Chain-of-Thought Reasoning in Transformers: From Theory to Practice

Yuejie Chi
Yale University

Harry Dong
Microsoft Research

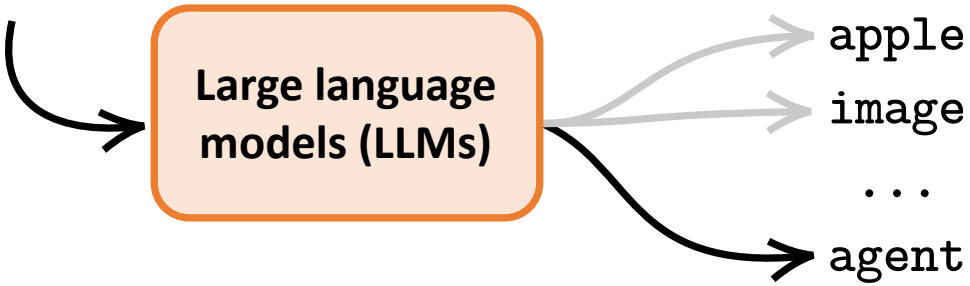
Tutorial, UAI 2026

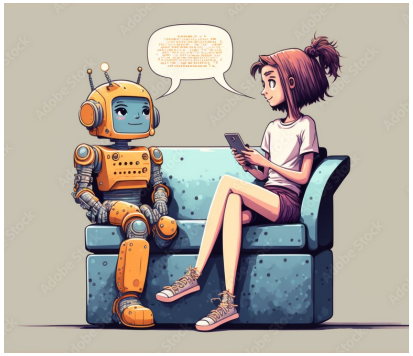
Large language models

The language models predict the next word based on previous words.

Prompt: Explain reinforcement learning (RL).

Answer: Reinforcement learning (RL) is a type of machine learning where an...





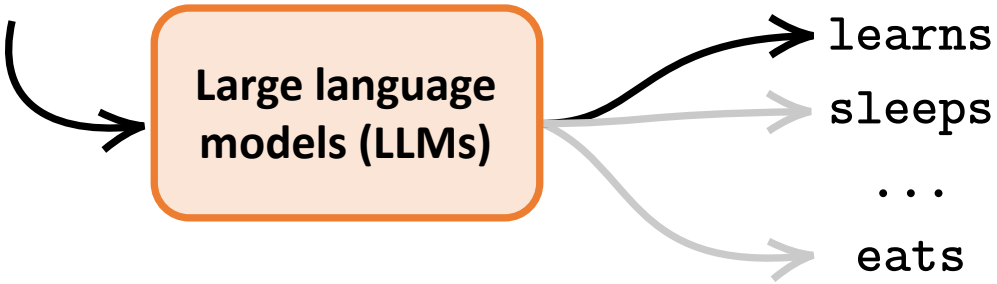
Applications in translation, Q&A, summarization...
and beyond!

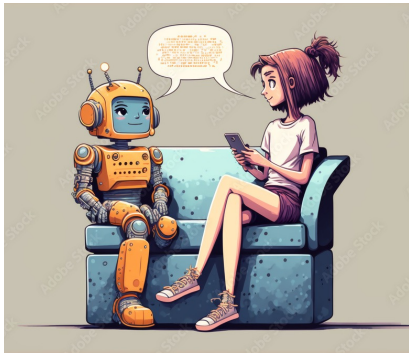
Large language models

The language models predict the next word based on previous words.

Prompt: Explain reinforcement learning (RL).

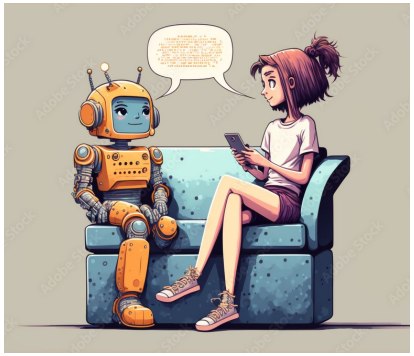
Answer: Reinforcement learning (RL) is a type of machine learning where an agent

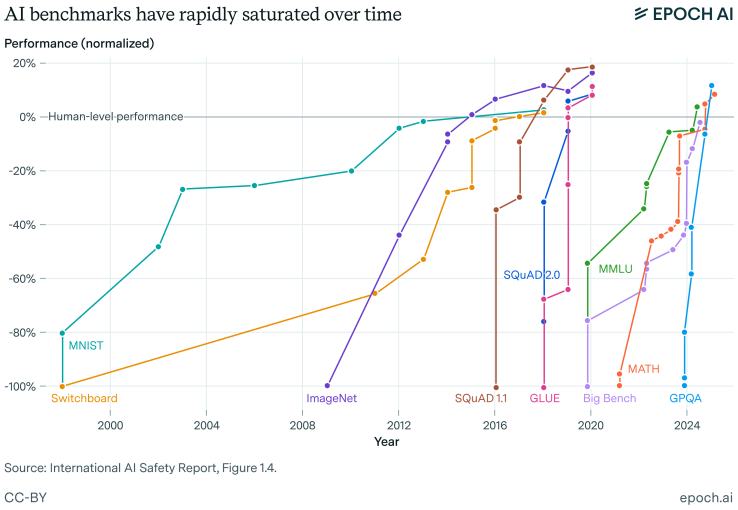


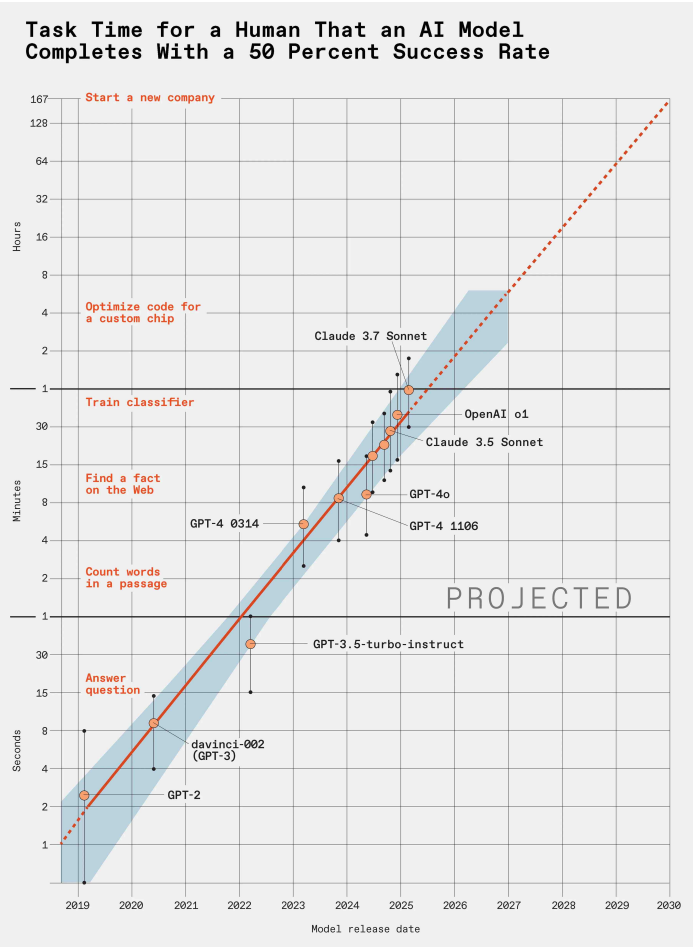


Applications in translation, Q&A, summarization...
and beyond!

The rise of large language models



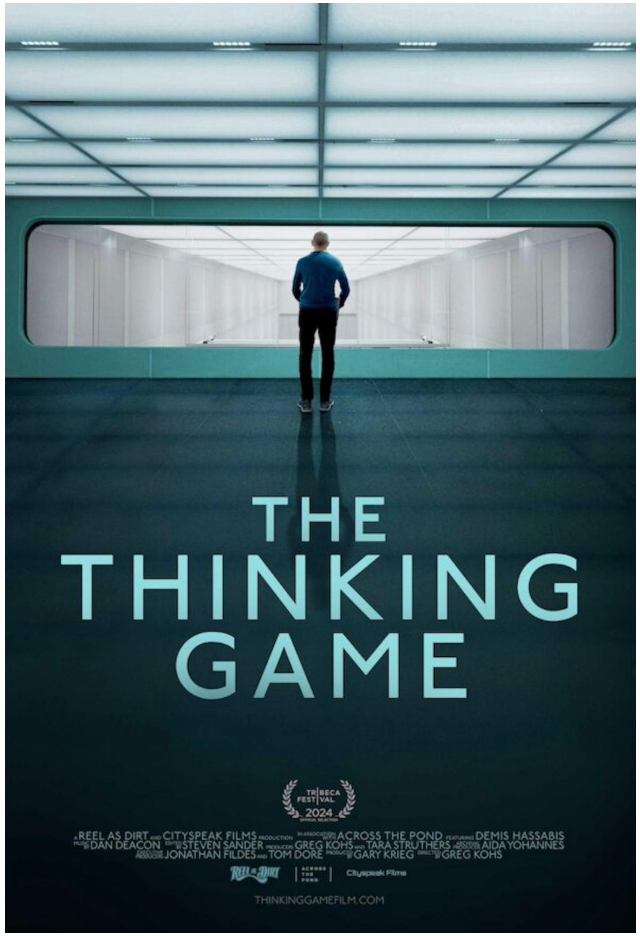




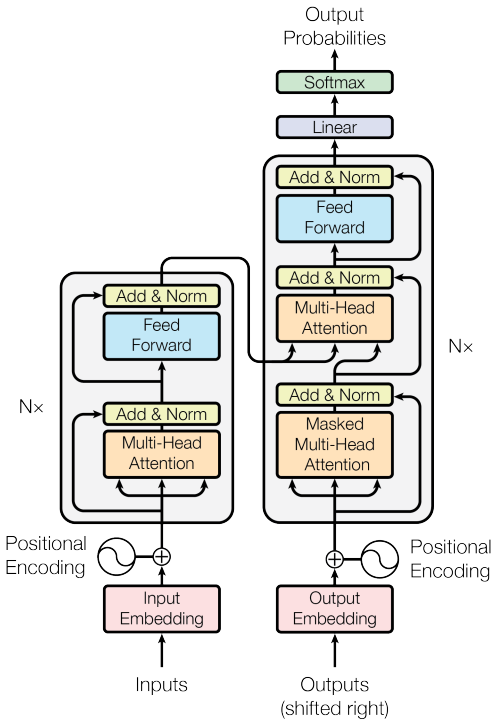
LLM benchmarking shows capabilities doubling every **7 months**.

`https://spectrum.ieee.org/large-language-model-performance`

The thinking machine



Transformers as the backbone architecture

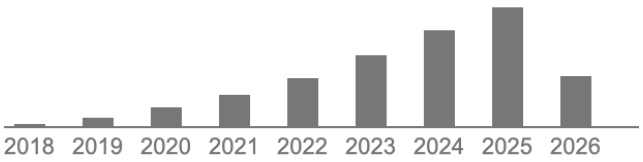


Attention is all you need

A Vaswani, N Shazeer, N Parmar, J

Advances in neural information proc

Cited by 263813



Chain-of-thought (CoT)

LLMs solve harder questions by reasoning **step-by-step**: generate intermediate reasoning steps before inferring an answer.*

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

<Question→Answer>

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

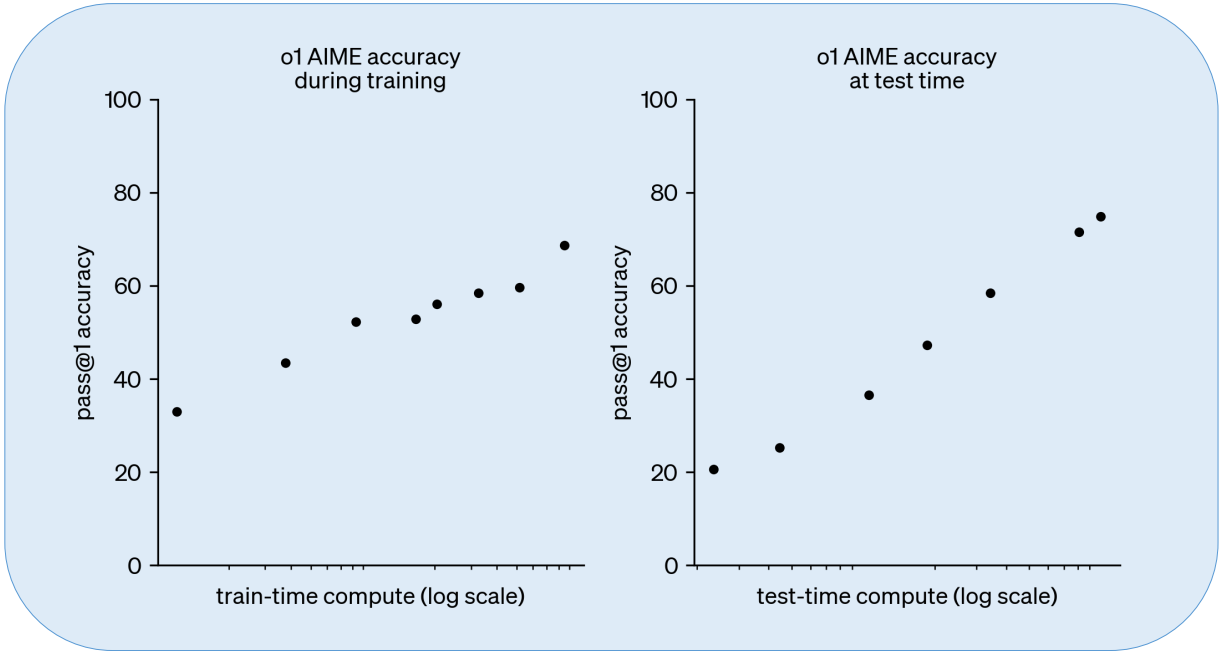
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

<Question→Rationale→Answer>

*Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.

Reasoning models and test-time scaling

Reasoning models: OpenAI released o1 in 2024, which was trained via reinforcement learning.



(Figure credit: o1)

Test-time scaling: reasoning performance improves with more test-time compute (e.g., thinking longer).

This tutorial

Demystifying Chain-of-Thought Reasoning in Transformers: From Theory to Practice

Part 1: Theory

- Expressivity
- Optimization and training dynamics
- Generalization

Part 2: Practice

- Practical training algorithms
- Efficient inference with test-time scaling
- Parallel scaling

Part 1: Mathematical theory of CoT

Expressivity:

- What are the provable advantage of CoT/test-time scaling in reasoning?

Optimization:

- How do trained transformers acquire such CoT capabilities?
- What is the difference between SFT and RL?

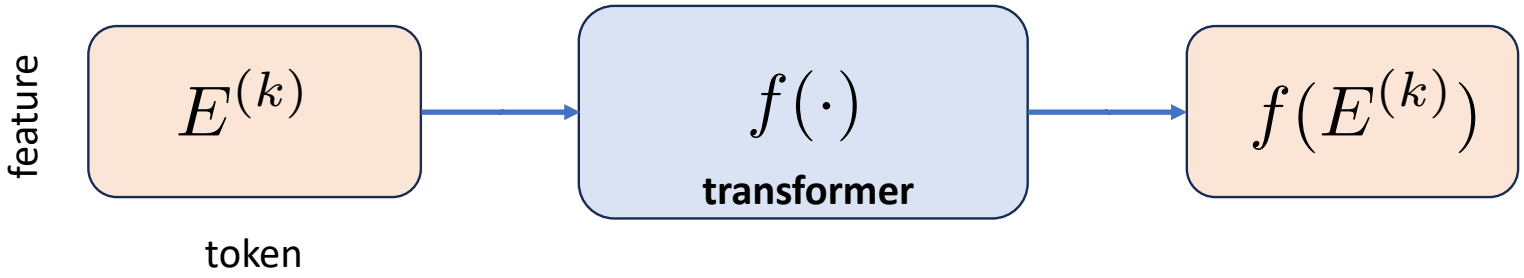
Generalization:

- How do the trained transformers generalize to unseen tasks?

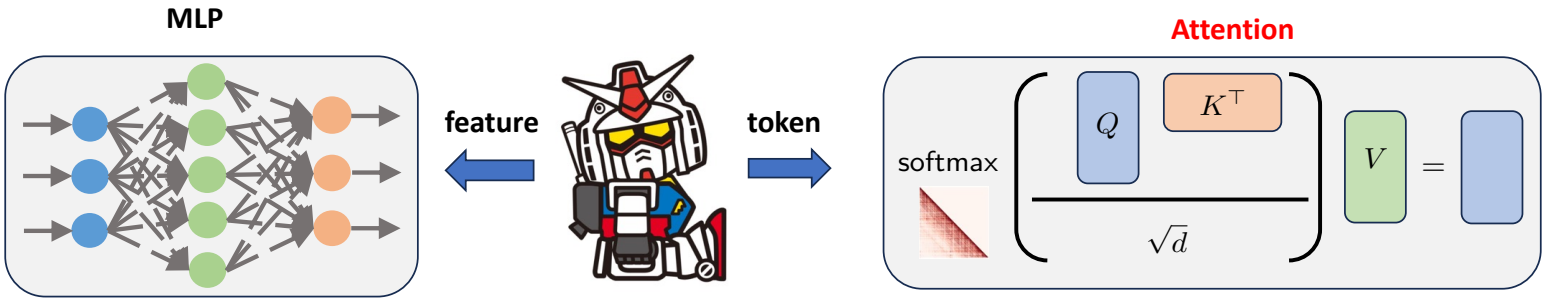
Primer: Transformer architectures

Decoder-only transformers

Sequence-to-sequence:

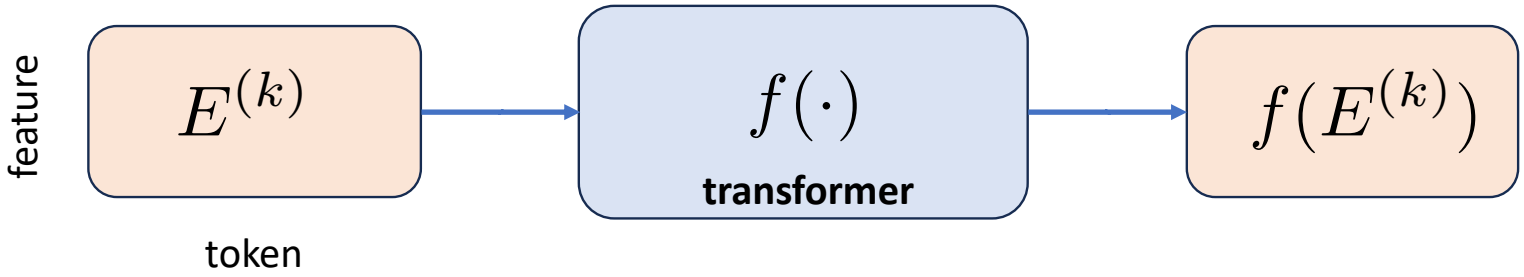


Transformer block = MLP ∘ attention

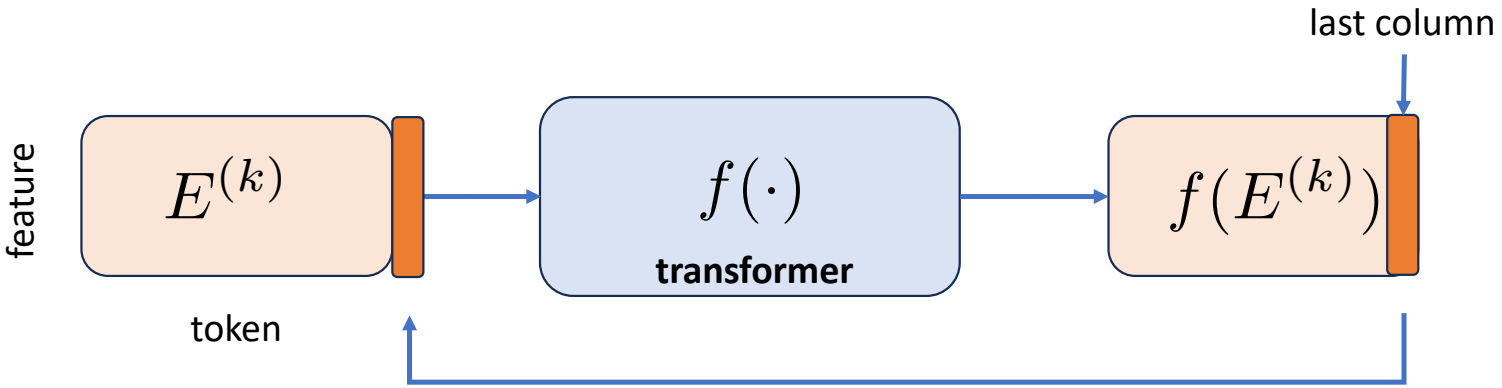


Decoder-only transformers

Sequence-to-sequence:



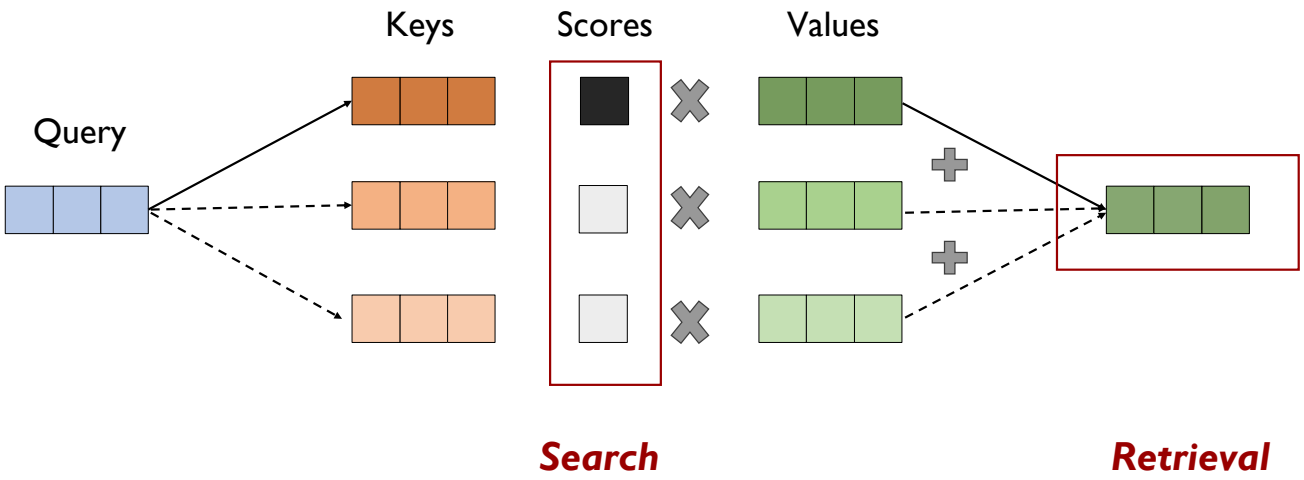
Autoregressive decoding:



Attention mechanism

Attention: For each query q, it matches against all the key-value pairs (k_i, v_i) and compute the value v:

$$\text{head}(q) = \sum \alpha_i v_i, \quad \text{where} \quad \alpha_i = \frac{\exp(q^\top k_i/\sqrt{d})}{\sum_u \exp(q^\top k_u/\sqrt{d})}.$$

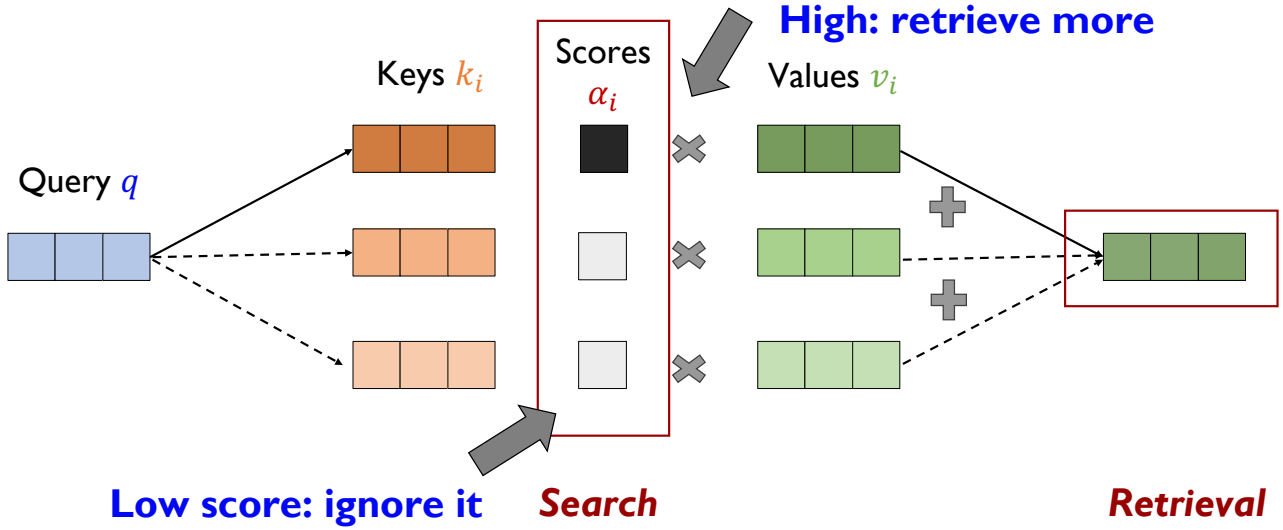


Attention searches and retrieves the values of relevant tokens

Attention mechanism

Attention: For each query q, it matches against all the key-value pairs (k_i, v_i) and compute the value v:

$$\text{head}(q) = \sum \alpha_i v_i, \quad \text{where} \quad \alpha_i = \frac{\exp(q^\top k_i/\sqrt{d})}{\sum_u \exp(q^\top k_u/\sqrt{d})}.$$



Attention searches and retrieves the values of relevant tokens

Self-attention mechanism

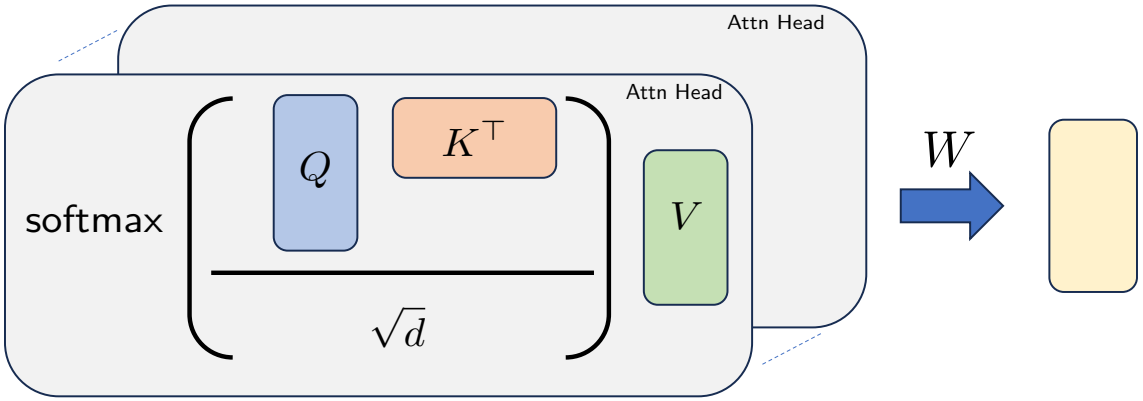
Given a vocabulary $\mathcal{V}$, and input sequence $X = (x_1, \ldots, x_n) \in \mathcal{V}^n$.

- compute the query $Q = \{q_i\}_{i=1}^n$, key $K = \{k_i\}_{i=1}^n$, and value $V = \{v_i\}_{i=1}^n$ using weight matrices W_Q, W_K, W_V as

$$q_i = W_Q x_i, \quad k_i = W_K x_i, \quad v_i = W_V x_i.$$

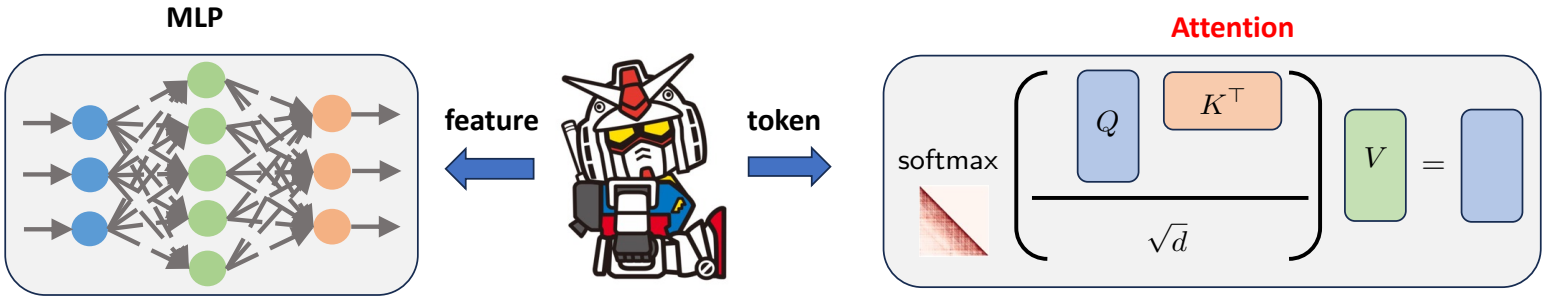
- compute the softmax score for all queries and the output with output weight W:

$$h_i = W \cdot \text{Head}(X), \qquad \text{with} \quad \text{Head}(X) := \text{softmax}\left(\frac{Q^\top K}{\sqrt{d}}\right) \cdot V.$$



Transformer block

A transformer block TF_θ further maps the output of a multi-head attention with a MLP.



- The multi-head output is

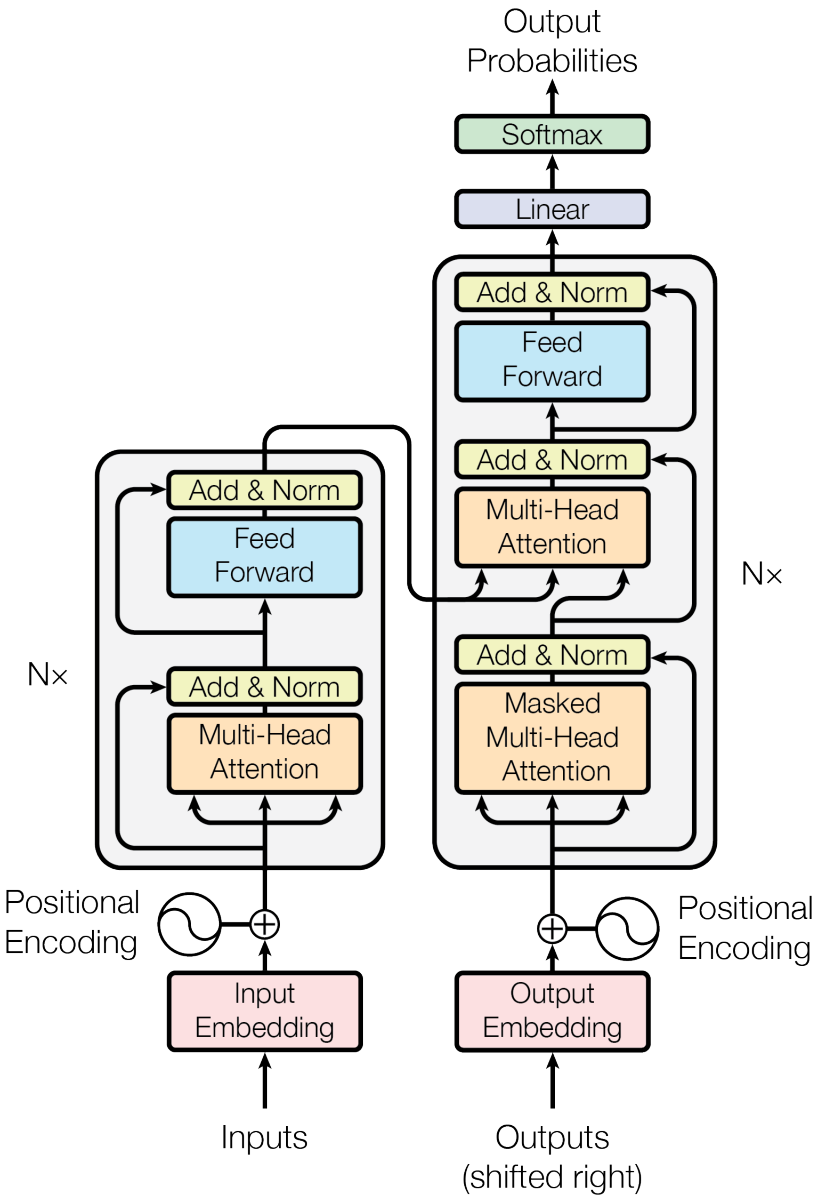
$$\mathrm{MHA}(X) = \sum_{h=1}^{H} W_h \cdot \mathrm{Head}_h(X)$$

- The output of multi-head attention is passed through a MLP

$$y = \mathrm{MLP}(\mathrm{MHA}(X)) = \underbrace{\mathrm{MLP} \odot \mathrm{MHA}}_{\mathrm{TF}_\theta}(X).$$

For example, $\mathrm{MLP}(h) = W_2 \mathrm{ReLU}(W_1 h + b_1) + b_2$.

Did we describe everything?



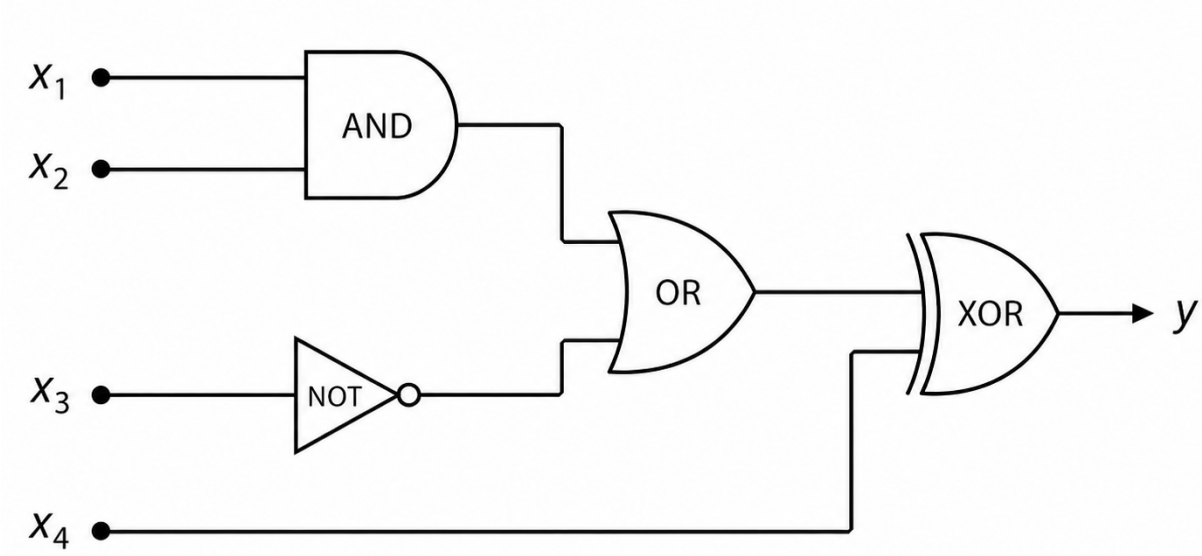
Expressiveness of CoT using circuit complexity

A crash course on circuit complexity

Given a vocabulary $\mathcal{V}$, a mapping $\mathcal{L} : \cup_{k \in \mathbb{N}^+} \mathcal{V}^k \to \mathcal{V}$ is called a *problem*.

Boolean circuit. A Boolean circuit over n variables is a directed acyclic graph where nodes are `AND`, `OR`, or `NOT` gates.

- `XOR` can be constructed by `AND`, `OR`, or `NOT`.



Circuit complexity classes

Class of problems when the input length is n.

- P/poly: problems solvable by polynomial-size $O(\text{poly}(n))$ circuits.
- NC^k: problems solvable in $O((\log n)^k)$ parallel runtime with a polynomial number of processors with fan-in of the gates at most 2.
- AC^k: same as NC^k except unbounded fan-in of `AND` and `OR` gates.
- TC^k: allowing `MAJORITY` in addition to AC^k.

It holds for all $k \in \mathbb{N}_+$:

$$\mathsf{NC}^k \subseteq \mathsf{AC}^k \subseteq \mathsf{TC}^k \subseteq \mathsf{NC}^{k+1}.$$

- We have $\mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$.
- It is widely conjectured $\mathsf{TC}^0 \neq \mathsf{NC}^1$.

What determines the expressivity of a transformer?

Consider a transformer model with L layers of transformer blocks.

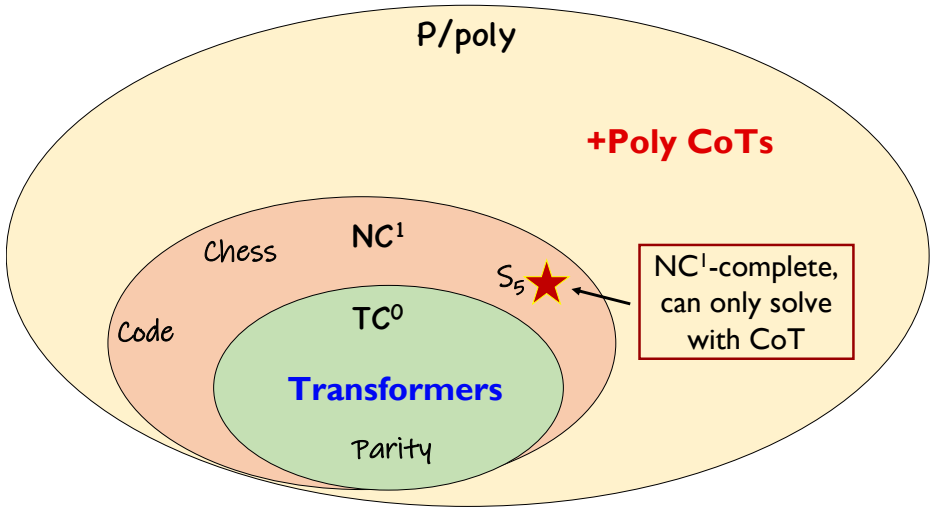
$$\mathsf{TF} = \mathsf{TF}_L \odot \mathsf{TF}_{L-1} \cdots \odot \mathsf{TF}_1,$$

where each $\mathsf{TF}_\ell = \mathsf{MLP}_\ell \odot \mathsf{Attn}_\ell$.

- **Network size:** depth and width
- **Sequence length:** complexity of the problem space
- **Embedding dimension:** representation capacity
- **Precision:** how are the parameters quantized?
- **Test-time/CoT scaling:** whether the model needs to produce the output in one shot.

For follow-up discussion, we assume the depth is constant (fixed), polynomial width, logarithmic embedding in the sequence length n.

Expressive power of transformers with CoT



assuming $TC^0 \neq NC^1$

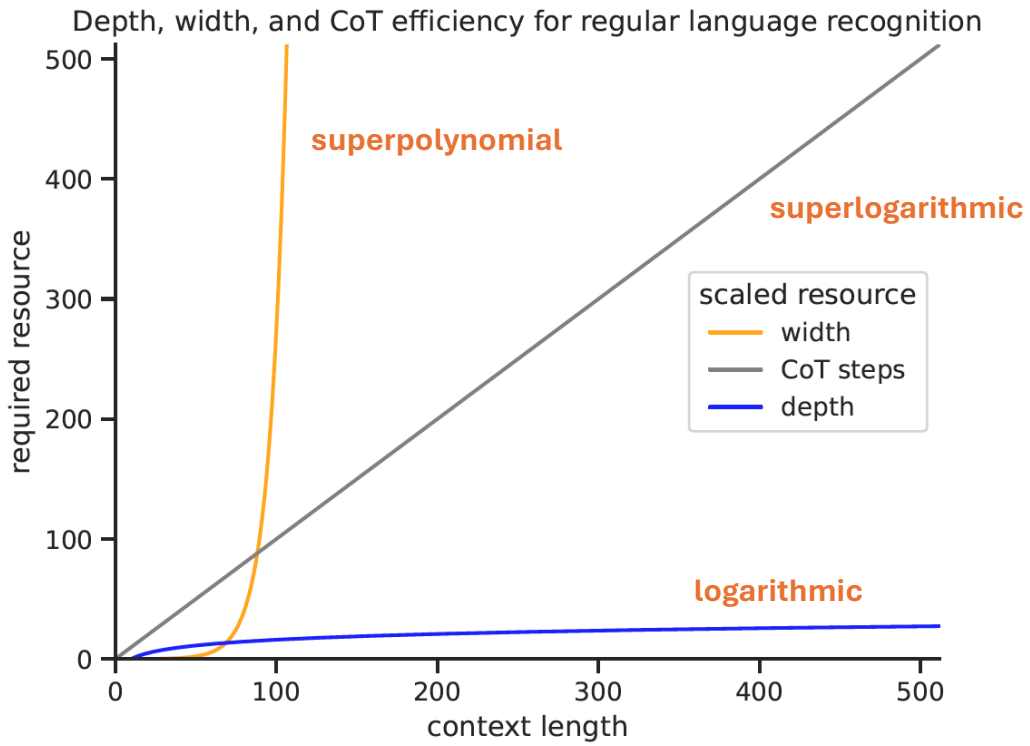
Expressiveness of constant-depth transformers: (Li et al., 2024; Liu et al., 2022; Merrill et al., 2023; Feng et al., 2023...)

- without CoT: logarithmic-precision transformer $\subset \mathsf{TC}^0$ and constant-precision $\subset \mathsf{AC}^0$;
- transformer with linear $O(n)$ CoTs $\supseteq \mathsf{NC}^1$;
- transformer with polynomial $O(\text{poly}(n))$ CoTs $\supseteq P/\text{poly}$.

With increasing CoT lengths, transformers become more powerful!

Aside: scaling network size versus scaling CoT

(Merrill and Sabharwal, 2025) benchmarked the impact of scaling depth (e.g., via looping), width, CoT for regular language recognition.



Training transformers for CoT capabilities

Two training paradigms

Problem

The battery charge in Mary's cordless vacuum cleaner lasts ten minutes. It takes her four minutes to vacuum each room in her house. Mary has three bedrooms, a kitchen, and a living room. How many times does Mary need to charge her vacuum cleaner to vacuum her whole house?

reasoning trace

Solution

Mary has 3 + 1 + 1 = 5 rooms in her house.
At 4 minutes a room, it will take her 4 * 5 = 20 minutes to vacuum her whole house.
At 10 minutes a charge, she will need to charge her vacuum cleaner 20 / 10 = 2 times to vacuum her whole house.

Final Answer

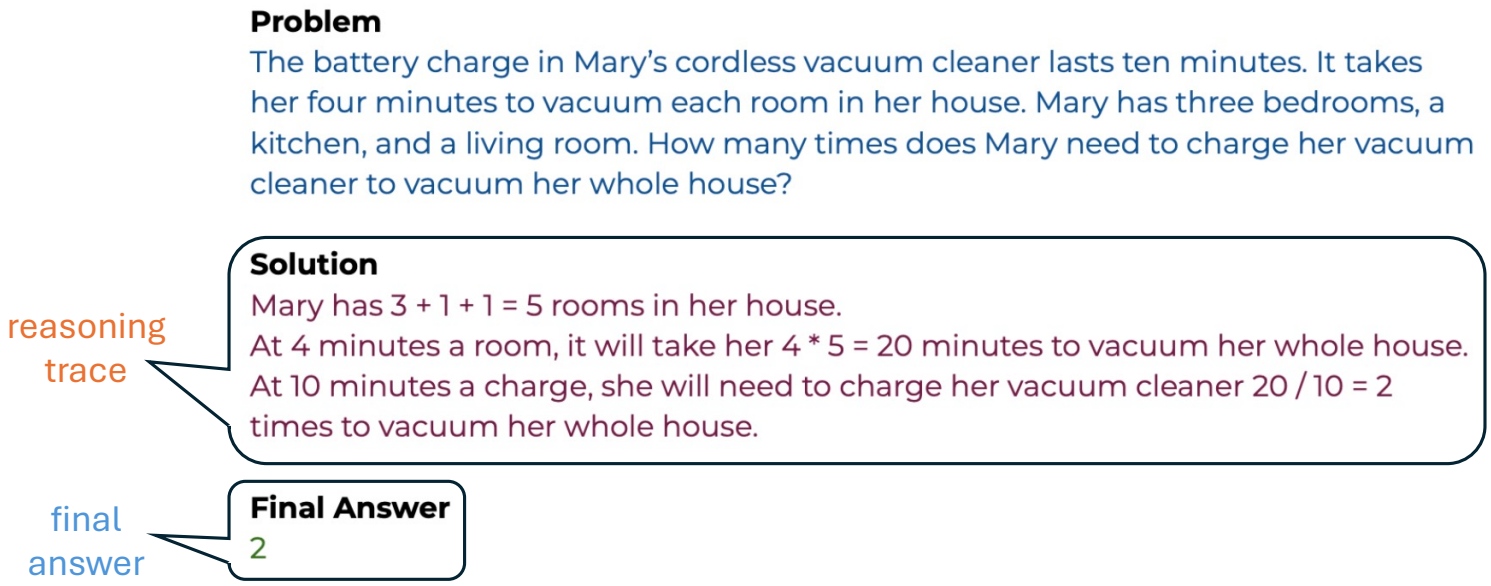
2

Supervised finetuning (SFT)

Next-token prediction over the reasoning trace

dense supervision

Two training paradigms



Supervised finetuning (SFT)

Next-token prediction over the reasoning trace

dense supervision

RL from verifiable reward (RLVR)

Reward only when the final answer is correct

sparse reward

Summary: SFT vs RL

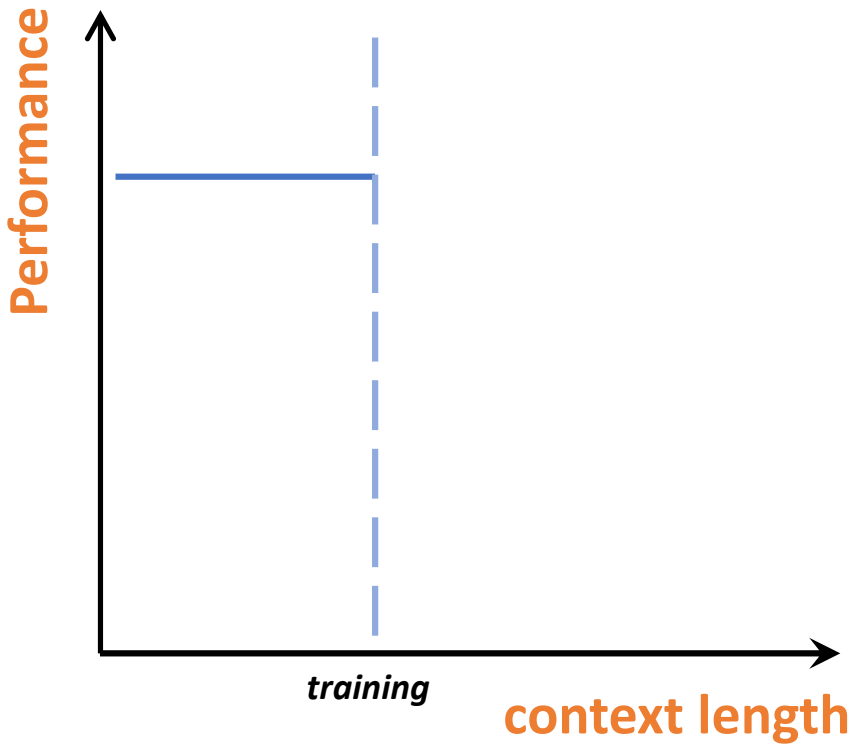
SFT: dense process supervision

- Stable and sample-efficient training
- Limited by quality of demonstrations
- Less exploration of diverse reasoning paths

RL: sparse outcome reward

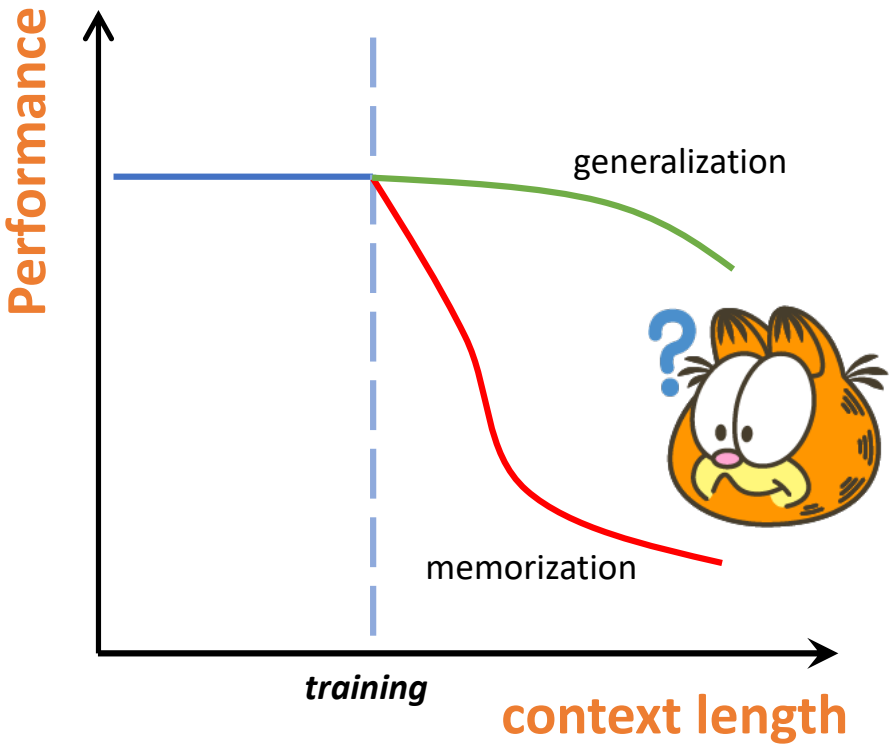
- Directly optimizes final correctness
- Can discover new reasoning strategies via exploration
- Difficult and slow in long-horizon reasoning

From training to generalization



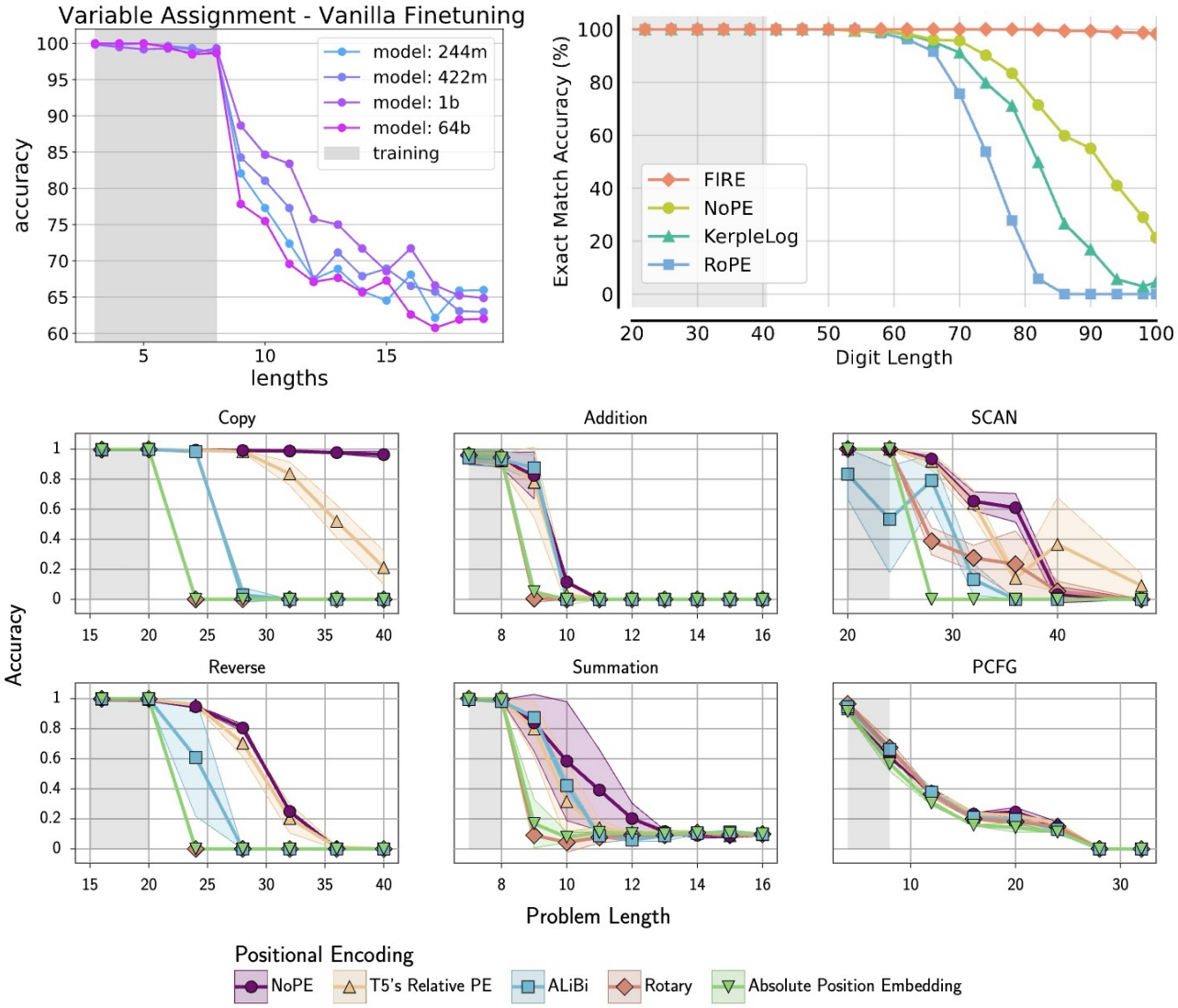
Can transformers trained on shorter tasks be generalized to longer ones?

From training to generalization



Can transformers trained on shorter tasks be generalized to longer ones?

Empirical evidence is rather mixed



(Figure credit: Anil et al., 2022; Kazemnejad et al., 2023; Zhou et al., 2024)

Key questions

Why do *trained* transformers perform multi-step reasoning beyond training examples?

Optimization

Generalization

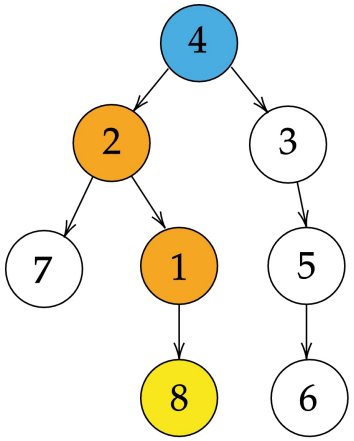
- What does gradient descent converge to?
- What is the trained attention at convergence?

- Will trained transformers generalize to unseen tasks?
- Will trained transformers generalize to longer lengths?

Supervised training from reasoning traces

Agenda: two case studies

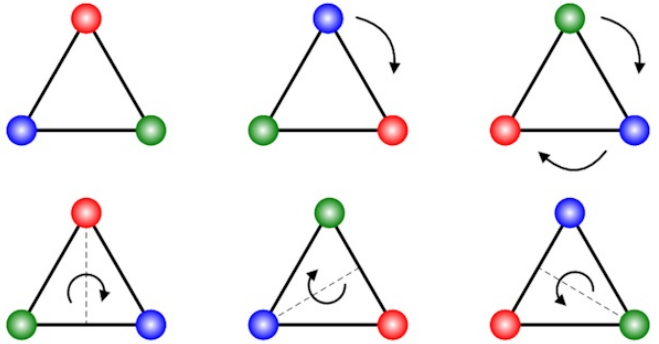
graph reasoning with attention only



Path-finding in a tree

(Yang et al., 2025)

reasoning with group action with attention + MLP

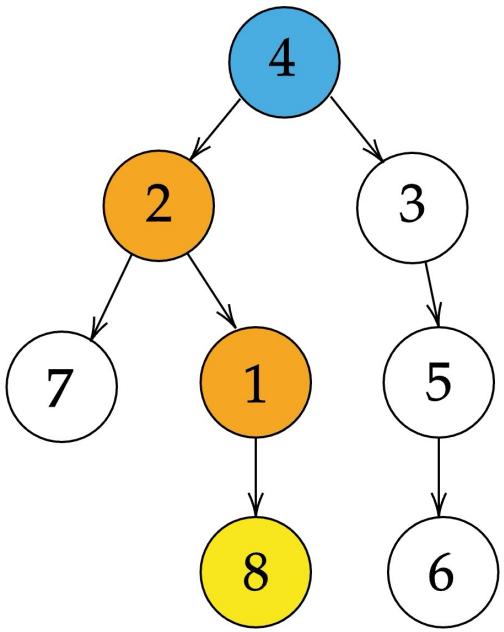


LEGO

(Huang et al., 2025)

Outlook: at the time of convergence, the concentration pattern of attention determines the generalization performance.

Understanding attention: path finding in trees



Prompt:

Edge set: $1 \to 8,\ 4 \to 2,\ 2 \to 7, 3 \to 5,$
$2 \to 1,\ 4 \to 3,\ 5 \to 6$

Root: 4

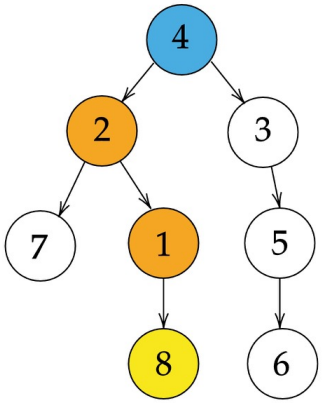
Goal-to-Root Path-finding:

Goal: 8 $\quad 8 \to 1 \to 2 \to 4$

Goal: 7 $\quad 7 \to 2 \to 4$

- Supervised training: the entire path is given in the training data.
- At each step, the transformer predicts the parent node of the current output.

Model setup



$$E^{(0)} = \begin{bmatrix} X^{(0)} \\ Y^{(0)} \end{bmatrix} = \left[\begin{array}{ccccccc|c|c} 1 & 4 & 2 & 3 & 2 & 4 & 5 & 0 & 8 \\ 8 & 2 & 7 & 5 & 1 & 3 & 6 & 4 & 0 \end{array}\right] \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

edges r g

query

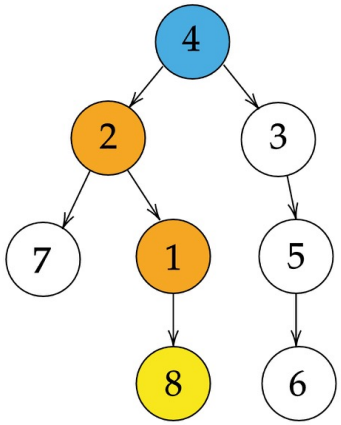
Embedding: let $A := (a_1, \ldots, a_S) \in \mathbb{R}^{d_1 \times S}$ be the embedding matrix with linearly independent columns, and $a_0 = 0_{d_1}$.

Attention-only mechanism: simplified attention model with trainable B, where $x_{-1}^{(k-1)}$ is the last column of $X^{(k-1)}$:

$$\text{softmax}\left(Y^{(k-1)\top} B x_{-1}^{(k-1)}\right) = \text{softmax}\left(\begin{bmatrix} X^{(k-1)\top} & Y^{(k-1)\top} \end{bmatrix} \underbrace{\begin{bmatrix} 0 & 0 \\ B & 0 \end{bmatrix}}_{W_Q^\top W_K} \begin{bmatrix} x_{-1}^{(k-1)} \\ y_{-1}^{(k-1)} \end{bmatrix}\right)$$

to ensure tractability.

Attention mechanism for path finding



$$E^{(0)} = \begin{bmatrix} X^{(0)} \\ Y^{(0)} \end{bmatrix} = \left[\begin{array}{ccccccc|c|c} 1 & 4 & 2 & 3 & 2 & 4 & 5 & 0 & 8 \\ 8 & 2 & 7 & 5 & 1 & 3 & 6 & 4 & 0 \end{array}\right] \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

edges r g

query

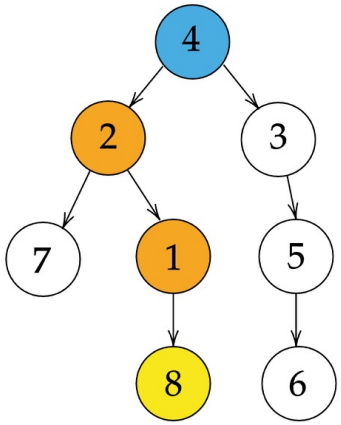
Attention searches the edge and retrieves the parent node

Attention output with fixed values

$$\widehat{o}^{(k)} = \underbrace{\begin{pmatrix} X^{(k-1)} \\ Y^{(k-1)} \end{pmatrix}}_{\text{values}} \cdot \underbrace{\text{softmax}\left(Y^{(k-1)\top} B x_{-1}^{(k-1)} \right)}_{\text{scores}}.$$

- The output is a linear combination of previous edges in the context.
- Attend to $(p(n^{(k-1)}), n^{(k-1)})$, and output $(p(n^{(k-1)}), n^{(k-1)})$.

Attention mechanism for path finding



$$E^{(0)} = \begin{bmatrix} X^{(0)} \\ Y^{(0)} \end{bmatrix} = \left[\begin{array}{ccccccc|cc} 1 & 4 & 2 & 3 & 2 & 4 & 5 & 0 & 8 \\ 8 & 2 & 7 & 5 & 1 & 3 & 6 & 4 & 0 \end{array}\right] \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

edges r g

search

query

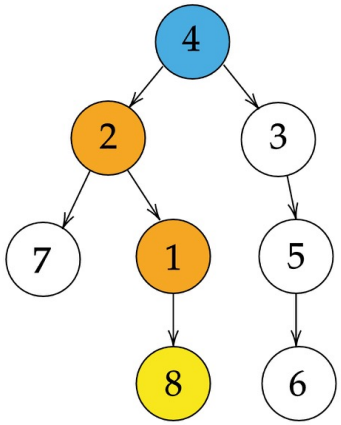
Attention searches the edge and retrieves the parent node

Attention output with fixed values

$$\widehat{o}^{(k)} = \underbrace{\begin{pmatrix} X^{(k-1)} \\ Y^{(k-1)} \end{pmatrix}}_{\text{values}} \cdot \underbrace{\text{softmax}\left(Y^{(k-1)\top} B x_{-1}^{(k-1)}\right)}_{\text{scores}}.$$

- The output is a linear combination of previous edges in the context.
- Attend to $(p(n^{(k-1)}), n^{(k-1)})$, and output $(p(n^{(k-1)}), n^{(k-1)})$.

Attention mechanism for path finding



$$E^{(0)} = \begin{bmatrix} X^{(0)} \\ Y^{(0)} \end{bmatrix} = \left[\begin{array}{ccccccc|cc} 1 & 4 & 2 & 3 & 2 & 4 & 5 & 0 & 8 \\ 8 & 2 & 7 & 5 & 1 & 3 & 6 & 4 & 0 \end{array}\right] \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

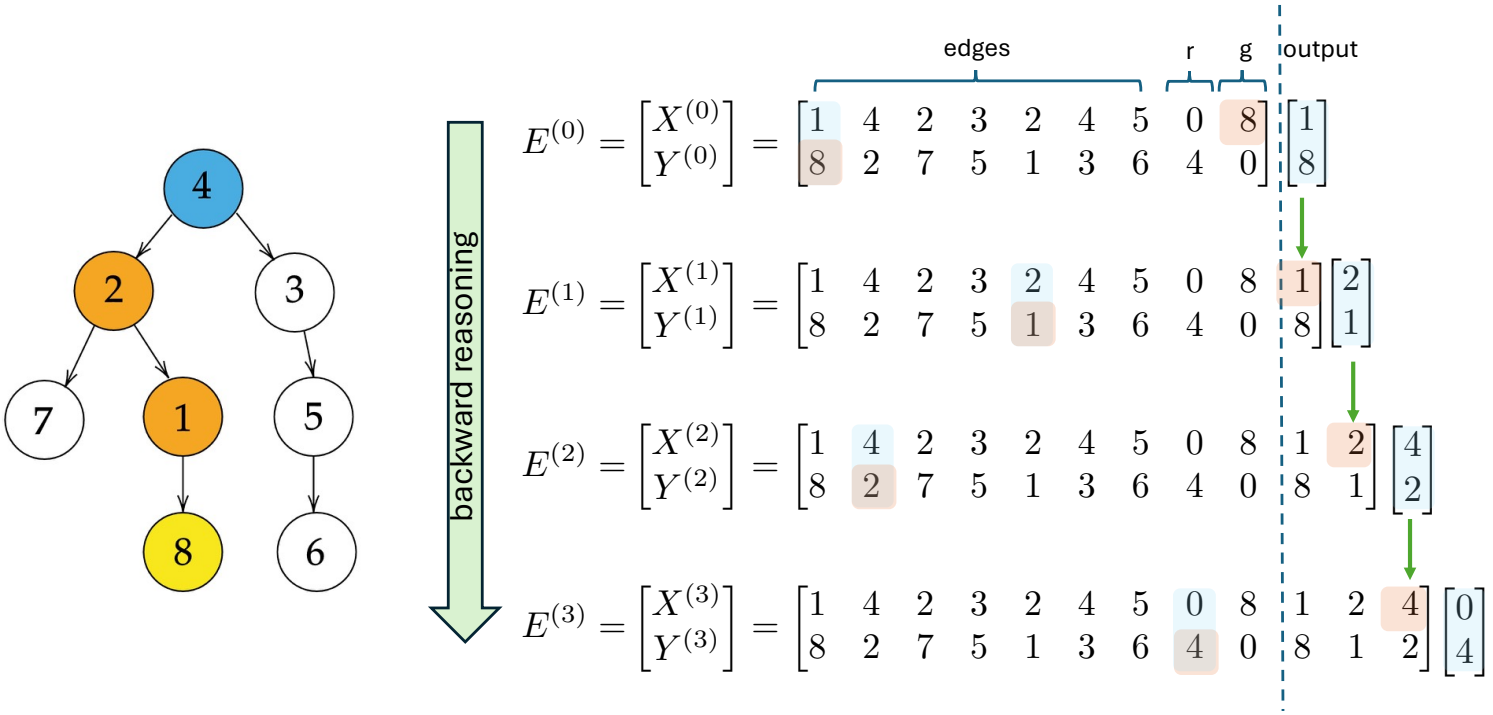
Attention searches the edge and retrieves the parent node

Attention output with fixed values

$$\widehat{o}^{(k)} = \underbrace{\begin{pmatrix} X^{(k-1)} \\ Y^{(k-1)} \end{pmatrix}}_{\text{values}} \cdot \underbrace{\text{softmax}\left(Y^{(k-1)\top} B x_{-1}^{(k-1)}\right)}_{\text{scores}}.$$

- The output is a linear combination of previous edges in the context.
- Attend to $(p(n^{(k-1)}), n^{(k-1)})$, and output $(p(n^{(k-1)}), n^{(k-1)})$.

Attention construction for path finding



Theorem (Construction for path finding)

Given linearly independent nodal embeddings, there exist attention constructions that implement this mechanism.

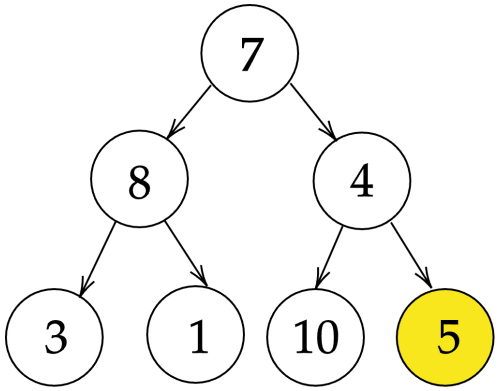
Assumptions for training dynamics

Can trained transfomers learn these mechanisms, i.e. the desired attention patterns?

Assumptions for training dynamics

Can trained transfomers learn these mechanisms, i.e. the desired attention patterns?

Training distribution: Let $m \geq 3$ and $S \geq 2^{m+1} - 1 =: N$. The training distribution P_{train} is the **uniform distribution** over **perfect binary trees** $\mathcal{T} = (\mathcal{V}(\mathcal{T}), \mathcal{E}(\mathcal{T}), g(\mathcal{T}))$ of depth m with distinct nodes chosen from $[S]$ and a goal node $g(\mathcal{T})$ being one of the leaf nodes.



Orthonormal embeddings: $\{a_i\}_{i \in [S]}$ are orthonormal vectors in $\mathbb{R}^{d_1}$.

Training objective

Autoregressive teacher-forcing: at each step $k \geq 2$, the input

$$E_{\text{train}}^{(k-1)}(\mathcal{T}) = (E_{\text{train}}^{(k-2)}(\mathcal{T}), o^{(k-1)}(\mathcal{T}))$$

is the concatenation of the input at step $k-1$ and the $(k-1)$-th column of the ground truth output. The output is

$$\widehat{o}_{\text{train}}^{(k)}(\mathcal{T}; B) = f(E_{\text{train}}^{(k-1)}(\mathcal{T}); B)_{:,-1}.$$

CoT training loss:

$$\mathcal{L}_{\text{train}}(B) = \frac{1}{2}\mathbb{E}_{\mathcal{T}\sim P_{\text{train}}} \|O(\mathcal{T}) - \widehat{O}_{\text{train}}(\mathcal{T}; B)\|_{\text{F}}^2.$$

Gradient descent: we initialize all parameters to be zero, and

$$B^{(t+1)} = B^{(t)} - \eta \nabla_B \mathcal{L}_{\text{train}}(B^{(t)}), \quad \forall t \geq 0.$$

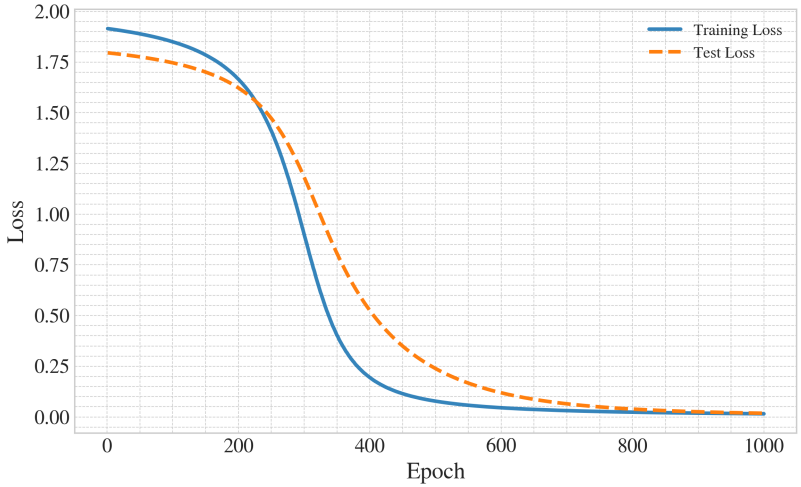
Optimization theory

Theorem (Yang et al., 2025)

We use zero initialization. For learning rate η and ϵ that are sufficiently small, we reach $\mathcal{L}_{\text{train}}(B^{(t)}) \leq \epsilon$ as soon as the number of iterations satisfies

$$T = \mathcal{O}\left(\frac{mS}{\eta\epsilon}\log\left(\frac{Nm}{\epsilon}\right)\right).$$

- Gradient descent converges at a rate of $\widetilde{\mathcal{O}}(1/\epsilon)$.



Theory of generalization

Generalization error: on unseen tree $\mathcal{T}$

$$\mathcal{L}_{\text{test}}(\mathcal{T}; B) = \frac{1}{2}\|O(\mathcal{T}) - \widehat{O}(\mathcal{T}; B)\|_{\text{F}}^2.$$

Theorem (Generalization)

Given any tree $\mathcal{T}$ at test time with $\widetilde{N}$ distinct nodes chosen from $[S]$ and a path length $\widetilde{m}$, there exists small enough $\epsilon_0 = \epsilon_0(m, \widetilde{N}, \widetilde{m}) > 0$ such that for any $\epsilon \in (0, \epsilon_0]$, when the training loss $\mathcal{L}_{\text{train}}(B^{(t)}) \le \epsilon$, it holds

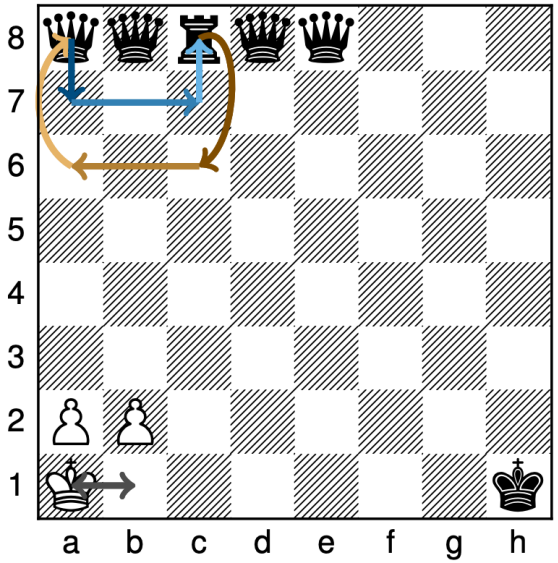
$$\mathcal{L}_{\text{test}}(\mathcal{T}; B^{(t)}) \le 4 \max\left\{\left(\frac{\widetilde{N} + \widetilde{m} - 1}{N + m - 2}\right)^2, 1\right\} \cdot \frac{\widetilde{m}}{m}\epsilon.$$

- Better length generalization on unseen trees (possibly with higher depth) for small training error.

Moving beyond attention: state tracking

Given an initial state and a sequence of *actions*, compute the final state.

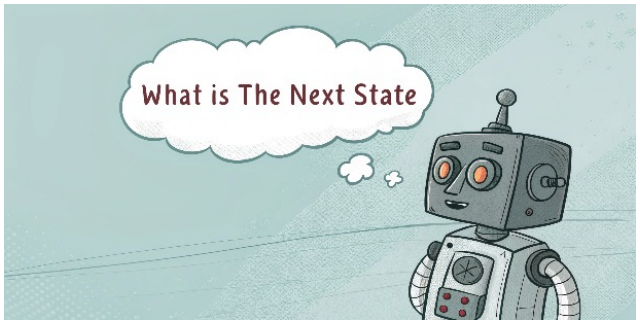
Playing Chess



Images modified from Merrill et al., 2024

Code evaluation

```
x = [0, 0, 1, 0, 0]
x[1], x[3] = x[3], x[1] # Swap 1, 3
```



Entities tracking

Alice, Bob, Carl, Dan, and Emma each have a coin. All are dimes except Carl's. Alice and Carl trade coins.

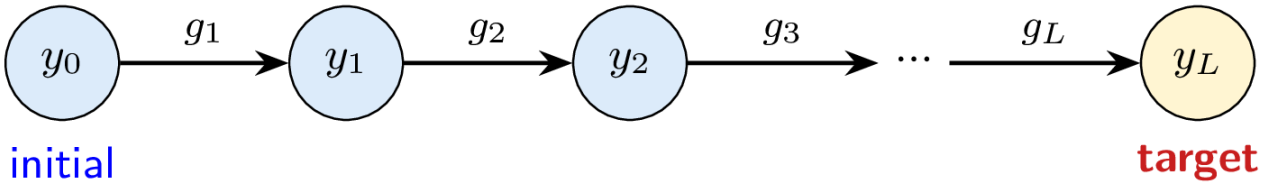
State tracking via LEGO

LEGO (Learning Equality and Group Operations) (Zhang et al., 2022):
with answer up to $m < L$

$$\underbrace{\boxed{x_1 = g_1(x_0)},\ \ldots,\ \boxed{x_L = g_L(x_{L-1})}}_{\text{predicate clauses}} \qquad \underbrace{\boxed{x_0 = y_0},\ \ldots,\ \boxed{x_m = y_m}}_{\text{answer clauses}}$$

with action $g_i \in \mathcal{G}$, value $y_i \in \mathcal{Y}$, variable $x_i \in \mathcal{X}$.

Goal: correctly calculate the last value $y_L = g_L(g_{L-1}(\cdots g_1(y_0)))$.



Examples of state tracking

Example from TC^0: parity check / modulo sum

$$x_1 = x_0 \oplus_2 1,\ x_2 = x_1 \oplus_2 1, \quad x_0 = 0,\ x_1 = ?,\ x_2 = ?$$

where x_2 counts the parity of the bit string $\{0, 1, 1\}$:

$$x_2 = x_1 \underbrace{\oplus_2 1}_{g_2} = (x_0 \underbrace{\oplus_2 1}_{g_1}) \oplus_2 1 = (\underbrace{0}_{y_0} \oplus_2 1) \oplus_2 1 \equiv 0 \pmod 2.$$

Example from NC^1: S_5 acts on $[5] = \{1, 2, 3, 4, 5\}$:

$$x_1 = g_1(x_0), \quad x_2 = g_2(x_1), \quad x_3 = g_3(x_2), \quad x_0 = 1,\ x_1 = ?,\ x_2 = ?,\ x_3 = ?.$$

Take $g_1 = (1\,4\,2)$, $g_2 = (2\,5)(3\,4)$, and $g_3 = (1\,3\,5)$. Then

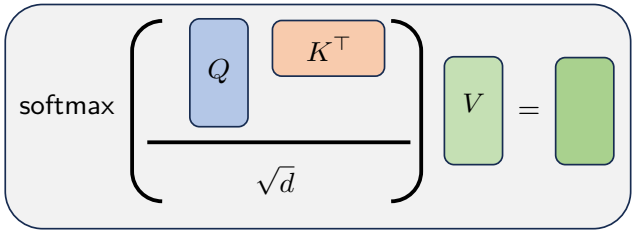
$$x_3 = g_3(g_2(g_1(\underbrace{1}_{y_0}))) = g_3(g_2(4)) = g_3(3) = 5.$$

One-layer attention + MLP model

Model mechanism: Attention (retrieval)

query

$$x_1 = g_1(x_0), \cdots x_{m+1} = g_m(x_m) \cdots, x_L = g_L(x_{L-1}), \quad x_0 = y_0, \cdots\cdots, x_m = y_m$$



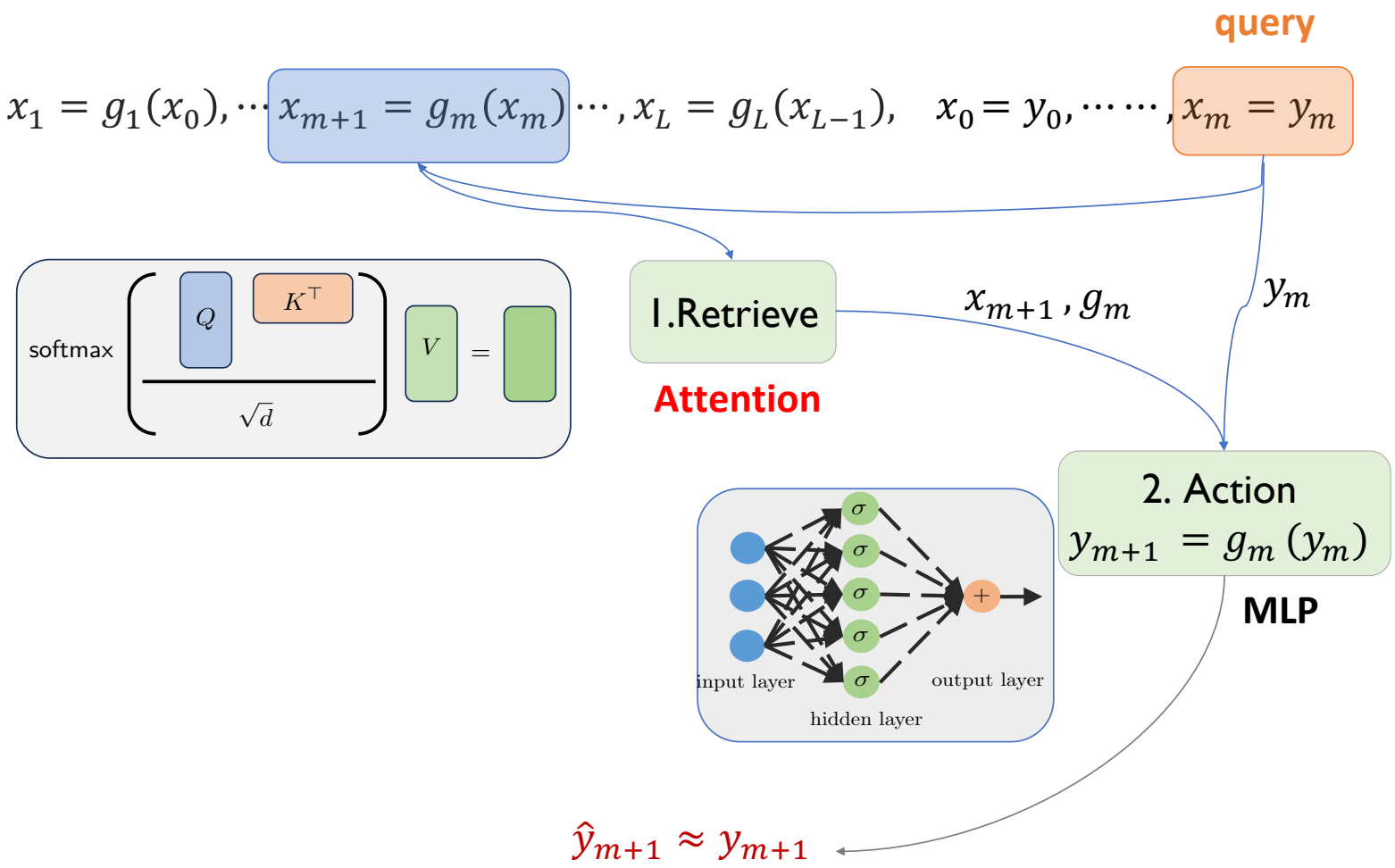
I.Retrieve

Attention

$$\hat{y}_{m+1} \approx y_{m+1}$$

One-layer attention + MLP model

Model mechanism: FFN (action) ∘ Attention (retrieval)



MLP training

MLP captures the atomic skills of group action

$$(g, y) \mapsto g(y)$$

and becomes fully trained at length $L = 1$ in the following sense:

When MLP receives $\frac{1}{2}(g + y)$, then for $j = \tau(g(y))$

$$\text{softmax}\Big(\text{MLP}_W\Big(\frac{1}{2}(g + y)\Big)\Big)_j = 1 - \frac{1}{\text{poly}(d)}$$

Here, $\tau : \mathcal{Y} \to [d]$ index the states for prediction.

Overparameterized regime: for each pair (g, y) with correct output $j = \tau(g(y))$, there exists a *unique* neuron r such that

$$\langle W_{j,r}, g\rangle = B, \quad \langle W_{j,r}, y\rangle = B, \quad \langle W_{j,r}, s\rangle = -B \text{ (other } s \neq g, y)$$
$$\langle W_{j,r'}, s\rangle = 0 \quad \text{(other } r' \neq r, s \in \mathcal{G} \cup \mathcal{Y})$$

where $B \asymp \log d$.

Tracking the attention dynamics

$$\mathbf{Z}^{L,m} \triangleq \underbrace{\boxed{x_1 = g_1(x_0)}}_{\mathbf{Z}_{\text{pred},1}}, \ \ldots, \ \boxed{x_L = g_L(x_{L-1})} \quad \underbrace{\boxed{x_0 = y_0}}_{\mathbf{Z}_{\text{ans},0}}, \ \ldots, \ \underbrace{\boxed{x_m = y_m}}_{\text{query } \mathbf{Z}_{\text{ans},m}}$$

Training objective: minimize the cross-entropy loss for *next-answer* prediction via gradient descent

$$\mathcal{L}(F) = \mathbb{E}\Big[\sum_{m=1}^{L} -\log p_F(y_{m+1} \mid \mathbf{Z}^{L,m})\Big]$$

where $d = |\mathcal{X}| + |\mathcal{G}| + |\mathcal{Y}|$ (roughly embedding dimension).

Tracking how attention concentrates on the key clauses and **quantify** the concentration *at loss convergence*

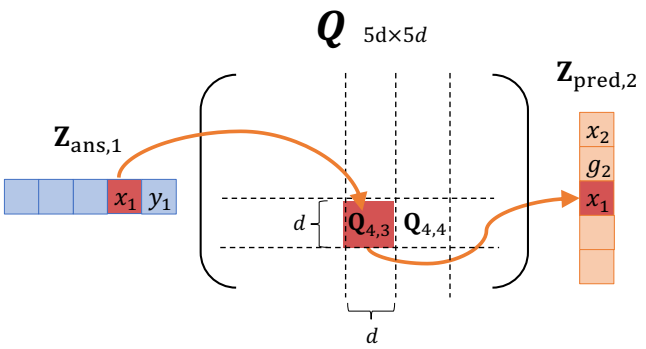
Tracking the attention dynamics

$$\mathbf{Z}^{L,m} \triangleq \underbrace{\boxed{x_1 = g_1(x_0)}}_{\mathbf{Z}_{\text{pred},1}}, \ldots, \boxed{x_L = g_L(x_{L-1})} \quad \underbrace{\boxed{x_0 = y_0}}_{\mathbf{Z}_{\text{ans},0}}, \ldots, \underbrace{\boxed{x_m = y_m}}_{\text{query } \mathbf{Z}_{\text{ans},m}}$$

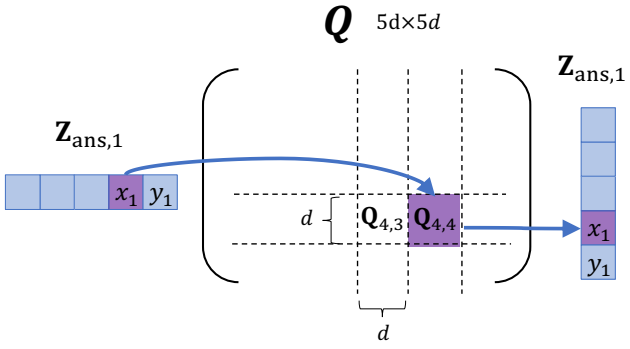
Attention $\mathbf{Attn}_{\text{ans},m\to\mathbf{k}}$: attention score from $\mathbf{Z}_{\text{ans},m} \to \mathbf{Z}_{\mathbf{k}}$;

Uniform initialization: attention scores are the same for all clauses.

Predicate look up

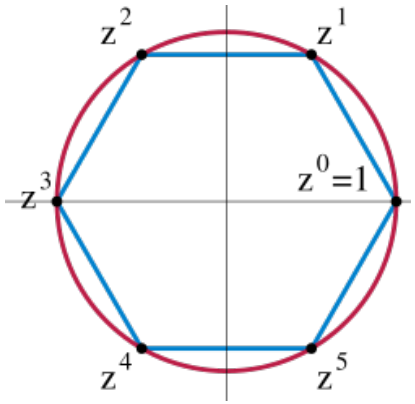


Self attention



Training dynamic: $\mathbf{Attn}_{\text{ans},\ell\to\text{pred},\ell} + \mathbf{Attn}_{\text{ans},\ell\to\text{ans},\ell} \to 1$

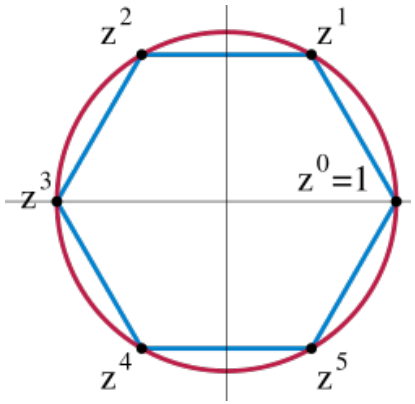
Two group actions

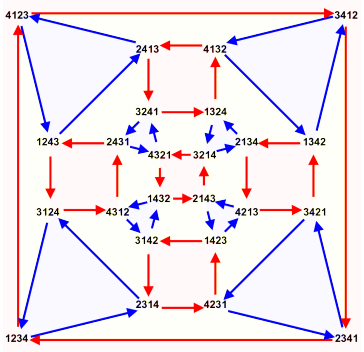


Simply transitive action:

- e.g., C_n, cyclic group (mod n)
- $\forall y, y'$, there is a **unique** $g \in \mathcal{G}$ s.t., $g(y) = y'$
- For example, cyclic group acts on $[n]$.

Two group actions





Simply transitive action:

- e.g., C_n, cyclic group (mod n)
- $\forall y, y'$, there is a **unique** $g \in \mathcal{G}$ s.t., $g(y) = y'$
- For example, cyclic group acts on $[n]$.

Symmetry action:

- e.g., S_n, all permutations of n elements
- $\mathcal{G}_{y \to y'} = \{g \in \mathcal{G} : \ g(y) = y'\}$ is **large**
- For example, S_n acts on $[n]$.

Length generalization for simple transitive group

Theorem (Short-to-poly length generalization)

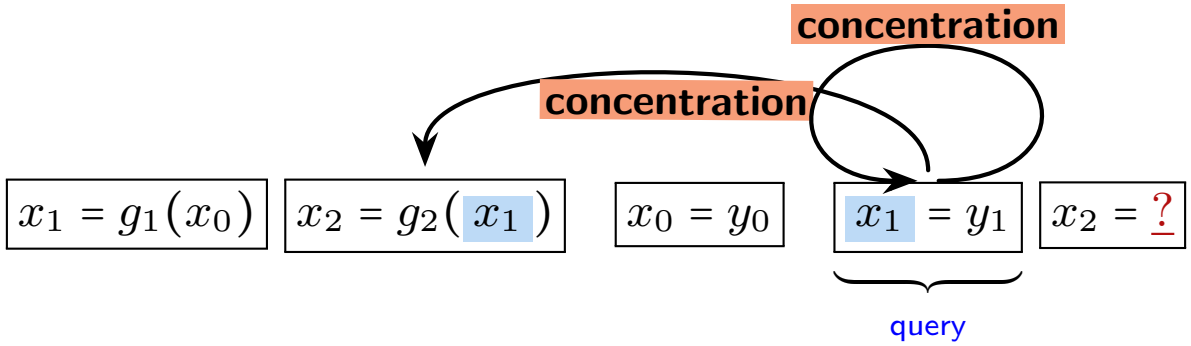
For simply transitive $\mathcal{G}$, the model trained via GD on the LEGO task of $L \leq 2$ can directly solve tasks for $L \leq O(d^c)$ with some constant $0 < c < 1$.

Length generalization for simple transitive group

Theorem (Short-to-poly length generalization)

For simply transitive $\mathcal{G}$, the model trained via GD on the LEGO task of $L \leq 2$ can directly solve tasks for $L \leq O(d^c)$ with some constant $0 < c < 1$.

Mechanism: MLP perfectly captures action; while attention learns to concentrate on the **relevant** clauses.



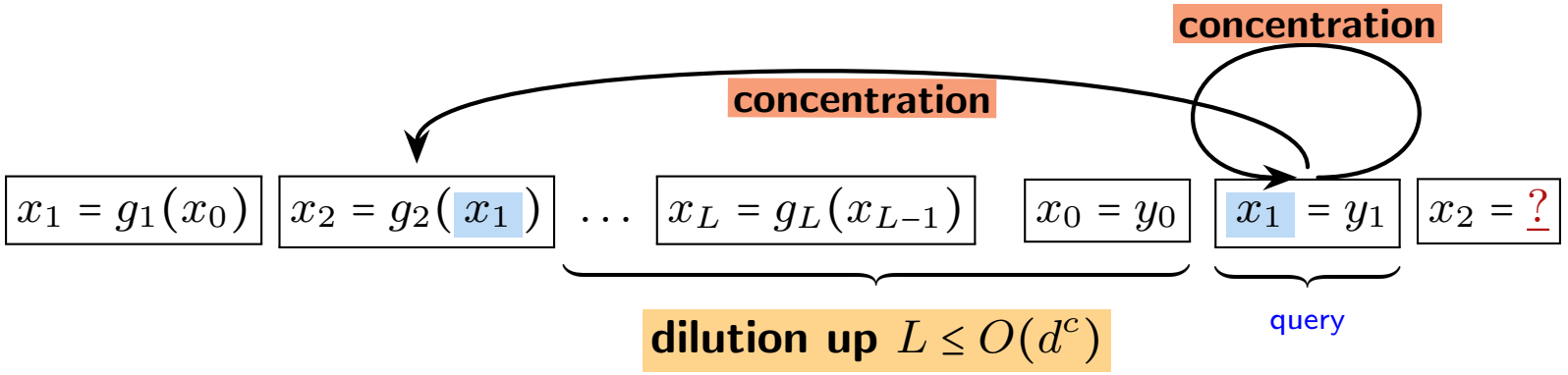
Attention concentration: $\mathbf{Attn}_{\text{ans},\ell\to\text{pred},\ell} + \mathbf{Attn}_{\text{ans},\ell\to\text{ans},\ell} = 1 - \frac{1}{O(d^c)}$

Length generalization for simple transitive group

Theorem (Short-to-poly length generalization)

For simply transitive $\mathcal{G}$, the model trained via GD on the LEGO task of $L \leq 2$ can directly solve tasks for $L \leq O(d^c)$ with some constant $0 < c < 1$.

Mechanism: MLP perfectly captures action; while attention learns to concentrate on the **relevant** clauses.



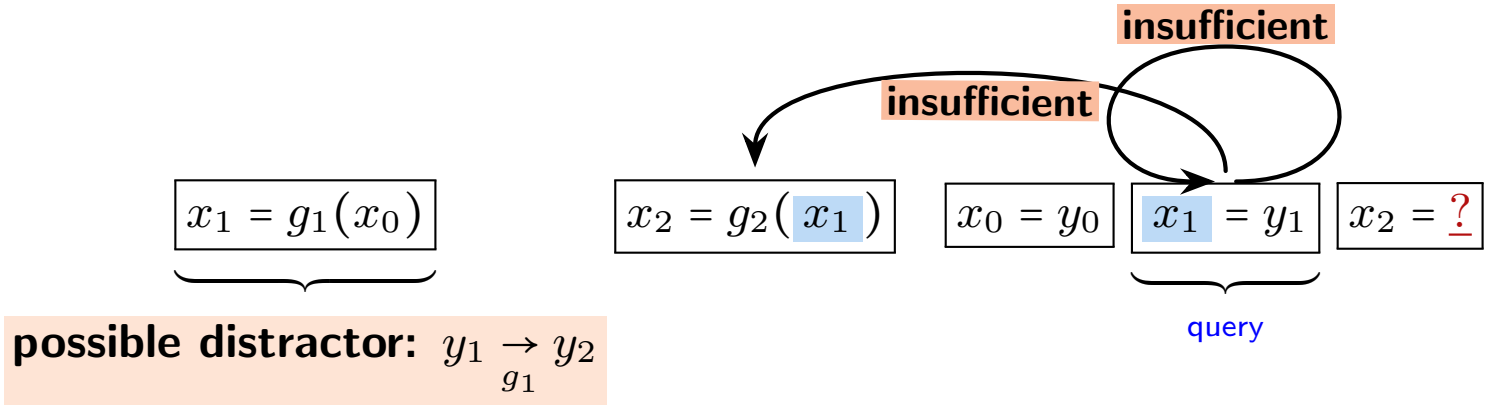
Attention concentration: $\mathbf{Attn}_{\text{ans},\ell\to\text{pred},\ell} + \mathbf{Attn}_{\text{ans},\ell\to\text{ans},\ell} = 1 - \frac{1}{O(d^c)}$

Length generalization for symmetry group

Hardness from symmetry $|\mathcal{G}_{y \to y'}| \gg 1$, simply transitive $|\mathcal{G}_{y \to y'}| = 1$.

Length generalization for symmetry group

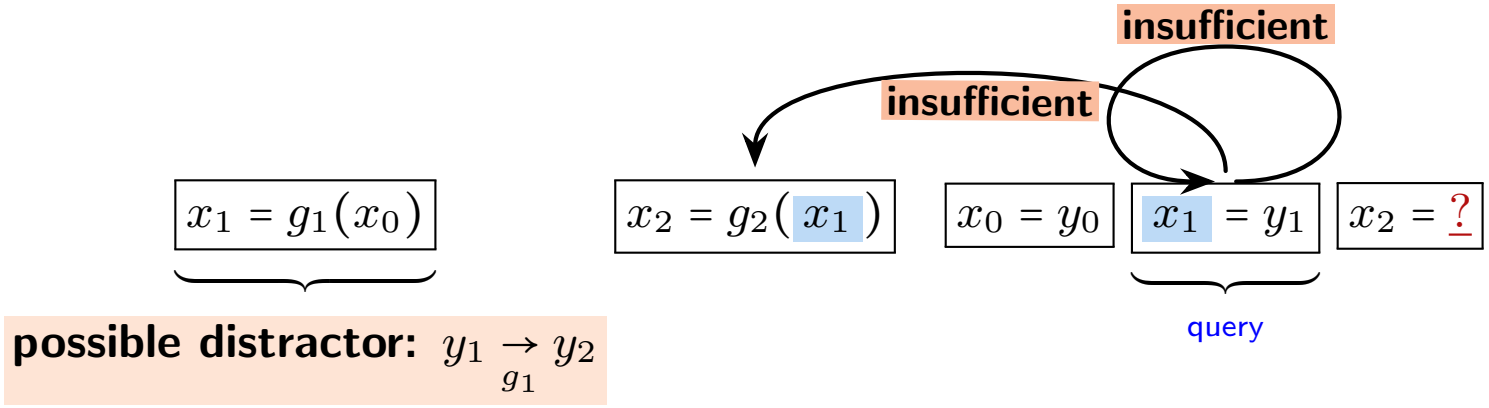
Hardness from symmetry $|\mathcal{G}_{y \to y'}| \gg 1$, simply transitive $|\mathcal{G}_{y \to y'}| = 1$.
Predicate lookup faces many **near-duplicates**:



$$\mathbf{Attn}_{\text{ans},\ell \to \text{pred},\ell} + \mathbf{Attn}_{\text{ans},\ell \to \text{ans},\ell} = 1 - \text{small constant}$$

Length generalization for symmetry group

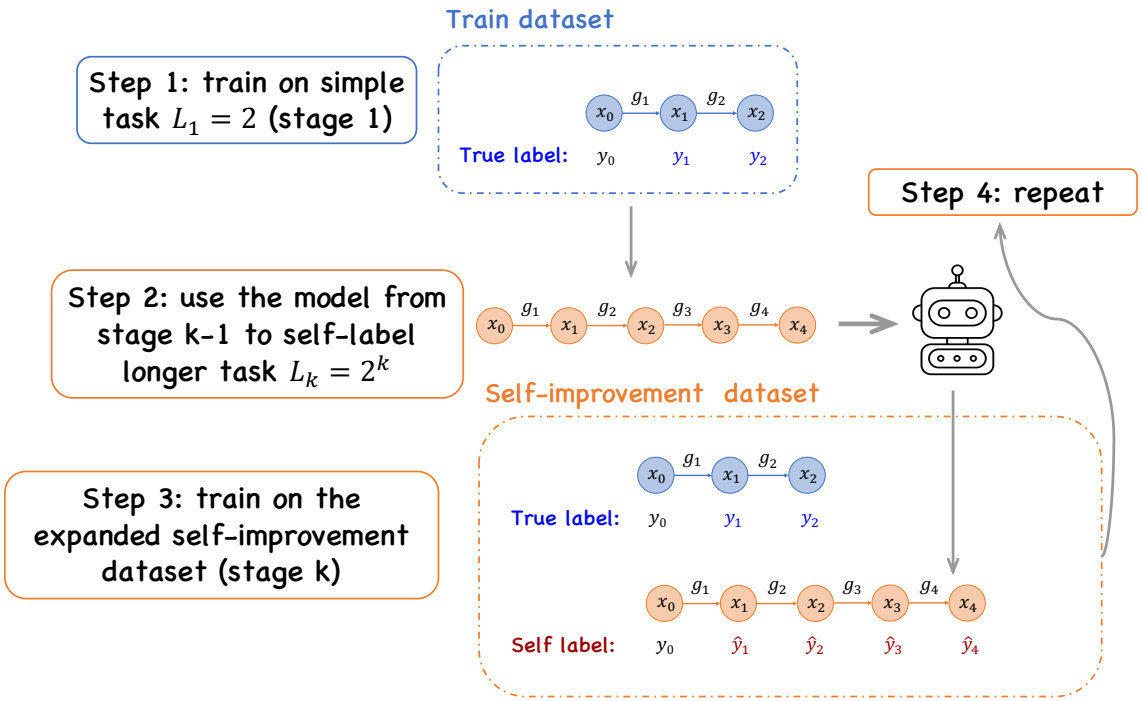
Hardness from symmetry $|\mathcal{G}_{y \to y'}| \gg 1$, simply transitive $|\mathcal{G}_{y \to y'}| = 1$.
Predicate lookup faces many **near-duplicates**:



$$\mathbf{Attn}_{\text{ans},\ell \to \text{pred},\ell} + \mathbf{Attn}_{\text{ans},\ell \to \text{ans},\ell} = 1 - \text{small constant}$$

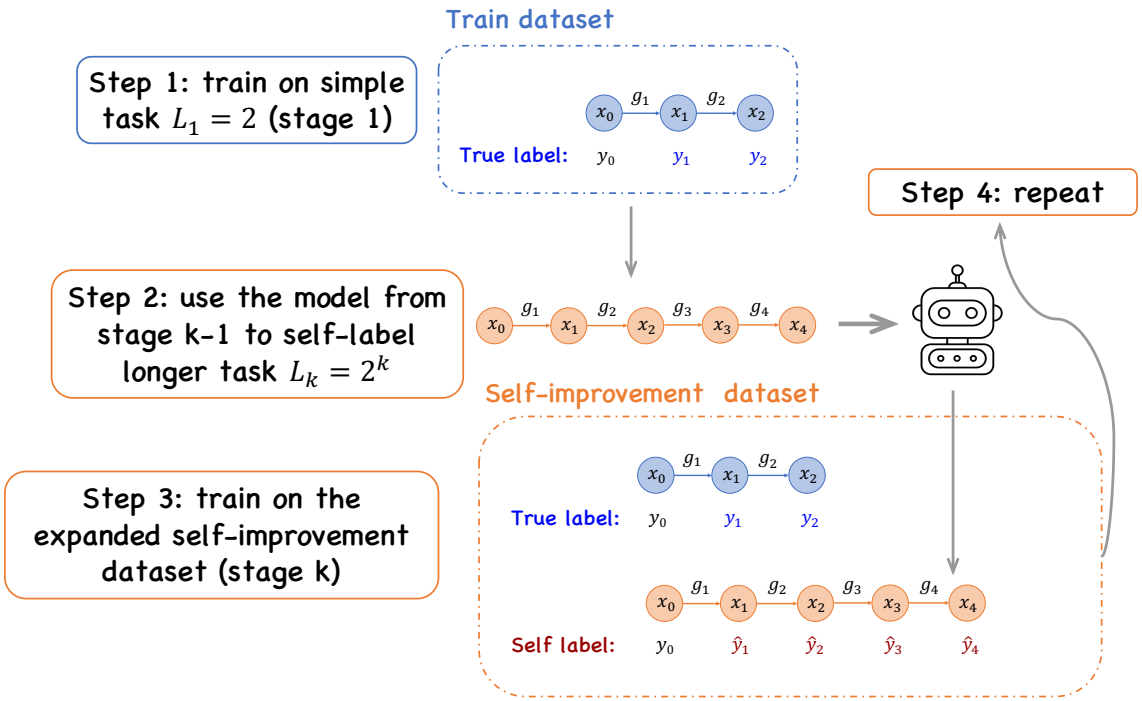
No guarantee beyond **constant**-factor length generalization!

Self-improvement provably helps



Inspired by (Lee et al., 2025)

Self-improvement provably helps

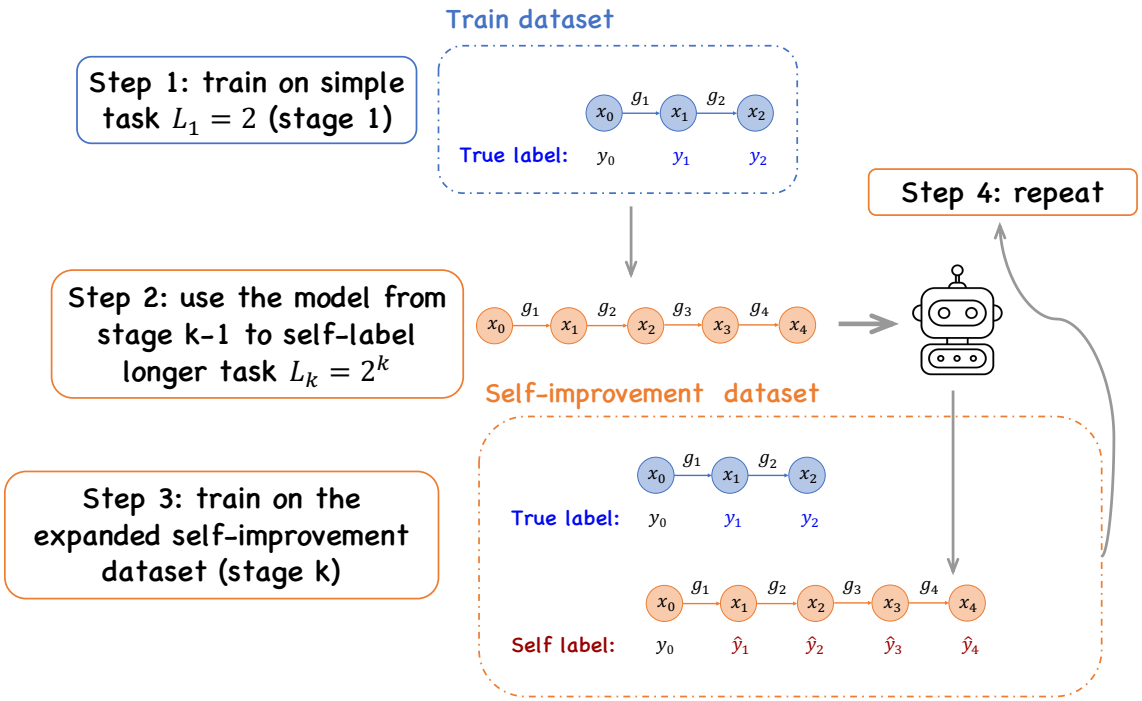


Inspired by (Lee et al., 2025)

Theorem (Self-improvement for length generalization)

If $\mathcal{G}$ is symmetry (S_n), then the model trained via self-training on $L = 1, 2, \cdots, 2^{k-1}$ can generalize to solve task of $L = 2^k$. After $\Theta(\log d)$ stages, it can solve tasks of $L \leq d$.

Self-improvement provably helps



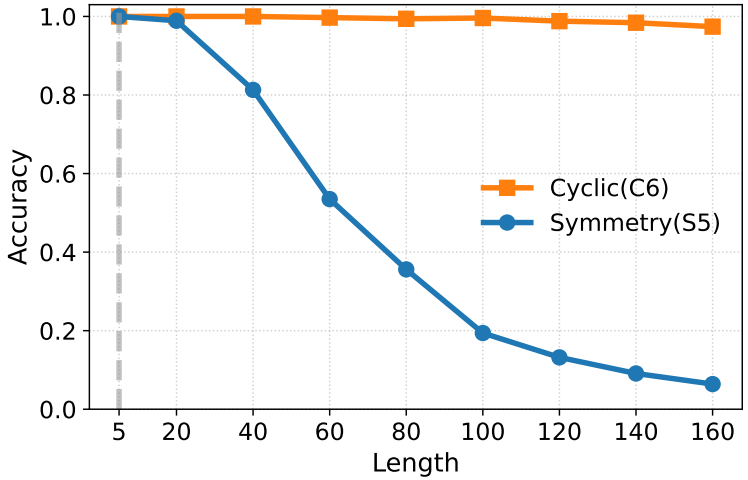
Inspired by (Lee et al., 2025)

Theorem (Self-improvement for length generalization)

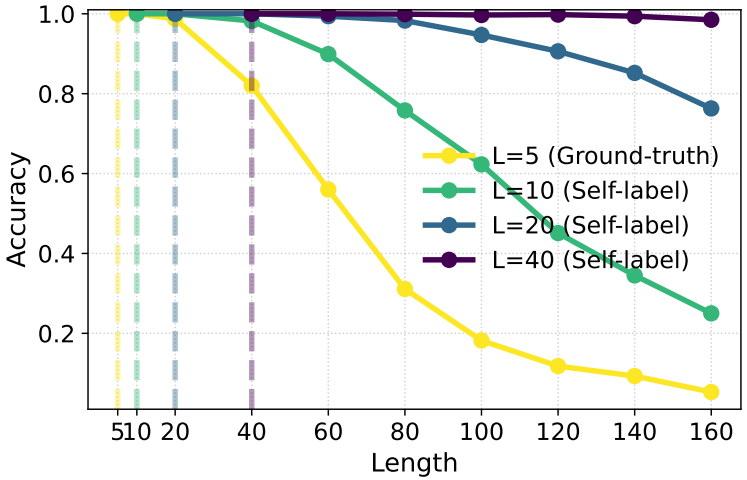
If $\mathcal{G}$ is symmetry (S_n), then the model trained via self-training on $L = 1, 2, \cdots, 2^{k-1}$ can generalize to solve task of $L = 2^k$. After $\Theta(\log d)$ stages, it can solve tasks of $L \leq d$.

Transformer with CoTs **learns** to solve problems in NC^1!

Numerical validation



direct length generalization



self-improvement

Self-improvement improves length generalization for harder tasks

Aside: statistical benefit of CoT

So far, we have assumed access to an infinite amount of data... but CoT can also impact the statistical learnability!

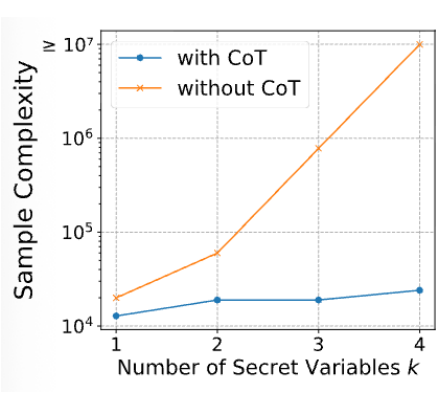
Aside: statistical benefit of CoT

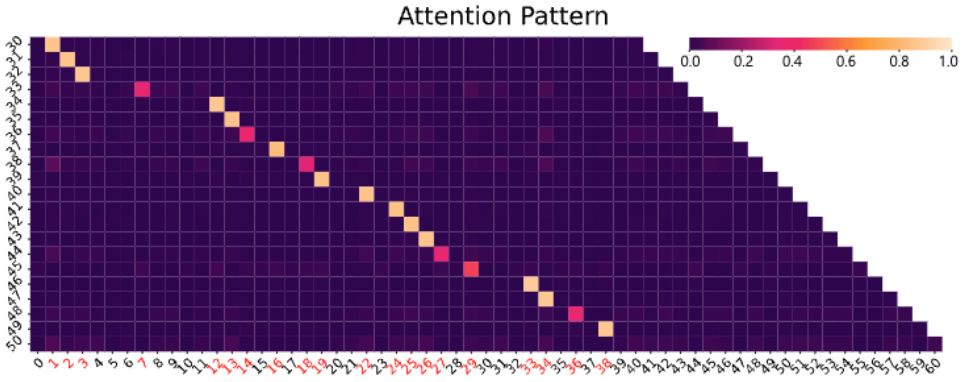
So far, we have assumed access to an infinite amount of data... but CoT can also impact the statistical learnability!

(Wen et al., 2024) shows that for parity learning problem with planted indices $\{i_1, \ldots, i_k\}$:

$$b_{i_1} \oplus b_{i_2} \oplus \ldots \oplus b_{i_k}$$

- without CoT, the sample size is at least exponential;
- with CoT, the sample size is within polynomial.





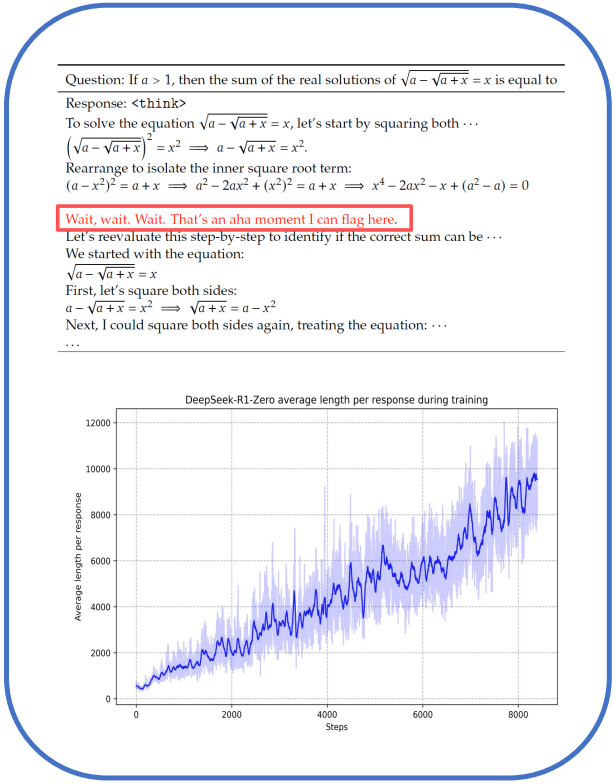
CoT allows sparse attention patterns that are statistically easier to learn.

Reinforcement learning with verifiable rewards

RL with verifiable rewards

RL with verifiable rewards (RLVR): In early 2025, Deepseek released R1, and revealed the training recipe called RLVR.

- The model is only rewarded for accuracy and format.
- No outcome or process neural reward model.



Does RLVR learn new capabilities?

"Reasoning patterns of larger models can be distilled into smaller models, resulting in **better performance** *compared to the reasoning patterns* **discovered through RL** *on small models."*

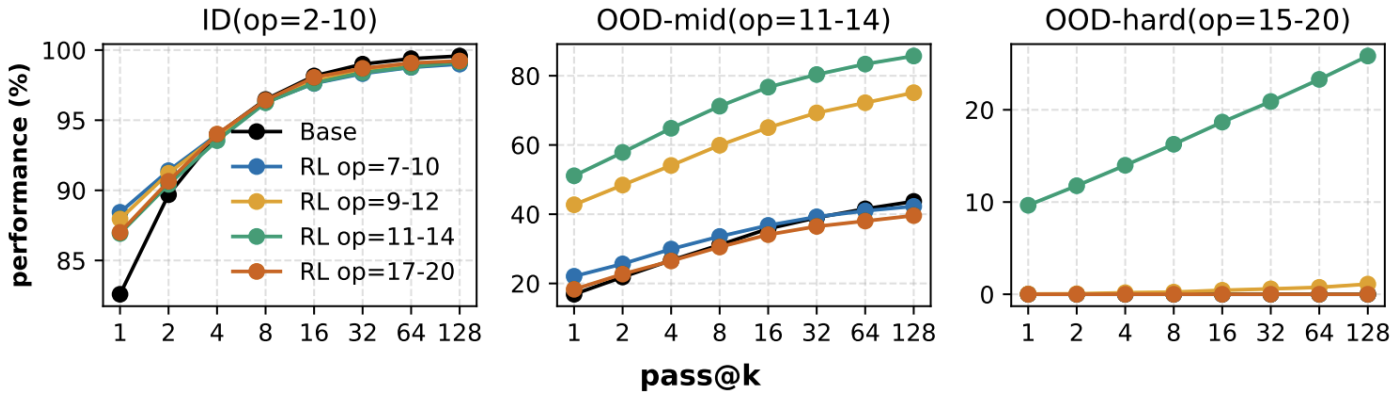
— DeepSeek-R1 [2025]

Model	AIME 2024		MATH	GPQA Diamond	LiveCode Bench
	pass@1	cons@64	pass@1	pass@1	pass@1
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9
Qwen2.5-32B-Zero	47.0	60.0	91.6	55.0	40.2
DeepSeek-R1-Distill-Qwen-32B	**72.6**	**83.3**	**94.3**	**62.1**	**57.2**

(Figure credit: Deepseek-R1)

The ceiling of RLVR seems to depend on the **capability** of base LLMs.

When is RL most effective?



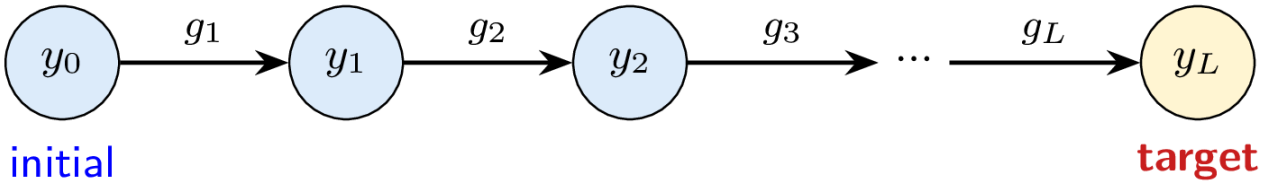
RL is most effective at the **edge of competence** [Yue et al. 2025]:

- No saturation: the task is not heavily covered during pre-training;
- "Just-right" difficulty: neither too easy nor too hard.

Our question: How does RLVR improve model's capabilities at the edge of competence?

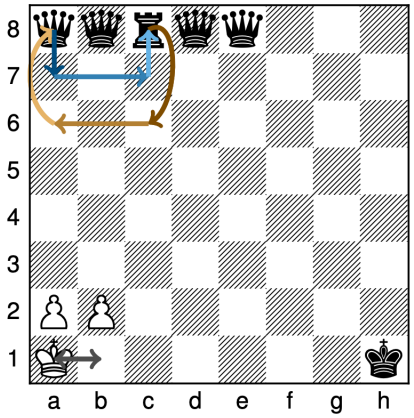
Revisit state tracking from an RL perspective

Given a group $\mathcal{G}$ acting on a finite value set $\mathcal{Y}$:



Goal: calculate the last value $y_L = g_L(g_{L-1}(\cdots g_1(y_0)))$ given the sequence of transitions $g_i \in \mathcal{G}$ with an initial value $y_0 \in \mathcal{Y}$.

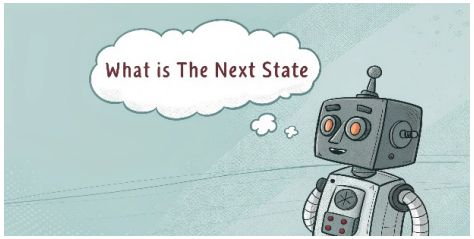
Playing Chess



Images modified from Merrill et al., 2024

Code evaluation

```
x = [0, 0, 1, 0, 0]
x[1], x[3] = x[3], x[1] # Swap 1, 3
```



Entities tracking

Alice, Bob, Carl, Dan, and Emma each have a coin. All are dimes except Carl's. Alice and Carl trade coins.

How does the transformer reason sequentially?

Transformer as autoregressive policy: given (prompt, y_0),

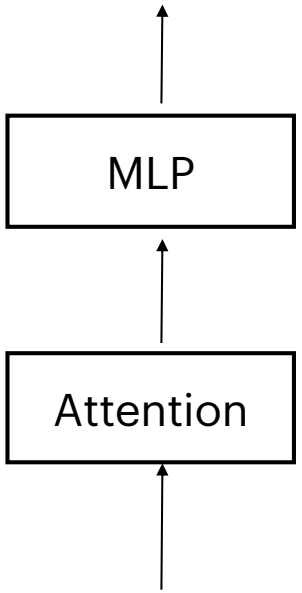
$$\pi_\theta(\widehat{y}^\ell \mid \text{prompt}, y_0) = \prod_{k=1}^{\ell} \pi_\theta(\widehat{y}_k \mid \text{prompt}, y_{k-1}), \qquad \ell = 1, \ldots, L,$$

where $\widehat{y}^\ell = (\widehat{y}_1, \ldots, \widehat{y}_\ell)$.

Model mechanism: TF = MLP (executor) $\circ$ Attention (retrieval)

Recursion of L-step reasoning: at each step ℓ, recall the target $y_\ell = g_\ell(y_{\ell-1})$.

- **Attention** = retriever: find correct g_ℓ from prompt at step ℓ – horizon-variant
- **MLP** = executor: $(g_\ell, y_{\ell-1}) \mapsto g_\ell(y_{\ell-1})$ with near-perfect accuracy – horizon-invariant



Outcome-based RL objective and policy gradient

Outcome-based RL objective:

$$\mathcal{J}_L(\theta) = \mathbb{E}_{\text{prompt},y_0}\left[\mathbb{E}_{\widehat{y}^L\sim\pi_\theta(\cdot|\text{prompt},y_0)}\left[r(\widehat{y}^L \mid \text{prompt}, y_0)\right]\right],$$

which uses only the terminal reward:

$$r(\widehat{y}^L \mid \text{prompt}, y_0) \triangleq \mathbf{1}\{\widehat{y}_L = y_L\}.$$

Outcome-based RL objective and policy gradient

Outcome-based RL objective:

$$\mathcal{J}_L(\theta) = \mathbb{E}_{\text{prompt},y_0}\Big[\mathbb{E}_{\widehat{y}^L\sim\pi_\theta(\cdot|\text{prompt},y_0)}\big[r(\widehat{y}^L \mid \text{prompt}, y_0)\big]\Big],$$

which uses only the terminal reward:

$$r(\widehat{y}^L \mid \text{prompt}, y_0) \triangleq \mathbf{1}\big\{\widehat{y}_L = y_L\big\}.$$

Length-normalized policy gradient via REINFORCE:

$$\theta^{(t+1)} = \theta^{(t)} + \eta\nabla_\theta\widetilde{\mathcal{J}}_L(\theta)$$

$$\nabla_\theta\widetilde{\mathcal{J}}_L(\theta) = \frac{1}{L}\,\mathbb{E}_{\text{prompt},y_0}\Big[r(\widehat{y}^L \mid \text{prompt}, y_0)\nabla\log\pi_\theta(\widehat{y}^L \mid \text{prompt}, y_0)\Big],$$

where $\widetilde{\mathcal{J}}_L(\theta) = \frac{1}{L}\mathcal{J}_L(\theta)$, and η is the learning rate.

Pretrained atomic skills

Pretrained MLP: we assume MLP is pretrained to capture the atomic skills of group action

$$(g, y) \mapsto g(y)$$

with weight W fixed during RL training.

Pretrained atomic skills

Pretrained MLP: we assume MLP is pretrained to capture the atomic skills of group action

$$(g, y) \mapsto g(y)$$

with weight W fixed during RL training.

When MLP receives $\frac{1}{2}(g+y)$, then for $j = \tau(g(y))$

$$\text{softmax}\Big(\text{MLP}_W\Big(\frac{1}{2}(g+y)\Big)\Big)_j = 1 - \frac{1}{\text{poly}(d)}.$$

Here, $\tau : \mathcal{Y} \to [d]$ index the states for prediction.

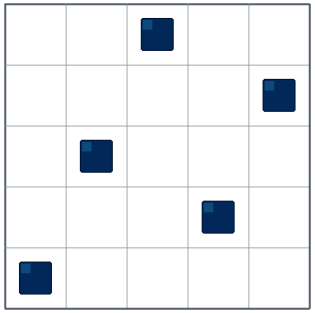
- Our prior work [Huang et al, 2025] provides guarantees for learning such atomic structures via overparameterized ReLU network, with activation level

$$B = C_{\text{mlp}} \log d,$$

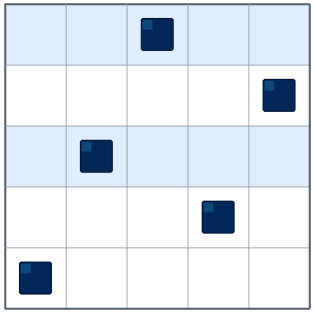
where C_{mlp} is a constant determining the **skill sharpness**.

Attention training

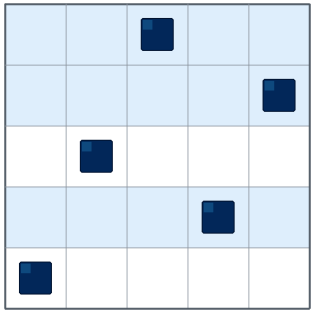
Attention mechanism reused across lengths: the ground truth retrieval pattern is a permutation at the longest possible reasoning length, measuring the *positional alignment*.



ground truth



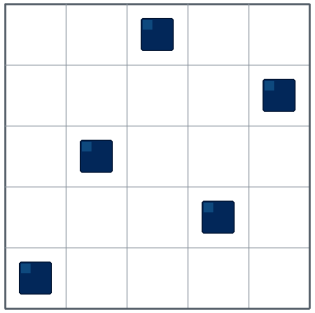
length-2



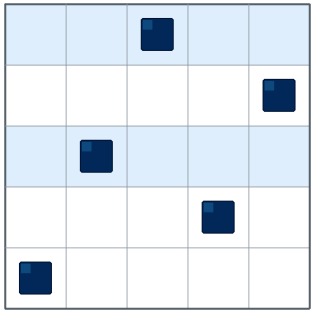
length-3

Attention training

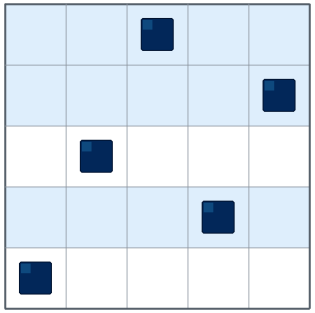
Attention mechanism reused across lengths: the ground truth retrieval pattern is a permutation at the longest possible reasoning length, measuring the *positional alignment*.



ground truth



length-2



length-3

Uniform attention at initialization: We train an attention matrix Q to learn the permutation pattern with

$$Q^{(0)} = 0 \qquad \text{(uniform attention)}.$$

Training dynamic tracks how the attention pattern concentrates throughout training.

Horizon barrier of compositional reasoning

Our first result reveals that RLVR is efficient for **short-horizon** reasoning, but suffers from **flat landscape** beyond a critical horizon.

Theorem (Horizon Barrier)

Assume $\mathcal{G}$ is non-abelian and simply transitive, then

- **Success of RL within short horizon $L < C_{\mathsf{mlp}}$:** *for large enough iterations $T = \widetilde{O}(\frac{\mathrm{poly}(d)}{\eta})$ with $\eta = \frac{1}{\mathrm{poly}(d)}$, the reward*

$$\mathcal{J}_L^{(T)} \geq 1 - \frac{1}{\mathrm{poly}(d)}.$$

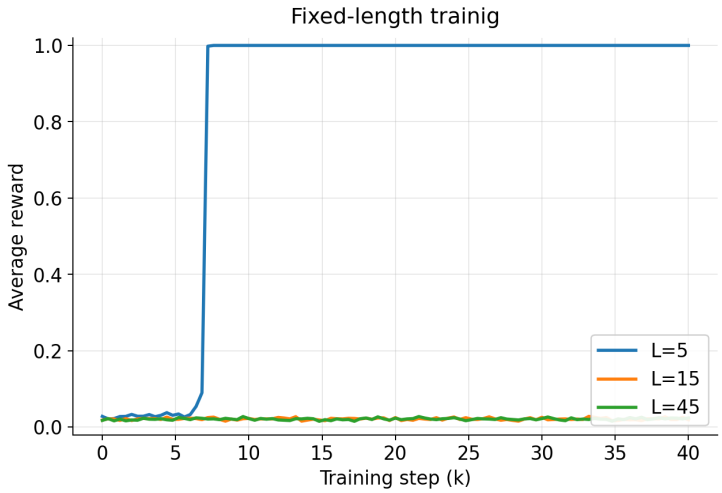
- **Exponentially flat region when $L \geq 2C_{\mathsf{mlp}}$:** *the gradients of $\mathcal{J}_L$ and rewards $\mathcal{J}_L^{(t)}$ satisfy*

$$\left|\left\langle \left[\nabla_Q \mathcal{J}_L^{(t)}\right] x, x' \right\rangle\right| \leq d^{-\Omega(L)}, \qquad \mathcal{J}_L^{(t)} = \frac{1}{d}(1 + o(1)),$$

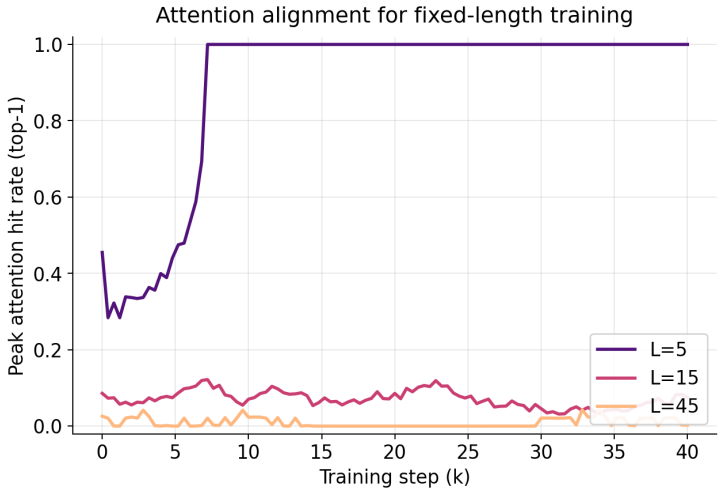
whenever $\max_{x,x' \in \mathcal{X}} \langle Q^{(t)} x, x' \rangle \leq 0.01$.

Single-length training

Experiment setup: we assume a cyclic group $\mathbb{Z}_{96}$ for reasoning length $L = 5, 15, 45$.



Average reward



Attention concentration

RL rapidly learns **short-horizon compositions**, whereas longer horizons exhibit a **near-flat reward plateau.**

Mixed-difficulty RLVR

The previous result showed that RL cannot operate beyond a critical horizon, even when the reward is non-zero by random guesses.

What if we **mix different horizons** together?

Mixed-difficulty distribution: we uniformly mix the horizons

$$\mathcal{L}_R = \{L_1, L_2, \ldots, L_K\}, \quad L_k = \min\{\lceil RL_{k-1} \rceil, L_{\max}\}.$$

Here, $R = L_{k+1}/L_k$ is the difficulty ratio, which controls how **smooth** the curriculum structure in the distribution.

Mixed-difficulty training objective:

$$\mathcal{J}_{\text{mix,R}} = \mathbb{E}_{L\sim\text{Unif}(\mathcal{L}_R)}[\mathcal{J}_L(\theta)]$$

$$y_0 \xrightarrow{g_1} \cdots \xrightarrow{g_{L_1}} y_{L_1}$$

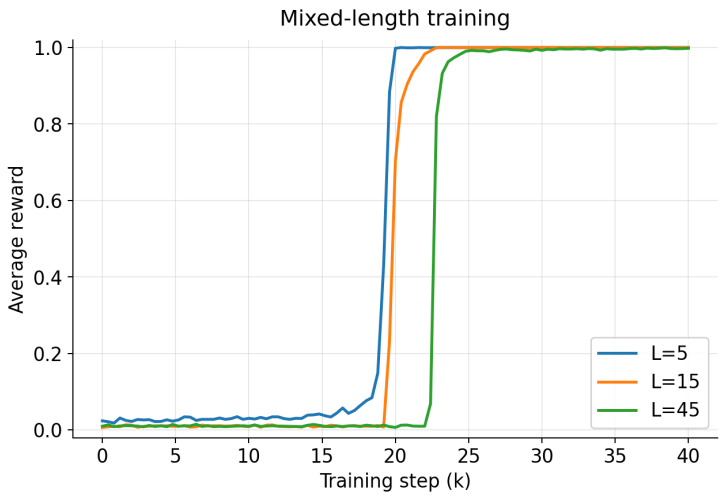
$$y_0 \xrightarrow{g_1} y_1 \xrightarrow{g_2} y_2 \xrightarrow{g_3} \cdots \xrightarrow{g_{L_2}} y_{L_2}$$

$$\vdots$$

$$y_0 \xrightarrow{g_1} y_1 \xrightarrow{g_2} y_2 \xrightarrow{g_3} y_3 \xrightarrow{g_4} y_4 \xrightarrow{g_5} \cdots \xrightarrow{g_{L_K}} y_{L_K}$$

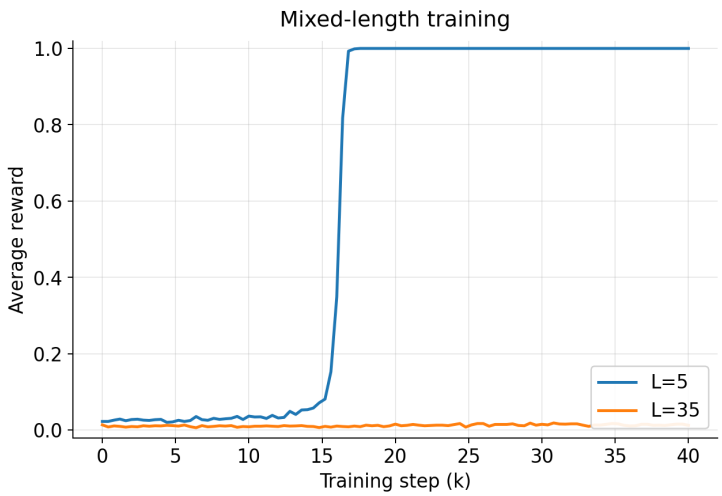
Learning behavior at different difficulty ratios

Relay



$R = 3$: a smoother difficulty spectrum facilitates relay dynamics across horizons.

Grokking



$R = 7$: a larger difficulty ratio yields a prolonged plateau at longer horizons.

The shared reasoning pattern generalizes across horizons through an implicit curriculum provided by the data mix.

RLVR dynamics under mixed difficulty

Stopping times: define the following stopping times for

- *Visible return*: $T_{\mathrm{vis},L} := \min_t\{\mathcal{J}_L^{(t)} \geq 0.01\}$;
- *Mastery*: $T_{\mathrm{mas},L} := \min_t\{\mathcal{J}_L^{(t)} \geq 0.99\}$.

Theorem (Huang et al., 2026)

Suppose $L_1 \leq C_{\mathsf{mlp}}$. Assume $\mathcal{G}$ is non-abelian and simply transitive,

- **Grokking (Long plateaus when $R \geq \omega(1)$):** *before the next horizon L_{k+1} sees visible returns, there is a long plateau:*

$$T_{\mathrm{vis},k+1} - T_{\mathrm{mas},k} \gtrsim \frac{L_{\max}}{\eta} \cdot d^{C_{\mathsf{mlp}}-1}.$$

- **Relay (Short plateaus when $R \leq O(1)$):** *before L_{k+1} sees visible returns, the plateau period is short:*

$$T_{\mathrm{vis},k+1} - T_{\mathrm{mas},k} \lesssim \frac{L_{\max}}{\eta} \cdot d^{C_{\mathsf{mlp}}\left(1-\frac{C_{\mathsf{mlp}}}{C_{\mathsf{mlp}}+R}\right)-1}.$$

- **(Phase transitions)** *Once horizon L_{k+1} reaches visible return, it arrives at mastery time quickly: $T_{\mathrm{mas},k} - T_{\mathrm{vis},k} \lesssim \frac{L_{\max}}{\eta} \cdot L_{k+1}$.*

Gradient characterization

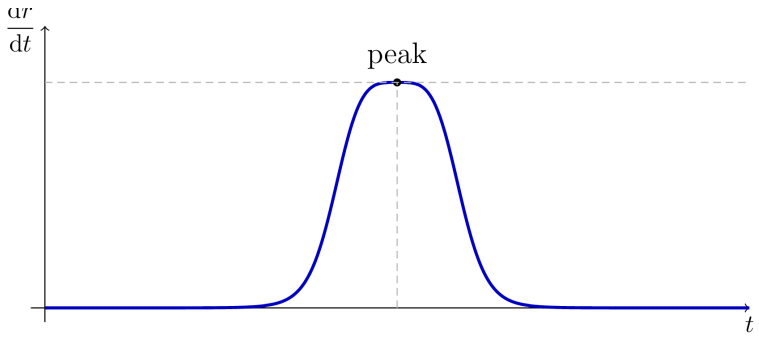
Step-wise probability margin: how much target dominates

$$\Delta_L :\approx p_{L,\text{target}} - p_{L,\text{non-target}}.$$

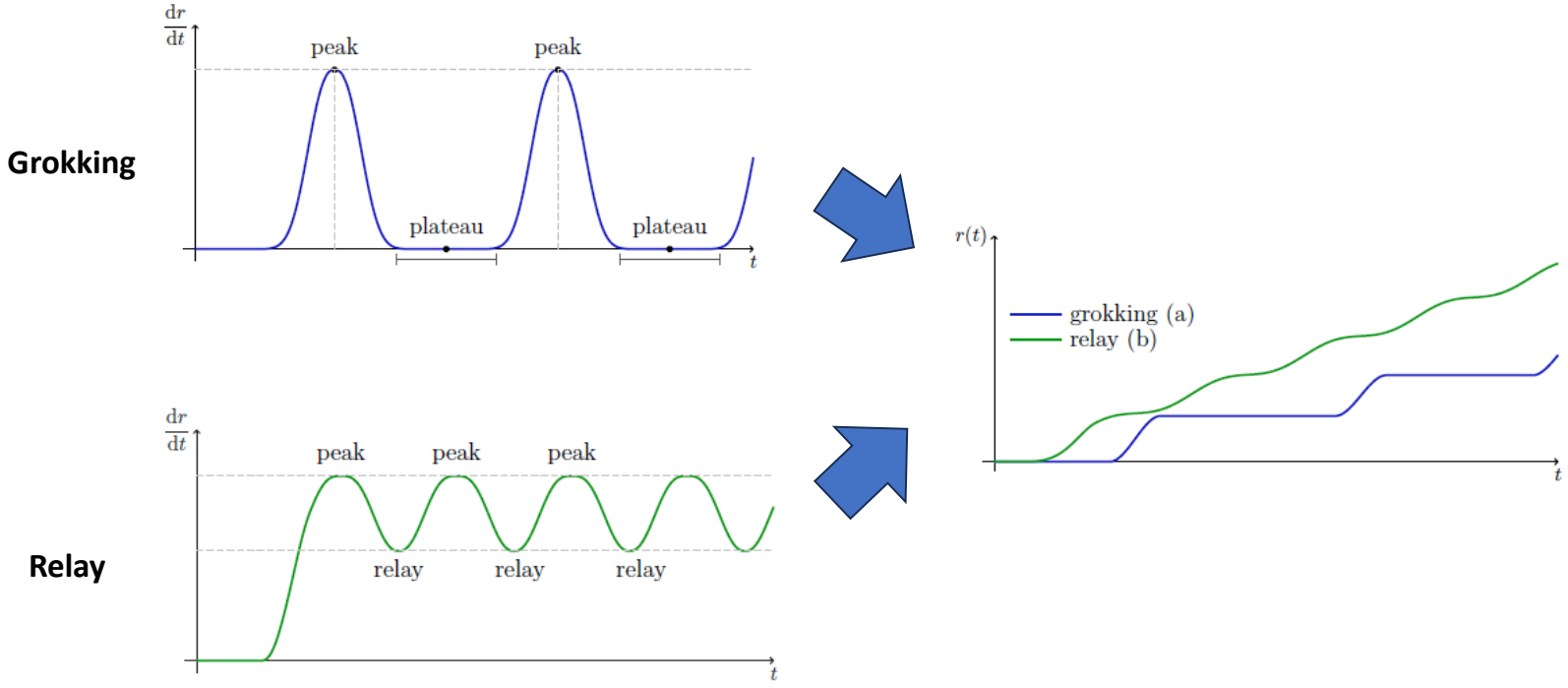
A key policy gradient characterization for aggregated attention q:

$$\nabla_q \widetilde{\mathcal{J}}_L \propto \underbrace{(\Delta_L)^L}_{\text{success over } L \text{ steps}} \cdot \underbrace{(1-\Delta_L)}_{\text{room to improve}}$$

- Δ_L small $\Rightarrow (\Delta_L)^L$ exponentially suppresses gradient (cold start)
- $\Delta_L \approx 1 \Rightarrow 1-\Delta_L$ vanishes (saturated)



Cartoon illustration of RLVR dynamics

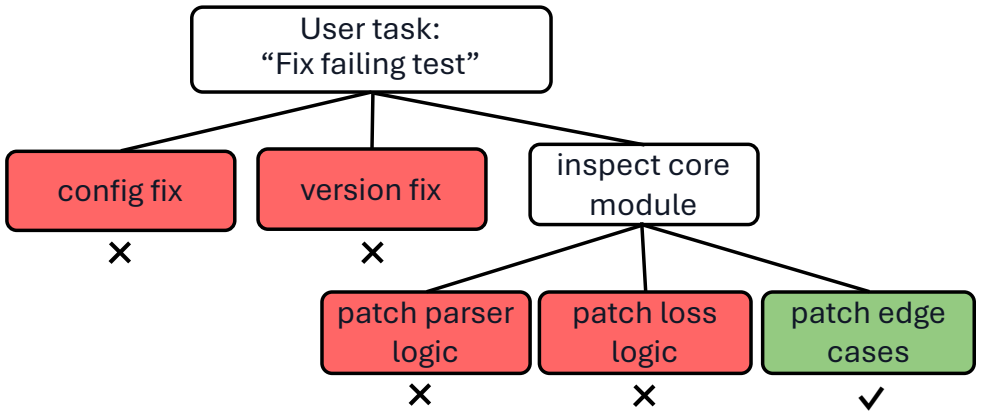


- **Grokking**: shorter-horizon saturates long before longer-horizon escapes random guessing.
- **Relay**: at the **edge of competence**, shorter-horizon still improves while longer-horizon already escapes randomness.

Reinforcement learning for agentic search

Search in agentic and reasoning tasks

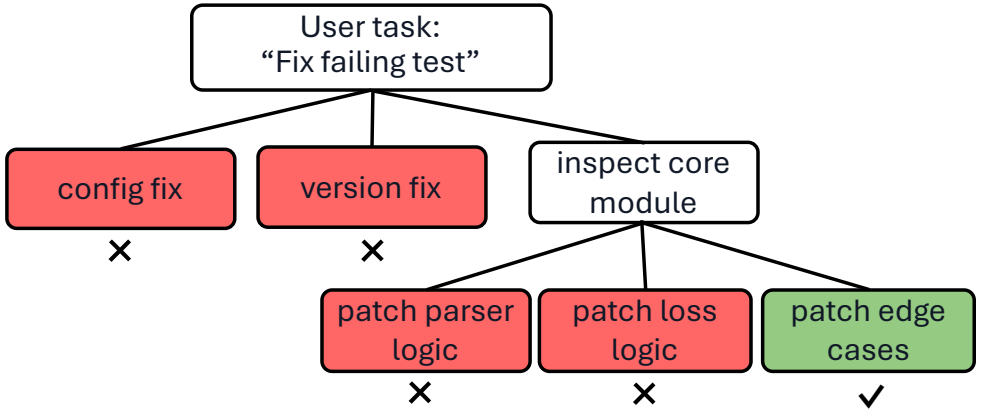
Search is a crucial algorithmic primitive in agentic and reasoning tasks.



- The structure of the search space and target of search are often a priori unknown to the agent.
- Agents need to perform backtracking to recover from failures.

Search in agentic and reasoning tasks

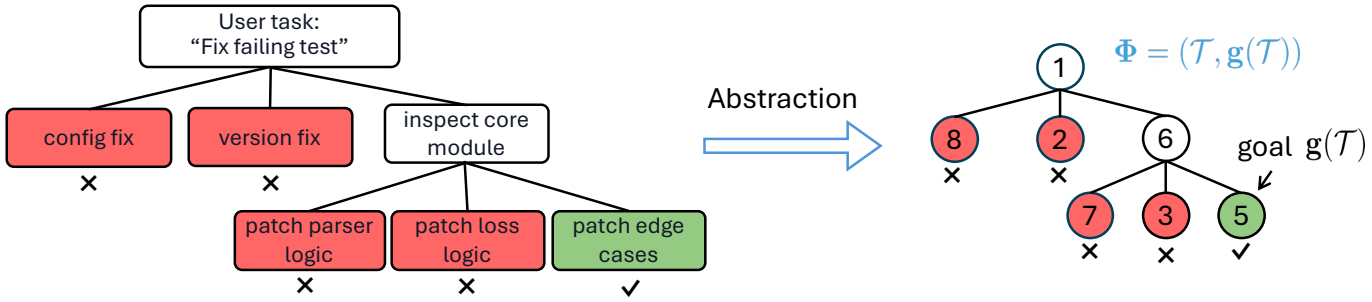
Search is a crucial algorithmic primitive in agentic and reasoning tasks.



- The structure of the search space and target of search are often a priori unknown to the agent.
- Agents need to perform backtracking to recover from failures.

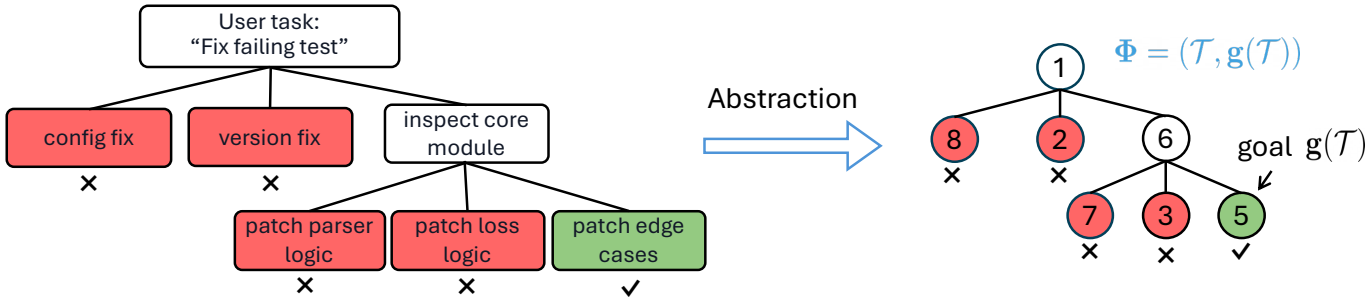
How does RL enable agents to search?

Abstraction as tree search



Goal: transformers act on the tree to find the goal node.

Abstraction as tree search

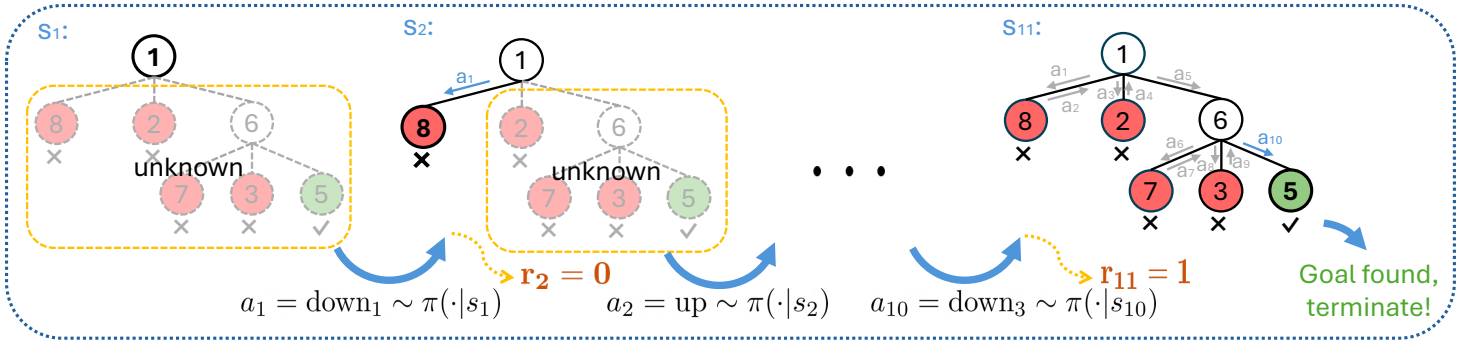


Goal: transformers act on the tree to find the goal node.

How does this differ from supervised fine-tuning (SFT)?

- In SFT, the complete tree and entire root-to-goal path is used to supervise a path-finding mechanism (Yang et al., 2025).
- In RL, the tree is *revealed* as the agent explores it through trial-and-error, with feedback about if it is a leaf or goal node.

RL environment



Episodic setting:

- A new tree is generated at the beginning of each episode.
- Transformer as an autoregressive policy over the action space:

$$\pi_\theta(\cdot|\text{context}) \sim \Delta(\mathcal{A}), \qquad \text{where} \quad \mathcal{A} = \{\text{up}, \text{down}_k\}$$

  it can visit children nodes, backtrack to parent nodes, etc...
- Environment reveals the next node in the tree, and its nodal properties (leaf/non-leaf).

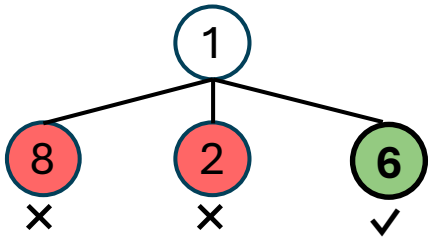
Can RL learn this search mechanism?

Yes!

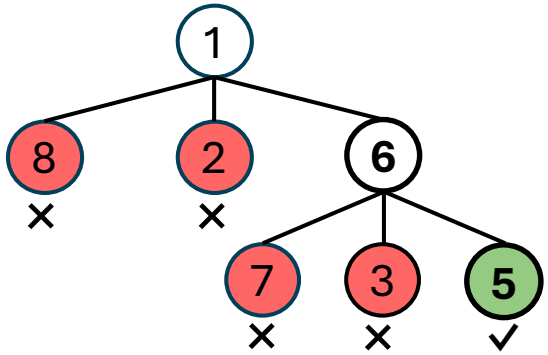
Theorem (Yang et al., 2026)

A one-layer two-head transformer learns to implement **depth-first search (DFS)** *via curriculum training on depth-1 and depth-2 trees sequentially. Furthermore, it generalizes directly to deeper trees.*

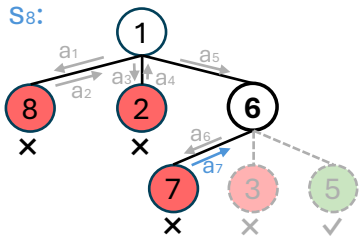
depth-1 trees



depth-2 trees



Attention head specialization



$$E^{(8)} = \begin{array}{c} \\ n \\ a \\ n' \\ c \end{array} \begin{array}{c} \text{query} \\ \begin{pmatrix} 0 & 1 & 8 & 1 & 2 & 1 & 6 & 7 \\ 0 & \text{down}_1 & \text{up} & \text{down}_2 & \text{up} & \text{down}_3 & \text{down}_1 & \text{up} \\ 1 & 8 & 1 & 2 & 1 & 6 & 7 & 6 \\ 0 & \times & 0 & \times & 0 & 0 & \times & 0 \end{pmatrix} \end{array}$$

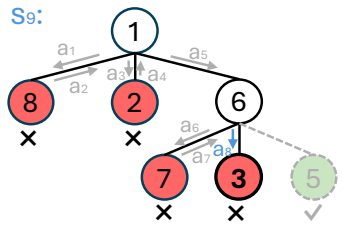
$$a_8 = \text{down}_2 \sim \pi_\theta(E^{(8)}) \approx (0, 0.5, 0.5, 0)^\top$$

Task specialization of attention heads:

- One head retrieves action history and suppresses tried children from being selected again;
- One head uses label history to promote action up at non-goal leaves and suppress up at internal nodes.

The two heads coordinate to implement depth-first search.

Attention head specialization

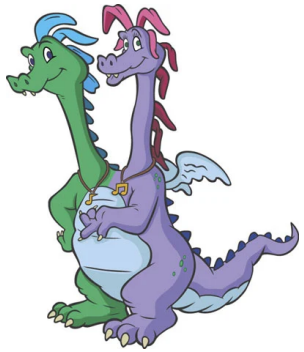


$$E^{(9)} = \begin{array}{c} \\ n \\ a \\ n' \\ c \end{array} \begin{array}{c} \text{query} \\ \begin{pmatrix} 0 & 1 & 8 & 1 & 2 & 1 & 6 & 7 & 6 \\ 0 & \text{down}_1 & \text{up} & \text{down}_2 & \text{up} & \text{down}_3 & \text{down}_1 & \text{up} & \text{down}_2 \\ 1 & 8 & 1 & 2 & 1 & 6 & 7 & 6 & 3 \\ 0 & \times & 0 & \times & 0 & 0 & \times & 0 & \times \end{pmatrix} \end{array}$$

$$a_9 = \text{up} \sim \pi_\theta(E^{(9)}) \approx (0, 0, 0, 1)^\top$$

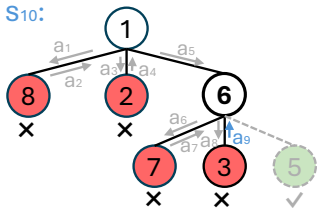
Task specialization of attention heads:

- One head retrieves action history and suppresses tried children from being selected again;
- One head uses label history to promote action up at non-goal leaves and suppress up at internal nodes.



The two heads coordinate to implement depth-first search.

Attention head specialization



Goal found, terminate!

$$E^{(10)} = \begin{array}{c} \\ n \\ a \\ n' \\ c \end{array} \begin{pmatrix} 0 & 1 & 8 & 1 & 2 & 1 & 6 & 7 & 6 & 3 \\ 0 & \text{down}_1 & \text{up} & \text{down}_2 & \text{up} & \text{down}_3 & \text{down}_1 & \text{up} & \text{down}_2 & \text{up} \\ 1 & 8 & 1 & 2 & 1 & 6 & 7 & 6 & 3 & 6 \\ 0 & \times & 0 & \times & 0 & 0 & \times & 0 & \times & 0 \end{pmatrix} \begin{array}{c} \\ \\ \\ \\ \text{query} \end{array}$$

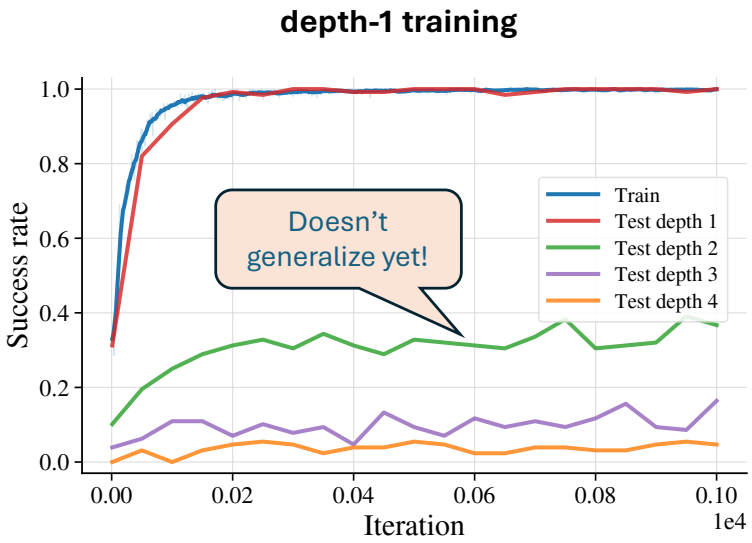
$$a_{10} = \text{down}_3 \sim \pi_\theta(E^{(10)}) \approx (0, 0, 1, 0)^\top$$

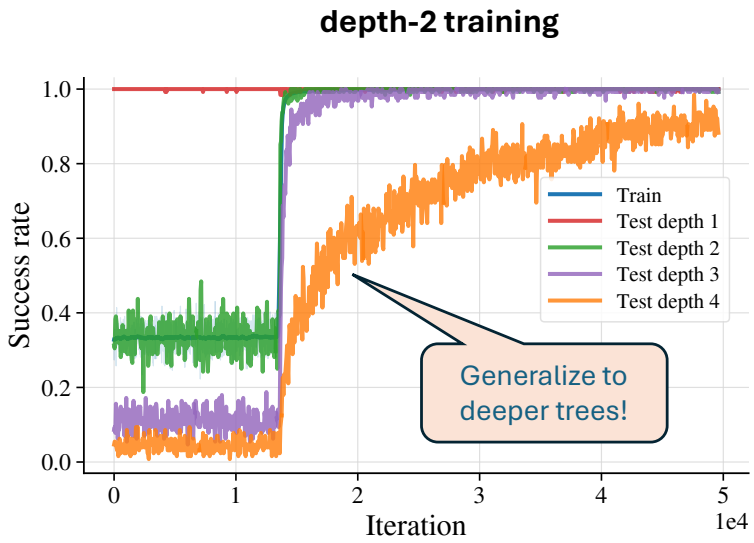
Task specialization of attention heads:

- One head retrieves action history and suppresses tried children from being selected again;
- One head uses label history to promote action up at non-goal leaves and suppress up at internal nodes.

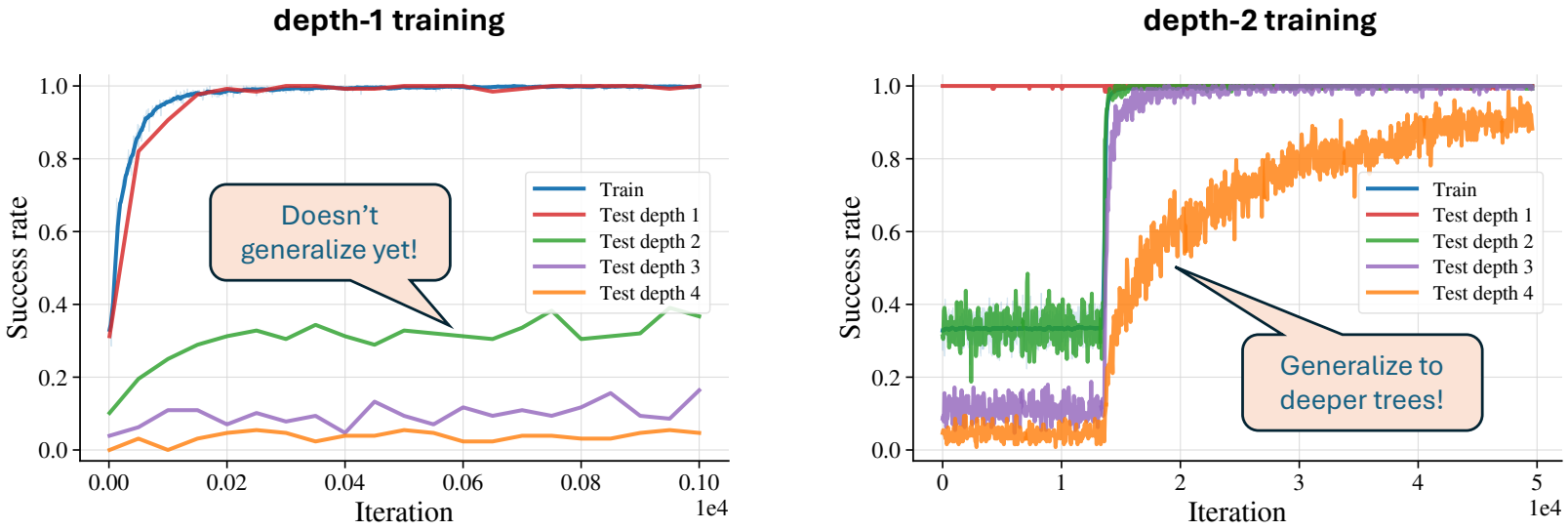
The two heads coordinate to implement depth-first search.

Numerical illustration





Numerical illustration



RL trains transformers to perform the depth-first search algorithm via curriculum learning.

Summary

Expressivity:

- CoT turns test-time tokens into **serial computational depth**, enabling fixed-depth transformers to perform inherently sequential computations.

Optimization:

- Training learns a reusable local algorithm: **attention retrieves and routes**, while the **MLP executes** the atomic update. **RLVR** learns much slower than **SFT**, but allows cold start without data.

Generalization:

- **Attention concentration** enables skill sharpening that succeeds beyond the training horizon, and **curriculum learning** allows faster training with better length generalization and long-horizon planning.

CoT supplies computational depth; training identifies the algorithmic primitives; curriculum carries it to longer horizons.

References I

- "*Attention Is All You Need,*" Vaswani et al., *NeurIPS*, 2017.
- "*Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,*" Wei et al., *NeurIPS*, 2022.
- "*Learning to Reason with LLMs,*" OpenAI, 2024.
- "*Chain of Thought Empowers Transformers to Solve Inherently Serial Problems,*" Li et al., *ICLR*, 2024.
- "*Transformers Learn Shortcuts to Automata,*" Liu et al., *ICLR*, 2023.
- "*The Expressive Power of Transformers with Chain of Thought,*" Merrill, Sabharwal, *ICLR*, 2024.
- "*Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective,*" Feng et al., *NeurIPS*, 2023.
- "*A Little Depth Goes a Long Way: The Expressive Power of Log-Depth Transformers,*" Merrill, Sabharwal, *NeurIPS*, 2025.
- "*Saturated Transformers are Constant-Depth Threshold Circuits,*" Merrill, Sabharwal, Smith, *TACL*, 2022.

References II

- "*The Parallelism Tradeoff: Limitations of Log-Precision Transformers,*" Merrill, Sabharwal, *TACL*, 2023.
- "*Exploring Length Generalization in Large Language Models,*" Anil et al., *NeurIPS*, 2022.
- "*Multi-head Transformers Provably Learn Symbolic Multi-step Reasoning via Gradient Descent,*" Yang, Huang, Liang, Chi, *NeurIPS*, 2025.
- "*Transformers Provably Learn Chain-of-Thought Reasoning with Length Generalization,*" Huang et al., *NeurIPS*, 2025.
- "*Unveiling Transformers with LEGO: a synthetic reasoning task,*" Zhang et al., *arXiv preprint*, 2022.
- "*Self-Improving Transformers Overcome Easy-to-Hard and Length Generalization Challenges,*" Lee et al., *ICML*, 2025.
- "*Learning Compositional Functions with Transformers from Easy-to-Hard Data,*" Wang et al., *COLT*, 2025.

References III

- "*From Sparse Dependence to Sparse Attention: Unveiling How Chain-of-Thought Enhances Transformer Sample Efficiency,*" Wen et al., *ICLR*, 2025.
- "*Transformers Provably Solve Parity Efficiently with Chain of Thought,*" Kim, Suzuki, *ICLR*, 2025.
- "*DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning,*" Guo et al., *Nature*, 2025.
- "*On the Interplay of Pre-Training, Mid-Training, and RL on Reasoning Language Models,*" Zhang, Neubig, Yue, *ICML*, 2026.
- "*Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?*" Yue et al., *NeurIPS*, 2025.
- "*On the Emergence of Implicit Curriculum in RLVR Learning Dynamics,*" Huang et al., *ICML*, 2026.
- "*Agentic Transformers Provably Learn to Search via Reinforcement Learning,*" Yang, Huang, Liang, Chi, *arXiv preprint*, 2026.

Chain-of-Thought Reasoning in Transformers: From Theory to Practice

Part 2

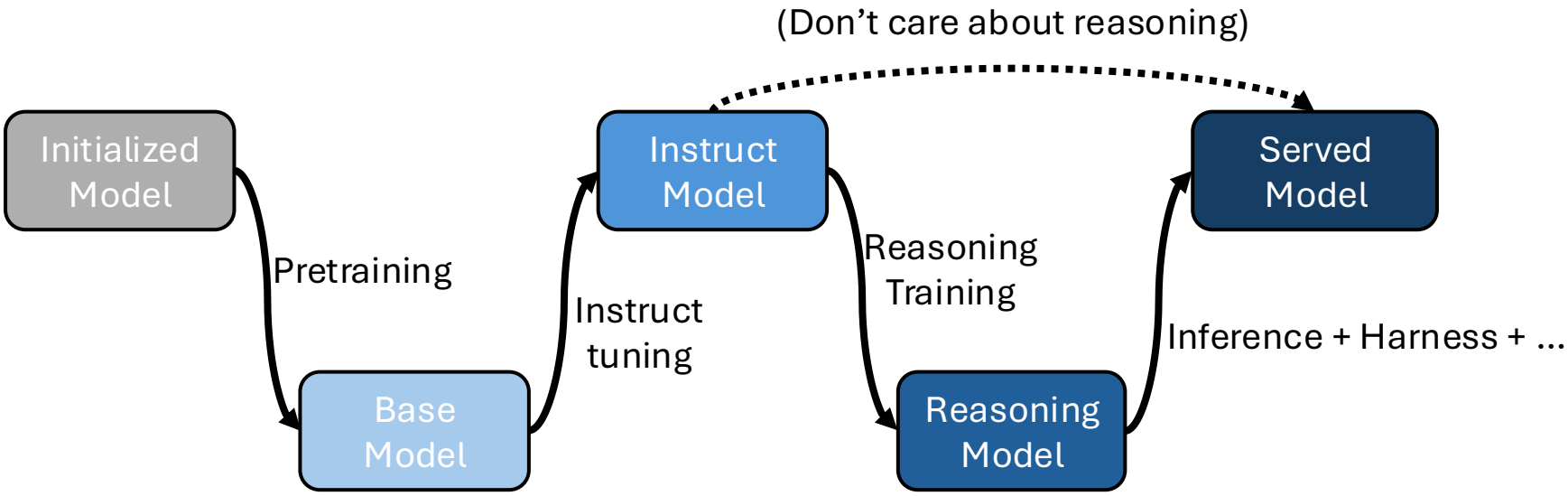
Part 2 goals

In Part 1, we covered the foundations of reasoning in LLMs: expressivity, optimization, and generalization.

In Part 2, we will explore how these methods are used in practice and the holistic considerations/tradeoffs we need to make.

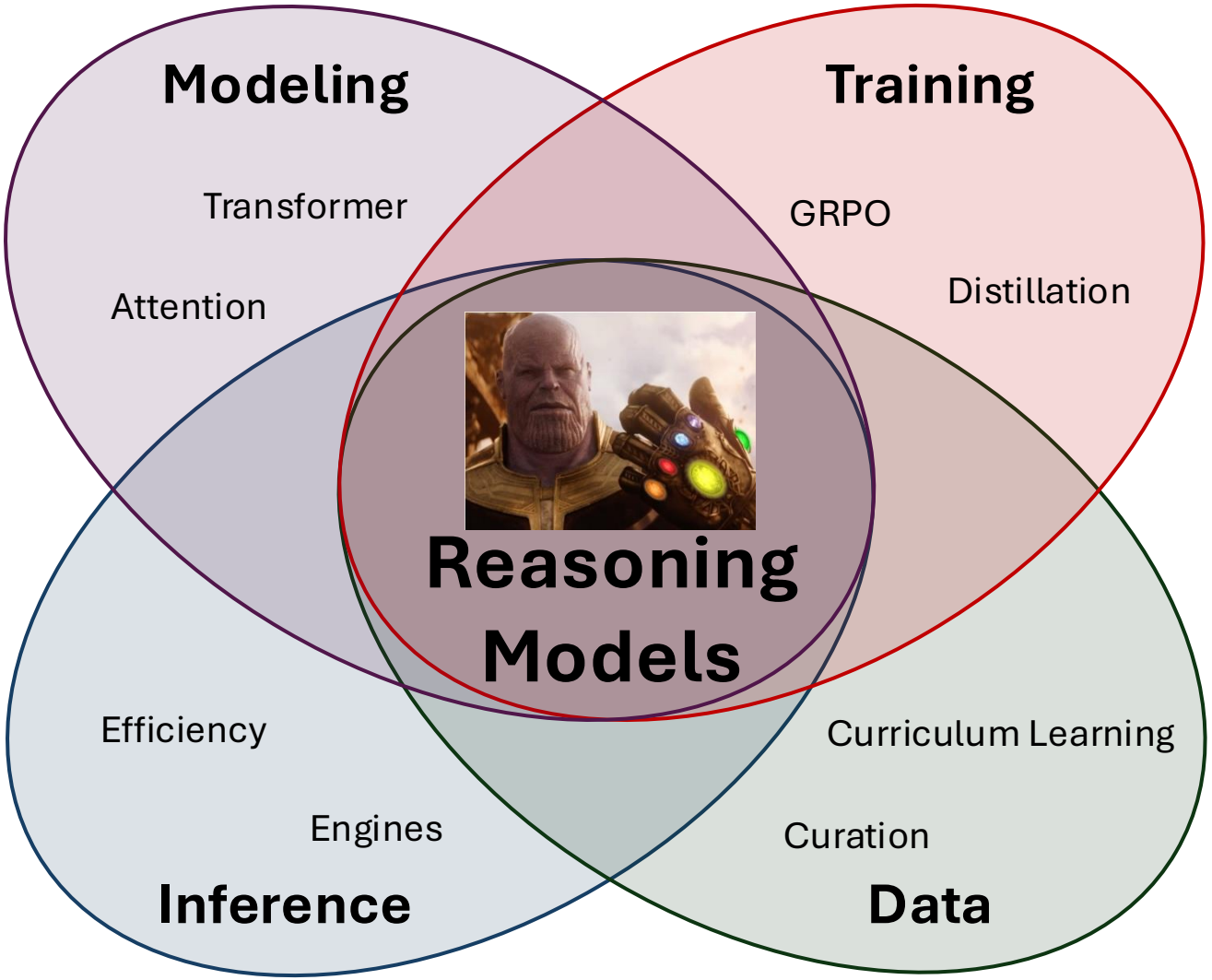
We will see how reasoning methods interact with the entire development and deployment process.

Training pipeline



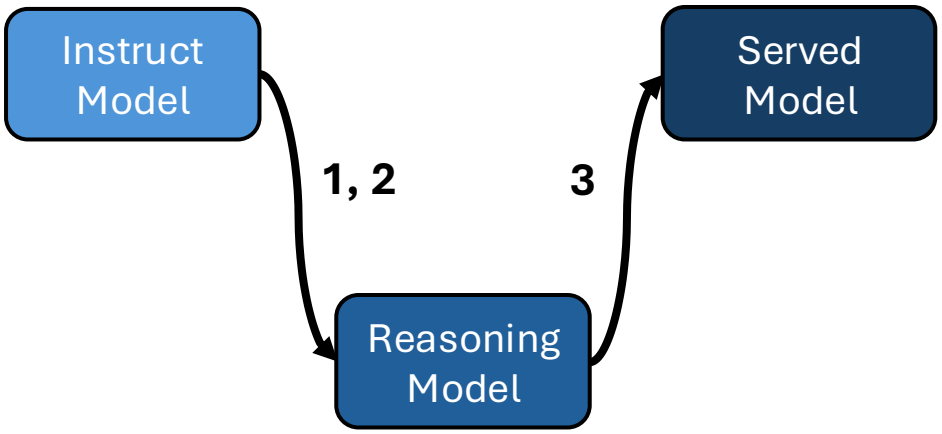
1. **Base Model:** understands general knowledge
2. **Instruct/Chat Model:** follows instructions and structured templates
3. **Reasoning Model:** expresses reasoning behaviors
4. **Served Model:** consumer-facing model

Why is reasoning possible?



Outline

1. Inducing Reasoning Overview
   a. SFT
   b. RLVR
2. Parallel Scaling
3. Inference and Deployment
4. What's Next



Inducing Reasoning

Life before: prompting to reason

A simple nudge for LLMs to use CoT when we can't train or do not have access to model weights.

Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

Few-shot CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

Zero-shot CoT

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

But how do we explicitly train models to naturally reason?

(Kojima et al., 2022)

Main ways to induce LLM reasoning

Recall from Part 1…

Two paradigms to teach LLMs to reason: SFT and RL

Supervised finetuning (SFT)
Next-token prediction over the reasoning trace
dense supervision

RL from verifiable reward (RLVR)
Reward only when the final answer is correct
sparse reward

Both are often used together.

Reasoning structure

Instruct models adopt a chat template, with a special format for reasoning.

System: *(e.g., You are a helpful assistant…)* } May be hidden from the user

User: *(e.g., Solve this problem…)*

Assistant:

<think>

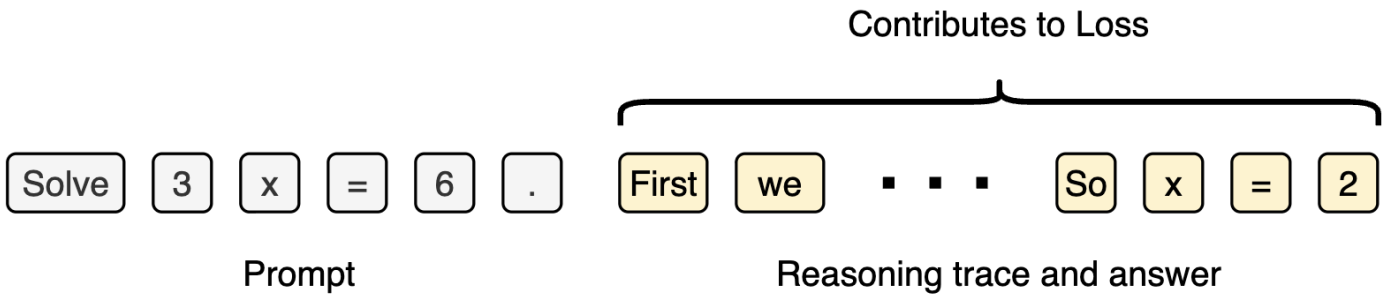
[Model reasoning trace] } May be hidden from the user

</think>

[Summarized CoT and answer]

SFT on reasoning traces

Curate a dataset of <problem> + <reasoning> + <answer> samples and train using SFT.



Reasoning traces are the most expensive part of the dataset to collect, so we must use synthetic data.

SFT distillation

Less powerful models can learn from the reasoning traces of a more powerful one.

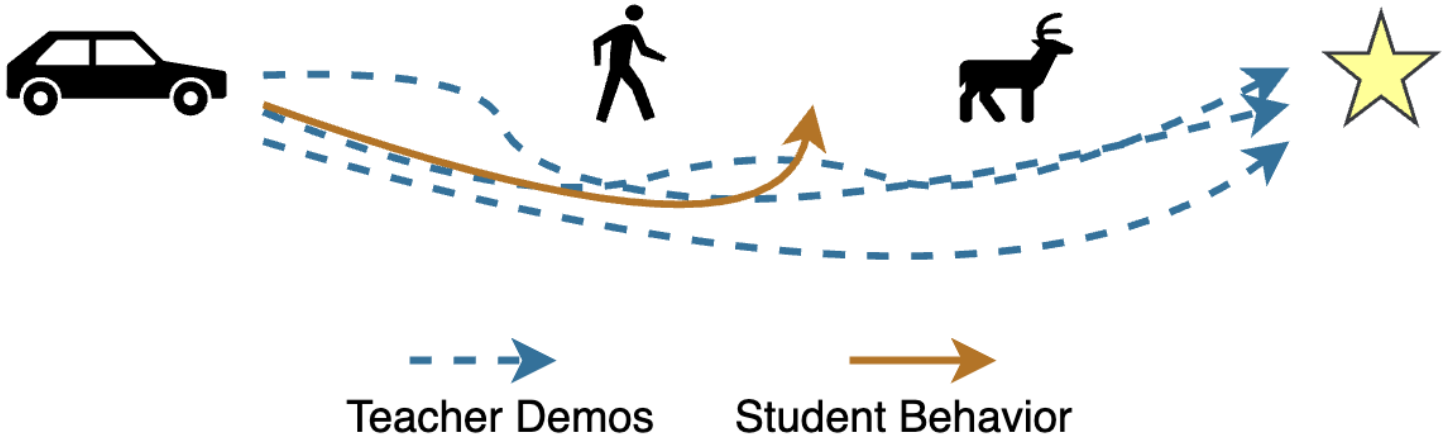
Steps:

1. For each prompt, multiple reasoning traces are sampled from a powerful teacher model.
2. Filter outputs by correctness and other desirable characteristics (e.g., length, format, etc.).
3. SFT

Off-policy learning

SFT from external reasoning traces forces the model to adopt behavior different from itself (off-policy).

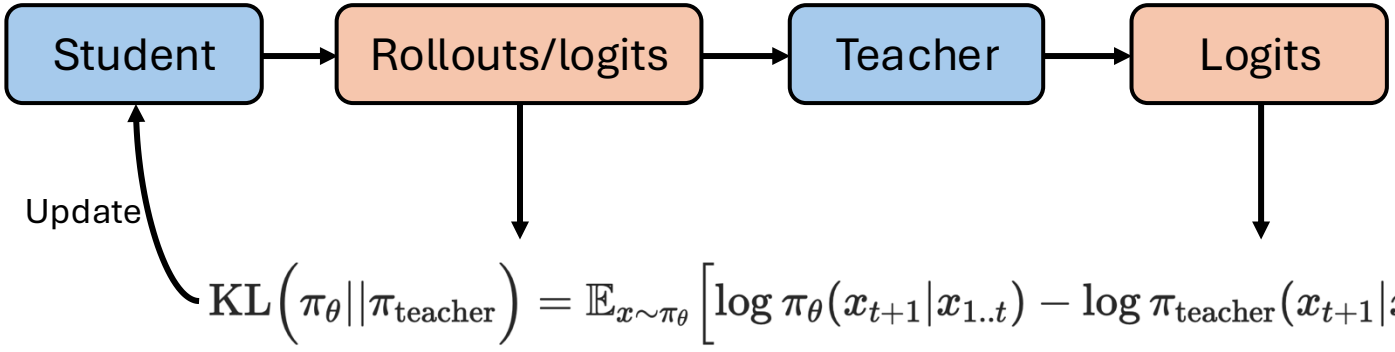
If a student deviates slightly during inference, it can be hard to recover.

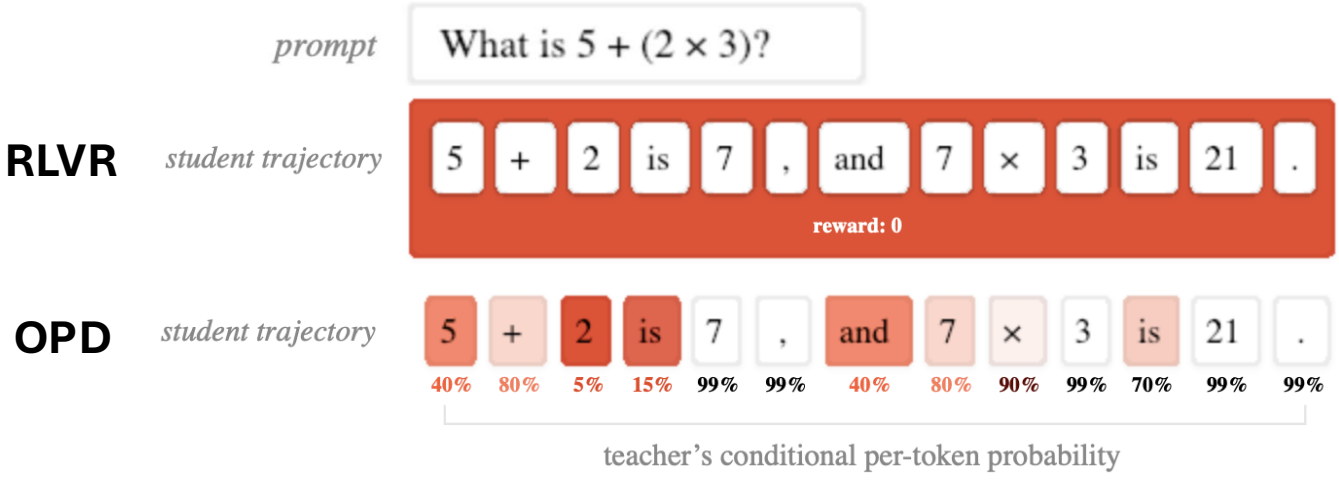


Is there a way to do distillation with on-policy data?

On-policy distillation (OPD)

A powerful teacher model can provide token-level logit feedback on student responses.

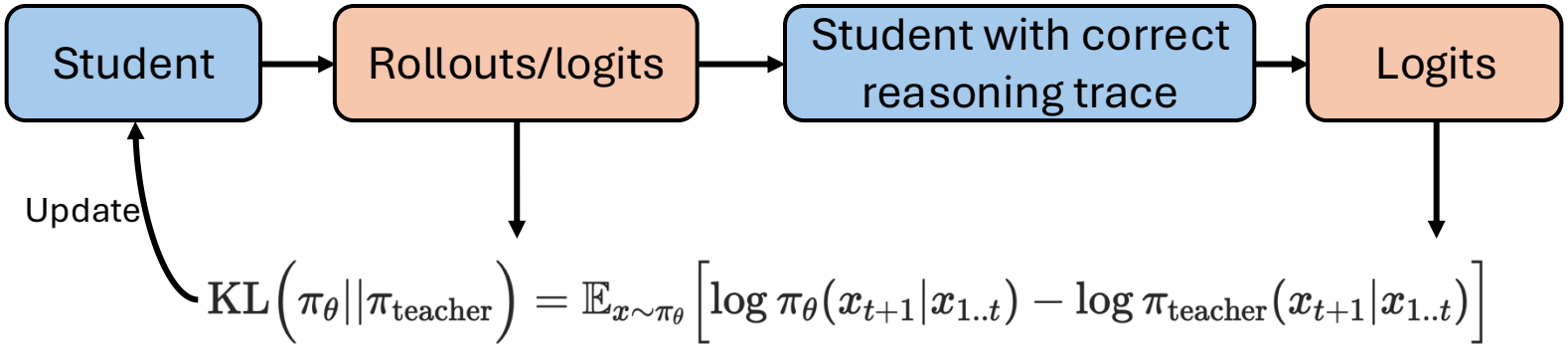




(Lu et al., 2025)

On-policy self-distillation (OPSD)

OPSD uses the student model with additional context (e.g., a correct reasoning trace) as the teacher.



Student Prompt

Problem: Find the derivative of $f(x) = 3x^2 + 2x - 5$ at $x = 2$

Answer:

Teacher Prompt

Problem: Find the derivative of $f(x) = 3x^2 + 2x - 5$ at $x = 2$

Here is a reference solution:
First find $f'(x) = 6x + 2$, then evaluate at $x = 2$: $f'(2) = 6(2) + 2 = 14$

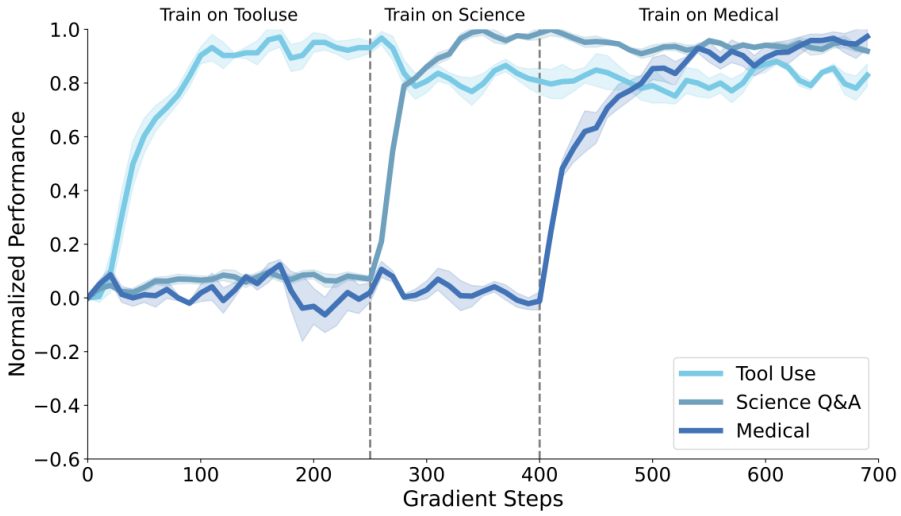
After understanding the reference solution, please try to solve this problem using your own approach below:

Answer:

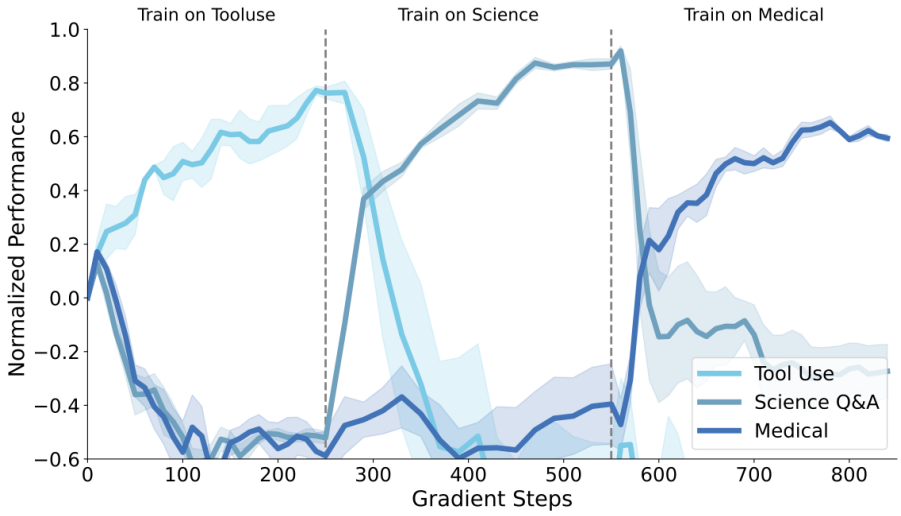
(Zhao et al., 2026)

Generalization

On-policy methods tend to preserve past capabilities.



OPSD



SFT

(Shenfeld et al., 2026)

SFT summary

SFT learns to reason from another model's reasoning ability.

We explored 3 ways to do this:

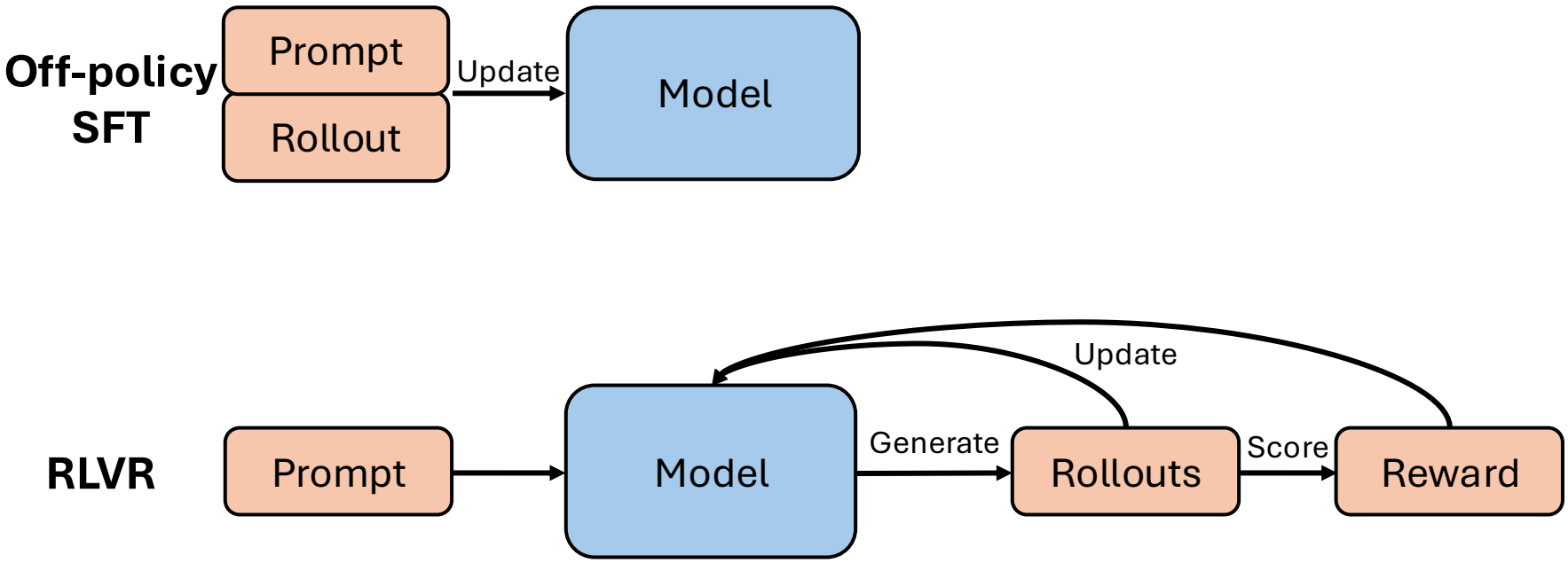
- Direct distillation from reasoning traces
- Distillation from teacher model feedback (OPD)
- Distillation from own feedback (OPSD)

In all cases, you need an external model or source of reasoning traces.

RLVR

Curate a dataset of <problem> + <answer> samples and train using RL—the simplest being RLVR.

- The model learns to generate its own reasoning traces.



Verifiable rewards

RLVR is useful when we can easily and objectively test output quality (i.e., verifiable rewards), bypassing subjective scores and labels.

Setting	Reward Signal
Math	Correctness
Puzzles/Games	Correctness, constraints, score
Code	Compilation, test cases
Agents	Task completion

RL Objective

Many use the group relative policy optimization (GRPO) objective, a variant of PPO.

PPO: For a prompt q, output o, and reward r:

$$\mathcal{J}_{\text{PPO}}(\theta) = \frac{1}{|o|}\sum_{s=1}^{|o|}\{\min[R_s(\theta)A_s, \text{clip}(R_s(\theta), 1-\epsilon, 1+\epsilon)A_s] - \beta D_{\text{KL}}(\pi_\theta||\pi_{\theta_{\text{ref}}})\}$$

Advantage computed with the help of a value function

GRPO: For a prompt q, outputs $o_1, \ldots, o_G$, and rewards $r_1, \ldots, r_G$:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{s=1}^{|o_i|}\left\{\min\left[R_s(\theta)\hat{A}_i, \text{clip}(R_s(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\right] - \beta D_{\text{KL}}(\pi_\theta||\pi_{\theta_{\text{ref}}})\right\}$$

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1,\ldots,r_G)}{\text{std}(r_1,\ldots,r_G)}$$

Estimate of the PPO advantage **without** a value function

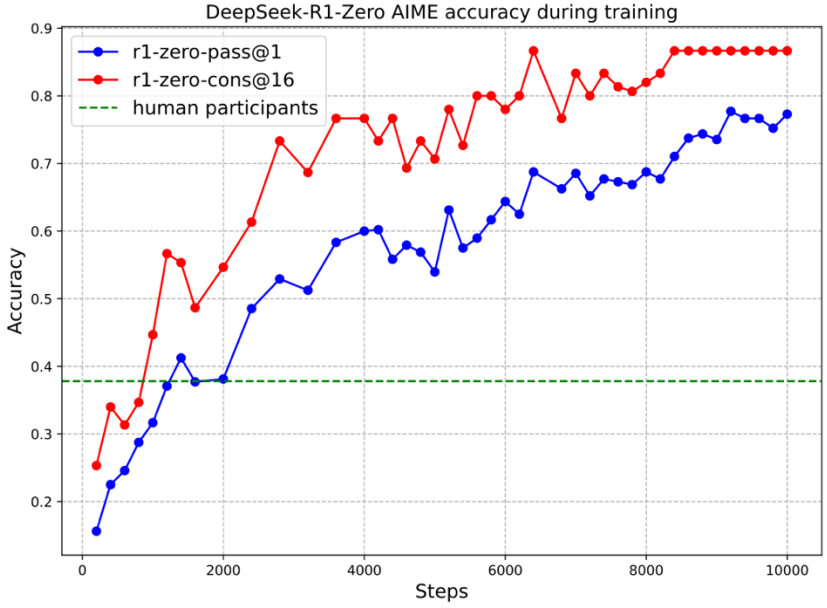
$$R_s(\theta) = \frac{\pi_\theta(o_s|q, o_{1:s-1})}{\pi_{\theta_{\text{old}}}(o_s|q, o_{1:s-1})}$$

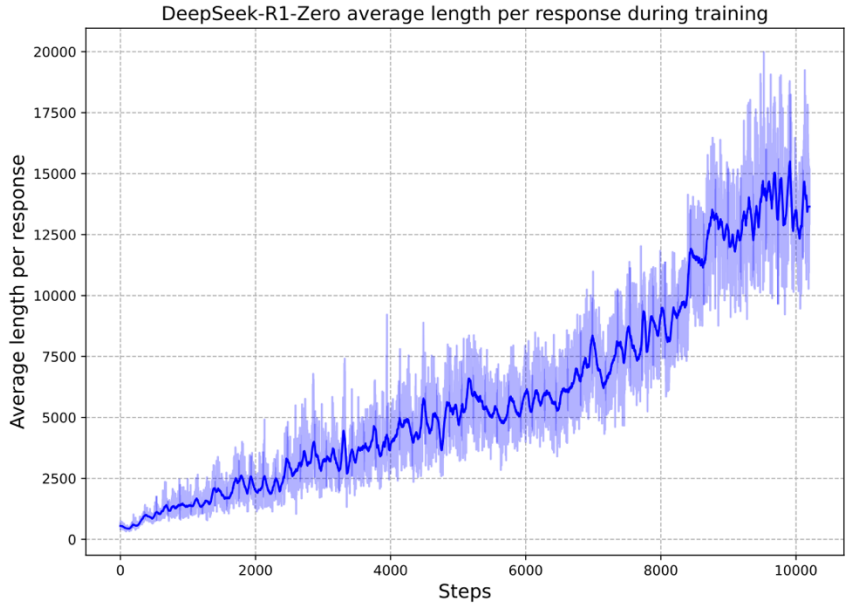
$$D_{\text{KL}}(\pi_\theta||\pi_{\theta_{\text{ref}}}) = \frac{\pi_{\theta_{\text{ref}}}(o_s|q, o_{1:s-1})}{\pi_\theta(o_s|q, o_{1:s-1})} - \log\frac{\pi_{\theta_{\text{ref}}}(o_s|q, o_{1:s-1})}{\pi_\theta(o_s|q, o_{1:s-1})} - 1$$

(Shao et al., 2024)

GRPO results

DeepSeek R1 was trained with GRPO with strong results.





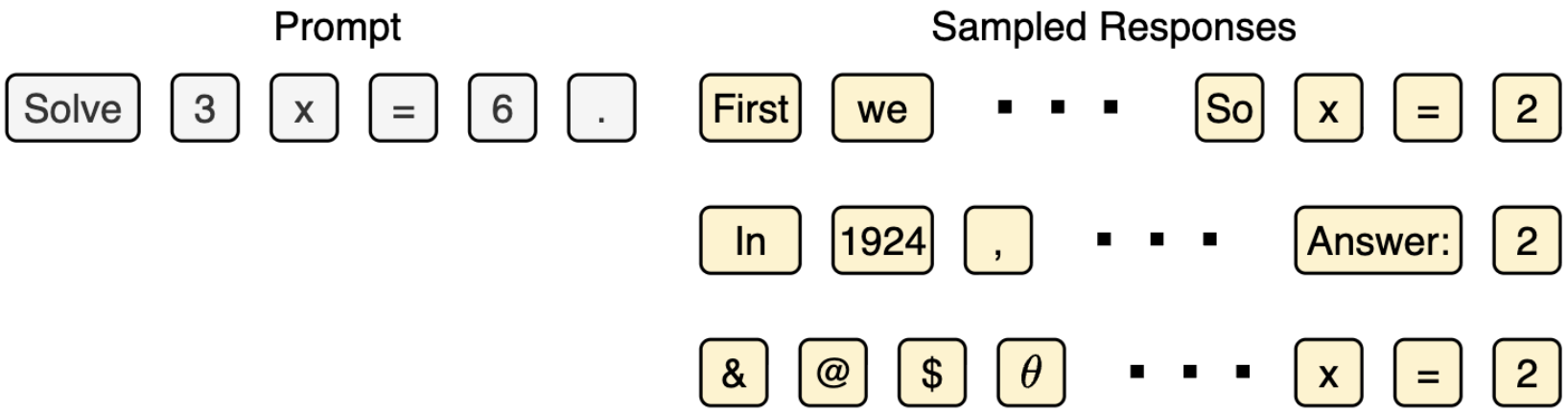
Since then, many variants of GRPO have emerged.

(Guo et al., 2025)

Combining SFT and RLVR

RLVR can also lead to uninterpretable behavior even if the performance is good.

In the eyes of RLVR, these have the same reward:



Often, SFT is applied first to ground the model before applying RL.

Advantage and knowledge boundary

Recall the GRPO advantage normalizes the reward within a group of responses to a single prompt:

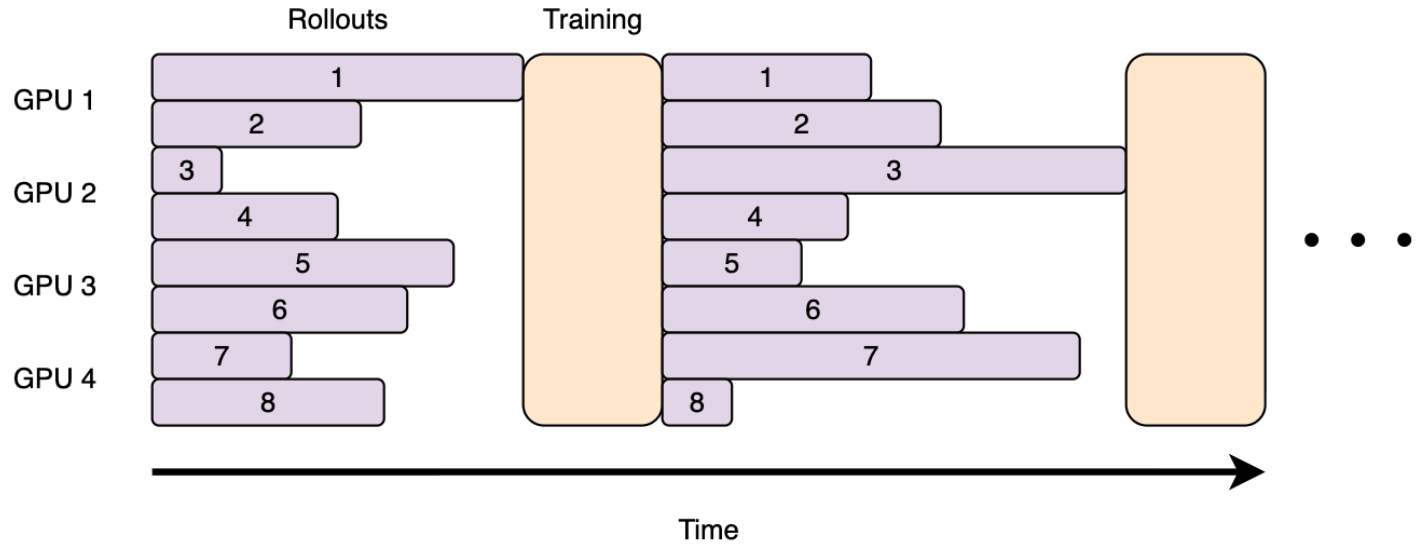
$$\hat{A}_i = \frac{r_i - \text{mean}(r_1,\ldots,r_G)}{\text{std}(r_1,\ldots,r_G)}$$

RLVR reward is often binary (0/1 for incorrect/correct)

- If all responses are correct or incorrect, nothing is learned!

RLVR is costly

At regular intervals during RLVR, we need to generate on-policy reasoning traces.



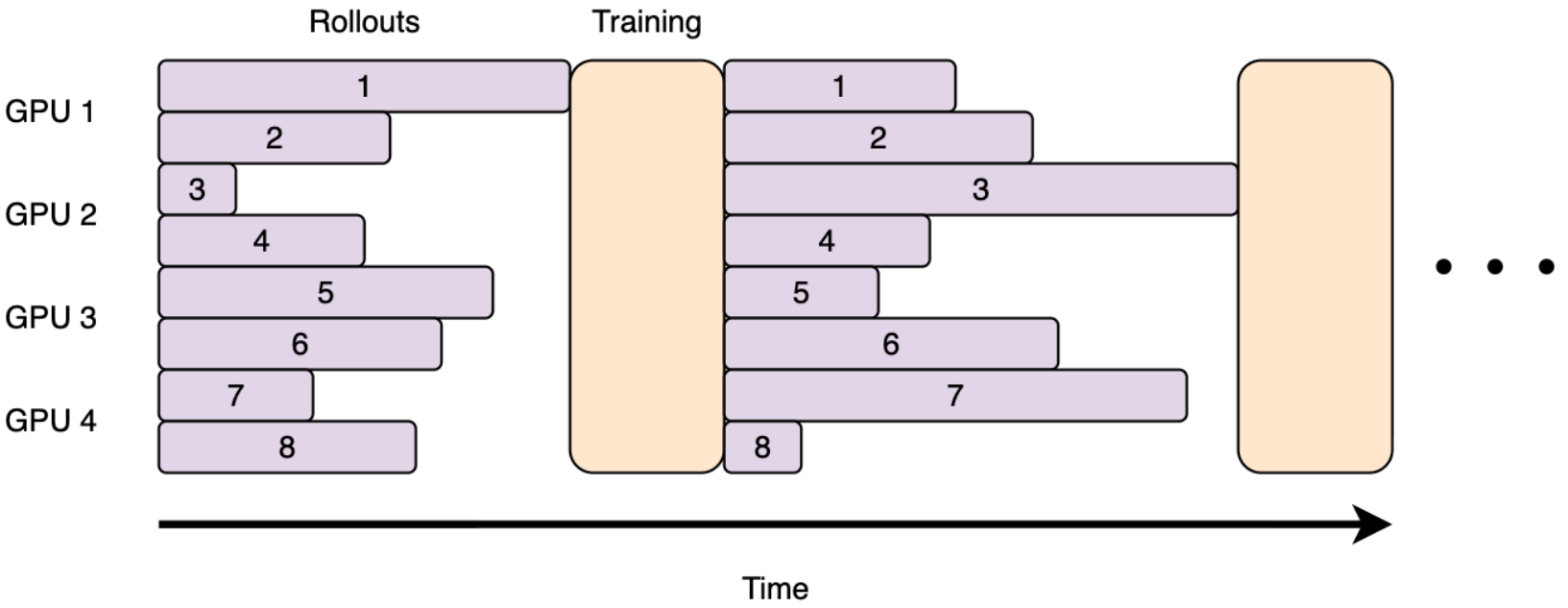
- Generative inference is autoregressive and reasoning traces can be on the order of 1K-10K tokens or more.
  - In contrast, direct SFT distillation processes tokens in parallel.
  - This ends up taking much longer than forward/backward passes!

RLVR Scalability

RLVR computational cost scales by many things:

- Model size/architecture
  - Can't modify this much
- Data
  - *Rollout length:* more tokens = more thinking but more time/memory
  - *Verification speed:* simple or complex scoring
  - *Learnability: prompts could be too easy/hard to learn from*
  - *Off policy steps*: more steps = more efficient but risks instability

Rollout length



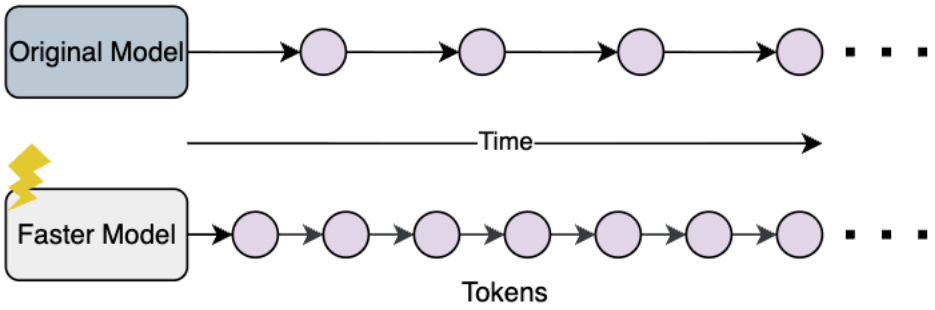
Rollouts are the biggest contributor to RLVR training latency, often taking up >50% of the time.

- Autoregressive LLM generation is sequential
- Prefill processes all tokens in parallel
- Over-truncation can lead to performance drops

Rollout length: acceleration

We can try to accelerate the rollout process.

- Algorithmic approximations (e.g., quantization, speculative decoding)
- Systems optimizations (e.g., vLLM)

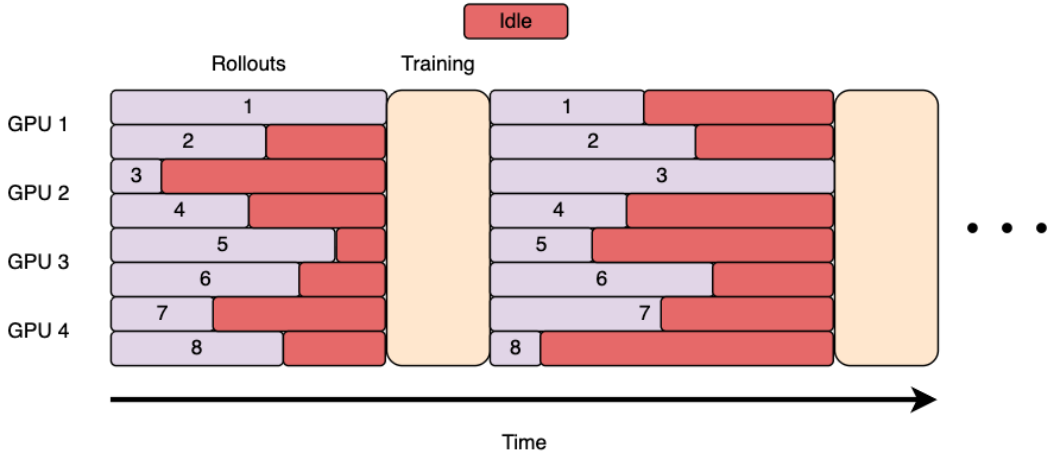


Risks of straying off-policy as training progresses and/or sequence length grows.

Rollout length: scheduling

Large differences in rollout lengths cause GPUs to be underultized.

- Phase synchronization means other GPUs wait for the slowest to finish.

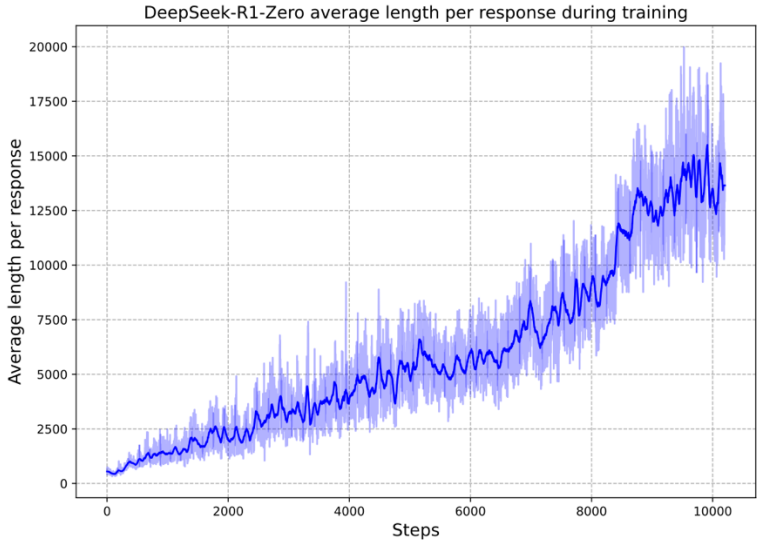


- But you also need all rollouts in a group to finish to calculate the advantage!

Several exisiting scheduling algorithms for RLVR try to minimize idle time.

Rollout length: growth

Models naturally use more CoT tokens as RLVR progresses, so average time between each gradient step and inference cost grow.



Are we using each token efficiently?

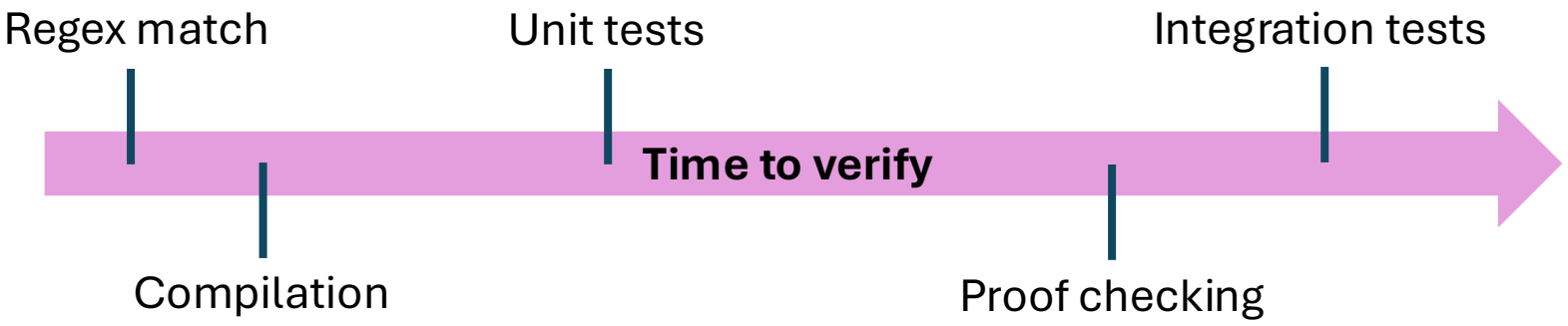
- Loss aggregation methods (e.g., DAPO and Dr. GRPO)
- Length regularization

(Guo et al., 2025)
(Yu et al., 2025)
(Liu et al., 2025)

Verification

Different domains have different methods of verification and scoring.

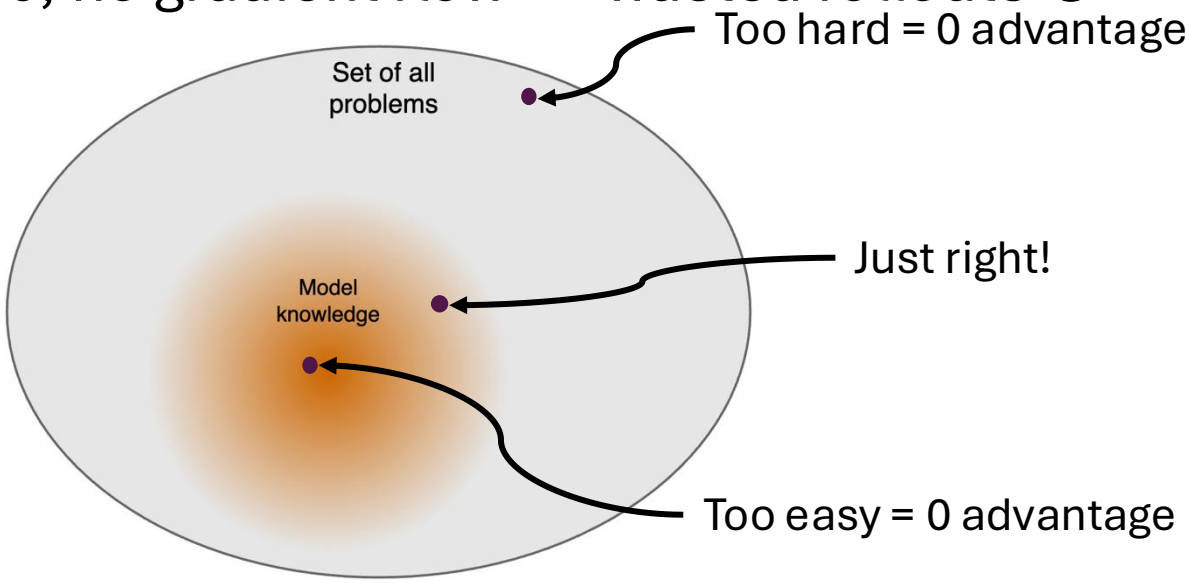
- Need to balance thoroughness and time.



Learnable samples

For one prompt, we need to sample multiple responses to calculate the advantage.

- If advantage is 0, no gradient flow => wasted rollouts ☹

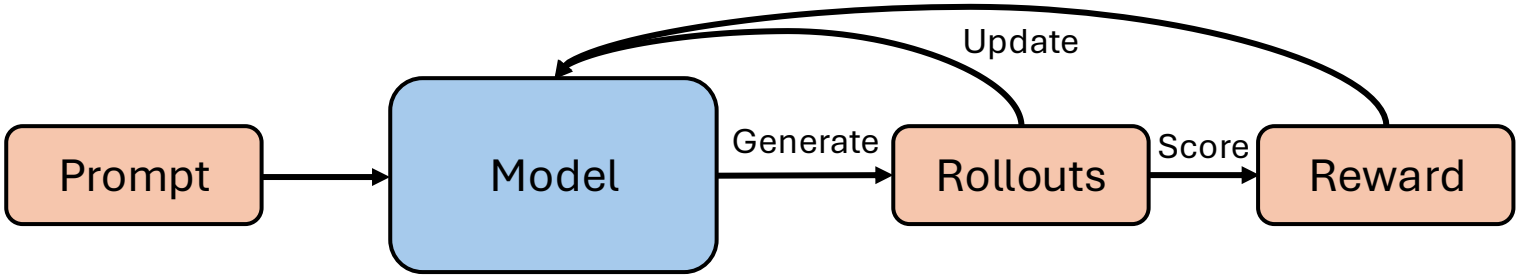


Adjustments

- Increase group size
- Denser rewards to create more diversity in rewards
- Curriculum learning from easy to hard prompts

Gradient steps between rollouts

A benefit of RL is training with on-policy data.



Rollouts are off-policy once the model is updated, but as we saw earlier, generating rollouts is expensive.

More off-policy updates speed up training but risk instability and collapse.

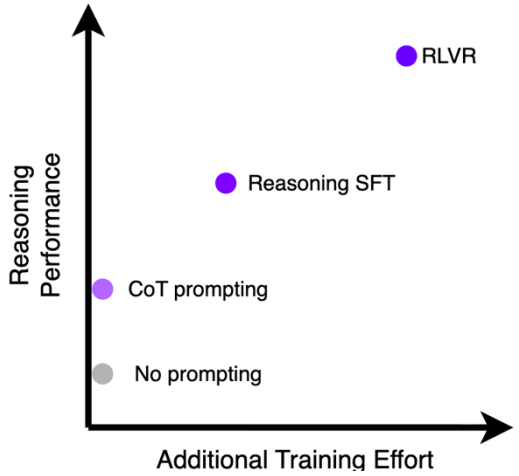
RLVR summary

RLVR is a valuable tool for training models to reason via exploration, but it comes with unique algorithmic and infrastructural challenges.

- *Rollout length:* strains memory, latency, and throughput
- *Verification:* balance quality with efficiency
- *Learnability:* too easy or too hard problems have no training signal
- *On/off-policy:* balance stability with efficiency

Section summary

We explored different ways to get LLMs to reason.



CoT Prompting

- Training-free
- Limited by pretrained abilities

SFT

- Train on strong model outputs
- Needs richer datasets and limited improvement

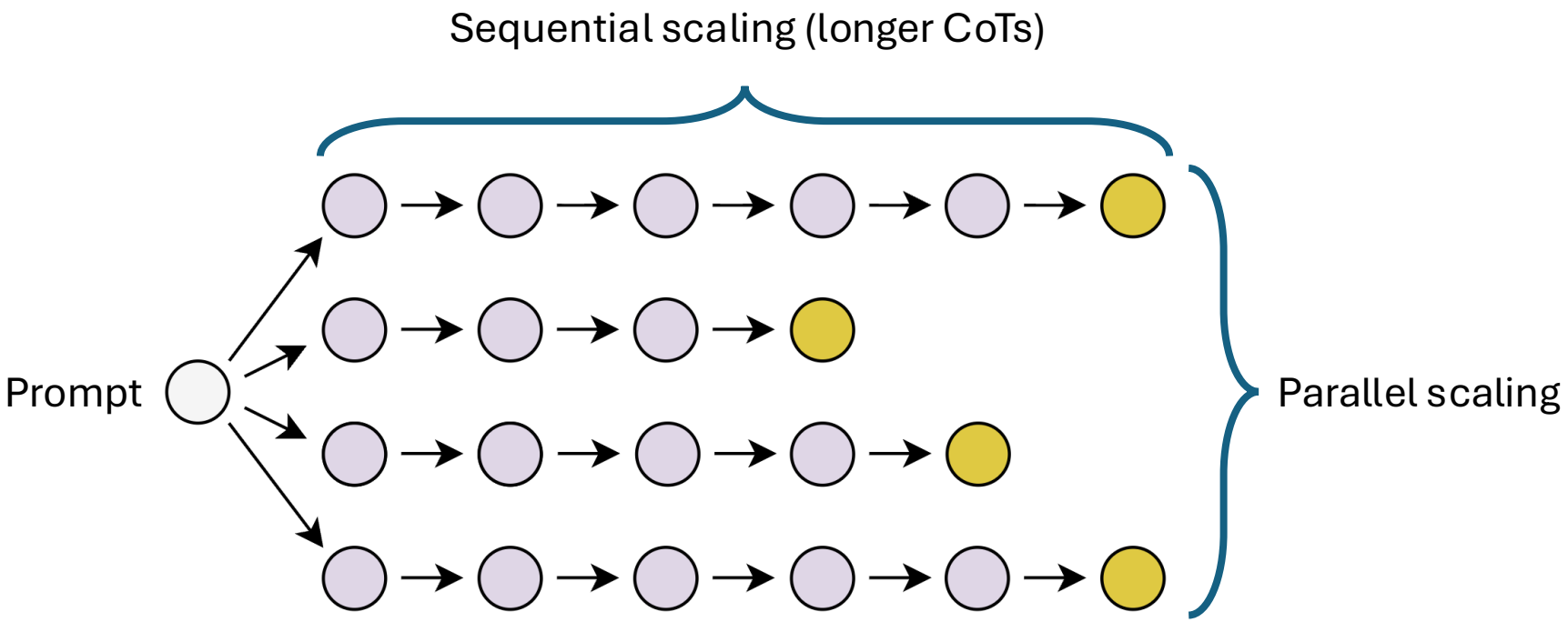
RLVR

- Model greatly enhances reasoning with exploration
- Computationally demanding

Parallel Scaling

Inference scaling axes

We've talked a lot about CoTs, but it is only one axis of test-time scaling.



Pass@k metrics

Multiple ways to evaluate a set of responses to a prompt.

Pass@k (i.e., coverage) is the probability that at least one of the *k* responses is correct.

- This measures whether a problem is within a model's capabilities.
- Special case: Pass@1 is accuracy

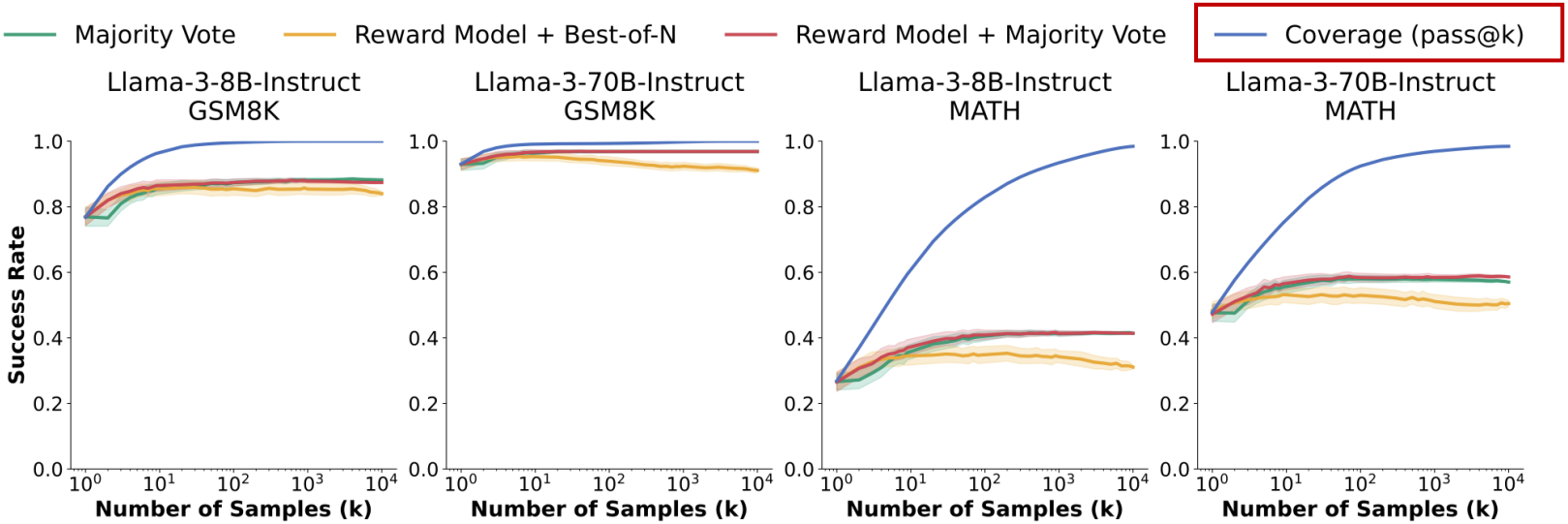
Prompt	Responses
1	✗ ✓ ✓ ✓
2	✗ ✗ ✓ ✗
3	✓ ✓ ✓ ✓
4	✗ ✗ ✗ ✗

Pass@1: 50%

Pass@4: 75%

Independent sampling

Simplest parallel scaling mechanism is to let the LLM make multiple independent attempts to a prompt.

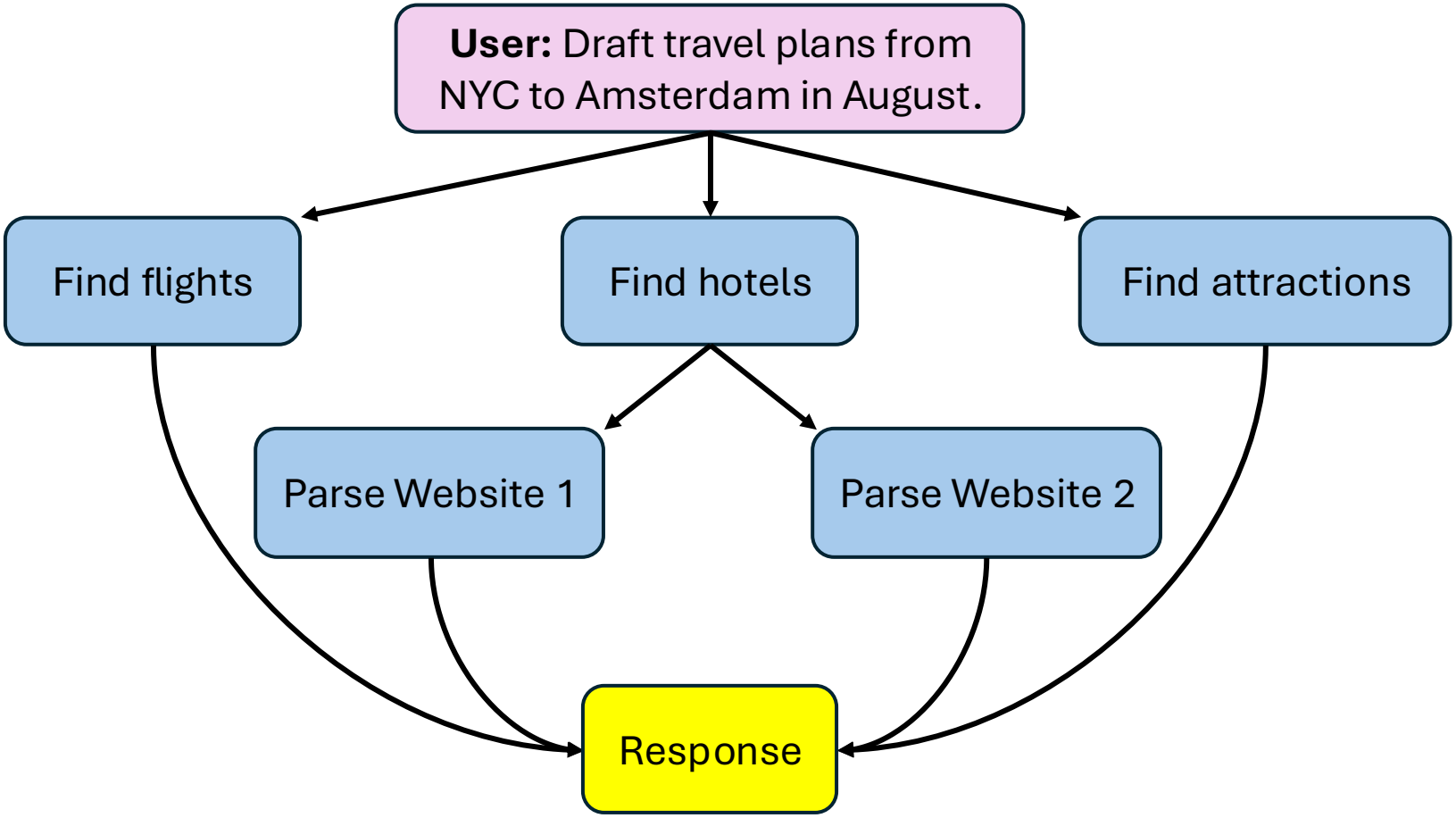


The probability of a correct answer being generated can be close to 1 with more samples.

(Brown et al., 2024)

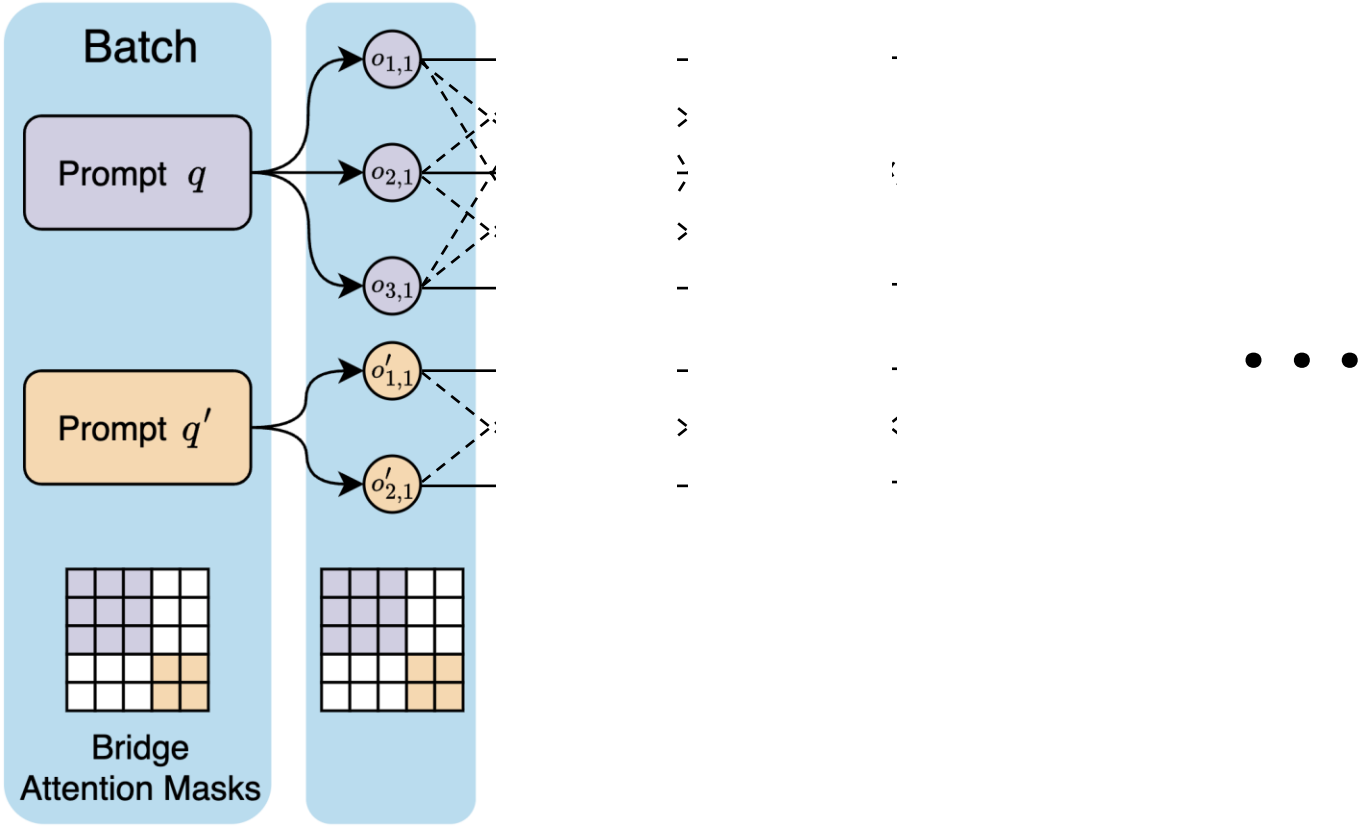
Decomposition

Teach a model to break down a task into parallel subtasks and merge their outputs.



Something in between: interdependence

We can also share latent information between rollouts at regular generation intervals, so useful information does not need to be generated from scratch every time.



(Dong et al., 2026b)

Interdependence on math

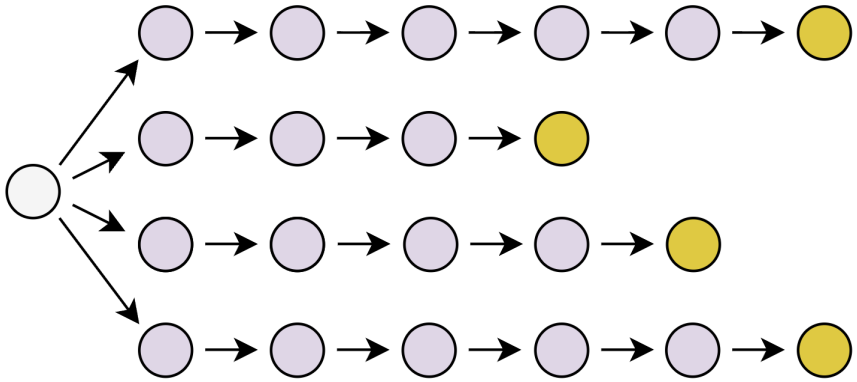
Interdependent sampling treats a batch of reasoning rollouts as a continuous unit for increased information flow and performance.

Model	MATH	AIME24	AIME25	AMC	BRU	CMI	HMMT	Avg	↑ Δ
DS-Qwen-1.5B	73.65	13.75	13.44	50.00	18.12	4.30	8.23	25.93	0.00
RLVR only	78.75	17.40	18.44	60.55	18.54	3.83	7.50	29.29	3.36
P-Match	78.65	18.12	19.17	**60.62**	20.94	5.08	8.54	30.16	4.23
Bridge	**81.30**	**20.11**	**20.00**	60.55	**21.36**	**5.63**	**9.79**	**31.25**	**5.32**
DS-Qwen-7B	82.15	23.44	21.88	66.02	23.75	5.63	11.98	33.55	0.00
RLVR only	**88.15**	29.06	23.85	74.30	28.33	7.97	**12.60**	37.75	4.20
P-Match	86.80	28.85	**25.73**	70.47	26.77	6.25	11.87	36.68	3.13
Bridge	**88.15**	**32.19**	25.41	**77.65**	**30.21**	**9.77**	12.40	**39.40**	**5.85**
DS-Llama-8B	73.40	15.42	13.12	57.97	15.62	2.73	8.23	26.64	0.00
RLVR only	76.70	18.12	18.12	63.44	15.83	5.47	10.52	29.74	3.10
P-Match	78.00	22.29	**20.21**	61.80	17.81	5.08	11.67	30.98	4.34
Bridge	**80.15**	**24.76**	18.18	**66.36**	**19.91**	**6.02**	**11.93**	**32.47**	**5.83**

(Dong et al., 2026b)

Getting one output

For parallel methods that generate multiple responses, there are several ways to produce one final response.

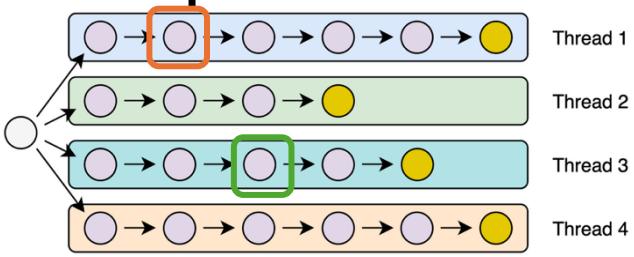


- *Self-consistency:* Majority voting.
  - Only useful if we can assign a discrete label to outputs.
- *Best-of-N:* Reward model extracts the best response.
  - Dependent on reward model performance.
- *Merge:* A model synthesizes all responses into one.
  - Concatenation can result in very long sequences.

Parallel scaling summary

Parallel scaling offers another axis to improve reasoning.

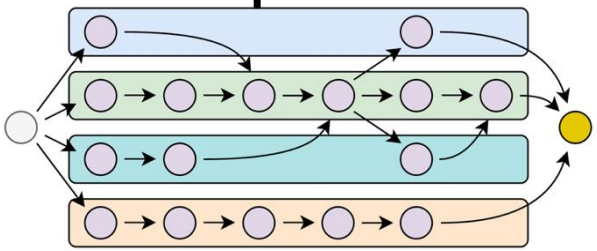
Independence



Info derived here is inaccessible there

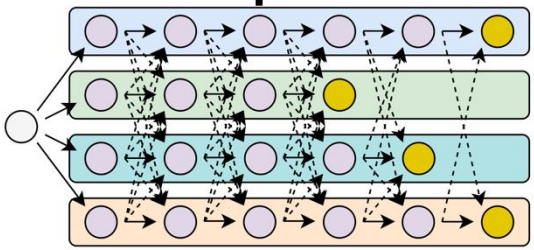
- Simple but no information flow across threads

Decomposition



- Information flow between threads

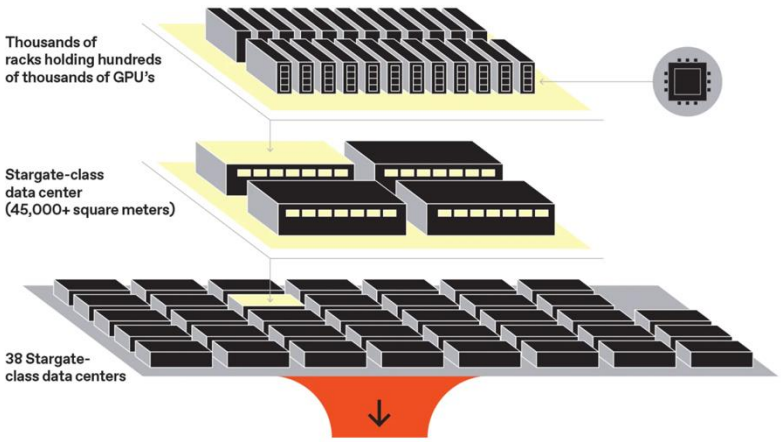
Interdependence

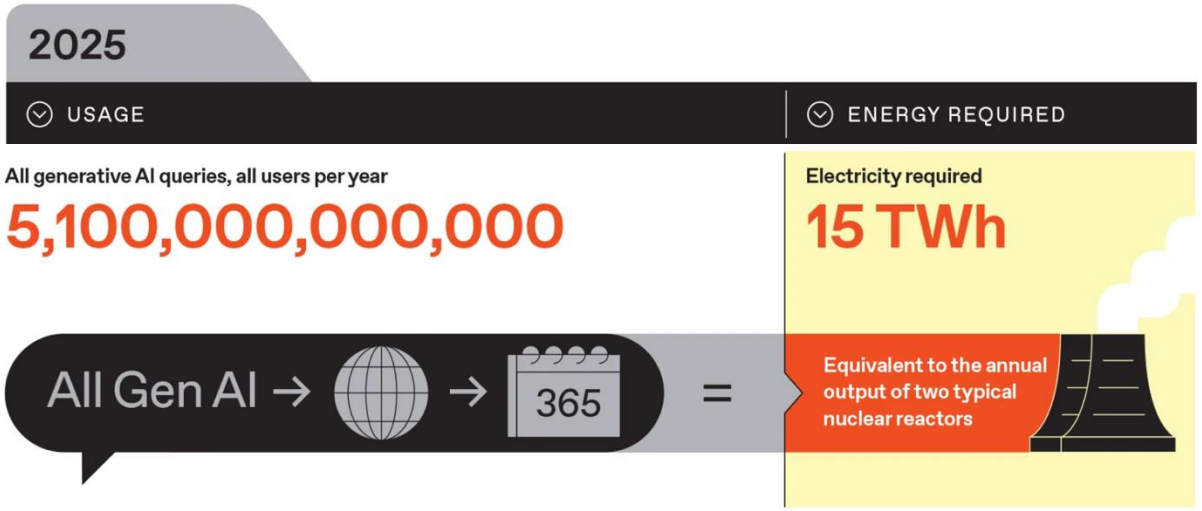


- Information flow between threads

Inference and Deployment

The case for efficient inference





Inference challenges

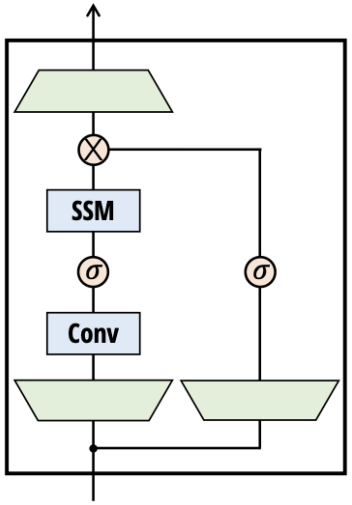
Inference is inefficient for many reasons.

- Sequential generation of tokens.
  - Leads to increased latency for longer generations.
- High memory costs from key-value (KV) caching.
  - Memory for KV caching can easily exceed the model size with long sequences and large batch sizes.
- Memory bound
  - Intermediate features and operations are dwarfed by model weights and KV cache.

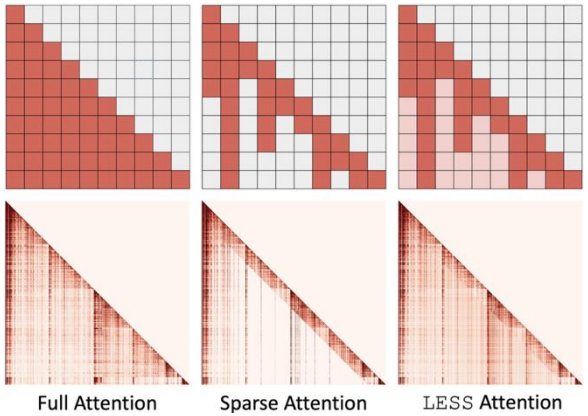
Efficient inference

Many relevant LLM generation efficiency works before reasoning models:

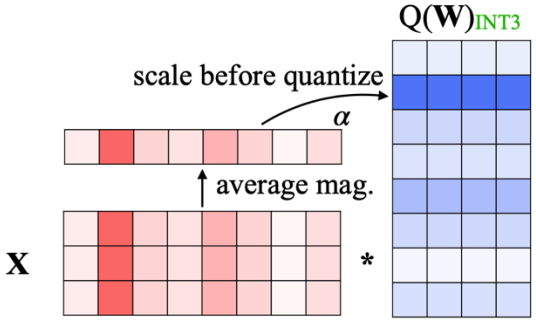
- Architecture: model adjustments for scalable behavior.
- Key-value (KV) caching: dynamic memory reduction for long sequences and high throughput.
- Pruning/quantization: model compression for low-memory/latency.



(Gu and Dao, 2024)



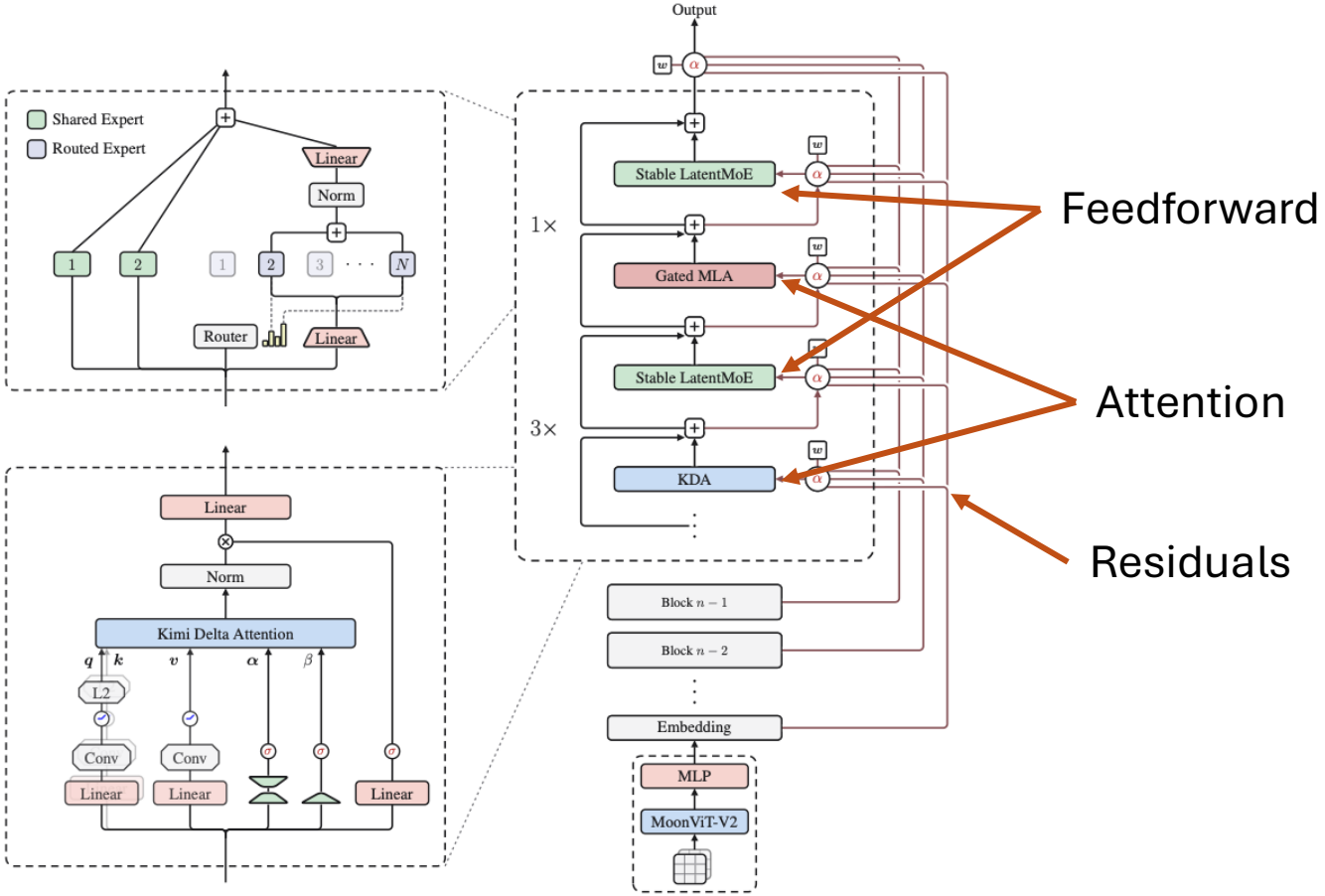
(Dong et al., 2024a)



(Lin et al., 2023)

Architecture

Architectural changes address efficiency at the source, but require retraining and often new training infrastructure.

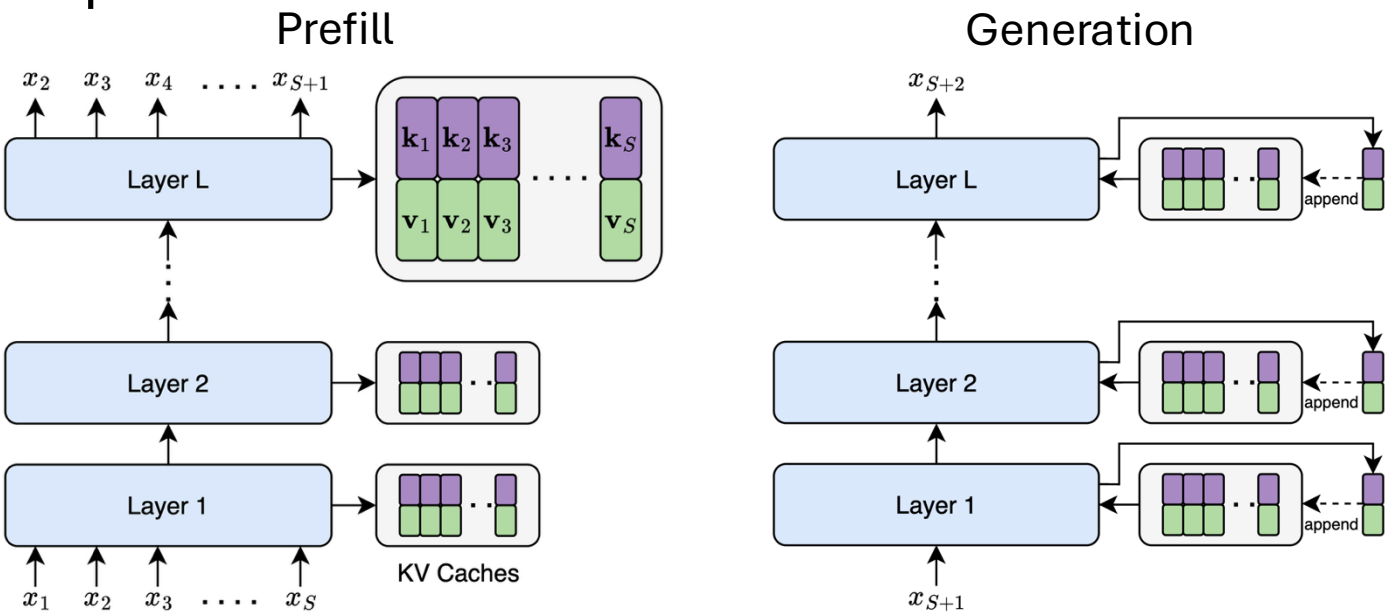


Recent models deviate significantly from the original transformer.

(Kimi Team et al., 2026)

KV caching

The KV cache stores intermediate attention features to accelerate inference at the cost of growing memory consumption.



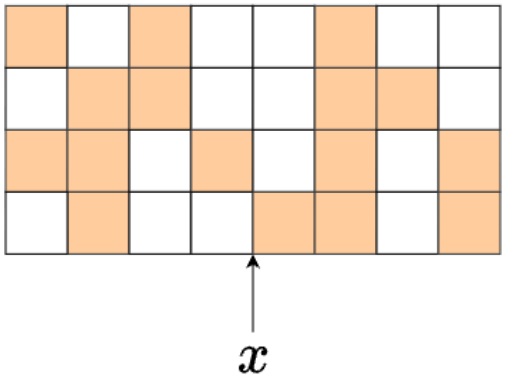
Long sequences make KV caches prohibitively large.

- Many KV caching algorithms have been developed before reasoning models

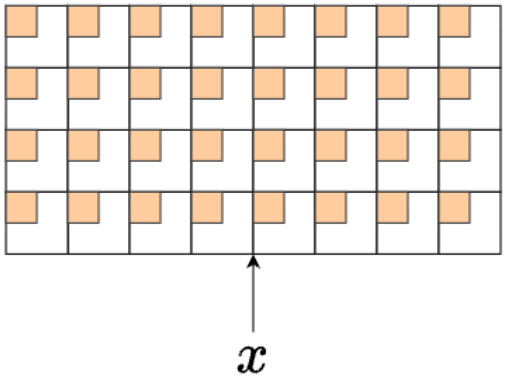
Model compression

The model itself comes with a large memory cost.

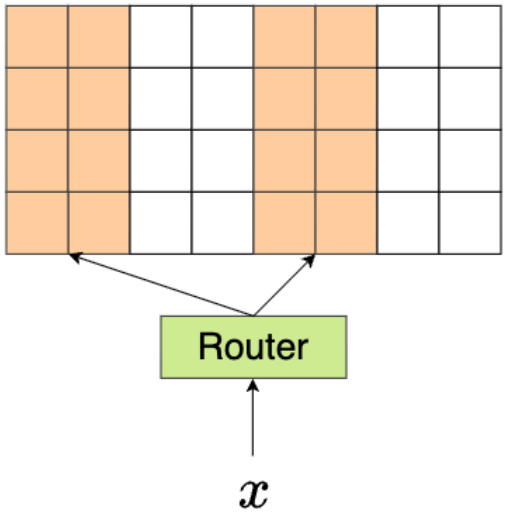
Pruning



Quantization

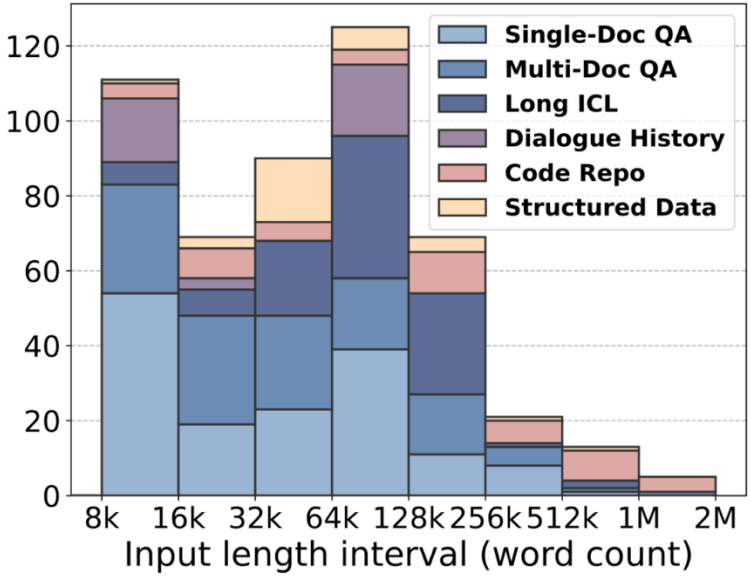


Mixture of Experts



Inference for reasoning

LLMs are thinking longer, tasks are getting more complex…



Reasoning LLMs strain inference bottlenecks due to long generations.

(Bai et al., 2025)

Reasoning is brittle

While performant on language tasks, many of inference methods break down when carried over to CoT/reasoning due to error accumulation!

Reasoning-aware inference acceleration can be effective.

	Math		Language			
Model	GSM8K	MATH	CoQA	QASPER	XSum	CNN/DailyMail
Llama 3.1 8B Instruct	53.98	13.94	63.88	15.16	21.97	25.98
GRIFFIN	20.47	6.16	63.37	12.63	21.53	25.89
Layer-wise Caprese	37.60	9.72	65.05	14.21	21.92	26.31
E2E Caprese	**51.40**	**12.64**	65.50	15.35	22.05	26.42
CATS	50.95	11.80	58.85	12.26	21.43	25.67
Layer-wise Caprese	51.93	13.66	63.92	14.34	22.58	25.79
E2E Caprese	**58.00**	**13.88**	64.27	14.42	22.08	26.13

Adaptive pruning methods

Low-rank reasoning recovery

(Dong et al., 2024b)
(Lee et al., 2024)
(Dong et al., 2026a)

Reasoning is *very* brittle

Mismatches in inference outputs can even occur organically from mathematically equivalent setups.

- Ex: numerical precision, GPU counts, associativity axiom

Standard Deviation of Accuracy with Greedy Decoding

	AIME'24			MATH500			LiveCodeBench-Easy		
	BF16	FP16	FP32	BF16	FP16	FP32	BF16	FP16	FP32
DeepSeek-R1-Distill-Qwen-7B	9.15%	5.74%	0	1.04%	1.12%	0.12%	1.67%	1.28%	0.37%
DeepSeek-R1-Distill-Llama-8B	4.60%	6.00%	5.8e-17	1.59%	0.73%	0.23%	2.31%	1.92%	0.29%
Qwen2.5-7B-Instruct	1.71%	1.45e-17	1.45e-17	0.83%	0.36%	1.16e-16	0.79%	0.48%	1.16e-16
Llama-3.1-8B-Instruct	1.92%	1.30%	0	0.94%	0.34%	0.13%	1.00%	0.67%	0.25%

High variance even with "deterministic" generation!

(Yuan et al., 2025)

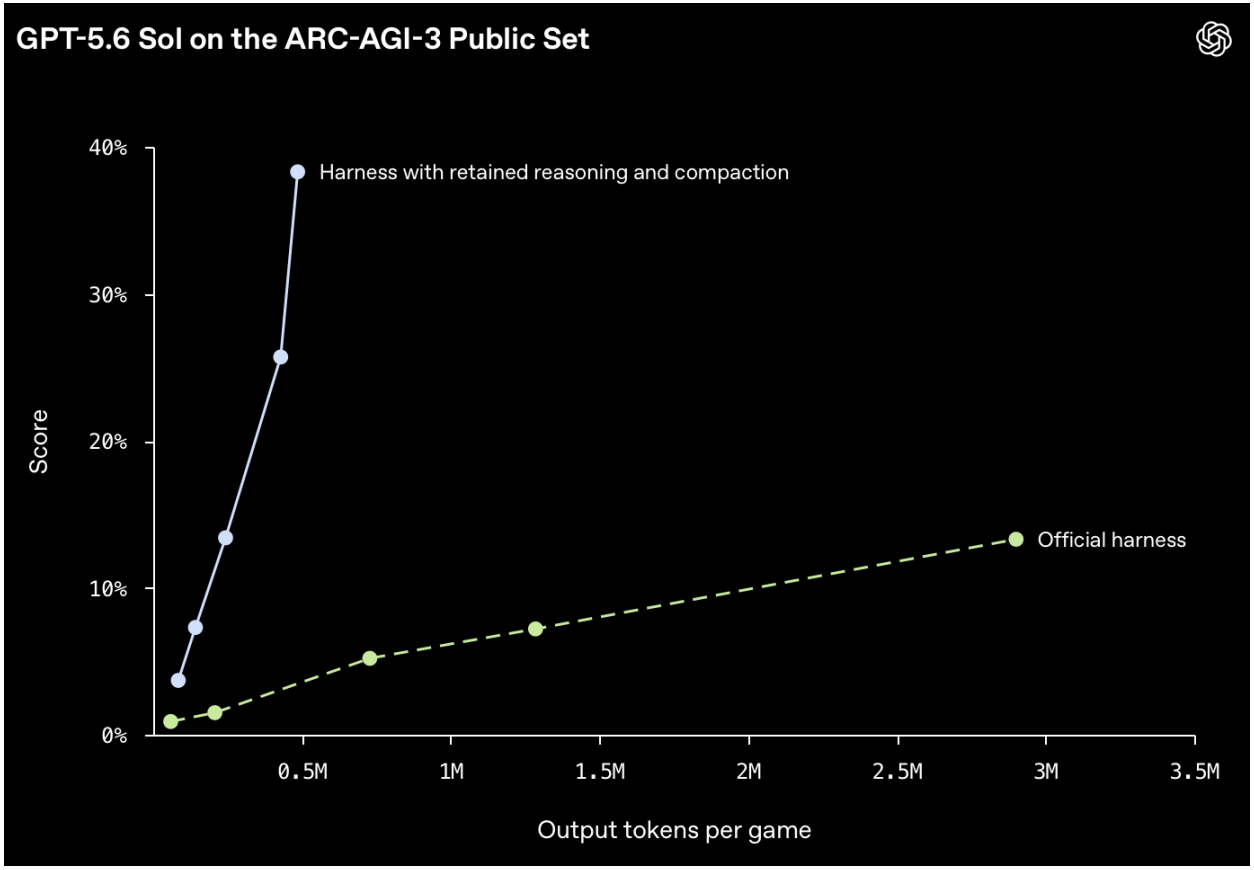
Harness

At deployment, you are likely not directly interacting with the raw LLM.

The harness is the infrastructure that chooses what to show, store, and retrieve information for the model without training the model.

- Formatting
- Retrieval-augmented generation (RAG)
- Edit/addition/deletion protocol
- Compaction

Benefits of harness



The harness alone can dramatically affect performance!

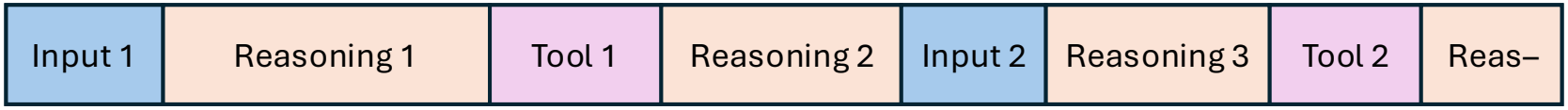
(Bigio and Sanders, 2026)

Context management

Reasoning models synthesize and generate many tokens.

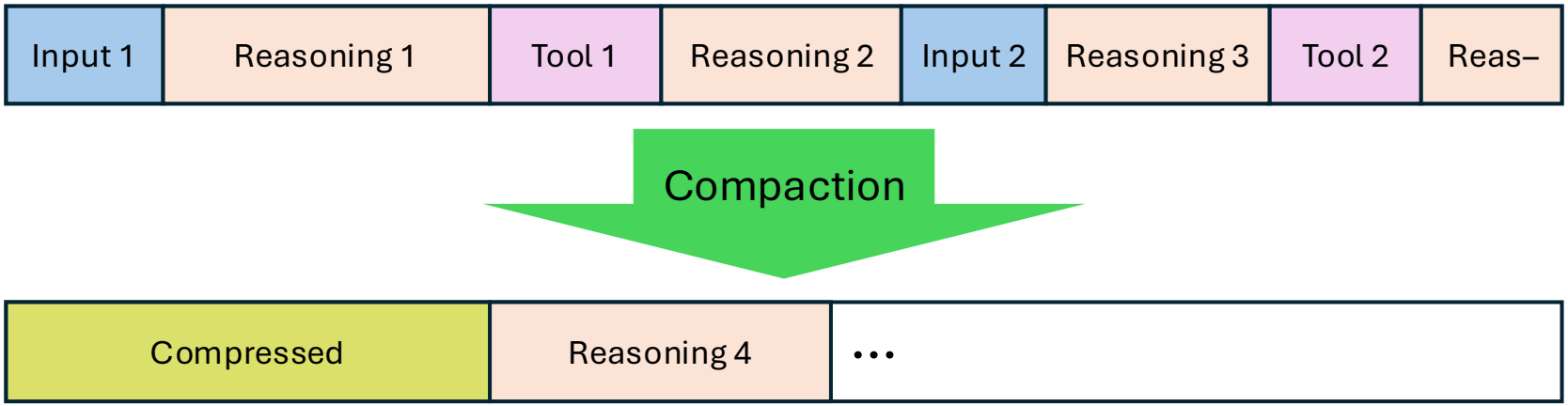
For many reasoning benchmarks:

But in realistic multi-turn settings:



Training longer context models only provides temporary relief and is not sustainable.

Compaction



Summarization: a model summarizes the history into fewer tokens.

Window: keep only the most recent history.

Offload: store the chunks of the history into fetch-able artifacts.

Inference and deployment summary

A reasoning model needs to easy to use!

Inference

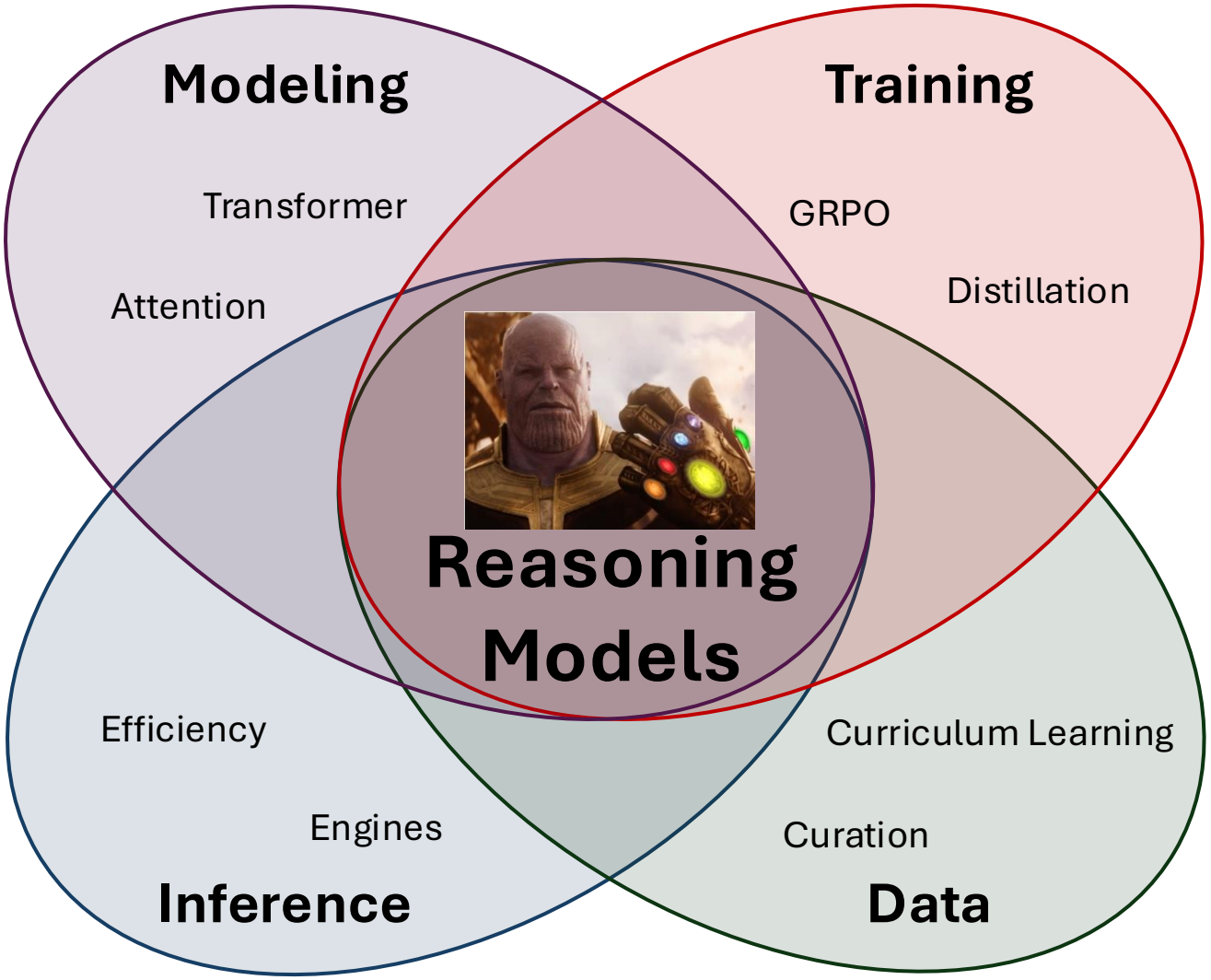
- Long generations of reasoning models accentuate existing inefficiencies of inference.
- Reasoning aware methods can help.

Deployment

- Harness engineering can play a big role in efficiency and performance.

What's Next

Why is reasoning possible?

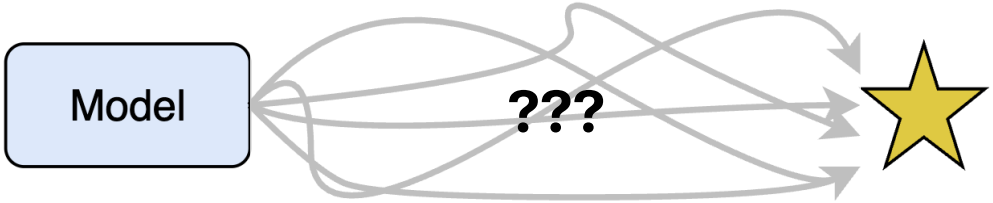


Progress along any of these directions directly helps LLMs reason better.

Actually evaluating reasoning

RLVR and many benchmarks primarily care about the final answer.

- Possible for the model to stumble upon the right answer despite errors or undesirable behavior.



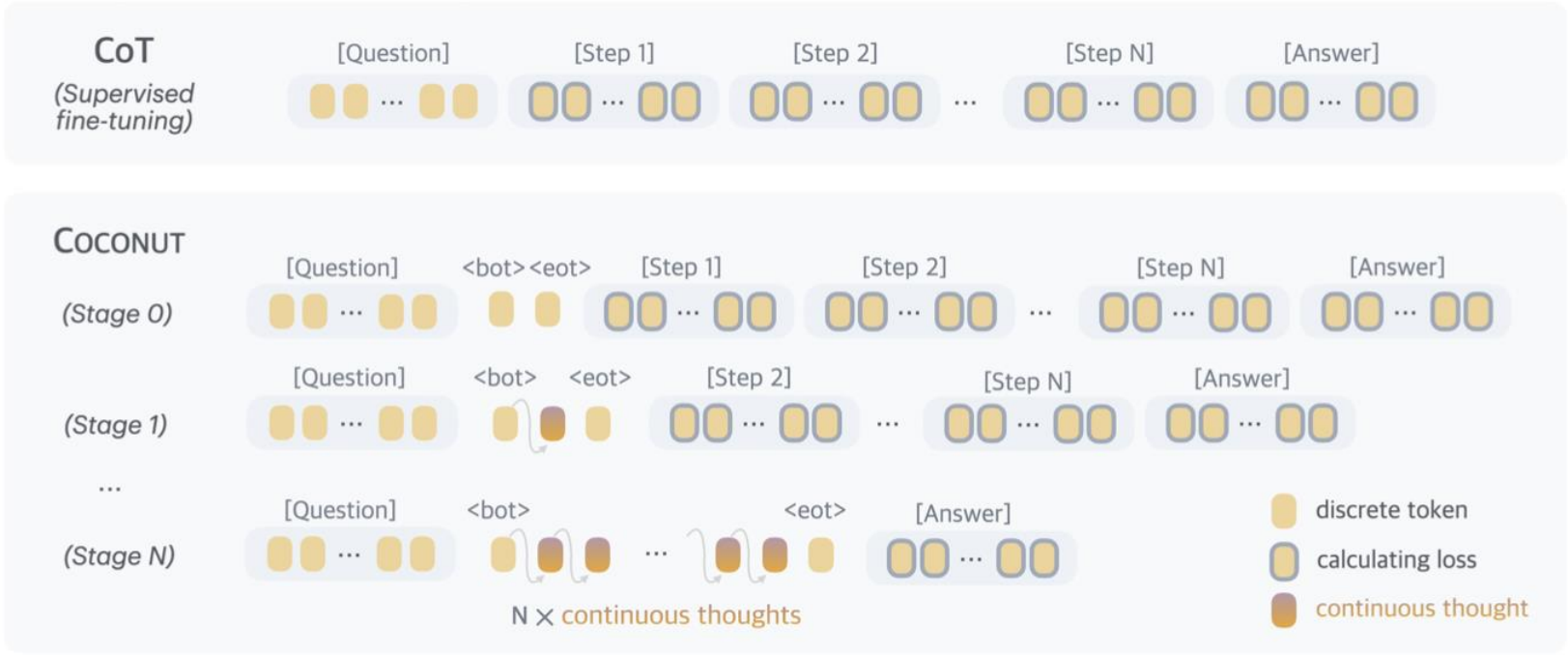
How can we systematically and reliably check reasoning quality beyond the final answer?

How can we evaluate the safety of these trajectories?

Latent reasoning

Reasoning currently takes the form of CoT discrete tokens.

- Good for human interpretability but is it artificially constraining for models?



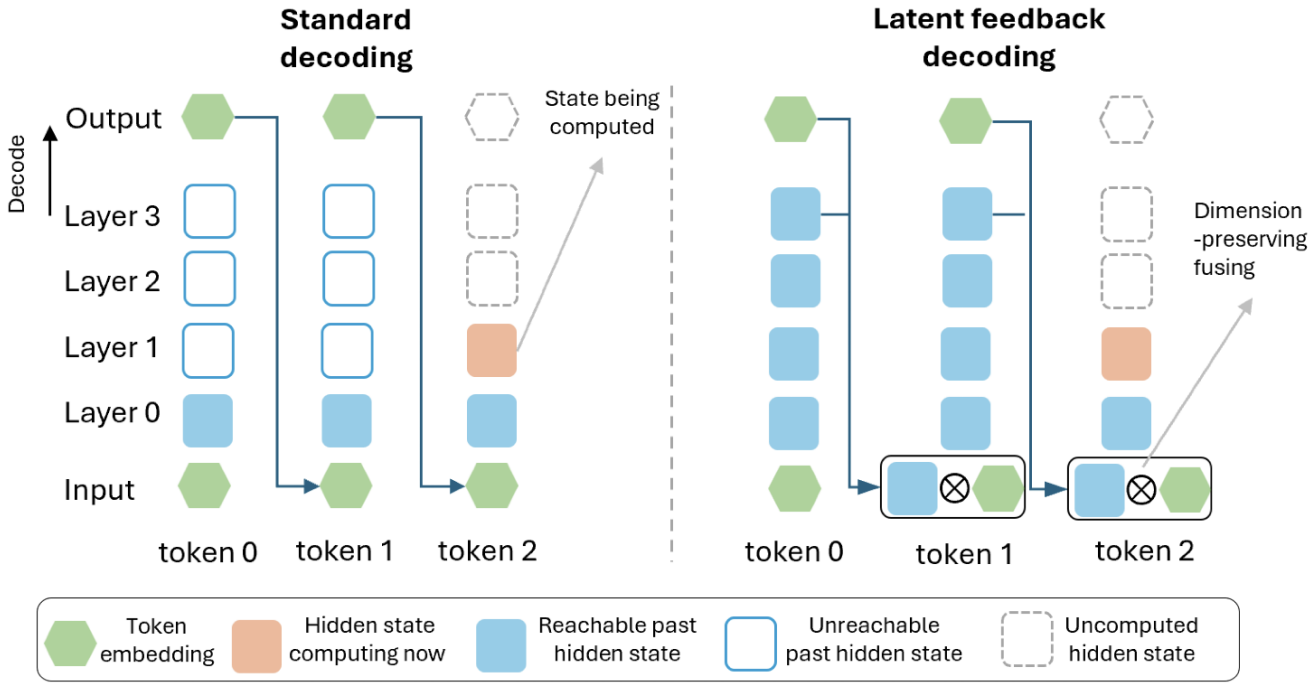
(Hao et al., 2025)

Information flow

We saw earlier that interdependent parallel sampling improves information flow across rollouts.

The transformer architecture also has inefficiencies in information utilization.

- Looped transformers and full-bandwidth transformers.



(Wang et al., 2026)

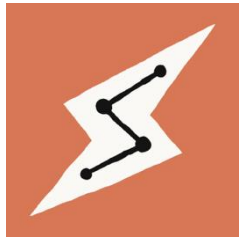
Application: agents

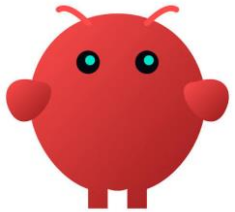
Let the model reason and *perform actions*.

- Calculator: no more mental math
- Internet access: look up esoteric/recent information
- Code environment: simplify redundant tasks

How can we…

- Integrate multimodal inputs
- Train orchestrators
- Create multi-agent collaboration/negotiation





Application: AI for science

So much progress in language, math, and coding…

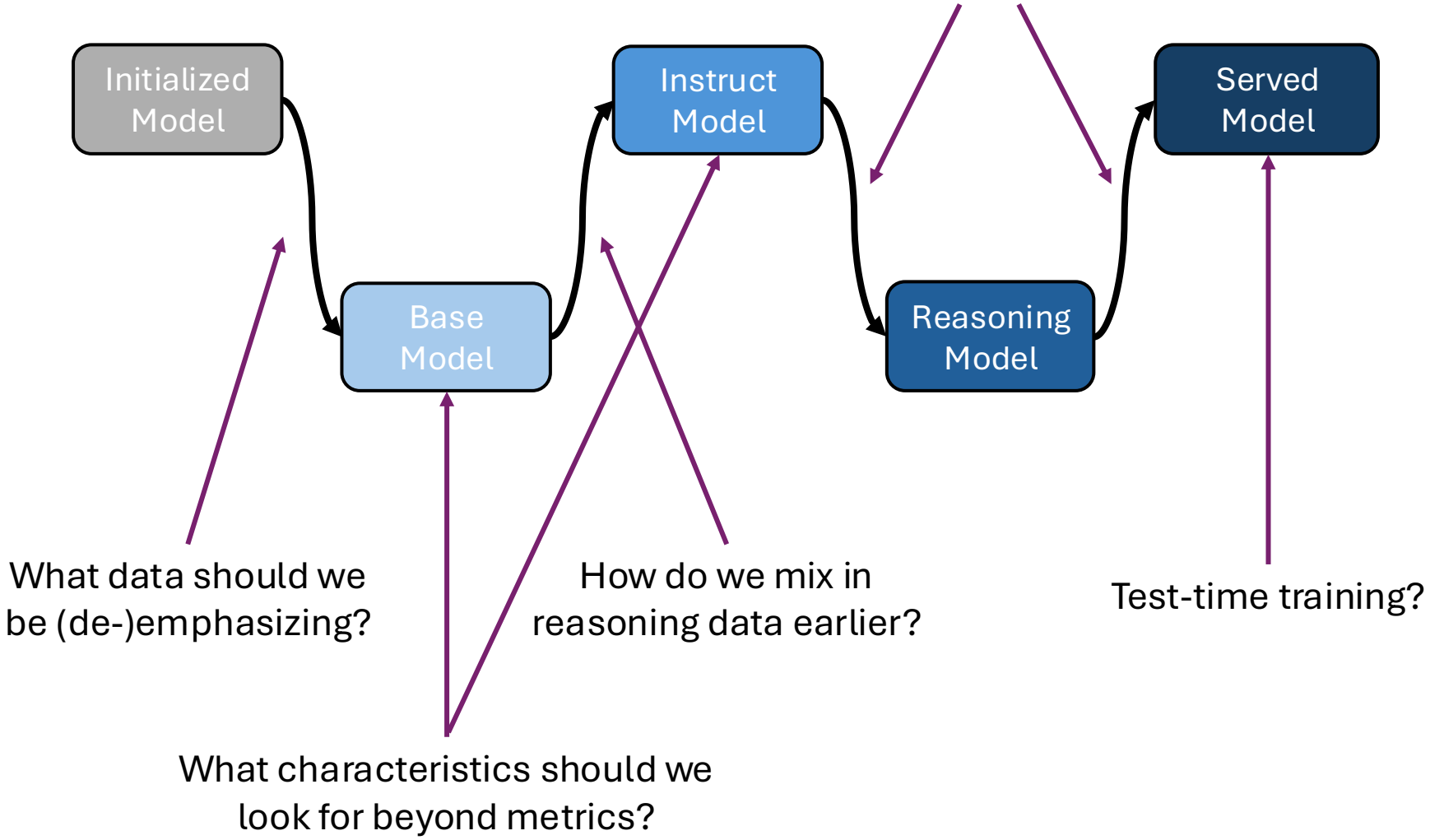
…perhaps it's time to think about scientific domains?

But scientific domains pose unique challenges

- Data scarcity
- Hard to verify
- Scalability
- Domain expertise and constraints

Training pipeline revisited

We have only touched upon a sliver of LLM development.



Summary

We explored the mechanisms and intracacies of training and inference with reasoning models.

- Training with SFT and RL
- Parallel scaling
- Inference challenges

These tools allow AI models to tackle increasingly challenging problems.

But there's still much more to do!

References I

- *"Large Language Models are Zero-Shot Reasoners,"* Kojima et al., *NeurIPS*, 2022.
- *"Cognitive Behaviors that Enable Self-improving Reasoners, or, Four Habits of Highly Effective Stars,"* Gandhi et al., *COLM*, 2025.
- *"On-policy Distillation,"* Lu et al., 2025.
- *"Self-Distilled Reasoner: On-Policy Self-Distillation for Large Language Models,"* Zhao et al., *ICML*, 2026.
- *"Self-Distillation Enables Continual Learning,"* Shenfeld et al., *ICML*, 2026.
- *"Deepseekmath: Pushing the Limits of Mathematical Reasoning in Open Language Models,"* Shao et al., *arXiv preprint*, 2025.

References II

- *"DeepSeek-R1 Incentivizes Reasoning in LLMs Through Reinforcement Learning,"* Guo et al., *Nature*, 2025.
- *"Mamba: Linear-Time Sequence Modeling with Selective State Spaces,"* Gu and Dao, *COLM*, 2024.
- *"Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference,"* Dong et al., *ICML*, 2024a.
- *"AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration,"* Lin et al., *MLSys*, 2024.
- *"Kimi K3: Open Frontier Intelligence,"* Kimi Team et al., *arXiv preprint*, 2026.
- *"Prompt-prompted Adaptive Structured Pruning for Efficient LLM Generation,"* Dong, Chen, and Chi, *COLM*, 2024.

References III

- *"CATS: Contextually-Aware Thresholding for Sparsity in Large Language Models,"* Lee et al., *COLM*, 2024.
- *"LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks,"* Bai et al., *ACL*, 2025.
- *"Scalable LLM Reasoning Acceleration with Low-rank Distillation,"* Dong et al., *CPAL*, 2026a.
- *"Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference,"* Yuan et al., *NeurIPS*, 2025.
- *"How enabling two settings tripled our scores on the ARC-AGI-3 benchmark,"* Bigio and Sanders, 2026.
- *"Large Language Monkeys: Scaling Inference Compute with Repeated Sampling,"* Brown et al., *arXiv preprint*, 2024.

References IV

- *"Generalized Parallel Scaling with Interdependent Generations,"* Dong et al., *ICLR*, 2026b.
- *"Training Large Language Models to Reason in a Continuous Latent Space,"* Hao et al., *COLM*, 2025.
- *"DAPO: An Open-Source LLM Reinforcement Learning System at Scale,"* Yu et al., *NeurIPS*, 2025.
- *"Understanding R1-Zero-Like Training: A Critical Perspective,"* Liu et al., *COLM*, 2025.
- *"Full-bandwidth Transformer,"* Wang et al., *arXiv preprint*, 2026.