Decoding & Test-time Scaling

EECE695D: Efficient ML Systems

Spring 2025

Today

- We talk about computational issues of **LLM decoding**
  - Pitfalls of greedy decoding
  - Computation-friendly solutions

Language modeling

- **Recall.** Language modeling is about approximating the ground-truth data-generating distribution

$$\hat{P} \approx P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$$

- So that we can:
  - Generate realistic samples $\vec{\mathbf{x}} \approx \hat{P}$
  - Make inference $\hat{P}(\mathbf{x} \mid \text{"Q. What color is an apple? A."})$
  - (and so on)

LLMs

- **LLM.** Most modern LLMs solve this by modeling the next-token probability
  - Input. A sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$
  - Output. Approximation of the conditional probability

$$\hat{P} \approx P(\mathbf{x}_{n+1} \mid \mathbf{x}_{1:n})$$

    - Very easy to train with unsupervised data

- **Question.** Suppose that we want to draw a length-L sample from $\hat{P}$.
  What should we do with this next token predictor?

Greedy decoding

- Naïvely, we would do **greedy decoding**:
  - For $n = 1, \ldots, L - 1$, repeat:

$$\hat{\mathbf{x}}_{n+1} = \operatorname{argmax}_{\mathbf{x}} \hat{P}(\mathbf{x} \mid \hat{\mathbf{x}}_{1:n})$$

- However, there are several pitfalls:
  - Resorts to a single, suboptimal solution (Pt 1: test-time scaling)
  - Difficult to parallelize (Pt 2: parallel decoding)

Test-time scaling

Greedy decoding

$$\hat{\mathbf{x}}_{n+1} = \text{argmax}_{\mathbf{x}} \hat{P}(\mathbf{x} \,|\, \hat{\mathbf{x}}_{1:n})$$

- Greedy sampling resorts to a **single solution**
    - The argmax operation is deterministic
    - Lacks diversity

- Worse, the sampled solution is **not always max-prob solution**

$$\hat{\mathbf{x}} \neq \text{argmax}_{\mathbf{x}} \hat{P}([\mathbf{x}_1, \ldots, \mathbf{x}_n])$$

- Greedy search is mypoic

Example: Myopic

- Suppose that we want to complete the sentence:

"I have (word 1) (word 2)"

- Suppose that we have:

$$\hat{P}(\text{"a"} \mid \text{"I have"}) = 0.7, \qquad \hat{P}(\text{"an"} \mid \text{"I have"}) = 0.3$$

$$\hat{P}(\text{"pear"} \mid \text{"I have a"}) = \hat{P}(\text{"cherry"} \mid \text{"I have a"}) = \hat{P}(\text{"banana"} \mid \text{"I have a"}) = 1/3$$

$$\hat{P}(\text{"apple"} \mid \text{"I have an"}) = 1$$

- The max-prob solution is: "an apple," w.p. 30%.
  - Greedy decoding will find something that starts with "a"

Random sampling

- One thing we can try is simply **random sampling**:
  - If the logits have been $\mathbf{z}_1, \ldots, \mathbf{z}_K$, then:

$$P(\hat{\mathbf{x}}_{n+1} = k) = \frac{\exp(\mathbf{z}_k)}{\sum_{i=1}^{K} \exp(\mathbf{z}_i)}$$

  - Can also do **temperature scaling**:

$$P(\hat{\mathbf{x}}_{n+1} = k) = \frac{\exp(\mathbf{z}_k/\tau)}{\sum_{i=1}^{K} \exp(\mathbf{z}_i/\tau)}$$

  - Diverse, but very suboptimal in many cases

Advanced sampling

- Advanced methods narrow down the options before sampling
  - **Top-k.** At each step, sample among top-K options only
  - **Nucleus.** Choose top tokens such that cumulative prob exceeds some p

Top-k: Select 3

$$[0.30, \quad 0.25, \quad 0.12, \quad 0.11, \quad \cdots]$$

Nucleus: (0.30+0.25+0.12+0.11) > p > (0.30+0.25+0.12)

Test-time scaling

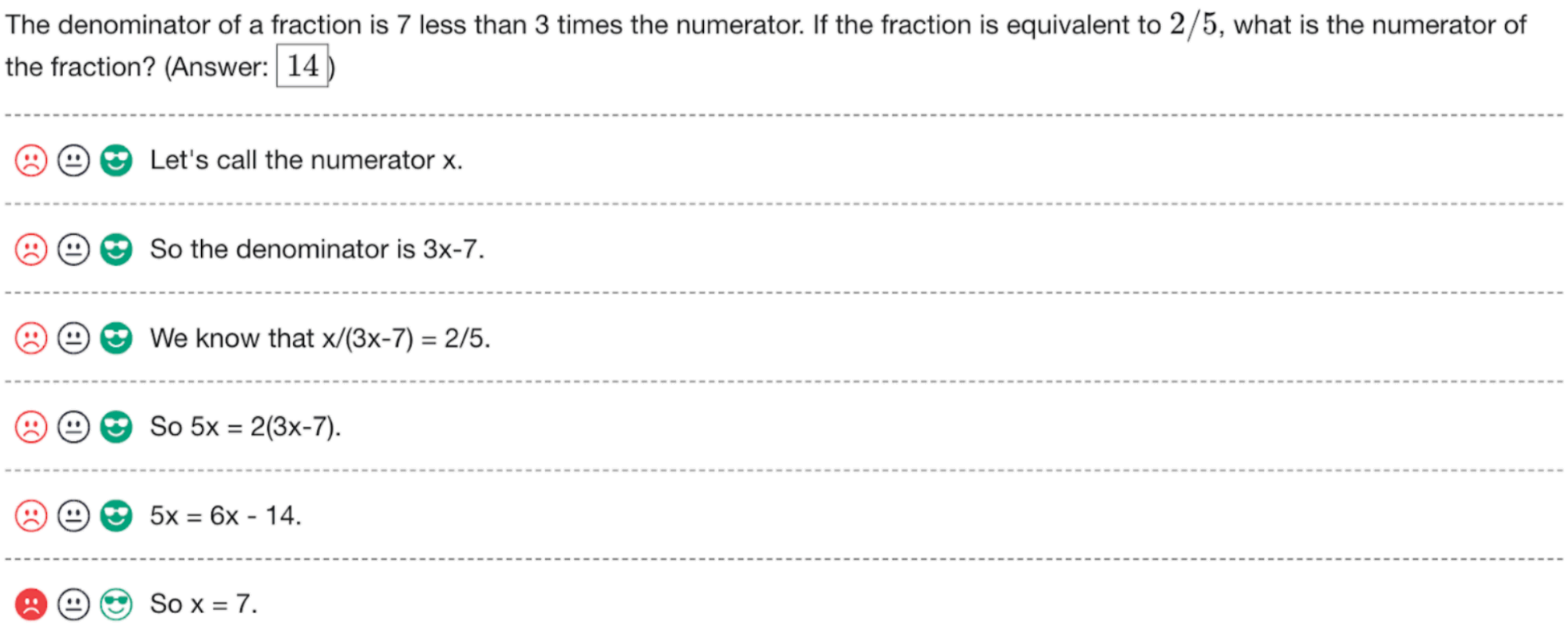
- One find higher-prob solution with a higher chance, using more samples
    - Uses extra computation (thus called test-time scaling)

- **A simple scaling method: Best-of-N**
    - Sample N sample sequences independently (w/ any sampling scheme)
    - Select the highest-probability one
        - $\log \hat{P}(\text{"word 1"}) + \log \hat{P}(\text{"word 2"} \mid \text{"word 1"}) + \cdots$
        - Take a majority vote of final answers, if applicable

Note. Sampling can be done in parallel, thus scalable in terms of latency

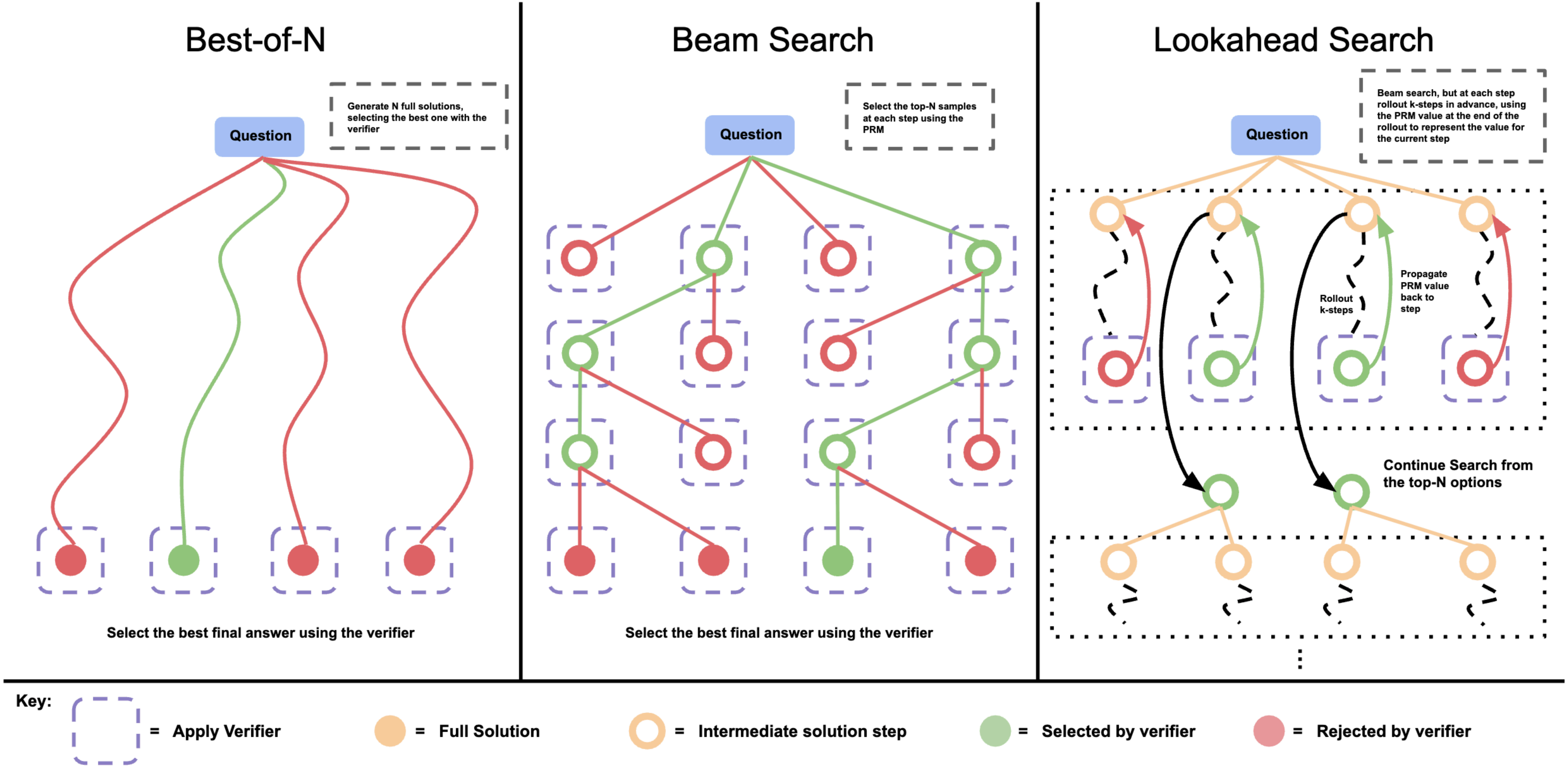
Test-time scaling

- One can replace “select the highest-probability” with **reward models**
  - Trained verifiers
- **Example.** “Let’s verify step-by-step” (Lightman et al., 2024)
  - Collected human feedback on the quality of the reasoning process, to train an evaluation model



Lightman et al., “Let’s verify step-by-step,” ICLR 2024

Fine-grained verification schedule



Snell et al., "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters," ICLR 2025

Parallel vs. Sequential

Snell et al., “Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters,” ICLR 2025

Recent lessons

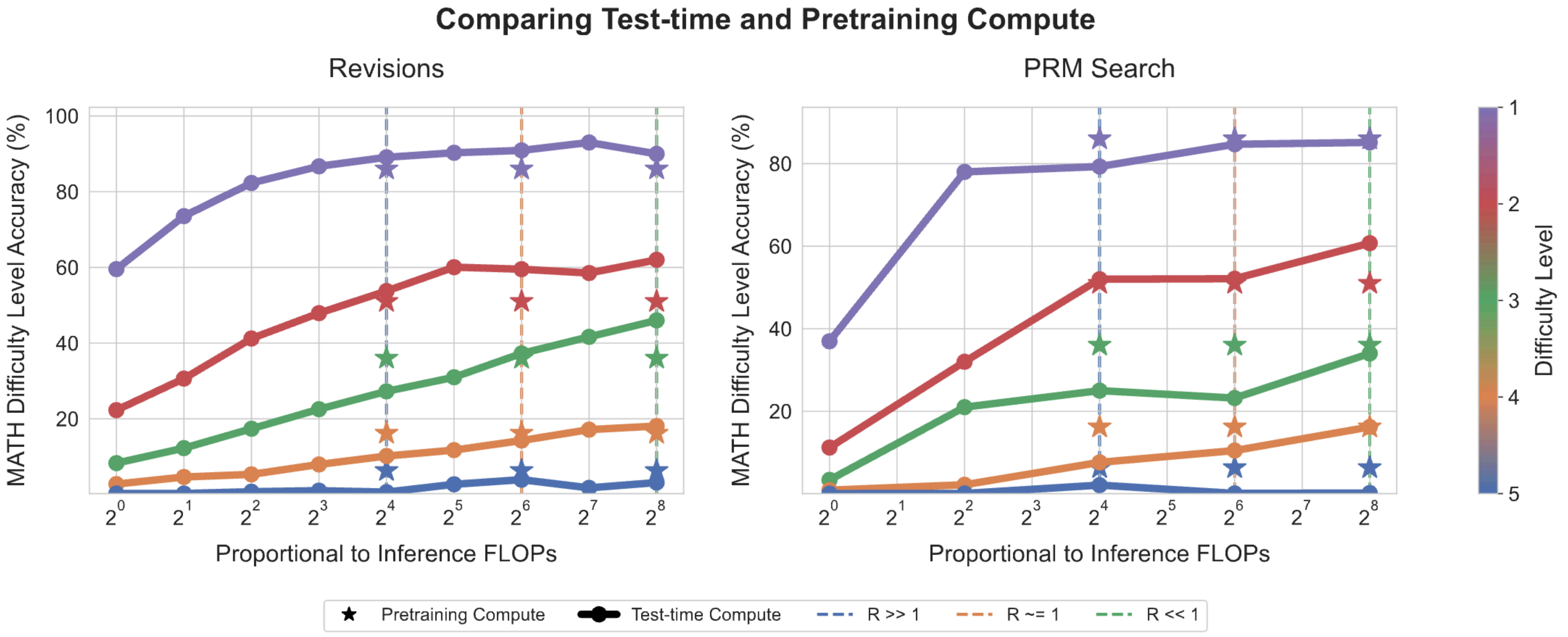
- Test-time scaling seems to be very powerful
  - Under certain scenarios, using compute for test-time scaling is better than using the same compute for pretraining

Question: Exchanging pretraining and test-time compute

Suppose a model was pre-trained with X FLOPs. Assume that we plan to run Y FLOPs of inference with this model. If we want to improve performance by increasing the total FLOPs budget by a factor of M (i.e., $M(X + Y)$ total FLOPs across both pretraining and inference), should we spend our FLOPs on increased pretraining compute or on additional test-time compute?

Snell et al., "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters," ICLR 2025

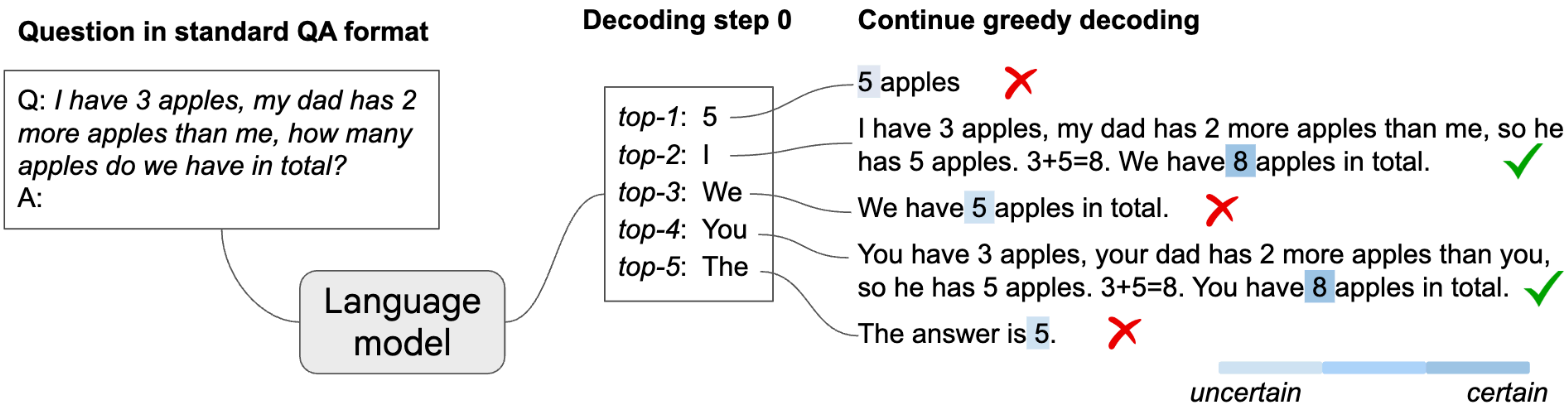
Recent lessons



Snell et al., “Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters,” ICLR 2025

Recent lessons

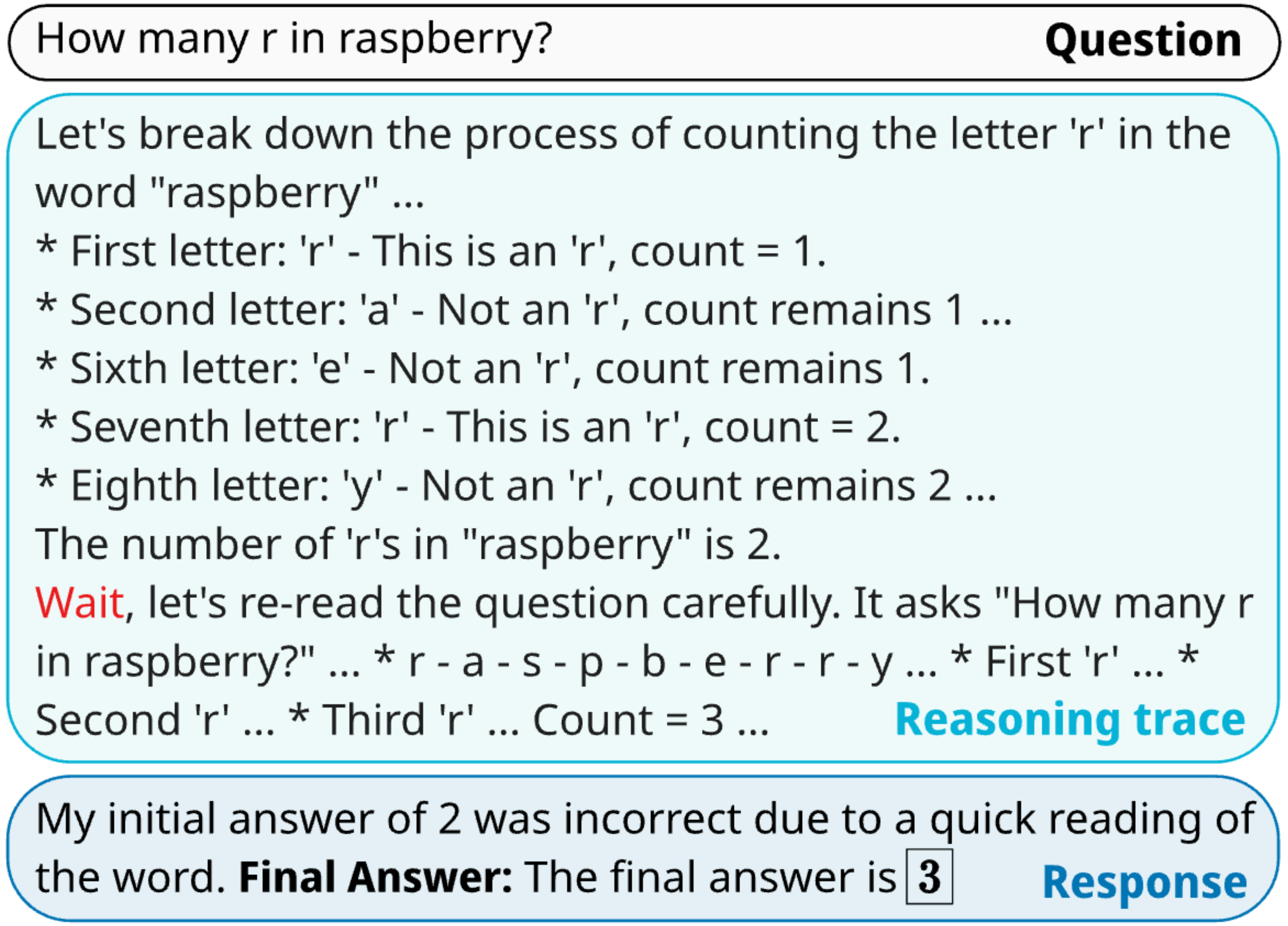
- Sampling policy can replace chain-of-thought prompting
  - Not using top-1 can elicit inherent chain-of-thought reasoning



Wang and Zhou., "Chain-of-Thought Reasoning without Prompting," NeurIPS 2024

Recent lessons

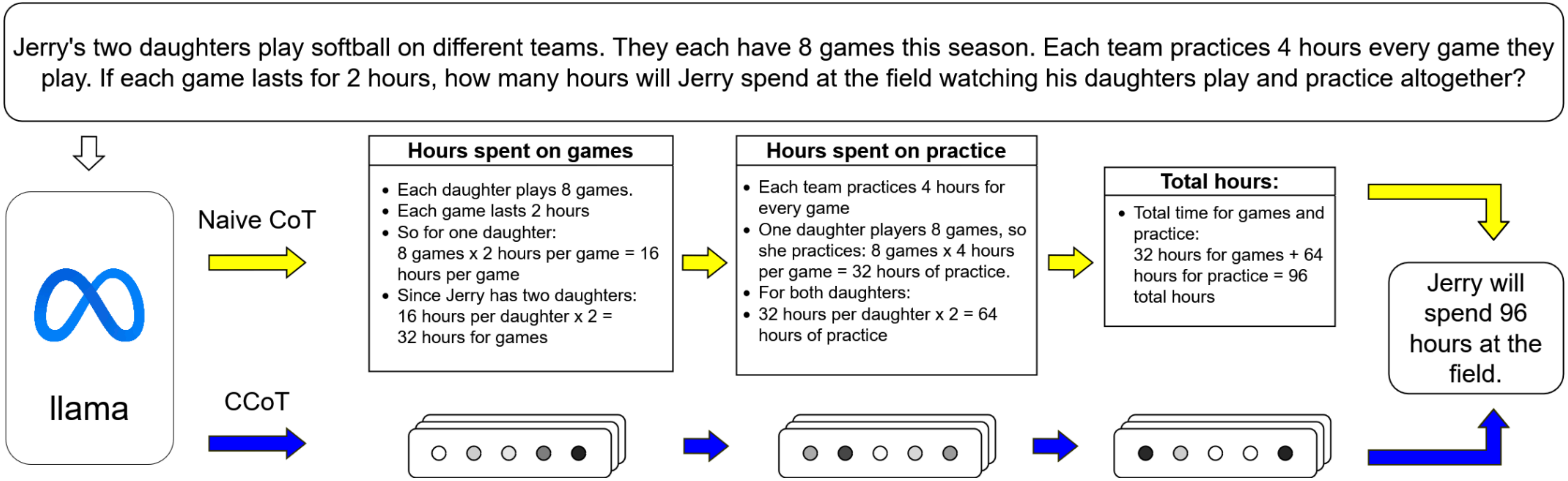
- A simple yet effective method to force sequential revision—and thus use more test-time compute—is to simply add “wait”
  - called “budget forcing”



Muenninghoff et al., “s1: Simple test-time scaling,” arXiv 2025

Recent lessons

- One can compress the reasoning process by extra fine-tuning
- **Example.** Compressed CoT compresses the reasoning procedure into a sequence of continuous “contemplation” tokens



Cheng and van Durme, “Compressed Chain of Thought: Efficient Reasoning through Dense Representations,” arXiv 2024

Further readings

- A nice survey:
  - https://arxiv.org/abs/2406.16838
- A neat tutorial blog post:
  - https://rentry.co/samplers

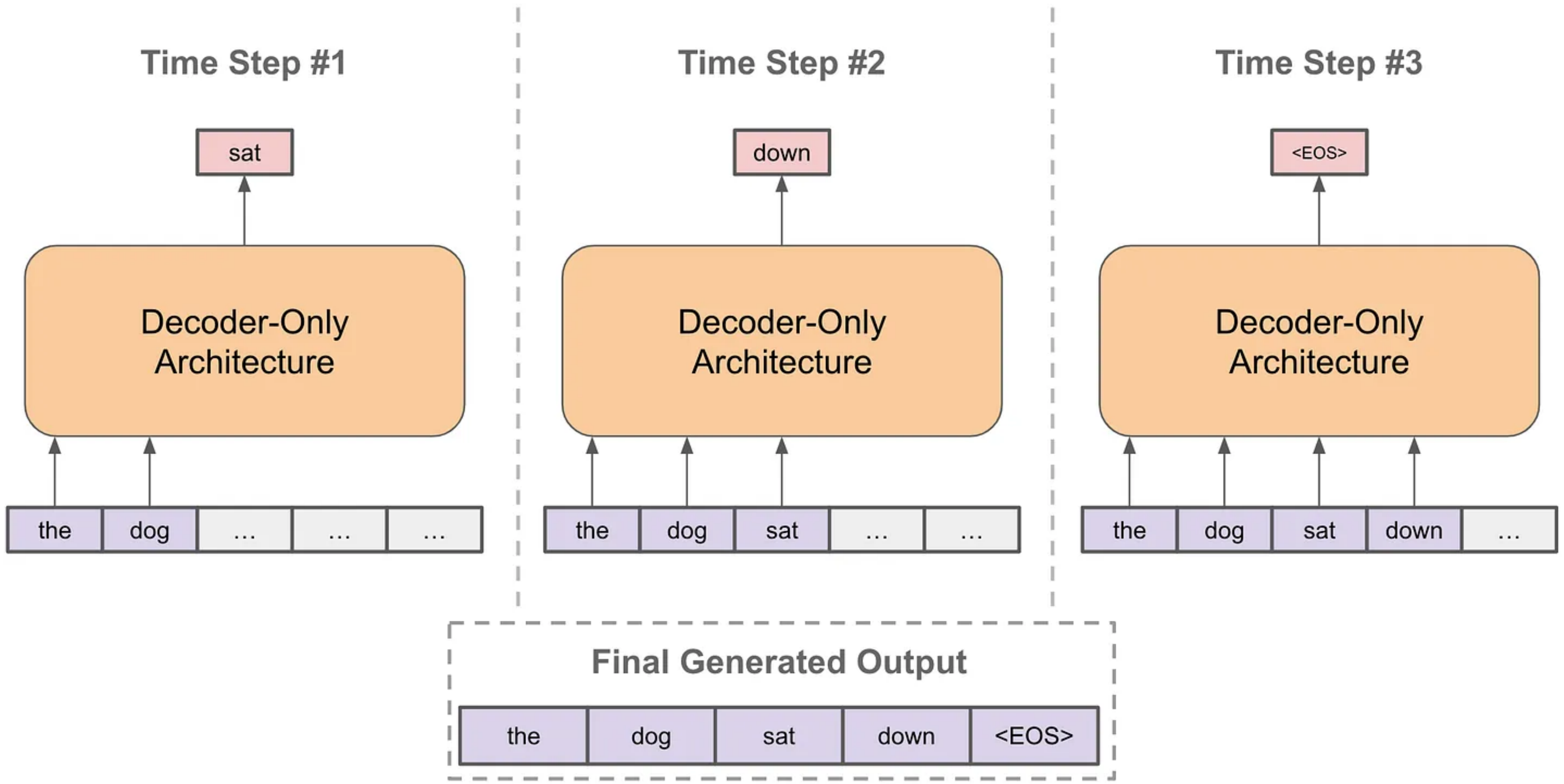
Parallel decoding

One-by-one decoding

- LLMs operate in a sequential manner

$$\text{Sample } \mathbf{x}_1 \rightarrow \text{Sample } \mathbf{x}_2 \rightarrow \text{Sample } \mathbf{x}_3 \rightarrow$$

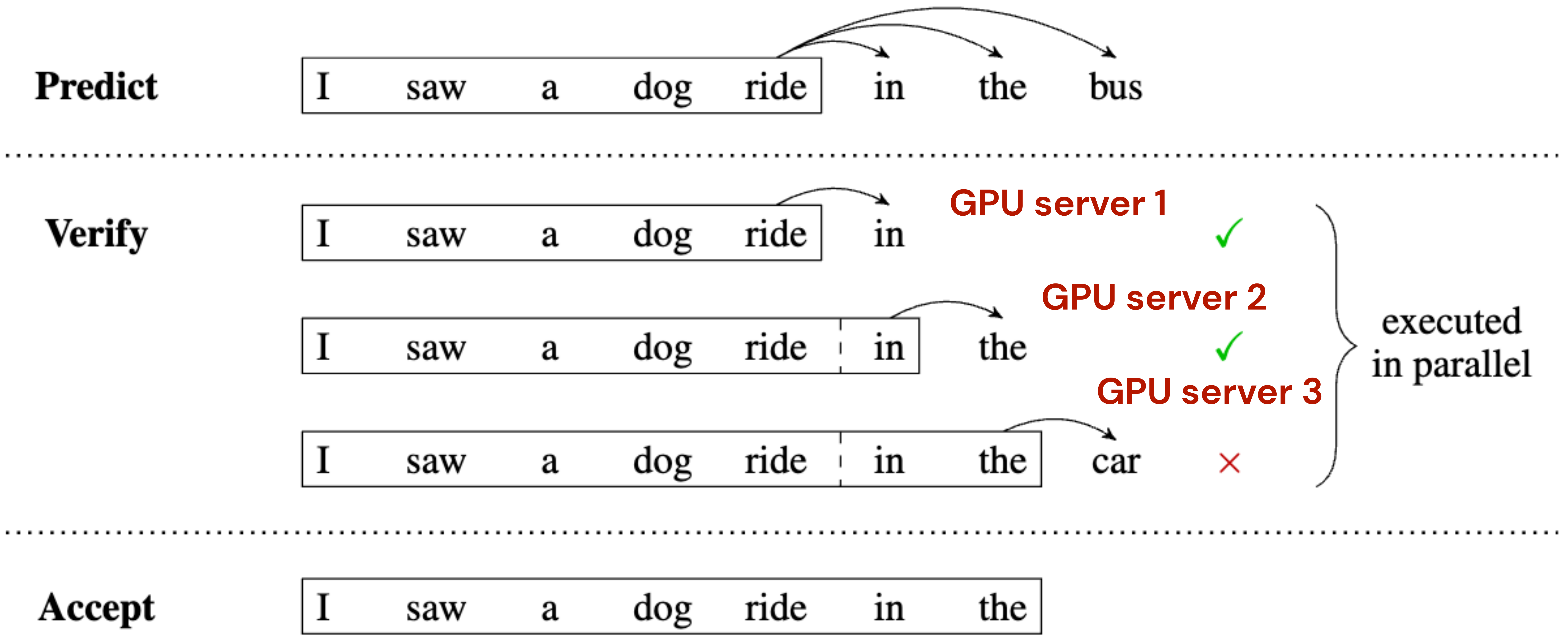
  - Cannot be parallelized effectively, per se.



https://cameronrwolfe.substack.com/p/language-model-training-and-inference

Parallelizing the verification

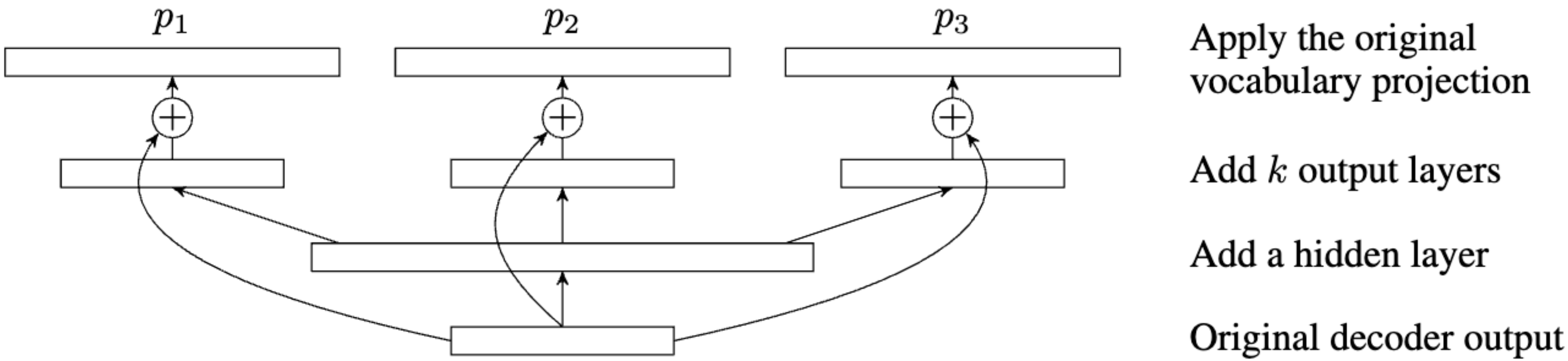
- **Idea.** We can verify in parallel!
  - Train a model that generate a block of tokens
  - Use multiple LLMs to verify up to which token is correct



Stern et al., "Blockwise Parallel Decoding for Deep Autoregressive Models" NeurIPS 2018

Parallelizing the verification

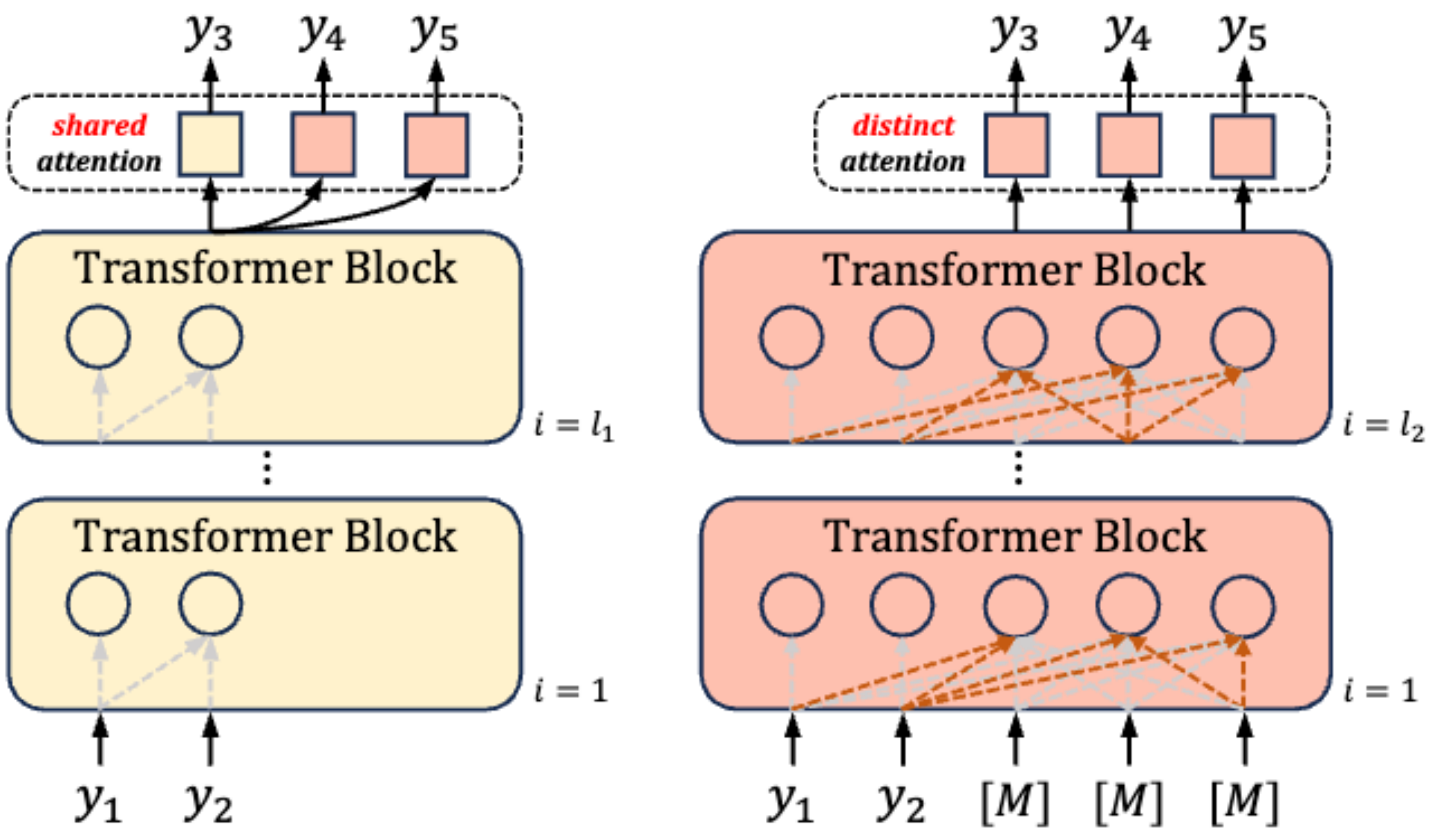
- **Question.** How do we generate multiple tokens?
- Option#1. Fine-tune additional heads
    - Limitation: predicting far-future tokens may require capturing different attention patterns



Stern et al., “Blockwise Parallel Decoding for Deep Autoregressive Models” NeurIPS 2018

Parallelizing the verification

- Option#2. Use a standalone small, autoregressive model (called "drafter")
  - Verification ensures that the results are identical as LLM
  - SLM often produces better result than LLM
    - Accept if top-k



(a) Blockwise Decoding (b) Spec-Drafter

Xia et al., "Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation," EMNLP 2023
Leviathan et al., "Fast Inference from Transformers via Speculative Decoding," ICML 2023

Random sampling + Speculative decoding

- Leviathan et al. (2023) extends the draft-then-verify framework to the case of generation-by-sampling
- **Example**
  - Suppose that the drafter generates with $\hat{Q}(\mathbf{x})$
    the verifier generates with $\hat{P}(\mathbf{x})$
  - We sample from $\hat{Q}(\mathbf{x})$, then do:
    - If $\hat{Q}(\mathbf{x}) \leq \hat{P}(\mathbf{x})$: Accept the sample
    - If $\hat{Q}(\mathbf{x}) > \hat{P}(\mathbf{x})$: Reject the sample w.p. $1 - \hat{P}(\mathbf{x})/\hat{Q}(\mathbf{x})$
      - Resample from $\mathrm{norm}(\max(0,\hat{P}(\mathbf{x}) - \hat{Q}(\mathbf{x})))$

Leviathan et al., "Fast Inference from Transformers via Speculative Decoding," ICML 2023

Further readings

- Self-speculative decoding
  - https://arxiv.org/abs/2309.08168
- Consistency LLMs (Jacobi decoding)
  - https://arxiv.org/abs/2403.00835
- Language modeling by Diffusion
  - https://arxiv.org/abs/2502.09992
- Medusa
  - https://arxiv.org/abs/2401.10774

That’s it for today 🙌