Inference Scaling

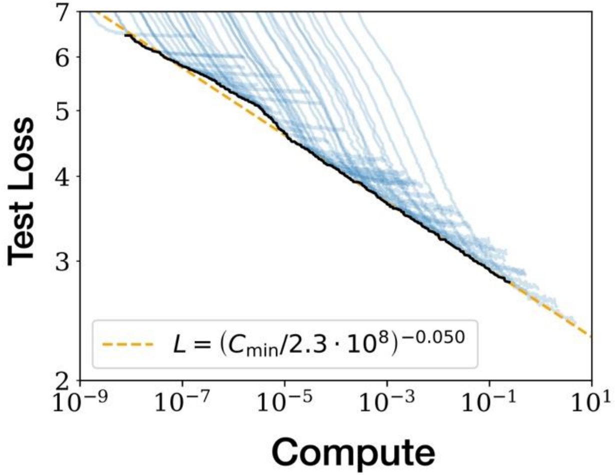
CSCI 601-471/671 (NLP: Self-Supervised Models)

Johns Hopkins Whiting School of Engineering

https://self-supervised.cs.jhu.edu/sp2025/

Scaling pre-training

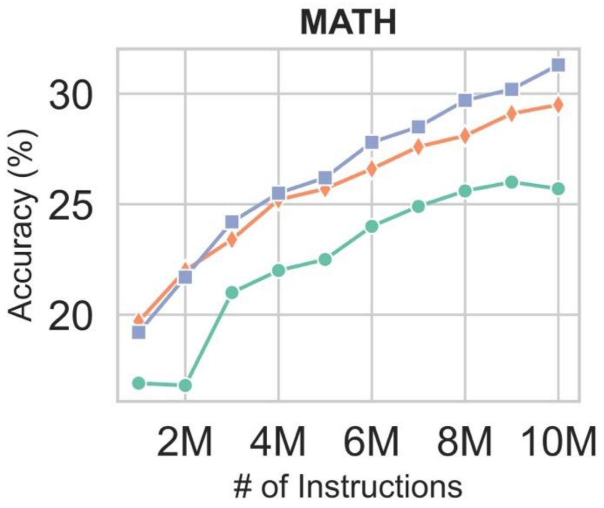
- The dominant focus of past few years has been on scaling training.
  - Involves multiple axes: model params, pre-training data and computing resources

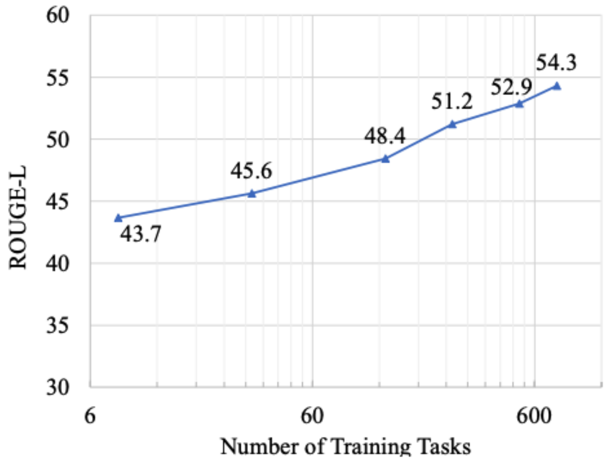


test loss predictably improves with increased pretraining compute
[Kaplan et al 2020]

Scaling fine-tuning

- We have also seen the benefits of scaling fine-tuning/alignment on diverse range of data.





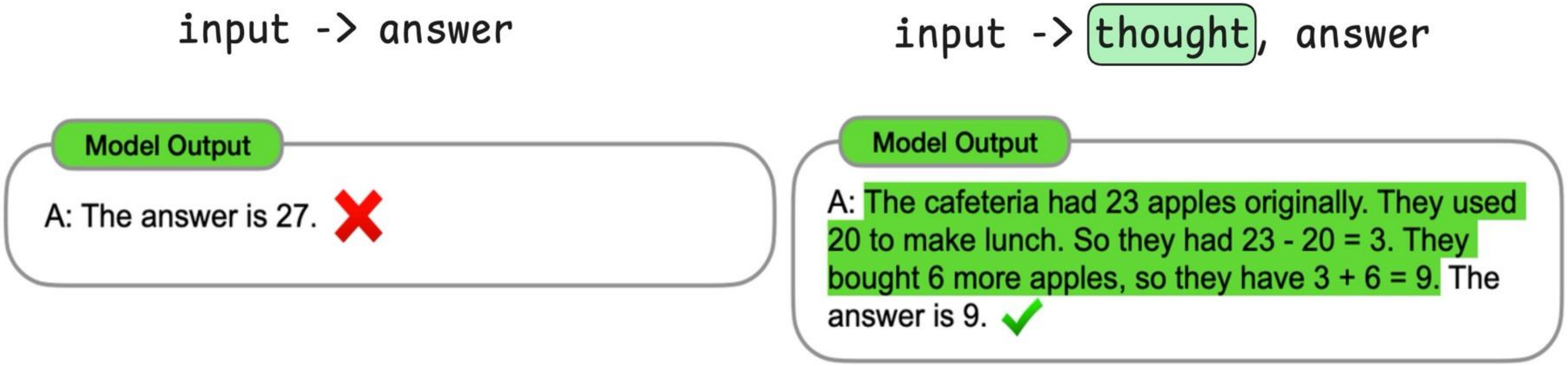
[MAmmoTH2: Scaling Instructions from the Web, 2024]

[Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks, Wang et al. 2022]

Johns Hopkins Whiting School of Engineering

Inference-time scaling: example (1)

- **Inference-time scaling:** allowing models to continue reasoning at inference time to get better results, at the cost of inference compute.
- **An earlier example:** Letting models produce "thought" tokens improves its accuracy.



[Slide credit: Sean Welleck]

Inference-time scaling: example (3)

- **Inference-time scaling:** allowing models to continue reasoning at inference time to get better results, at the cost of inference compute.
- **Example from different domain:** Training vs inference scaling laws in board games.
  - Notice that one can make up for <u>less</u> training with <u>more</u> inference-time scaling.

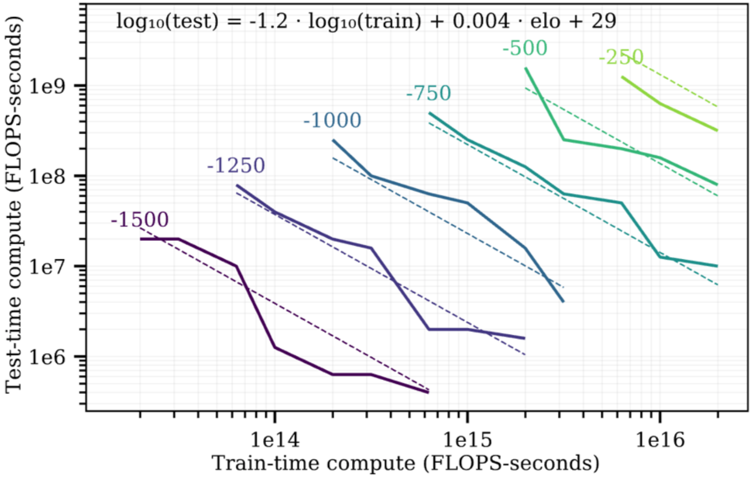
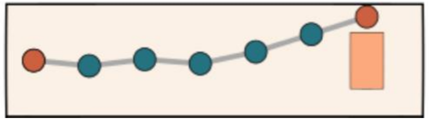


Fig. 9. The trade-off between train-time compute and test-time compute. Each dotted line gives the minimum train-test compute required for a certain Elo on a 9×9 board

[Scaling Scaling Laws with Board Games, 2021]

Johns Hopkins Whiting School of Engineering

Inference scaling: problem setup



- **Input problem** is given: x
- **Intermediate thoughts:** $z_{1:T}$
  - The "thoughts" are characterized by a policy model (the "thinker"): $p_\theta(.|x)$.
  - The "thoughts" may also depend on a reward: $r_\phi(.|x)$.
    - (Other names for "reward": verifier, teacher, feedback.)
  - Note: these may accept partial thoughts: $r_\phi\left(z_{t+1}\middle|x, z_{1:t}\right)$ to score thought continuation.
- **Ultimate answer:** $y \sim p(.|x, z_{1:T})$
  - Answer given question and thoughts.
  - Sampled from the policy model.

[Figure: Sasha Rush]

Inference-time scaling: dimensions

- Here are the key issues we need to figure out:
  1. How should we think about the reward function $r_\phi(.)$?
  2. What we do during inference, given $r_\phi(.)$ and $p_\theta(.)$ that are fixed?
  3. How we train the policy $p_\theta(.)$ and reward model $r_\phi(.)$?

Inference-time scaling: dimensions

- Here are the key issues we need to figure out:
  1. **How should we think about the reward function $r_\phi(.)$?**
  2. What we do during inference, given $r_\phi(.)$ and $p_\theta(.)$ that are fixed?
  3. How we train the policy $p_\theta(.)$ and reward model $r_\phi(.)$?

The Reward Function

Reward mechanisms

- A **reward** function $r_\phi(.)$ tells you whether the response is going in the “right” direction.
  - The reward can be a neural network or a non-statistical model.
- **Non-statistical examples:**
  - Python Linter check (was there a syntax errors?)
  - SQL execution response (did the query find the right columns?)
  - Regular expression (did the math equations adhere to proper syntax?)
  - Chess engine (did the policy model win the game against the opponent?)
- **Statistical examples:**
  - A classifier trained to predict how likely the current thoughts lead to an answer.
  - An LM prompted with certain goal (“LLM-as-a-judge”, e.g., "Constitutional AI")

Reward mechanisms: flavors

- The reward may be binary $r_\phi(.) \in \{0, 1\}$ or continuous $r_\phi(.) \in (0, 1)$.
- Some rewards may need full thoughts, but some may work for partial thoughts.

Outcome or process? → **Learned or not learned? ↓**	**Outcome reward (ORM)**	**Process Reward (PRM)**
Learned typically, $r_\phi(.) \in (0, 1)$		
Not learned typically, $r_\phi(.) \in \{0, 1\}$		

[Uesato et al., 2022, Lightman et al., 2023]

Inference-time scaling: dimensions

- Here are the key issues we need to figure out:
  1. How should we think about the reward function $r_\phi(.)$?
  2. **What we do during inference, given $\boldsymbol{r_\phi}(.)$ and $\boldsymbol{p_\theta}(.)$ that are fixed?**
  3. How we train the policy $p_\theta(.)$ and reward model $r_\phi(.)$?

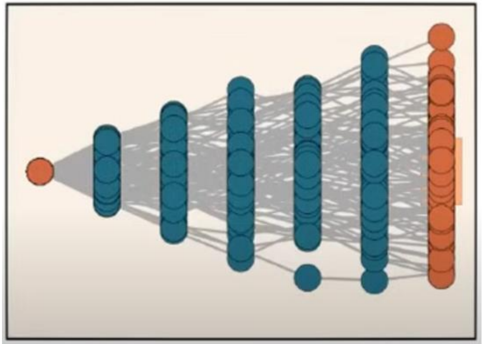
Inference Algorithms

Inference scaling: inference objective

- **Thoughts as latent variables** — Suppose $r_\phi(.)$ and $p_\theta(.)$ are fixed.
- We care about $p(y|x)$, but that depends on intermediate thoughts.
- Think about a marginalization over all possible thoughts $z_{1:T}$:

$$\max_y p(y|x) = \max_y \mathbb{E}_{z \sim p_\theta(.|x)}[p(y, z_{1:T}|x)]$$

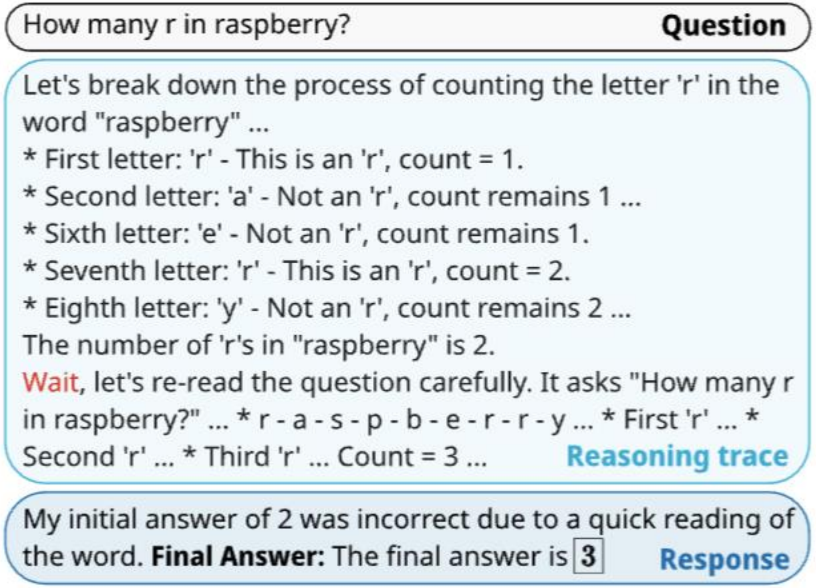
- In other words, if we were to magically explore the whole space of "thoughts" what is the best solution?
- But the space of thoughts is combinatorially large! 🙅‍♀️
  - So, we need to think of a cost-efficient way to explore the the space of "thoughts".



[figure credit: Sasha Rush]

Inference-time scaling:
Just let your LLM speak/think

- Get your LLMs spell out its thoughts.
  - **Challenge:** LLMs tend to stop early if they believe they've already answered.
- How can you *force* models to keep thinking?
- A simple heuristic is to keep "nudging" the model to continue generating responses; if it stops, append "wait" or "wait, are you sure?" to encourage further output.

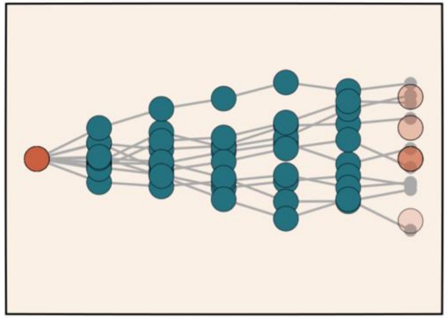


[s1: Simple test-time scaling, 2025](#)

Inference-time scaling: self-consistency

- **Self-Consistency or Majority voting** — Generate collections of isolated "thoughts" (parallel) and then aggregate them.

- For N samples:
  - Sample thoughts: $z_{1:T}^{n} \sim p(.|x)$
  - Sample answers: $y^{n} \sim p(.|x, z_{1:T}^{n})$
- Pick majority vote: $y \sim p(.|x, \{z_{1:T}^{n}, y^{n}\}_{n=1}^{N})$

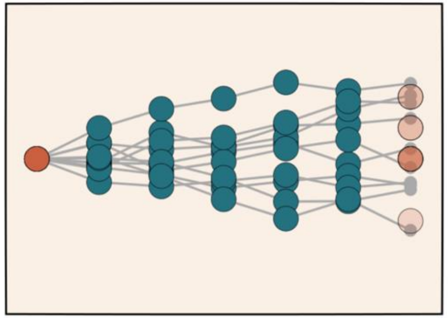


[Slide credit: Sasha Rush]

Inference-time scaling: best-of-N

- **Rejection Sampling (best of N)** — Generate collections of isolated "thoughts" (parallel) and their corresponding answer. Then pick one/best.

- For N samples:
  - Sample thoughts: $z_{1:T}^n \sim p(.|x)$
  - Sample answers: $\mathrm{y}^n \sim p(.|x, z_{1:T}^n)$
- Pick the best — few choices:
  - Based on the reward: $\max_n r(y^n|x, z_{1:T}^n)$
  - Based on the policy + reward: $\max_n \left[p(y^n|x, z_{1:T}^n) r_\phi(y^n|x, z_{1:T}^n)\right]$

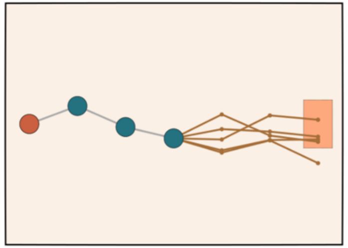


[Slide credit: Sasha Rush]

Inference-time scaling: search

- **Search algorithm**— Using the intermediate scores at each step, i.e., reward $r_\phi(.)$ and policy $p_\theta(.)$, progressively guide the generation of thoughts using A*, Beam Search, Monte Carlo Tree Search (MCTS), etc.

- Think of **Beam search** which is a heuristic search algorithm.
- For T steps:
  - Maintain a beam of size B, # of candidate sequences at each step.
  - Given a partial "thought" $z_{1:t} \sim p(x)$, expand each sequence in the beam by considering the top k probable next steps based on $p(.)$.
  - Score each sequence (based on reward or policy).
  - Prune out the search space by keeping the top B sequences.



[figure credit: Sasha Rush]

Inference-time scaling: dimensions

- Here are the key issues we need to figure out:
  1. How should we think about the reward function $r_\phi(.)$?
  2. What we do during inference, given $r_\phi(.)$ and $p_\theta(.)$ that are fixed?
  3. **How we train the policy $p_\theta(.)$ and reward model $r_\phi(.)$?**

Training a good inference-scaler

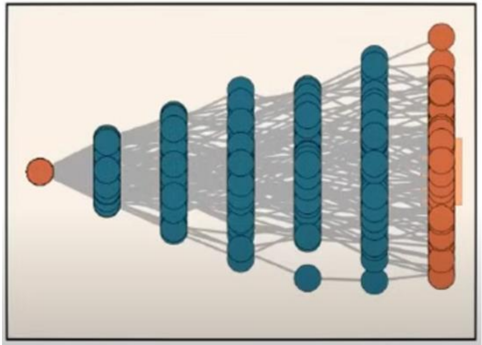
Thinking about the inference objective

- **Thoughts as latent variables** — Suppose $r_\phi(.)$ and $p_\theta(.)$ are fixed.
- We care about $p(y|x)$, but that depends on intermediate thoughts.
- Think about a *marginalization* over all possible thoughts $z_{1:T}$:

Can be replaced with $r_\phi(.)$

$$\max_y p(y|x) = \max_y \mathbb{E}_{z \sim p_\theta(.|x)}[p_\theta(y, z_{1:T}|x)]$$

- In other words, if we were to magically explore the whole space of "thoughts" what is the best solution?
- But the space of thoughts is combinatorially large! 🙅
  - So, we need to think of a cost-efficient way to explore the the space of "thoughts".



[figure credit: Sasha Rush]

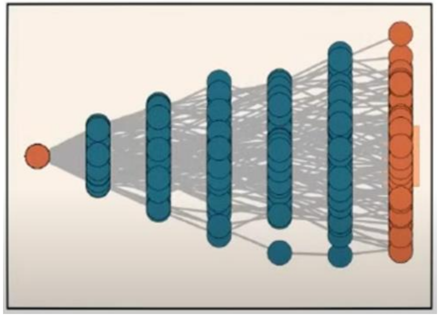
Training for scale inference

- The central question here is how do you train the "policy" (the "thinker") to be more conducive to inference-scaling?

- **Why is this non-trivial?** Remember the inference goal:

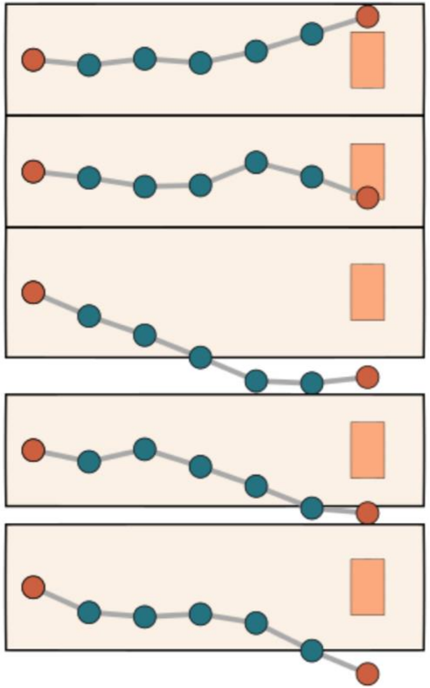
$$p_\theta(y|x) = \mathbb{E}_{z \sim p_\theta(.|x)}[p_\theta(y, z_{1:T}|x)]$$

- Even if I give you training data (x, y) (or equivalently, a perfect verifier of final answers), you still can't train it since you don't know what thoughts lead to a given correct answers)



Approach 1: Guess and Check

- **Guess and Check:**
  - Sample N chain of thoughts
  - Check if successful with ORM
  - Train on the good ones

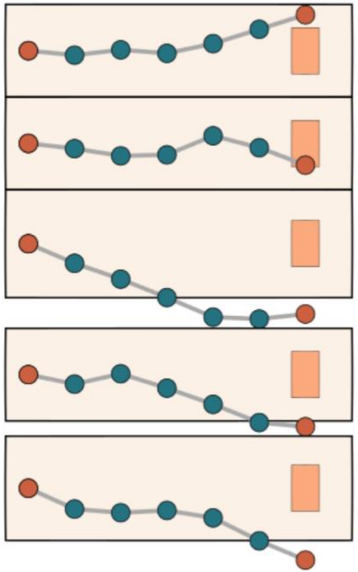


Guess and Check, formalized

- There is a principled way to describe "Guess and Guess" via Expectation Maximization (EM).
- The training goal: $\max_{\theta} \{\mathbb{E}_{z \sim p_\theta(.|x)}[p_\theta(y, z_{1:T}|x)]\}$

1. **E-Step:** Explore in the space of likely thoughts
   - For N samples:
     - Sample thoughts: $z_{1:T}^n \sim p(.|x)$
     - Sample answers: $y^n \sim p(.|x, z_{1:T}^n)$
   - Using ORM (gold label or verifier), keep the "good" thoughts: Z_{good}.
2. **M-Step:** Update parameters to maximize our objective.

$$\theta' \leftarrow \max_{\theta} \sum_{Z_{\text{good}}} p_\theta(y, z|x)$$



[Slide credit: Sasha Rush]

Guess and Check as EM: Few Examples

- Different people refer to this approach under different names:
    - Self-Training [Yarowsky, 1995]
    - Best-of-N Training [Cobbe et al., 2021]
    - STaR [Zelikman et al., 2022]
    - ReST [Gulcehre et al., 2023]
    - ReST-EM [Singh et al., 2023]
    - Filtered Rejection Sampling [Nakano et al., 2021]

ReST-EM

Algorithm 1: ReST (Expectation-Maximization). Given a initial policy (e.g., pre-trained LM), ReST^{EM} iteratively applies `Generate` and `Improve` steps to update the policy.

Input: $\mathcal{D}$: Training dataset, $\mathcal{D}_{val}$: Validation dataset, $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \theta)$: loss, $r(\boldsymbol{x}, \boldsymbol{y})$: Non-negative reward function, I: number of iterations, N: number of samples per context

for $i = 1$ *to* I **do**

`// Generate (E-step)`

Generate dataset $\mathcal{D}_i$ by sampling: $\mathcal{D}_i = \{ (\boldsymbol{x}^j, \boldsymbol{y}^j)|_{j=1}^N \text{ s.t. } \boldsymbol{x}^j \sim \mathcal{D},\ \boldsymbol{y}^j \sim p_\theta(\boldsymbol{y}|\boldsymbol{x}^j) \}$

Annotate $\mathcal{D}_i$ with the reward $r(\boldsymbol{x}, \boldsymbol{y})$.

`// Improve (M-step)`

while *reward improves on* $\mathcal{D}_{val}$ **do**

Optimise θ to maximize objective: $J(\theta) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_i} [r(\boldsymbol{x}, \boldsymbol{y}) \ \log p_\theta(\boldsymbol{y}|\boldsymbol{x})]$

end

end

Output: Policy p_θ

[Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models, 2023]

ReST-EM

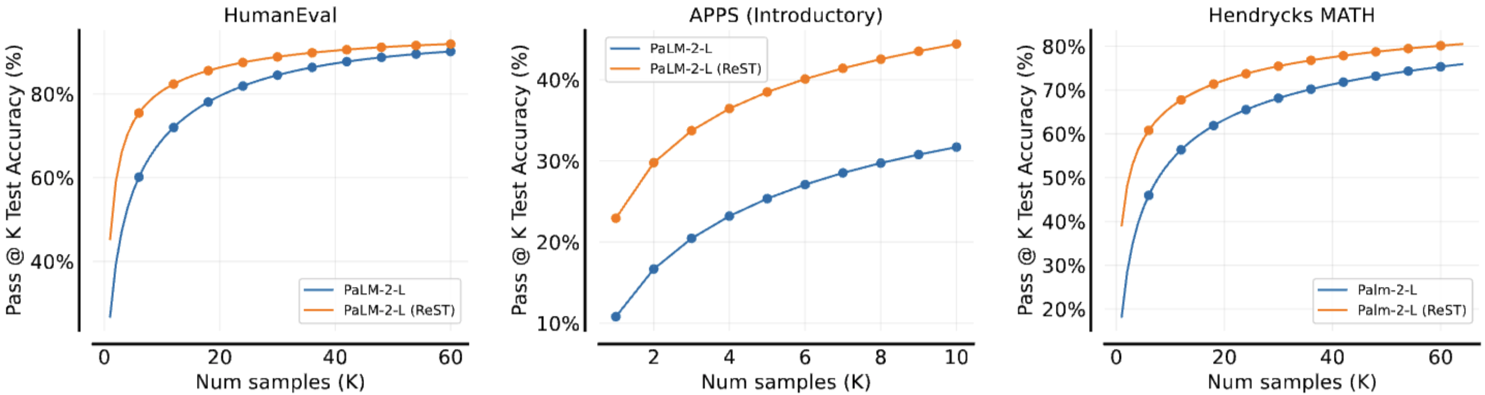
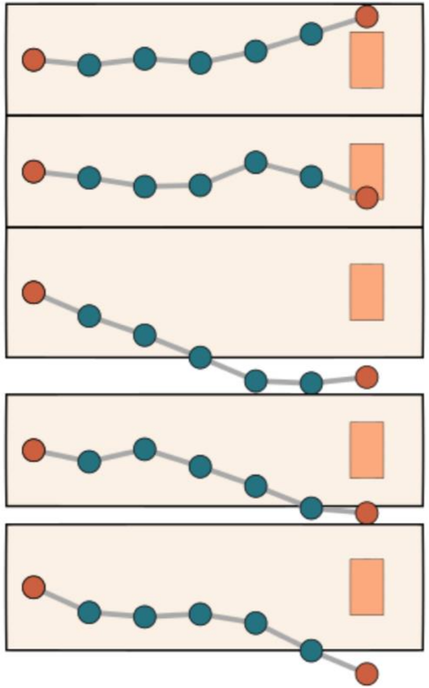


Figure 5 | **Pass@K results** for PaLM-2-L pretrained model as well as model fine-tuned with ReST^{EM}. For a fixed number of samples K, fine-tuning with ReST^{EM} substantially improves Pass@K performance. We set temperature to 1.0 and use nucleus sampling with $p = 0.95$.

Guess and Check: challenges

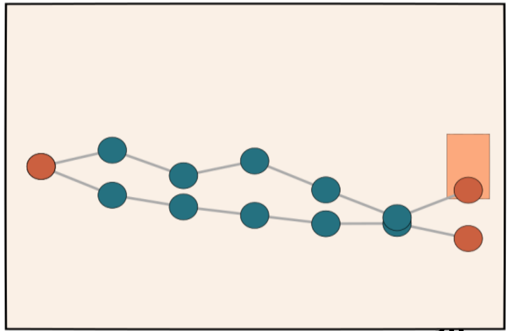
- **Collapse:** "Good" and "bad" thoughts may share most of the path. Often, one or few small mistakes may lead to a bad outcome. If the training data is not able to distinguish such nuances then the model may not learn anything useful.

- **Cost efficiency:** Will sampling around give us reasonable thoughts? If the problem is complex this may take forever!



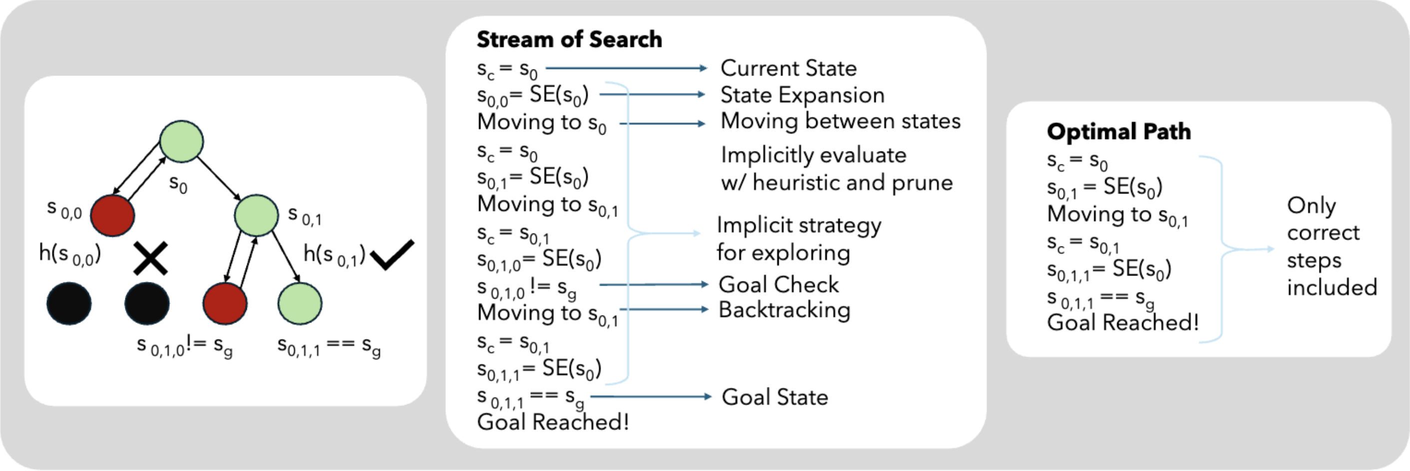
Approach 2: Roll-out and Re-organize

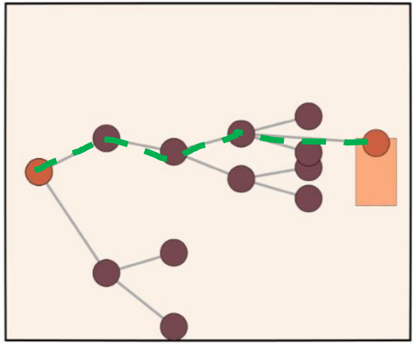
- Like "guess and check", but carefully organize the roll-outs:
- **Guess and re-organize:**
  - Sample N chain of thoughts and label the good and bad ones with ORM.
  - Re-organize the roll-outs:
    - Identify the [likely] good portions of bad roll-outs
    - Construct "thoughts" that contain transitions from "bad" to "good".
  - Train on the re-organized roll-outs.

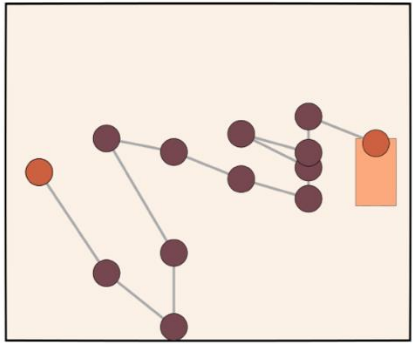


Example: Stream of Search

- Through a careful search, find a "good" path (thought).
  - Say, this gives us $z^*_{1:T}$ path (the shortest path from start to end)
  - To find this path we had to go through several rabitholes and backtrack since the search found out that these were not good.
  - The idea is to linearize this logic and train on it.



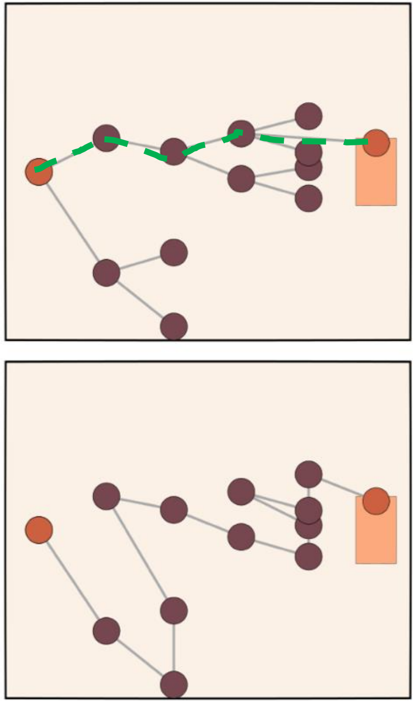




[Stream of Search (SoS): Learning to Search in Language, 2024]

Example: Stream of Search

- Through a careful search, find a "good" path (thought).
  - Say, this gives us $z^*_{1:T}$ path (the shortest path from start to end)
  - To find this path we had to go through several rabitholes and backtrack since the search found out that these were not good.
  - The idea is to linearize this logic and train on it.
- Specifically, find an alternative path $\hat{z}_{1:T'}$ $(T' > T)$ which encodes the whole search (including its "bad" segments). Basically, the chain includes transitions from "bad" segments to the "good" ones.
  - We can find $\hat{z}_{1:T'}$ through linearizing tree process.
- Upon finding $\hat{z}_{1:T'}$, we train an autoregressive model on it.
  - Inference time is just linear generation (no fancy search).
  - Intuitively, this model learns to pivot, if it goes through "bad" direction. (learning to recover)

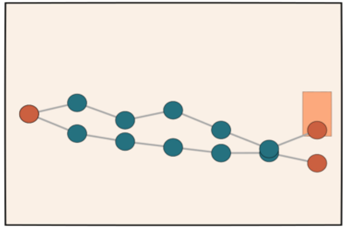


[Stream of Search (SoS): Learning to Search in Language, 2024](#)

Bonus

Example: learning to "self-correct"

- **Learning to Self-Correct:**
  - Given an input x, sample N chain of thoughts and score them with ORM.
  - Pair of "thoughts" z and z' such that:
    - They're similar (lots of overlap)
    - One is improving upon the other, i.e., ORM(z) > ORM(z')
  - Train a "corrector" model to produce z, given input x and z'.
- **Applying it at the inference time:**
  - Given input x, generate an initial solution z
  - Iteratively apply the corrector to z to improve/correct it.

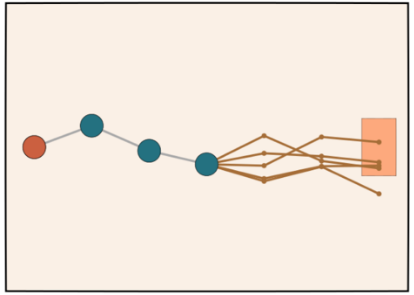


[Generating Sequences by Learning to Self-Correct, 2022]

Approach 3: Training process rewards with search mechanisms

Bonus

- If you can train a process reward (PRM) $r_\phi(.)$ that approximately tell us whether the partial thought is going in the right direction, that will be a strong signal during inference. **But how do you train it?**
- It's going to be difficult to annotate data for training a PRM. It's a massive space!*
- It may be more feasible to infer labels for PRM, using an ORM!
  - Consider MCTS but can be done with any search algorithm.
  - At each step, we have a partial rollout: $z_{1:t}$ up to t.
  - Over-sample rollouts from t onward, score them with ORM, and score the chances that $z_{1:t}$ would lead to a good outcome.
  - Train PRM with such estimates.

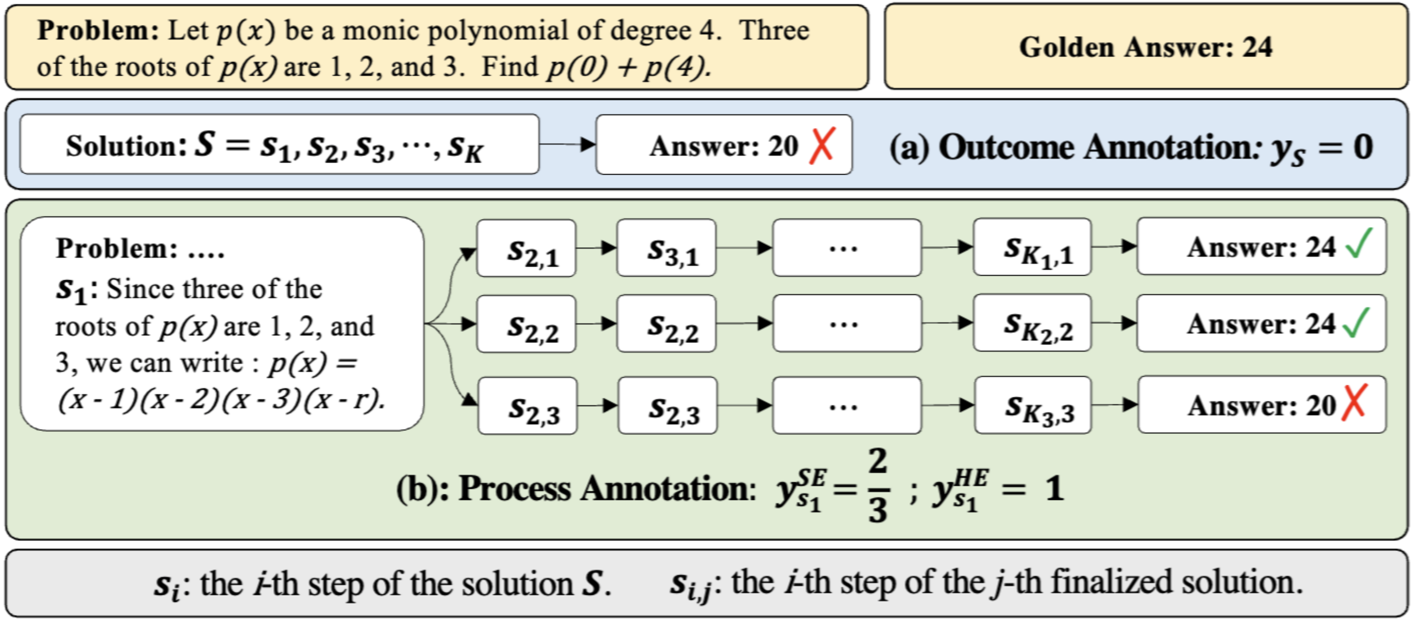


[figure credit: Sasha Rush]

* Still some folks chose to do human annotations! [Uesato et al., 2022, Lightman et al., 2023]

Bonus

Process Reward via MCTS: example



[Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations, 2023]

Bonus

Process reward via MCTS: example

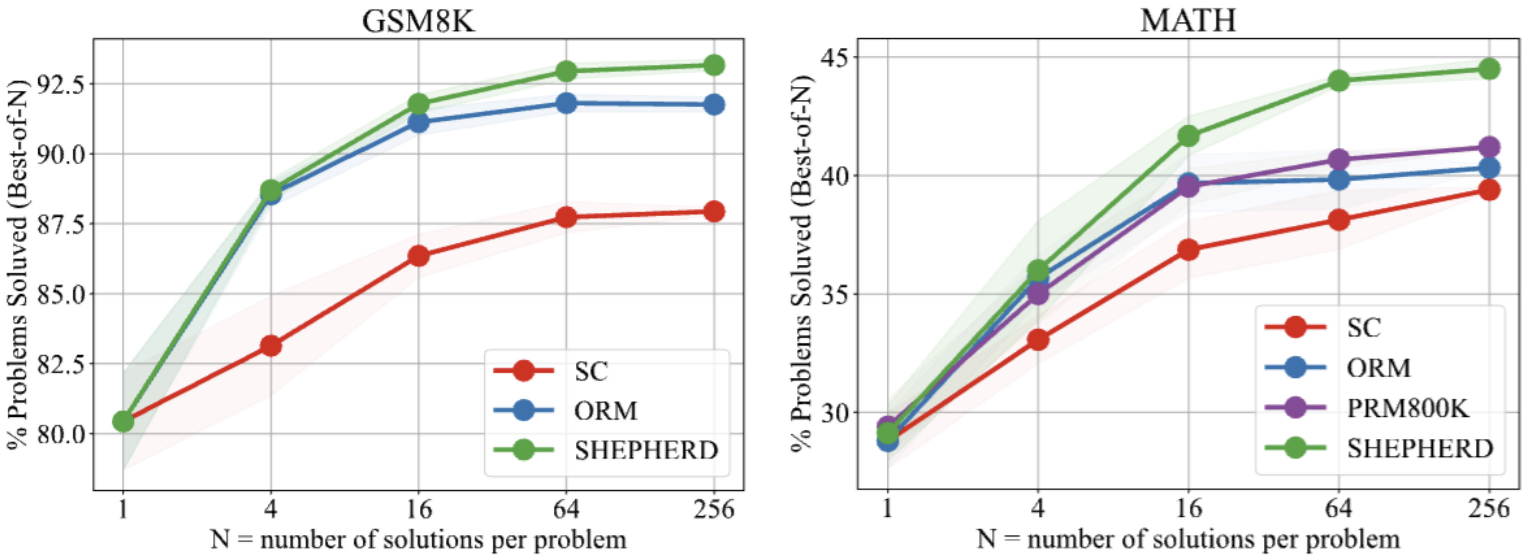


Figure 3: Performance of LLaMA2-70B using different verification strategies across different numbers of solution candidates on GSM8K and MATH.

[Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations, 2023]

Bonus

Example: AlphaZero

- Goal is to train:
  - a policy (the decision maker)
  - a value function (estimates the expected outcome of the game from that state)
- Self-play using guided-search with exploration
  - The exploration is through MCTS.
  - The search explores moves based on visit counts and value estimates.
  - The final move is selected based on these MCTS evaluations.
- Label final outcomes of self-play games
- Train the policy and the value function

Figure source

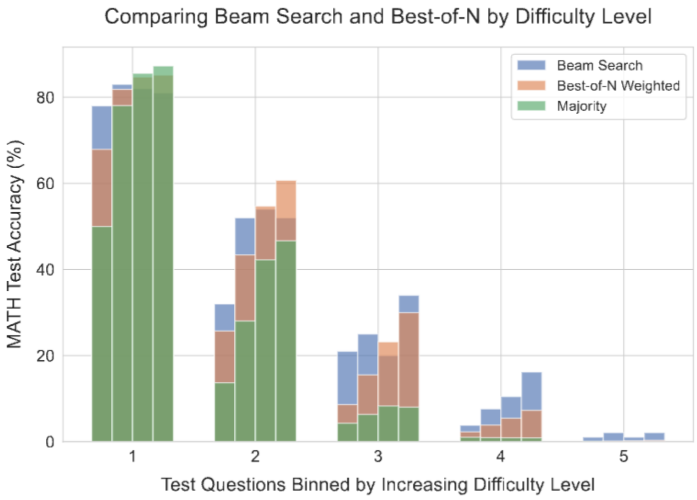
Candidate moves?

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017

Bonus

Which approach is better? Depends

- The effectiveness of different approaches to scaling test-time compute critically varies depending on the difficulty of the prompt/task.
- Figure: Comparing beam search and best-of-N binned by difficulty level.
  - The four bars correspond to increasing test-time compute budgets (4, 16, 64, and 256 generations).
  - On the easier problems (bins 1 and 2), beam search shows signs of over-optimization with higher budgets, whereas best-of-N does not.
  - On the medium difficulty problems (bins 3 and 4), we see beam search demonstrating consistent improvements over best-of-N.



[Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024]

What is OpenAI doing?

Do we know what OpenAI is doing?

- We don't know the details.
- But from their blog post, we can guess:
  - Uses RL during **training**; therefore, there is *explicit training* to incentivize better inference-scaling.
  - CoT: At the **test** time, it articulates long sequence of "thoughts". Hence, no explicit search algorithm at inference.
  - Data efficient: ?
- Let's look at a few examples!

> Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the

> problem. Through reinforcement learning, o1 learns to hone its chain of thought and refine the strategies it uses. It learns to recognize and correct its mistakes. It learns to break down tricky steps into simpler ones. It learns to try a different approach when the current one isn't working. This

https://openai.com/index/learning-to-reason-with-llms/

Do we know what OpenAI is doing?

- A key design consideration is how to train a model for this:
  - Would you get humans to annotate follow up thoughts (off-policy data) or let the model annotate candidate answers and labels good/bad ones (on-policy data).
  - Seems like OpenAI went for the latter.

"When training a model for reasoning, one thing that immediately jumps to mind is to have humans write out their thought process and train on that. ***When we saw that if you train the model using RL to generate and hone its own chain of thoughts it can do even better than having humans write chains of thought for it.*** *That was the "Aha!" moment that you could really scale this."*

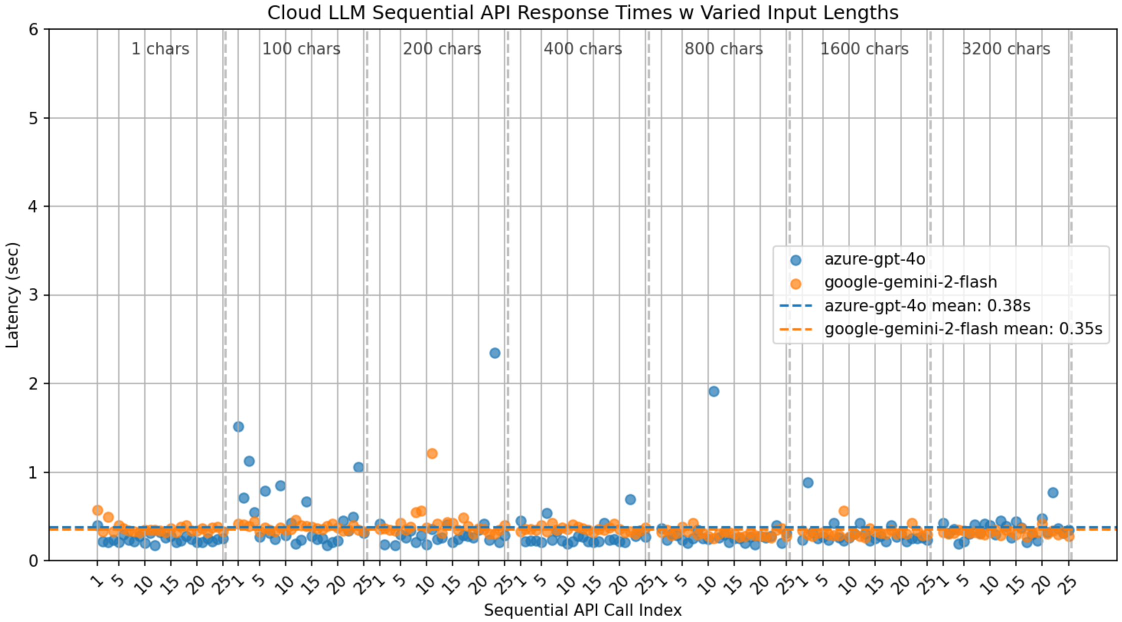
https://www.youtube.com/watch?v=tEzs3VHyBDM

Thinking about cost and ...

- Obviously, long inputs/outputs cost money.
- Generally output tokens cost a bit more than input tokens.
  - Output tokens are generated one token at a time (one forward pass per token)
  - Input tokens are parallel, but require parallel processing
- How about latency?

Thinking about cost and latency

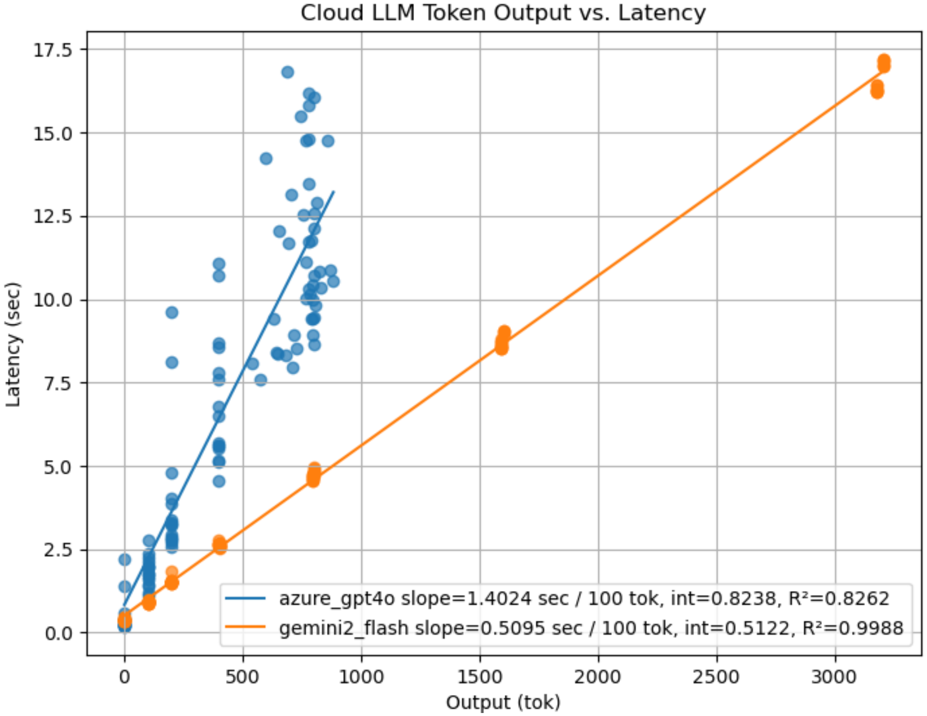
- Two cloud-based models: (1) Azure GPT4 on Azure; (2) Gemini2-flash on GCP.
- **Reading** long sequences is **quite fast**, across different input lengths (~0.5 sec)



[Results credit: Owen Bianchi]

Thinking about cost and latency

- Generating long sequences can be slow. Latency cost grows linearly in length.
  - Azure GPT4: 1.4 sec / token
  - Gemini2-flash: 0.5 sec / token



[Results credit: Owen Bianchi]

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING