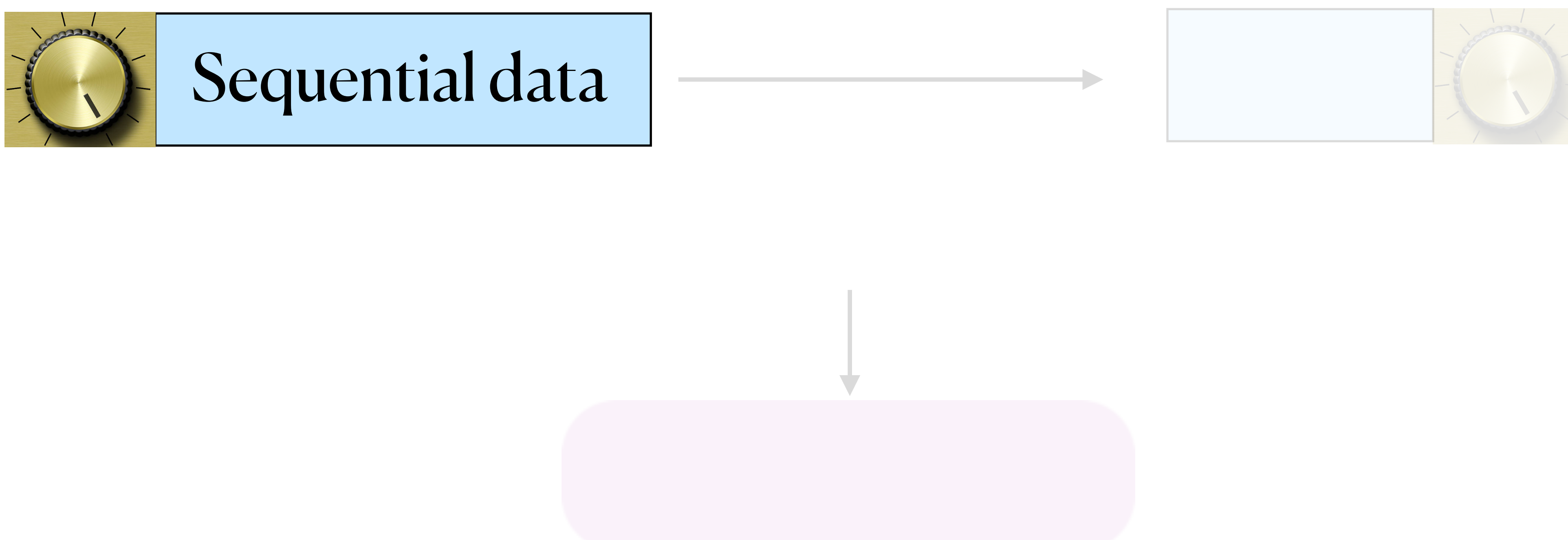
(Consistent pattern for different loss functions, optimizers, and real-world datasets)


Topic models
Transformers
Sequential data
Optimization
Model
How do they learn?



[Nichani et al. 2024]

Factual recall



What is Factual recall?

What is the capital of France?

In which country was Marie Curie born?

Paris

Poland

The association (France, capital, Paris) is stored somewhere within the weights

Factual recall

Motivating Questions:

- How do LLMs learn to store such facts within their parameters?
- What is the relationship between parameter count and the number of facts?

Factual recall

Motivating Questions:

- How do LLMs learn to store such facts within their parameters?
- What is the relationship between parameter count and the number of facts?

Main results

- Theoretical model for analyzing factual recall via **associative memories**
- Proving that transformers can memorize facts with near-optimal capacity

[Nichani et al. 2024]

Associative memories

- Input vocabulary $[N]$ and output vocabulary $[M]$
- Ground truth *association function* $f^* : [N] \to [M]$
- Embedding vectors $\{e_x\}_{x \in [N]}$ and unembedding vectors $\{u_y\}_{y \in [M]}$, sampled uniformly on sphere
- Transformer model $F : \mathbb{R}^d \to \mathbb{R}^d$
- Argmax decoding: $\hat{f}(x) = \arg\max_{y \in [M]} u_y^\top F(e_x)$
- Perfect memorization: $f^*(x) = \hat{f}(x), \forall x \in [N]$
- How many parameters does F need to achieve perfect memorization?

Associative memories

$[N]$ $[M]$

$$f^* : [N] \to [M]$$

$\{e_x\}_{x \in [N]}$

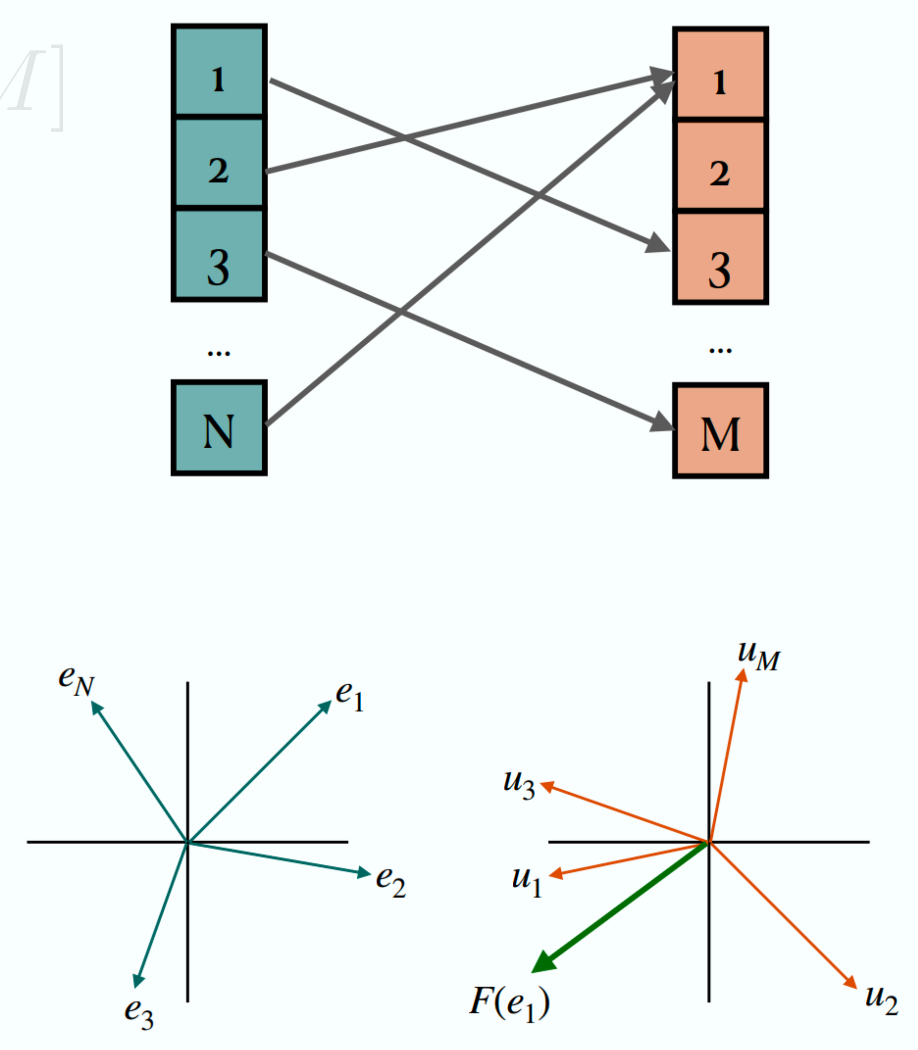
$\{u_y\}_{y \in [M]}$

$F : \mathbb{R}^d \to \mathbb{R}^d$

$$\hat{f}(x) = \arg\max_{y \in [M]} u_y^\top F(e_x)$$

$$f^*(x) = \hat{f}(x), \forall x \in [N]$$

F



Model from Cabannes et al., 2024.

Associative memories: Main results

Linear associative memory: $F(z) = Wz, W \in \mathbb{R}^{d \times d}$

Theorem: Assume f^* is injective. If $N = \tilde{O}(d^2)$, then with high probability there exists a W such that $\hat{f}(x) = f^*(x), \forall x \in [N]$.

- Obtained by construction $W = \sum_{x \in [N]} u_{f^*(x)} e_x^\top$. Superposition of outer products

Associative memories: Main results

$$F(z) = Wz, W \in \mathbb{R}^{d \times d}$$

f^* $N = \tilde{O}(d^2)$

W $\hat{f}(x) = f^*(x), \forall x \in [N].$

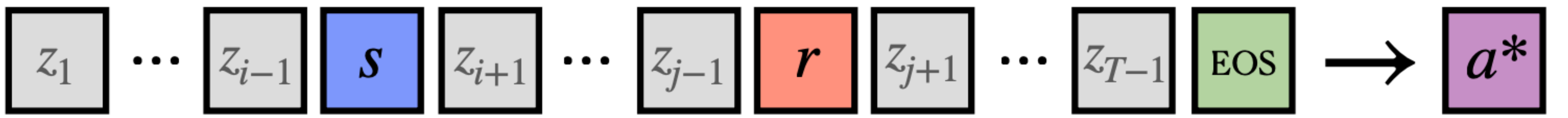
$$W = \sum_{x \in [N]} u_{f^*(x)} e_x^\top .$$

MLP associative memory: $F(z) = V^\top \sigma(Wz)$ for $V, W \in \mathbb{R}^{m \times d}$

Theorem: If $N = \tilde{O}(md)$, then with high probability there exists a W such that $\hat{f}(x) = f^*(x), \forall x \in [N].$

- Improvement over one-hot embeddings, which can only store $N \propto d$ associations
- Matching lower bounds

Synthetic task for Factual recall



The capital of France is Paris

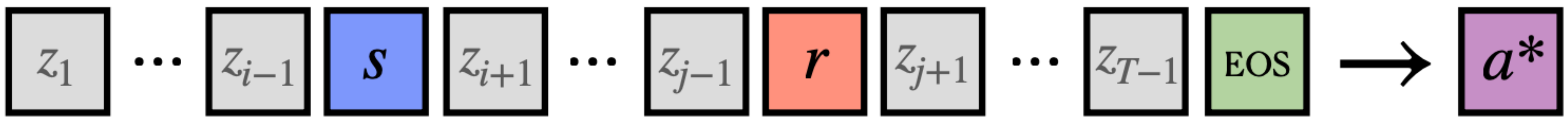
$s \in \mathcal{S}$: subject token

$r \in \mathcal{R}$: relation token

$a^*(s, r) \in \mathcal{A}_r$: attribute/fact to be stored

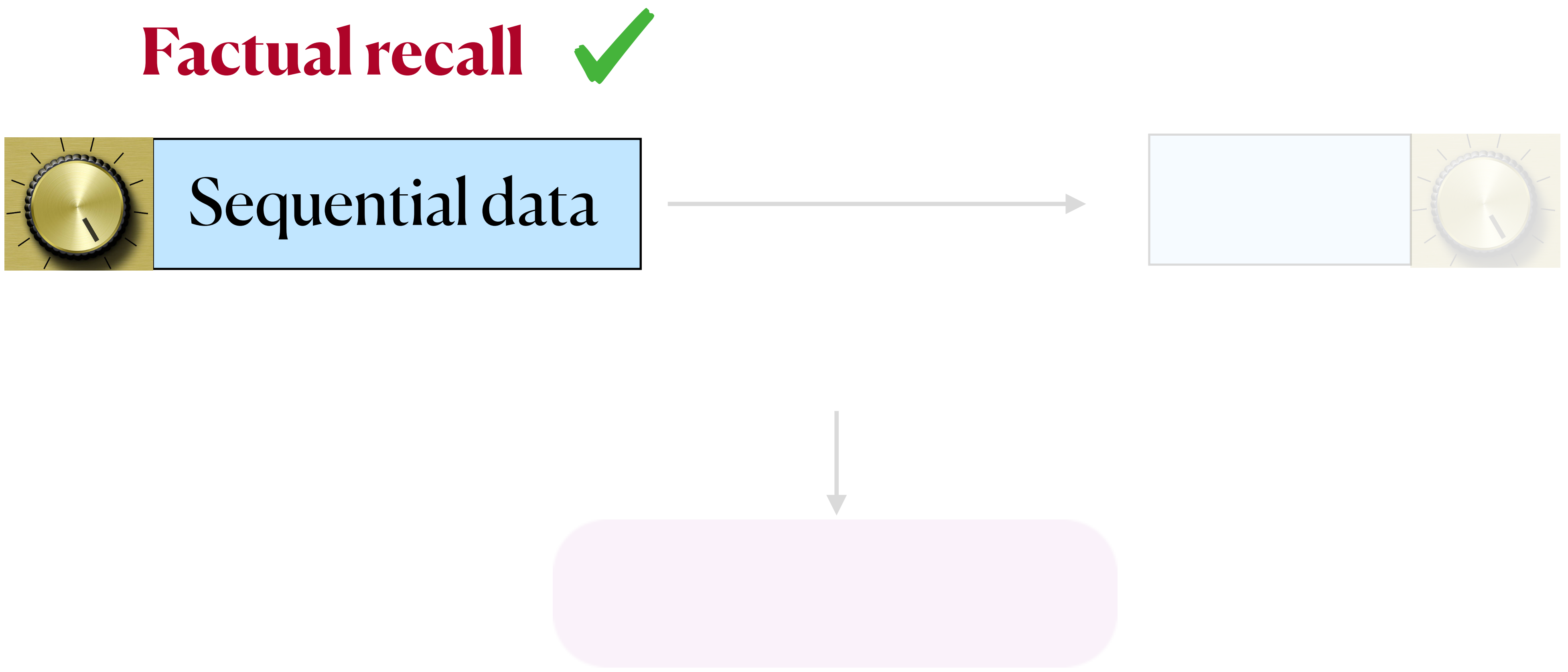
$z_i \in \mathcal{N}$: noise tokens

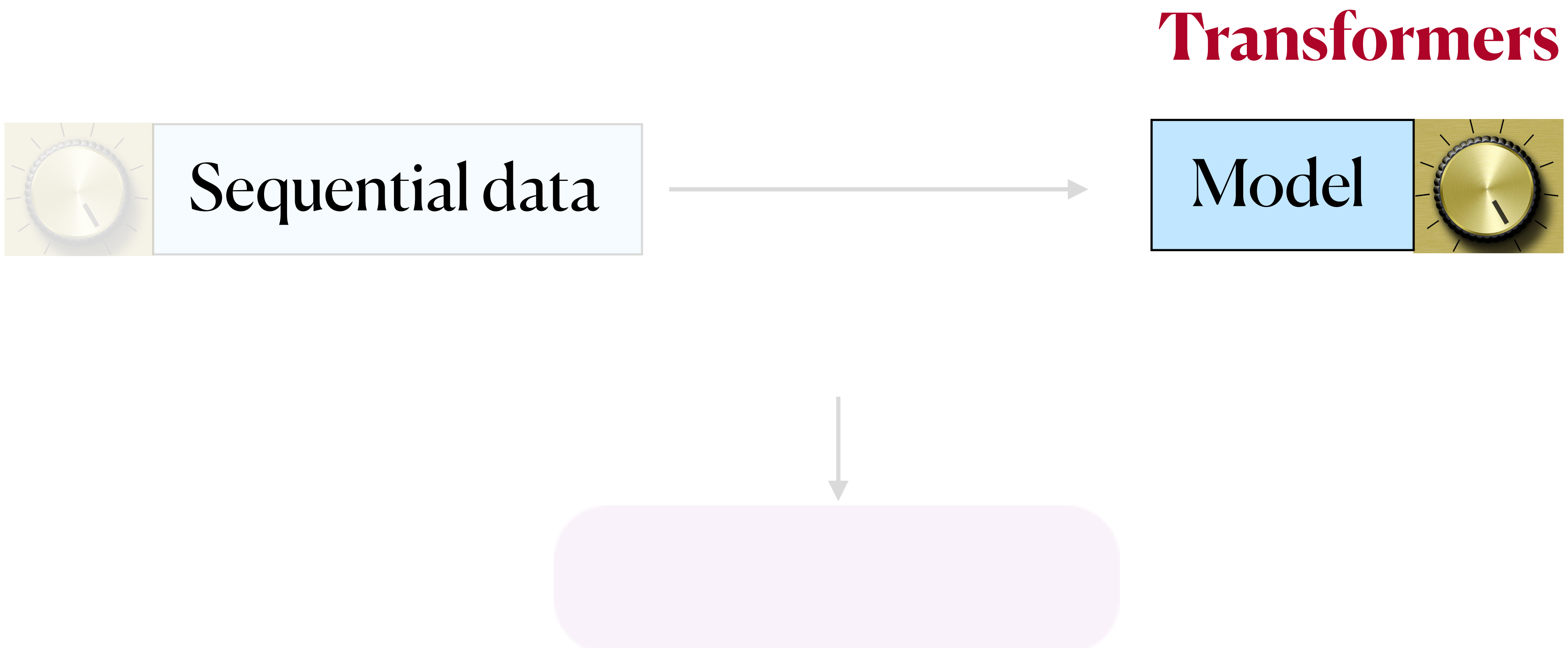
Synthetic task for Factual recall



The capital of France is Paris

How many parameters do Transformers need to solve this?

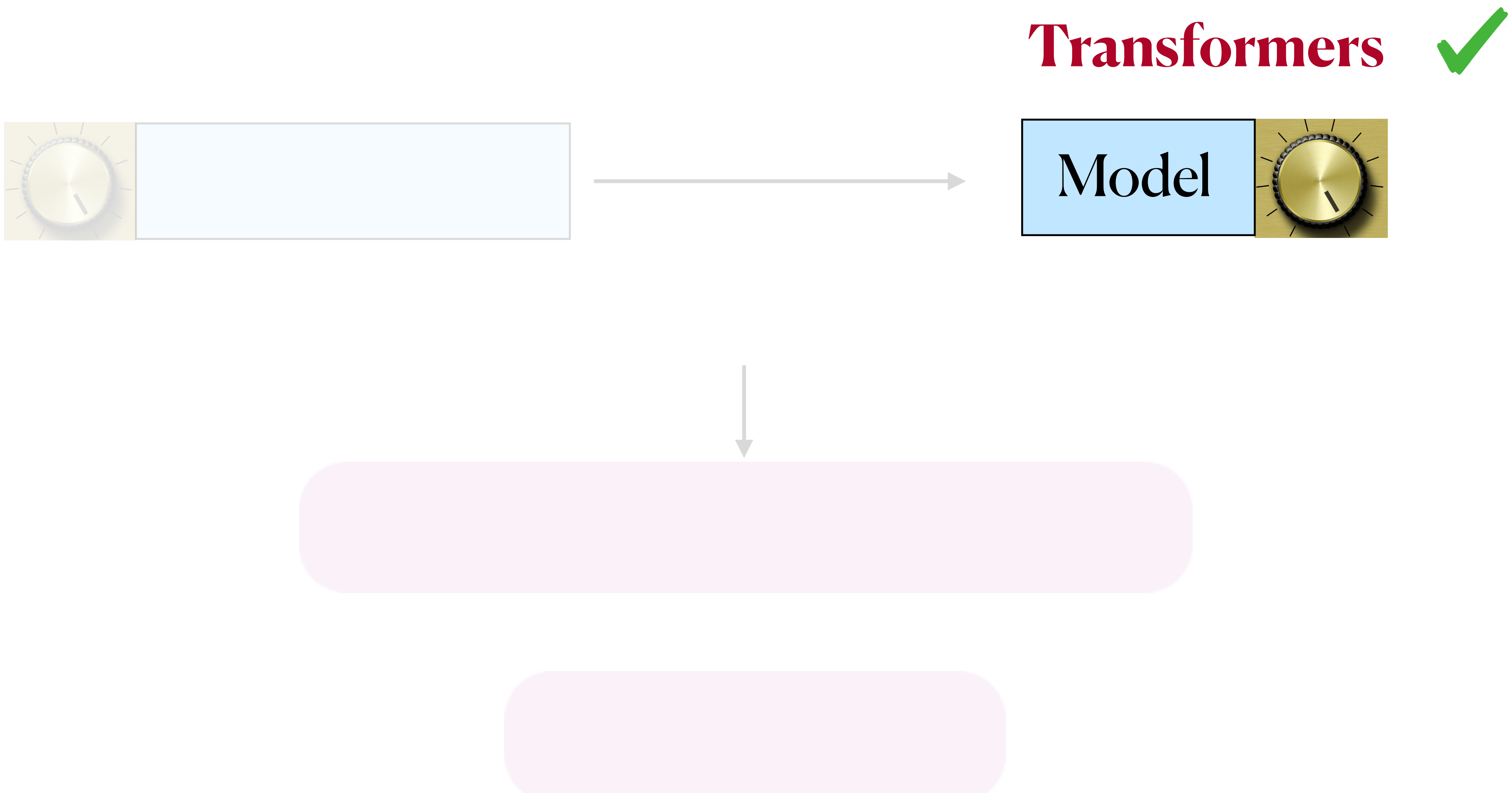

Factual recall
Sequential data


Transformers
Sequential data
Model

Single layer transformer with MLP

- Random embeddings

- Embedding dimension d, head dimension d_h, MLP width m, H heads


Transformers
Model

How many parameters to they need?

How do they learn?

How many parameters to they need?

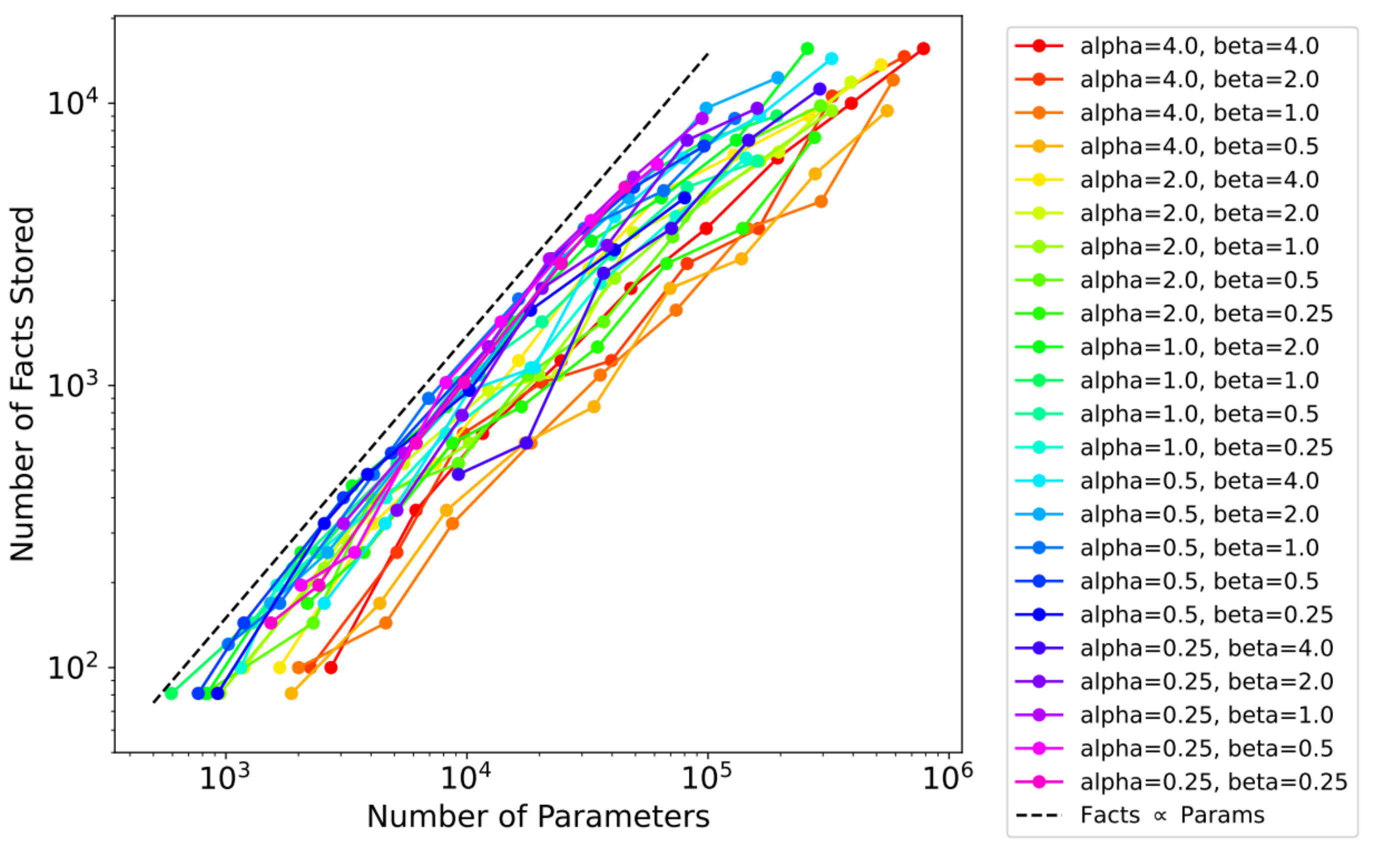
Theorem (informal)

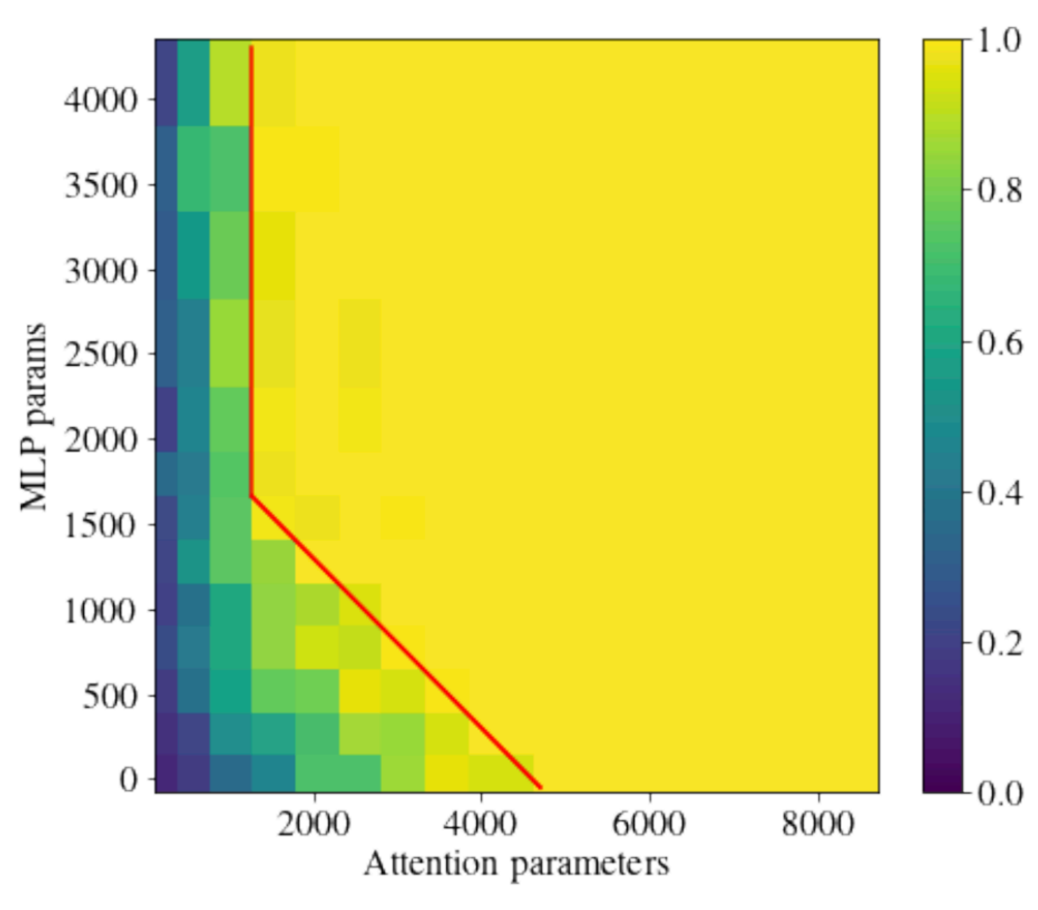
- Attention + MLP: $Hd_h \gtrsim S + R$ and $md \gtrsim SR$ suffices
- Attention-only: $d \gtrsim R + A_{\max}$ and $Hd_h \gtrsim S$ suffices $(A_{\max} \triangleq \max_r |\mathcal{A}_r|)$

How many parameters to they need?

Theorem (informal)

- Attention + MLP: $Hd_h \gtrsim S + R$ and $md \gtrsim SR$ suffices
- Attention-only: $d \gtrsim R + A_{\max}$ and $Hd_h \gtrsim S$ suffices $(A_{\max} \triangleq \max_r |\mathcal{A}_r|)$



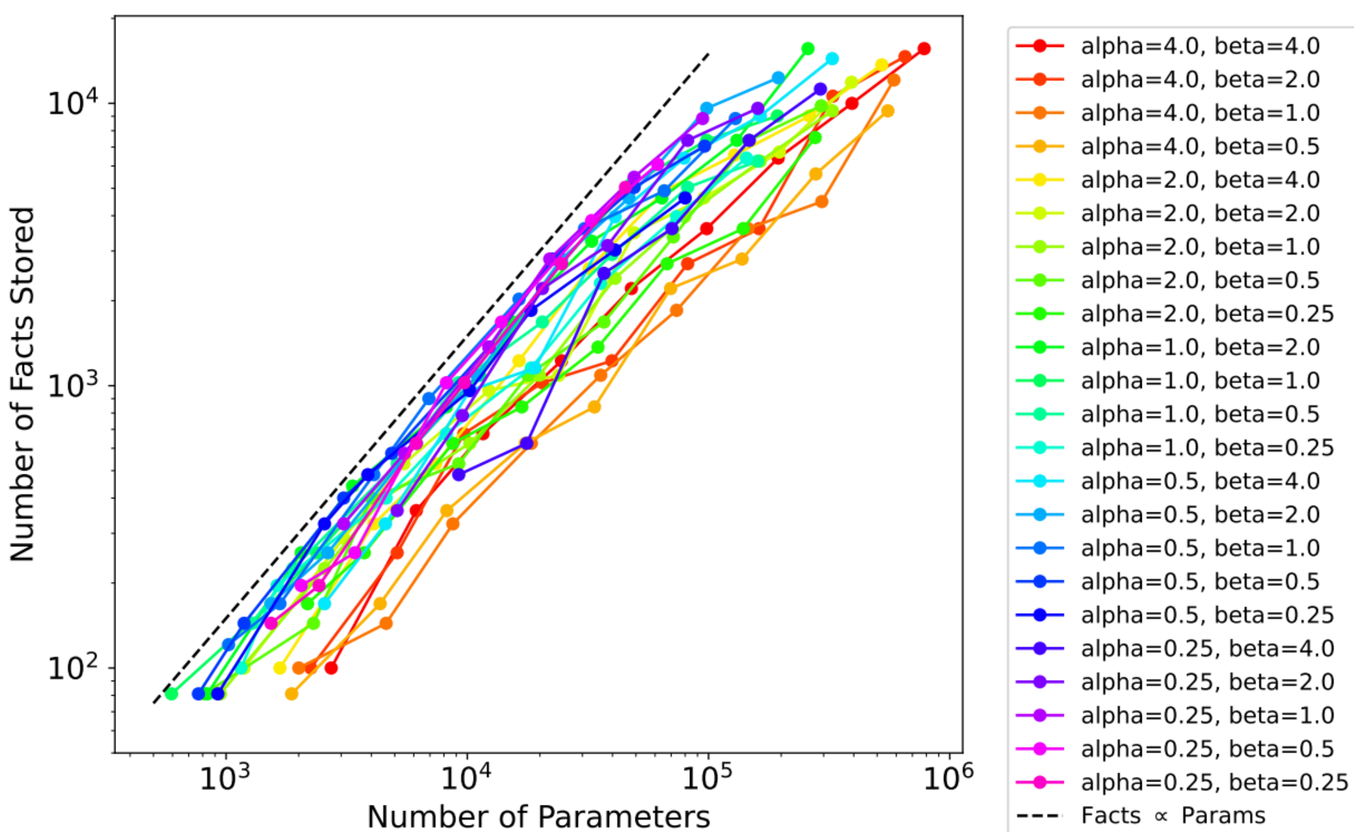


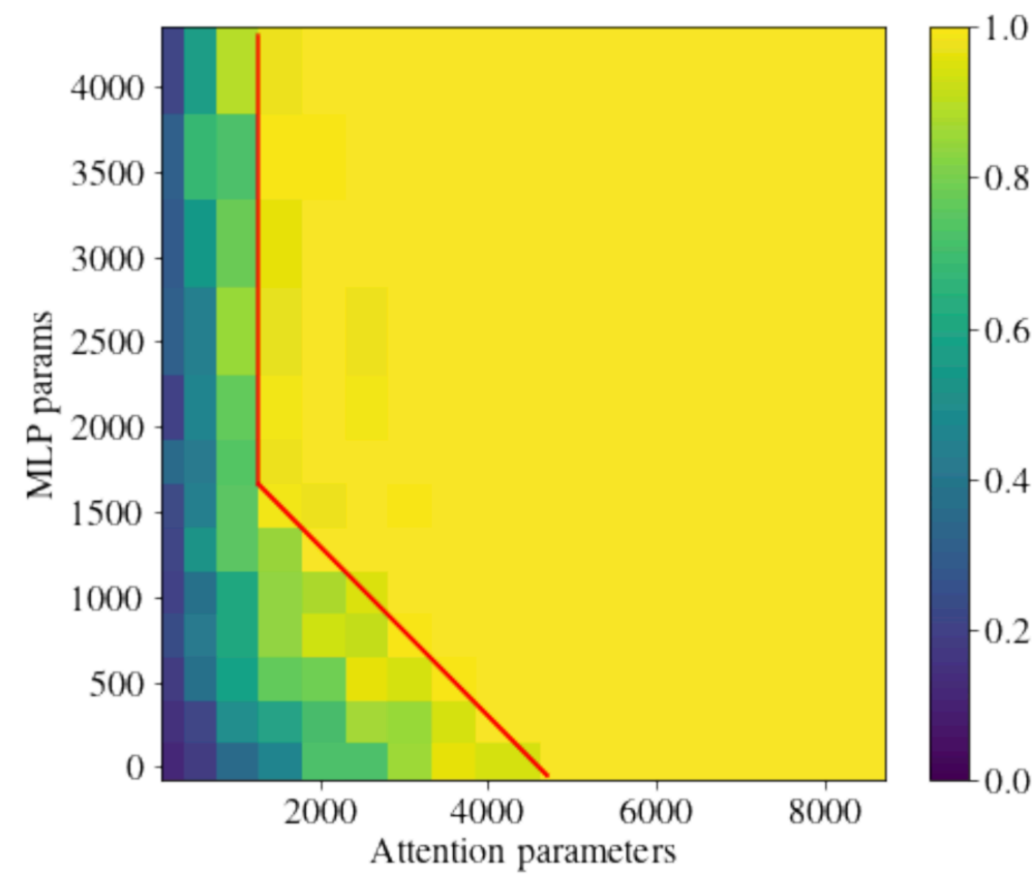
Number of facts stored scale linearly with parameter size

$$Hd_h \gtrsim S + R \qquad md \gtrsim SR$$

$$d \gtrsim R + A_{\max} \qquad Hd_h \gtrsim S \qquad (A_{\max} \triangleq \max_r |\mathcal{A}_r|)$$

[Allen-Zhu et al. 2024]





How many parameters to they need? ✓

How do they learn?

How do they learn?

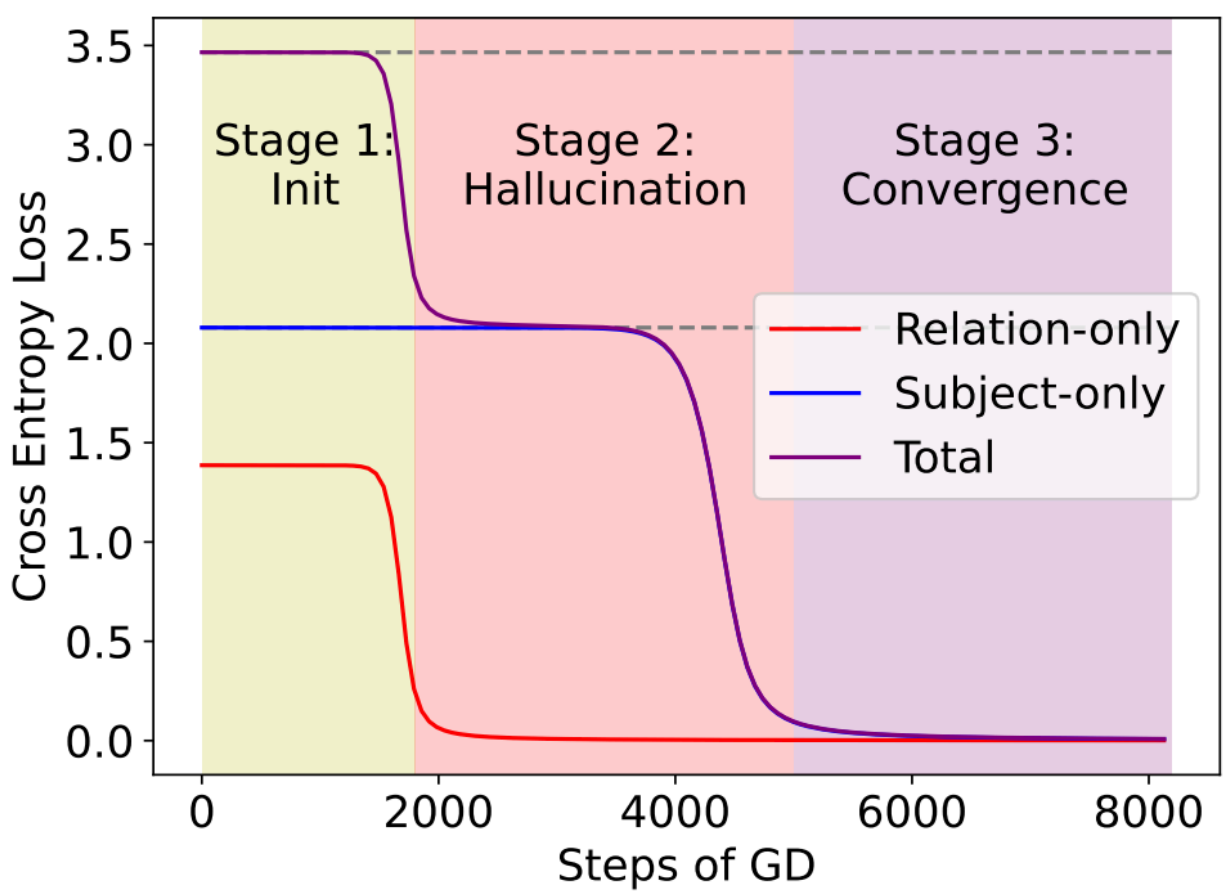
- **Linear attention, one-hot embeddings**
- Gradient flow with initialization $W_{OV}(a, z), w_{KQ}(z) \approx \alpha > 0$

Theorem (informal)

- Global convergence to zero loss
- Intermediate phase where model predicts with $p(a \mid r)$ instead of $p(a \mid s, r)$

How do they learn?

$p(a \mid r)$ $p(a \mid s, r)$



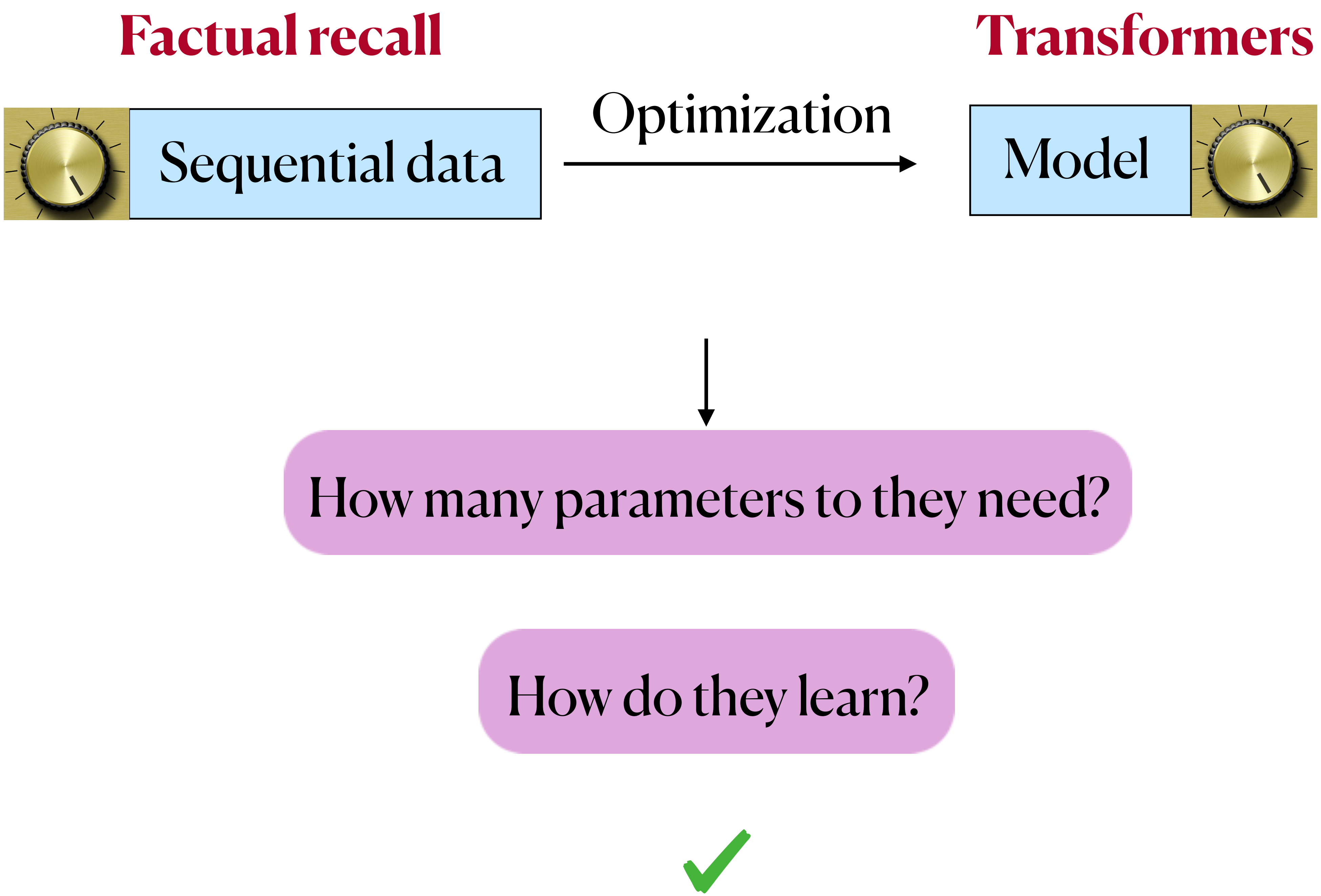
Hallucination stage where prediction is only based on the relation

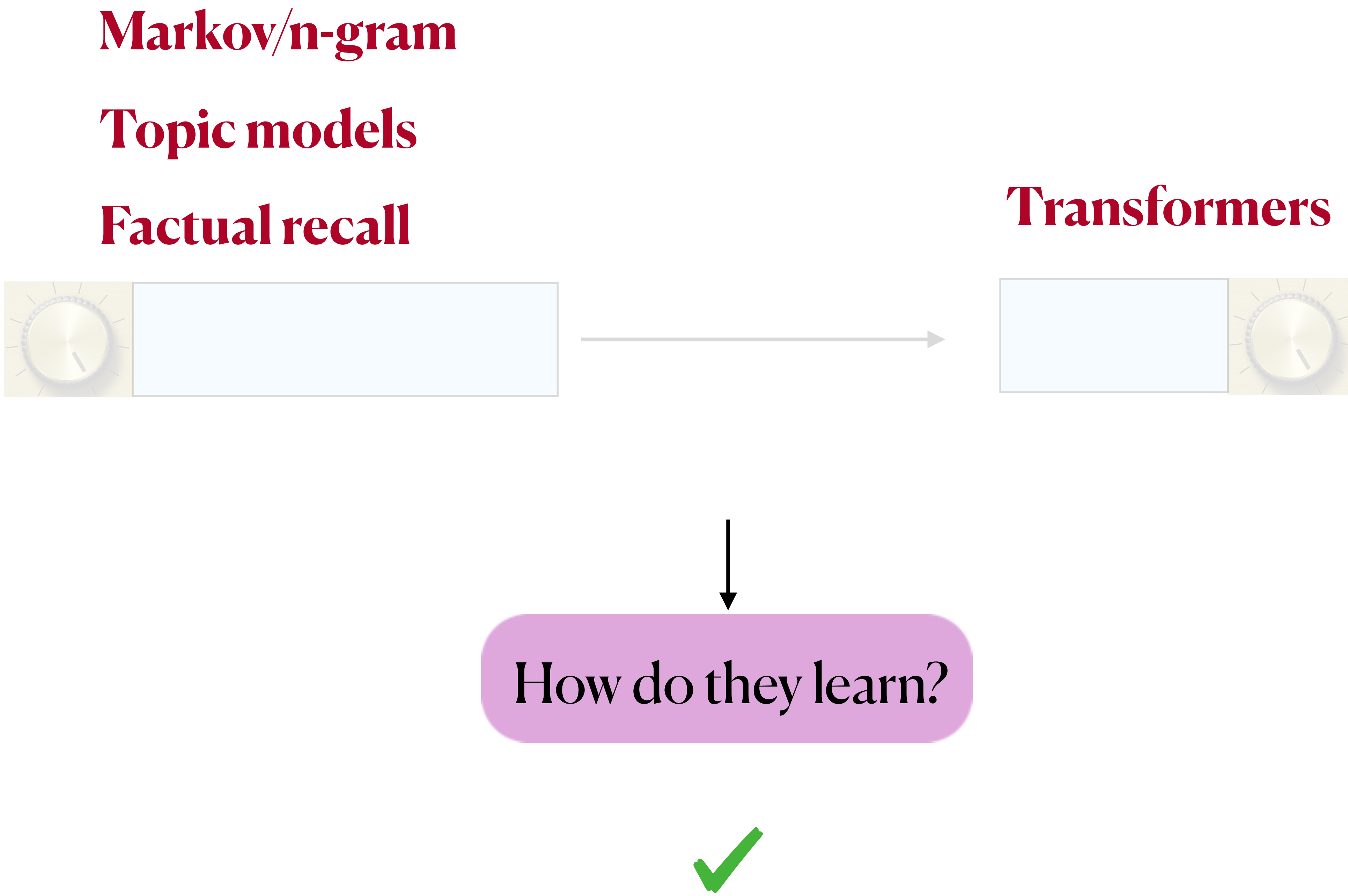
Stage-wise with sub-n grams

Key Takeaways

Transformers memorize facts with near-optimal capacity

Sequential learning behavior—first using only the relation, then both subject and relation

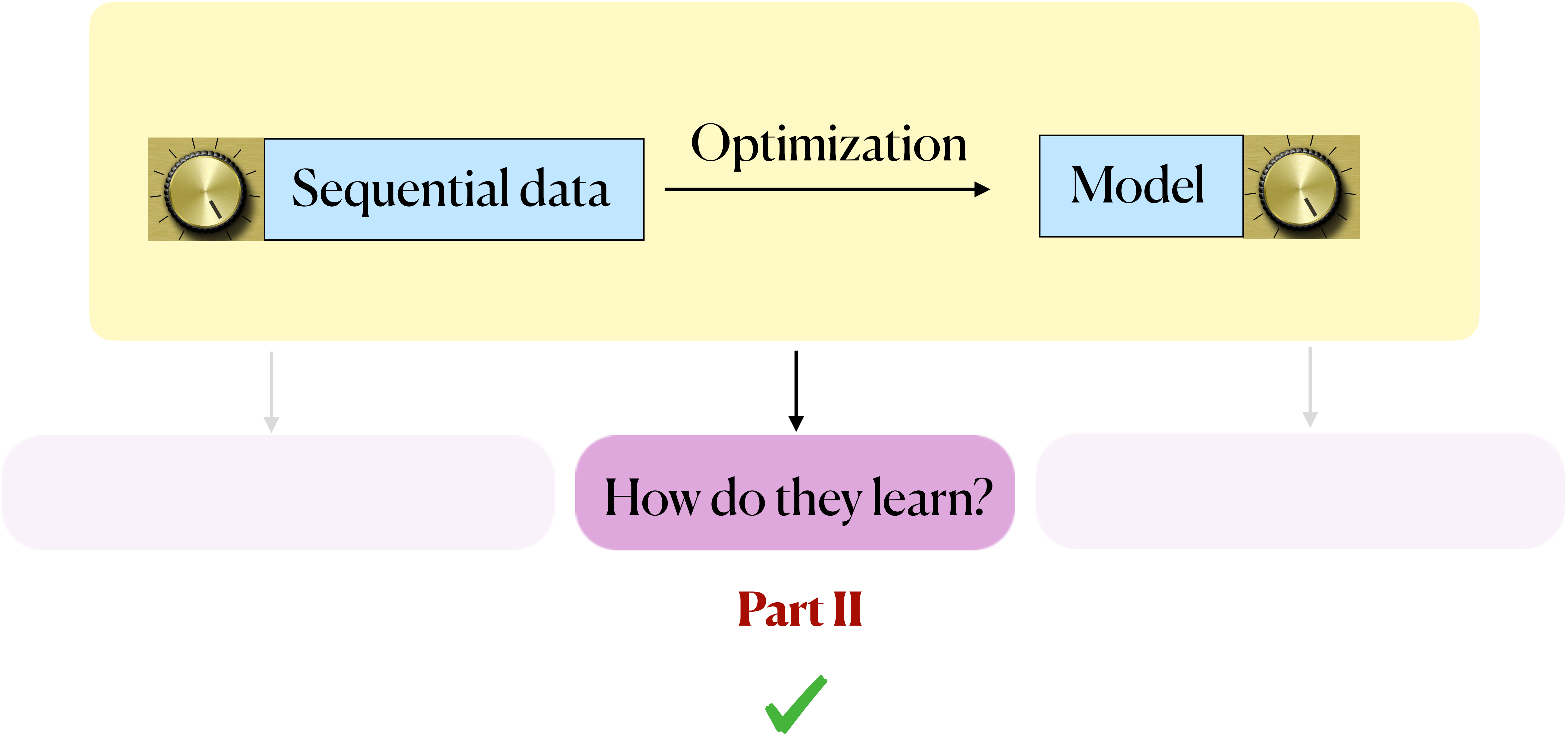

Factual recall
Transformers
Sequential data
Optimization
Model
How many parameters to they need?
How do they learn?

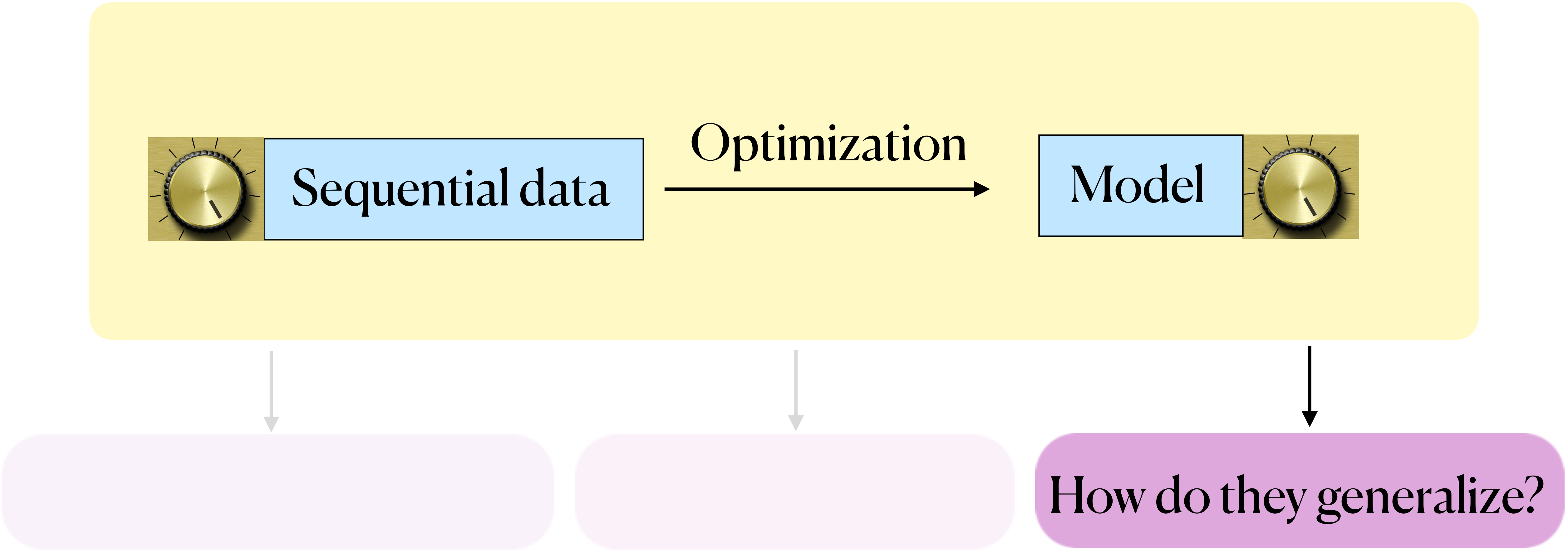

Markov/n-gram
Topic models
Factual recall
Transformers
How do they learn?

More...

Compositional/Multi-hop reasoning **Transformers**

[Wang et al. 2025, Guo et al. 2024]


Sequential data
Optimization
Model
How do they learn?
Part II


Sequential data
Optimization
Model
How do they generalize?

Part III

Part III
Generalization

(properties of practical solutions)

Recap

Part I: representability — *existence of solutions?*

- Upper bound: e.g. automata; induction head.
- Lower bound: depth (Transformer) and width (Transformer + RNNs).

↓

Provable benefit of depth/CoT: length **generalization**

Part II: optimization — *searching for solutions?*

- Can find (local) optima for e.g. Markov data, topic model, linear regression.
- Caution: different optimal solutions may **generalize** differently.

Part III — (a tiny bit about) generalization

1. Length generalization

- Setup: train on small, test on large.
- Challenging — causes & mitigations?
  - **Proper measure of size**: e.g. parity: $\sum_i x_i$ matters (more than T).
  - **RASP-L conjecture**: short RASP-L program → length generalize ✓.

[Zhou et al. 23]

  - Takeaways: 1) positions cause issues; 2) potential new "hierarchy".

e.g. RASP-L is more fine-grained

parity (×), majority (✓) $\in TC^0 \backslash AC^0$

Part III — (a tiny bit about) generalization

1. Length generalization

- Setup: train on small, test on large.
- Challenging — causes & mitigations?
  - **Proper measure of size**: e.g. parity: $\sum_i x_i$ matters (more than T).
  - **RASP-L conjecture**: short RASP-L program → length generalize ✓.
  - Takeaways: 1) positions cause issues; 2) potential new "hierarchy".

↑

e.g. RASP-L is more fine-grained

parity (×), majority (✓) $\in TC^0 \backslash AC^0$

❖ C-RASP [Yang et al. 24]

Part III — (a tiny bit about) generalization

2. Same-length generalization

- Setup: same-length OOD sequences.
- Can the model always learn a *robust* solution?
  - No, even for a naive task (2-layer representation; easy to learn).

Part III — (a tiny bit about) generalization

2. Same-length generalization

- Setup: same-length OOD sequences.
- Can the model always learn a *robust* solution?
  - No, even for a naive task (2-layer representation; easy to learn).

Flip-flop: a task that's easy to represent and learn.

→ 2-layer, constant-size **(Part I)**

→ great in-distribution accuracy **(Part II)**

… yet with imperfect OOD generalization.

[Liu et al. 23]

Part III — (a tiny bit about) generalization

2. Same-length generalization

- Setup: same-length OOD sequences.
- Can the model always learn a *robust* solution?
    - No, even for a naive task — inherent limitations of attention.

Flip-flop: a task that's easy to represent and learn.

2-layer, constant-size
(Part I)

great in-distribution accuracy
(Part II)

... yet with imperfect OOD generalization.

[Liu et al. 23]

Summary

via studying sandboxes

Part I … *what* are *the solutions in practice?*

- Tools for upper and lower bounds on the solution size.
- Implications: depth-width tradeoff; architecture comparison & improvement.

Part II … *how well can the solutions be found?*

- Implicit bias of gradient-based methods, canonical reparametrization.
- Simplicity bias, stage-wise training.

Part III… *why* & *how models (fail to) generalize?*

- Length generalization: proper measure of size; RASP-L.
- Same-length OOD generalization: inherent limitations of attention.

Summary

benefits of sandboxes

Understanding & clarity

- e.g. architecture choice (Part I); length generalization (Part I & III).
- e.g. training dynamics (Part II).

Diagnoses & stress test

- e.g. 1-layer models fail to learn 1st-order Markov chains (Part II)
- e.g. attention's limitations revealed by flip-flop (Part III).

Algorithm design

- e.g. hybrid models (Part I); structured assumptions to improve OOD.

Summary

limitations of sandboxes

Gap between sandboxes and real-world

Data:

- Which data structures to use?
- Single sandbox → a mixture?

Model:

- What architectures choices are essential?
  e.g. layers? tokenization?
- Assumptions on pretrained models?
  e.g. (Fourier) features? ICL/emergence?

Learning from Sandboxes

A lot of more to do! We need you :)

Understanding & clarity

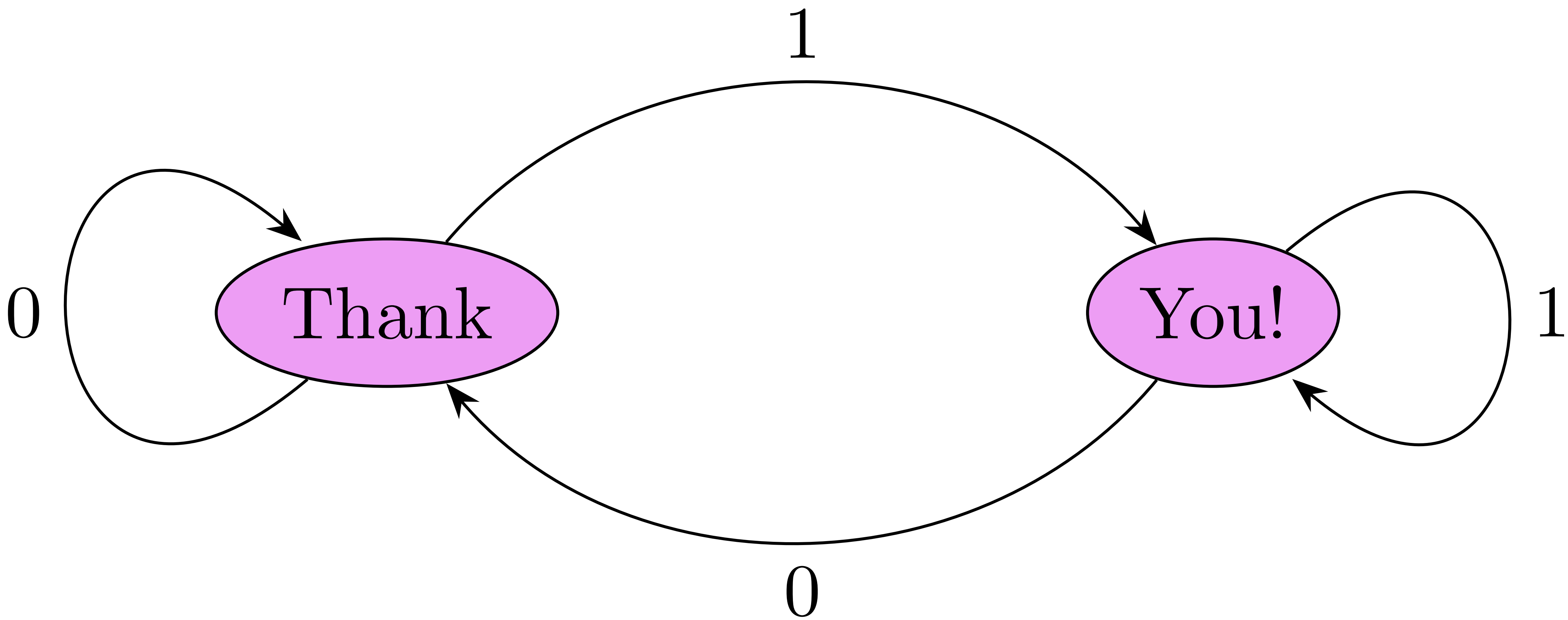
- e.g. architecture choice (Part I); length generalization (Part I & III).
- e.g. training dynamics (Part II).

Diagnoses & stress test

- e.g. 1-layer models fail to learn 1st-order Markov chains (Part II)
- e.g. attention's limitations revealed by flip-flop (Part III).

Algorithm design

- e.g. hybrid models (Part I); structured assumptions to improve OOD.


1
0
Thank
You!
1
0