Advanced Inference Strategies

Sean Welleck | CMU Advanced NLP

March 25, 2025

Recap: Training and Inference

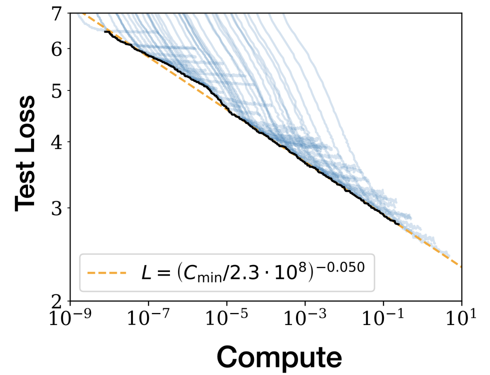
Training: use data and a loss to obtain a model $p_\theta(y|x)$:

- Pre-training (*Lecture #6*)
- Post-training
  - Fine-tuning (*Lecture #9*)
  - Reinforcement learning (*Lecture #11*)

Recap: Training and Inference

Training scaling: improve performance with larger model and dataset

(*Lecture #6*)



$$\text{Compute} \propto \text{Model size} \times \text{Data size}$$

Recap: Training and Inference

Inference: generate outputs with a model and algorithm $g(p_\theta, x)$:

- Decoding algorithms (*Lecture #7*)
  - Sampling
  - Optimization (e.g., beam search)

Recap: Training and Inference

Inference: generate outputs with a model and algorithm $g(p_\theta, x)$:

- Decoding algorithms (*Lecture #7*)
  - Sampling
  - Optimization (e.g., beam search)
- Basic prompting patterns (*Lecture #8*)
  - Chain-of-thought
  - Prompt chains

Recap: Training and Inference

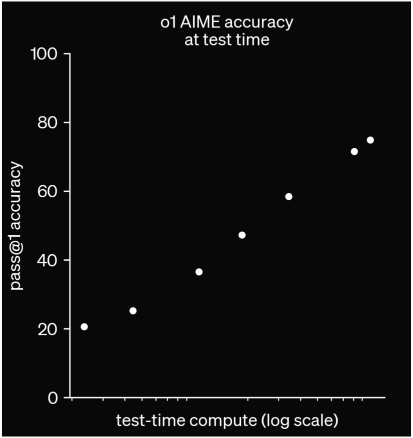
Inference: generate outputs with a model and algorithm $g(p_\theta, x)$:

- Decoding algorithms (*Lecture #7*)
  - Sampling
  - Optimization (e.g., beam search)
- Basic prompting patterns (*Lecture #8*)
  - Chain-of-thought
  - Prompt chains
- *Today: advanced inference strategies*

Training and Inference

Inference scaling: improve performance by generating more tokens

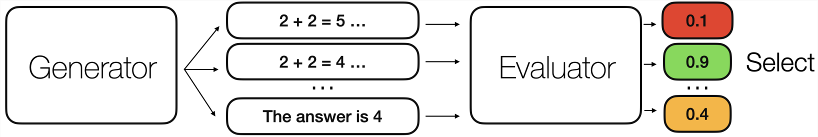
(*This lecture*)



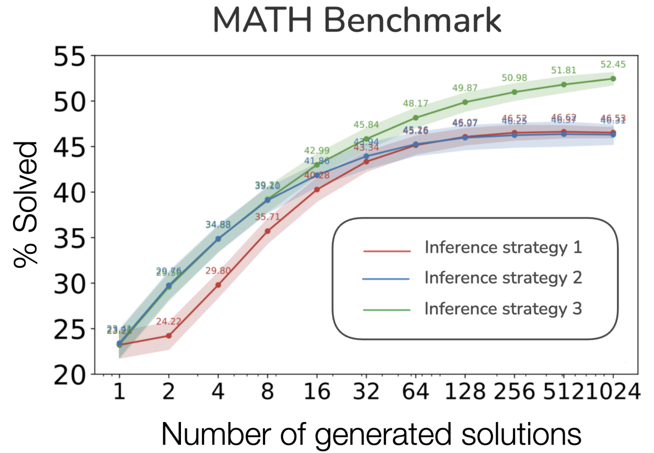
$$\text{Compute} \propto \text{Model size} \times \text{Generated tokens}$$

Advanced inference strategies

1. Generate multiple times

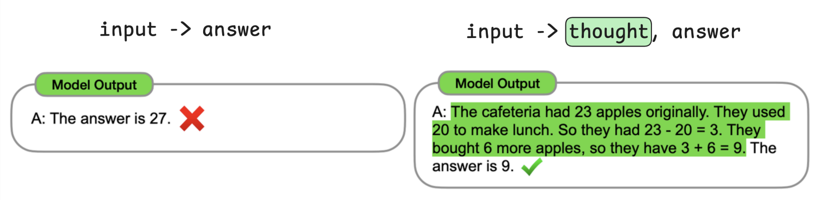


1. Generate multiple times



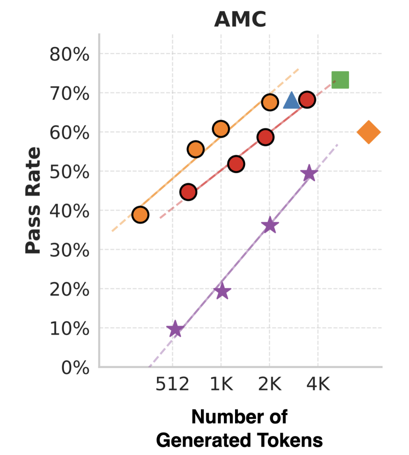
Advanced inference strategies

1. Generate multiple times
2. Generate longer outputs



Advanced inference strategies

1. Generate multiple times
2. Generate longer outputs



Today's lecture: Advanced inference strategies

1. **Part 1:** Generate multiple times
   - *Meta-generation*: chain, parallel, refinement, tree search

2. **Part 2:** Generate longer outputs
   - *Long chain-of-thought*

Today's lecture: Advanced inference strategies

1. Part 1: Generate multiple times
   - *Meta-generation*: chain, parallel, refinement, tree search

2. Part 2: Generate longer outputs
   - *Long chain-of-thought*

Recap: generation and decoding algorithms

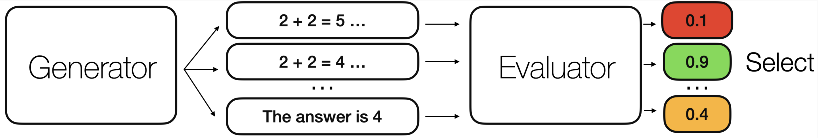
Generator: Generates a sequence with a language model.

- Example: calling an LLM API
- Decoding algorithms (*Lecture #7*)
    - Greedy decoding
    - Temperature sampling
    - ...

$$y \sim g(p_\theta, x; \phi)$$

Meta-Generation Algorithm

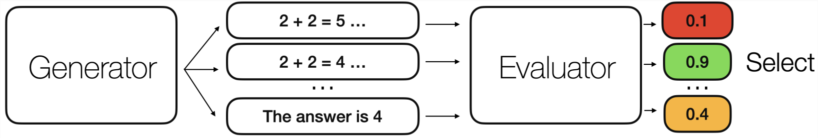
Meta-generator: Strategies for calling a generator multiple times



- Example: call API multiple times, select the best sequence with a separate model

Meta-Generation Algorithm

Meta-generator: Strategies for calling a generator multiple times

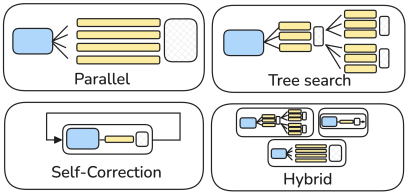


- Example: call API multiple times, select the best sequence with a separate model

$$y \sim G(x, g; \Phi)$$

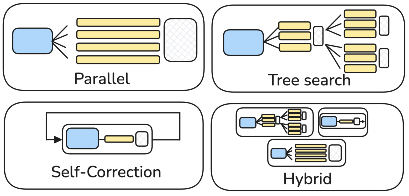
Meta-generators | outline

- Strategies
  - Parallel
  - Tree search
  - Refinement/self-correction

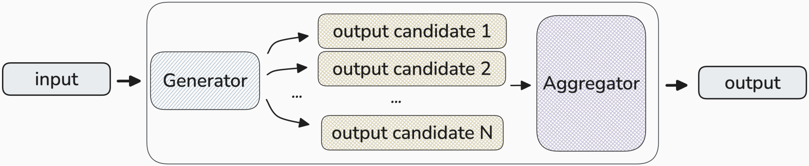


Meta-generators | outline

- Strategies
  - **Parallel**
  - Tree search
  - Refinement/self-correction



Meta-generators | parallel



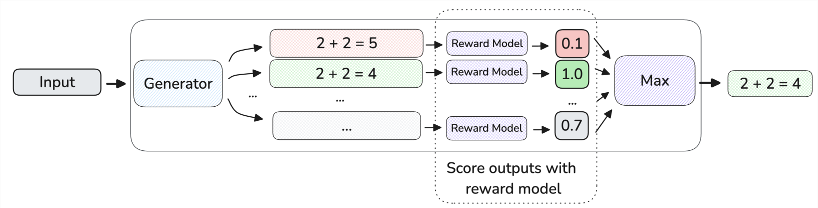
- Generate candidates:

$$\{y^{(1)}, \ldots, y^{(N)}\} \sim G(\cdot|x)$$

- Aggregate:

$$y = h(y^{(1)}, \ldots, y^{(N)})$$

Parallel | Best-of-N[1]



$$\underset{\{y^{(1)},\ldots,y^{(N)}\}}{\arg\max} \underbrace{v(y)}_{\text{reward model}}$$

[1][Stiennon et al., 2020, Nakano et al., 2022]

Parallel | Best-of-N

x:

Input:
Let $f(r) = \sum_{j=2}^{2008} \frac{1}{j^r} = \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{2008^r}$. Find $\sum_{k=2}^{\infty} f(k)$.

Llemma 34B solution:
We have

z:

$$\sum_{k=2}^{\infty} f(k) = \sum_{k=2}^{\infty}\sum_{j=2}^{2008} \frac{1}{j^k} = \sum_{j=2}^{2008}\sum_{k=2}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\sum_{k=0}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\frac{1}{1-\frac{1}{j}}$$
$$= \sum_{j=2}^{2008} \frac{1}{j(j-1)} = \sum_{j=2}^{2008}\left(\frac{1}{j-1} - \frac{1}{j}\right)$$
$$= \left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \cdots + \left(\frac{1}{2007} - \frac{1}{2008}\right)$$
$$= 1 - \frac{1}{2008}$$
$$= \boxed{\frac{2007}{2008}}.$$

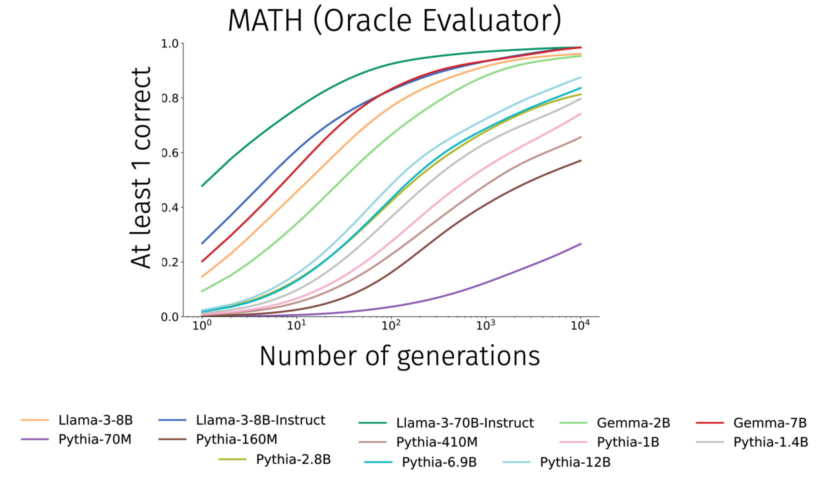
y:

Final Answer: The final answer is $\frac{2007}{2008}$.

Example: solve a math problem

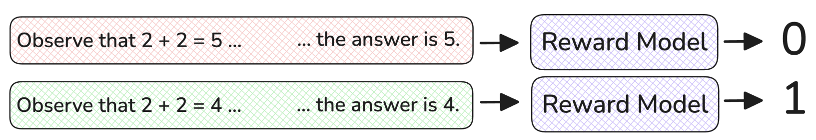
Best-of-N

What if we had a perfect reward model $v^*(y)$?



Parallel | Best-of-N

Learned reward model $v(y) \to [0, 1] \approx R(y)$:

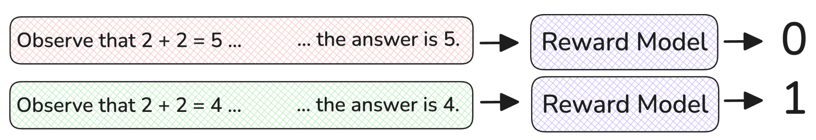


Train reward model with correct and incorrect examples.[2]

[2] E.g., [Cobbe et al., 2021]

Parallel | Best-of-N

Learned reward model $v(y) \rightarrow [0, 1] \approx R(y)$:



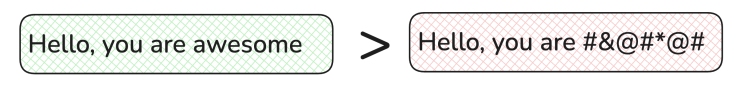
Train reward model with correct and incorrect examples.[2]

Terminology: Reward model $\approx$ evaluator $\approx$ critic $\approx$ verifier $\approx$ value $\approx$ scoring model

[2] E.g., [Cobbe et al., 2021]

Parallel | Best-of-N

Learned reward model $v(y) \rightarrow [0,1] \approx R(y)$:



Train reward model with preference data.[2]

[2] E.g., [Stiennon et al., 2020]

Parallel | Best-of-N

Why Best-of-N?

- Approximates maximum (true) reward:

$$
\begin{aligned}
\text{Best-of-}N &= \underset{y \in \{y^{(1)}, \ldots, y^{(N)}\}}{\arg\max}\ v(y) \\
&\approx \arg\max_{y} v(y) && (1) \\
&\approx \arg\max_{y} R(y) && (2)
\end{aligned}
$$

Parallel | Best-of-N

Why Best-of-N?

- Approximates maximum (true) reward:

$$
\begin{aligned}
\text{Best-of-}N &= \underset{y \in \{y^{(1)}, \ldots, y^{(N)}\}}{\arg\max} \; v(y) \\
&\approx \arg\max_{y} v(y) && (1) \\
&\approx \arg\max_{y} R(y) && (2)
\end{aligned}
$$

(1) gets better as number of generations N increases!

Parallel | Best-of-N

Why Best-of-N?

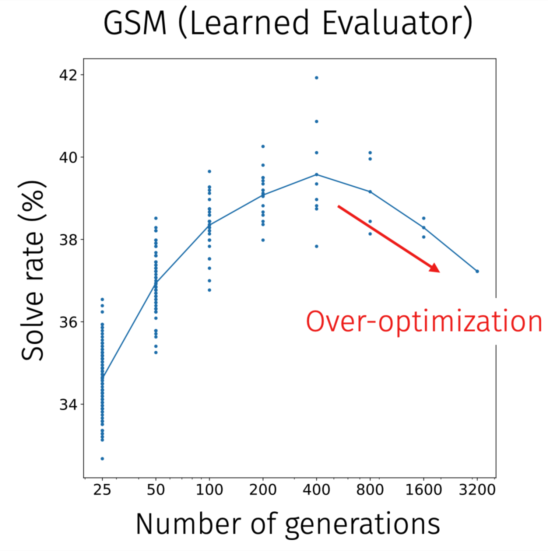
- Approximates maximum (true) reward:

$$\begin{aligned}\text{Best-of-}N &= \underset{y \in \{y^{(1)},\ldots,y^{(N)}\}}{\arg\max}\; v(y) \\ &\approx \arg\max_y v(y) \qquad (1) \\ &\approx \arg\max_y R(y) \qquad (2)\end{aligned}$$

(1) gets better as number of generations N increases!

(2) Suffers from imperfect reward model, aka "over-optimization"

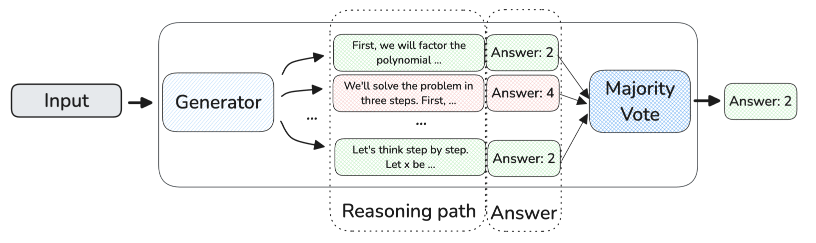
Parallel | Best-of-N[3]



[3] Plot adapted from *Training Verifiers to Solve Math Word Problems* [Cobbe et al., 2021]

Parallel | voting

Voting aggregation:[4]

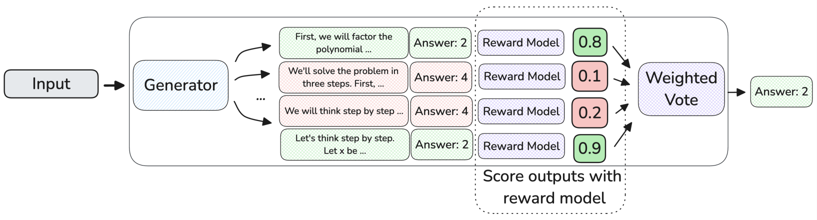


$$\arg\max_{a} \sum_{i=1}^{N} 1\{y^{(i)} = a\},$$

[4] Also called *self-consistency* [Wang et al., 2023]

Parallel | weighted voting[5]

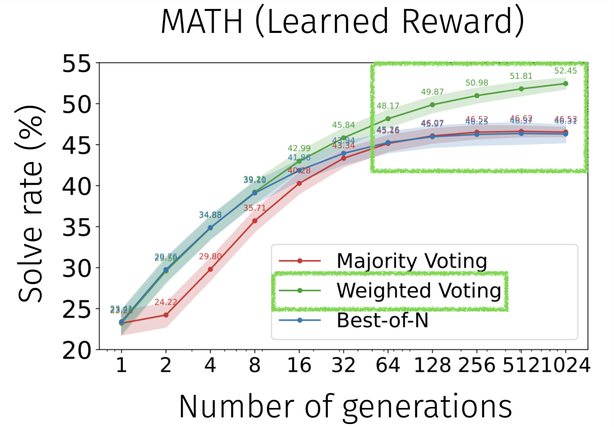
Weighted Voting:



$$\arg\max_{a} \sum_{i=1}^{N} \underbrace{v(y^{(i)})}_{\text{reward model}} \cdot \mathbb{1}\{y^{(i)} = a\},$$

[5][Li et al., 2023]

Can outperform Best-of-*N*, e.g.:[6]



[6] [Sun et al., 2024] *Easy-to-Hard Generalization: Scalable Alignment Beyond Human Supervision.*

Parallel | why (weighted) voting?

As the number of candidates $N \to \infty$, voting accuracy converges to...[7]

$$\frac{1}{M}\sum_{i=1}^{M}\mathbb{I}\left[a_i^* = \arg\max_a \underbrace{\sum_z v(x,z,a)g(z,a|x)}_{\text{“Marginalize out paths } z\text{”}}\right]$$

Notation:

- (x, z, a): (input, solution, answer)
- M: number of test examples

[7] Theorem 2, [Wu et al., 2024b] *Inference Scaling Laws.* Y. Wu, Z. Sun, S. Li, S. Welleck, Y. Yang.

Parallel | why (weighted) voting?

As the number of candidates $N \to \infty$, voting accuracy converges to...[7]

$$\frac{1}{M}\sum_{i=1}^{M}\mathbb{I}\left[a_i^* = \arg\max_a \underbrace{\sum_z v(x,z,a)g(z,a|x)}_{\text{“Marginalize out paths } z\text{”}}\right]$$

Takeaway 1: Will accuracy keep improving with more samples?

- **No**, it eventually converges to the accuracy shown above

[7] Theorem 2, [Wu et al., 2024b] *Inference Scaling Laws.* Y. Wu, Z. Sun, S. Li, S. Welleck, Y. Yang.

Parallel | why (weighted) voting?

As the number of candidates $N \to \infty$, voting accuracy converges to...[7]

$$\frac{1}{M}\sum_{i=1}^{M}\mathbb{I}\left[a_i^* = \arg\max_a \underbrace{\sum_z v(x,z,a)g(z,a|x)}_{\text{“Marginalize out paths } z\text{”}}\right]$$

Takeaway 2: When is weighted voting better than voting?

- When $v \cdot g$ assigns more total mass to correct answers than g

[7] Theorem 2, [Wu et al., 2024b] *Inference Scaling Laws.* Y. Wu, Z. Sun, S. Li, S. Welleck, Y. Yang.

Parallel | why (weighted) voting?

As the number of candidates $N \to \infty$, voting accuracy converges to...[7]

$$\frac{1}{M}\sum_{i=1}^{M}\mathbb{I}\left[a_i^* = \arg\max_a \underbrace{\sum_z v(x,z,a)g(z,a|x)}_{\text{“Marginalize out paths } z\text{”}}\right]$$

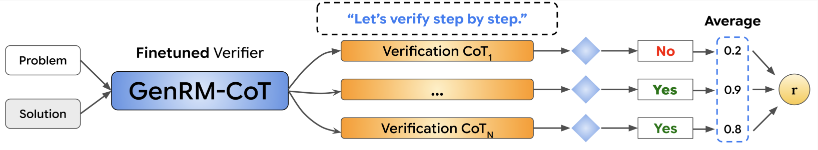
Takeaway 3: How do we improve performance further?

- Improve the reward model v
- Improve the generator g (better model and/or better algorithm)

[7] Theorem 2, [Wu et al., 2024b] *Inference Scaling Laws.* Y. Wu, Z. Sun, S. Li, S. Welleck, Y. Yang.

Parallel

Improve the reward model:



Parallel generation *in the reward model too*[8]

Active area of research!

[8] [Zhang et al., 2024]

Meta-generation strategies | parallel

Parallel

- Explores output space by generating full sequences
- Large performance gains in practice
- Bounded by the quality of the evaluator and generator

Parallel

- Explores output space by generating full sequences
- Large performance gains in practice
- Bounded by the quality of the evaluator and generator

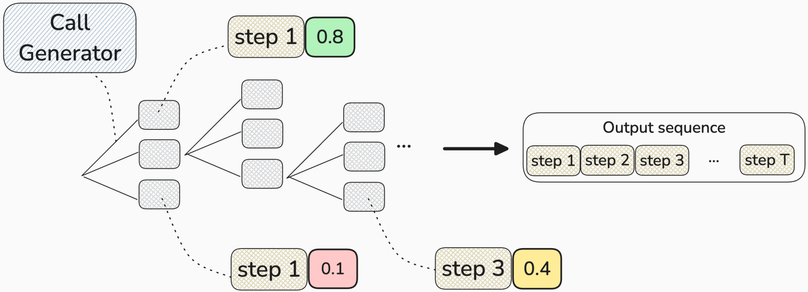
Insight: only uses the verifier at the end (on full sequences)

- *Next:* Can we better leverage *intermediate* evaluation?

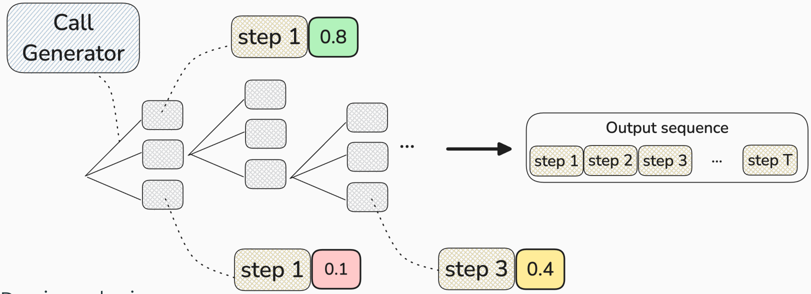
Strategies

- Strategies
  - Parallel
  - <u>Tree search</u>
  - Refinement

Tree search | basic idea



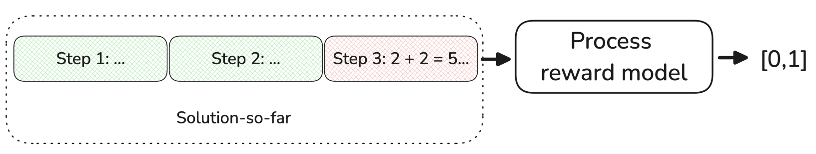
Tree search | basic idea



Design choices:

- States s
- Transitions $s \to s'$
- Scores $v(s)$
- Strategy (breadth-first, depth-first, ...)

1. Scores: "process reward model (PRM)"[9]

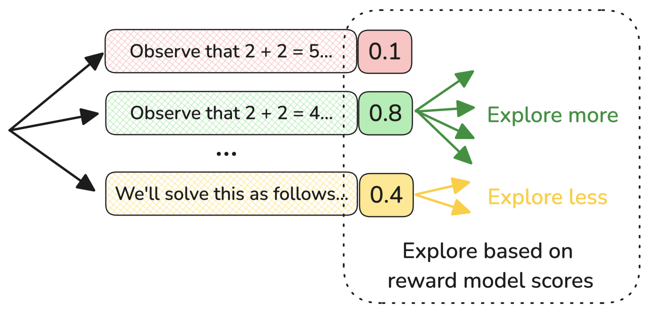


$$v(x, s_1, s_2, \ldots, s_t) \rightarrow [0, 1]$$

[9] [Uesato et al., 2022, Lightman et al., 2024, Wang et al., 2024a]

Tree search | example (REBASE)

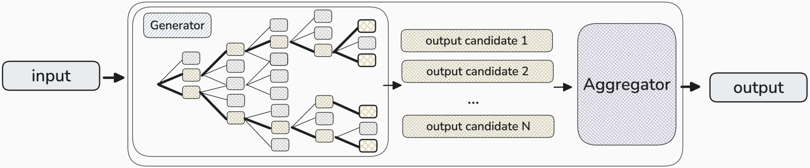
2. *Reward Balanced Search (Rebase)*[10]



$$\text{explore}_i = \text{Round}\left(\text{Budget}\frac{\exp\left(v(s_i)/\tau\right)}{\sum_j \exp\left(v(s_j)/\tau\right)}\right), \tag{3}$$

[10] [Wu et al., 2024b] *Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference.*

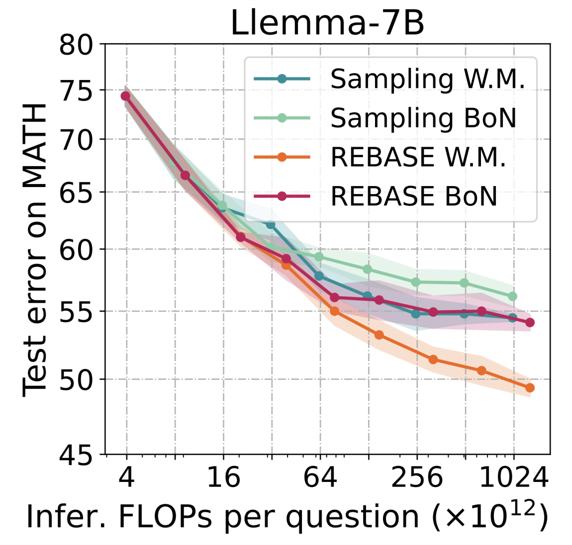
Tree search | example



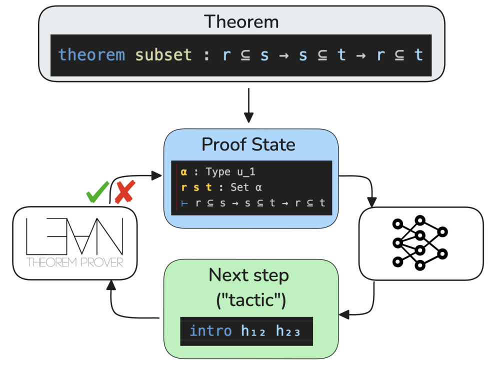
Run tree search to get candidates for aggregation (e.g., voting).

- **Key idea:** Leverages scores on *intermediate* states
  - Backtracking
  - Exploration

Tree search | examples[11]

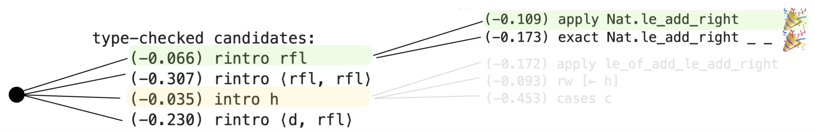


[11] [Wu et al., 2024b] *Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference.*



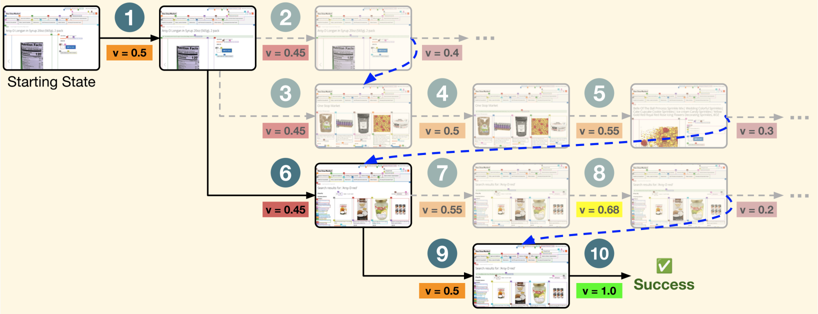
Formal theorem proving [Polu and Sutskever, 2020]

Tree search | examples



Best-first search in formal theorem proving

Tree search | examples



Best-first search in web agents [Koh et al., 2024]

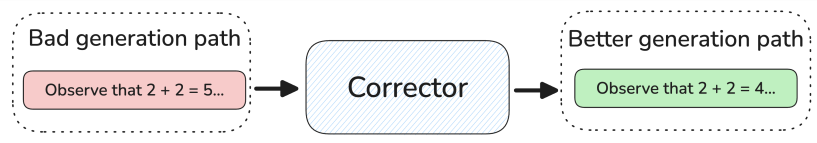
Tree-search

- Can backtrack and explore using intermediate scores
- Requires a suitable environment and value function
  - Decomposition into states
  - Good reward signal

Strategies

- Strategies
  - Parallel
  - Tree search
  - Refinement

Refinement / self-correction

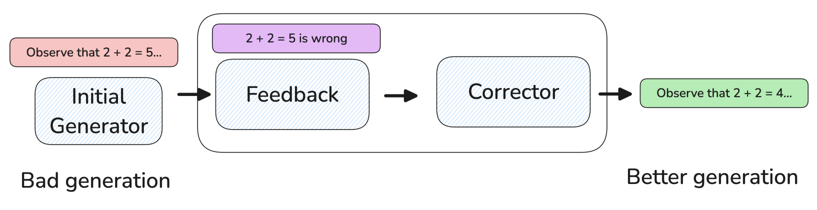


Improve a generation

Repeat:

- $y^{(i+1)} \sim g(x, y^{(i)})$

Refinement / self-correction

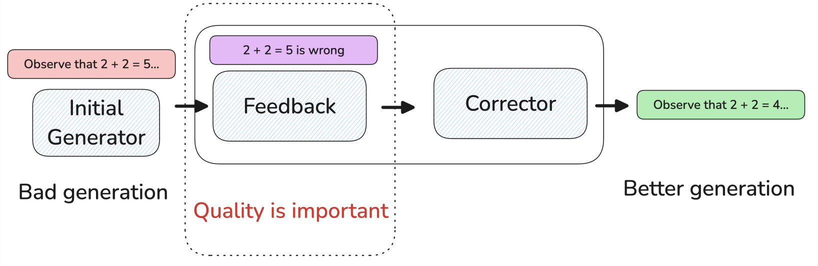


Improve a generation using feedback

Repeat:

- $y^{(i+1)} \sim g(x, y^{(i)}, F(y^{(i)}))$

Refinement / self-correction

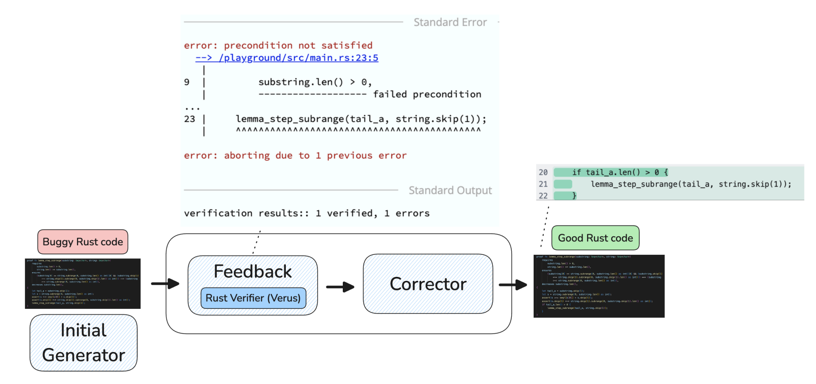


Improve a generation using feedback

In practice, the quality and source of feedback is crucial:

- **Extrinsic**: external information at inference time
- **Intrinsic**: no external information at inference time

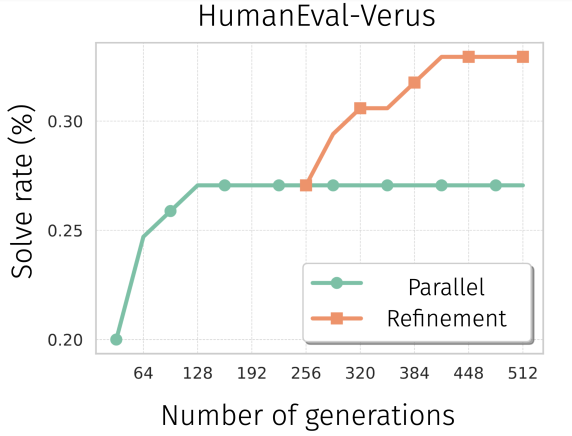
1. **Extrinsic**: external feedback



Feedback: external program verifier[12]

[12] [Aggarwal et al., 2024], *AlphaVerus*. P. Aggarwal, B. Parno, S. Welleck.

1. **Extrinsic**: external feedback



AlphaVerus. P. Aggarwal, B. Parno, S. Welleck.

Meta-generators | refinement | extrinsic

1. **Extrinsic**: external feedback

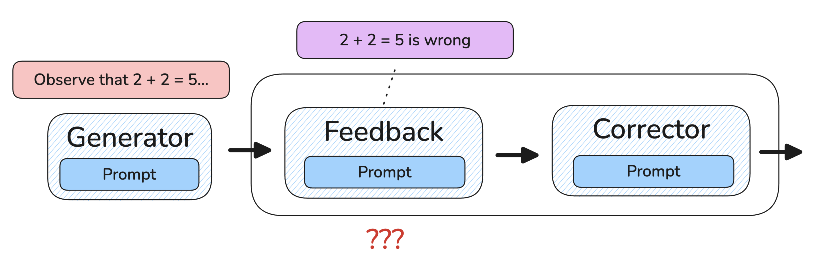
Several success cases:

- Verifiers [Aggarwal et al., 2024]
- Code interpreters [Chen et al., 2024]
- Retrievers [Asai et al., 2024]
- Tools + agent environment[12]
- ...

Intuition: adds new information, can detect and localize errors

[12] https://x.com/gneubig/status/1866172948991615177

2. **Intrinsic**: Re-prompt the same model:



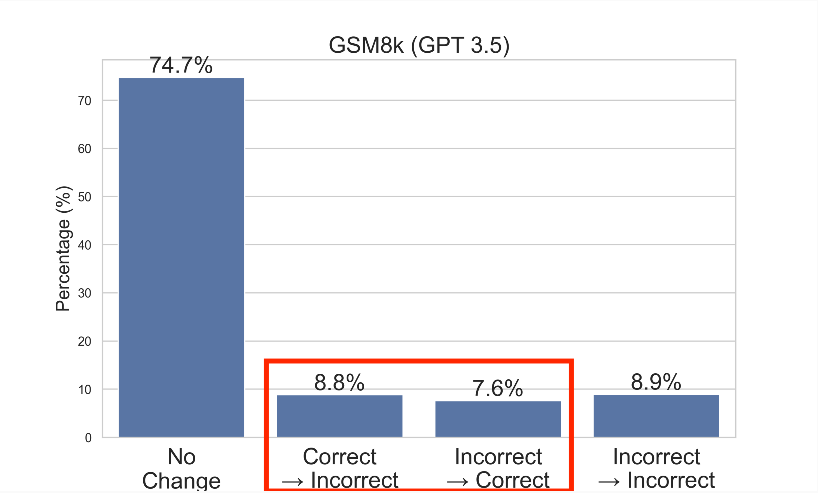
Re-prompt a single LLM, e.g. [Madaan et al., 2023]

Mixed results:

- Easy to evaluate tasks: positive [Wang et al., 2024b]
    - E.g., missing info [Asai et al., 2024]
- Mathematical reasoning: mixed[13]

[13] E.g., [Huang et al., 2024] *Large Language Models Cannot Self-Correct Reasoning Yet*

Meta-generators | refinement | intrinsic

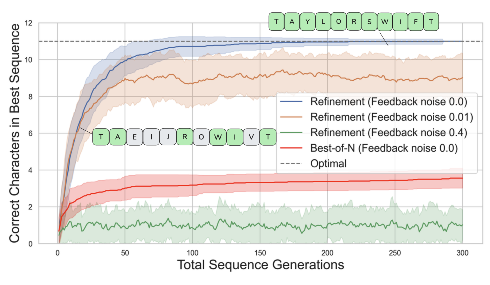


Takeaway: feedback is too noisy From [Huang et al., 2024]

Meta-generators | refinement

Generate "TAYLORSWIFT"

- Generator:
  - p(character)
- Feedback:
  - Incorrect characters
- Corrector:
  - Regenerate incorrect



Meta-generators | refinement / self-correction

Refinement / self-correction

- Extrinsic
    - Positive results for environments that detect or localize errors
- Intrinsic
    - Mixed results, depends on difficulty of verification

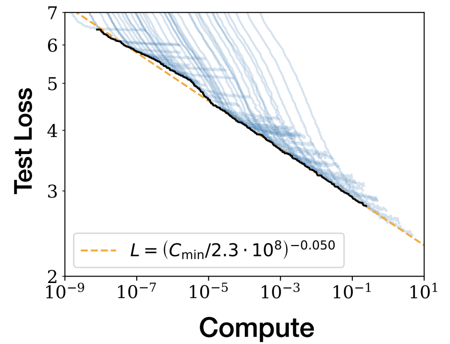
Meta-generators | outline

- Strategies
  - Parallel
  - Tree search
  - Refinement
- *Inference scaling laws*

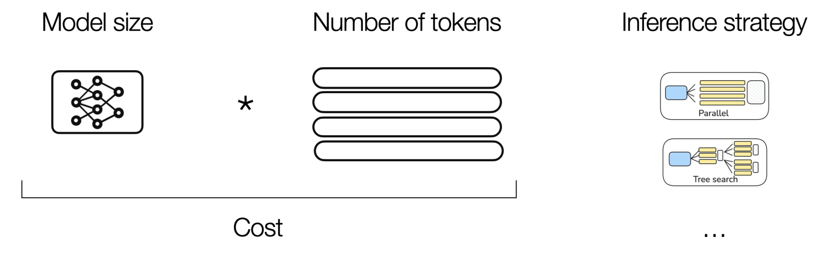
Pre-training scaling laws

Recap: **pre-training** scaling laws (*Lecture #6*)

1. (Model size, # training tokens): blue
2. **Compute optimal**: black
3. **Scaling law**: orange

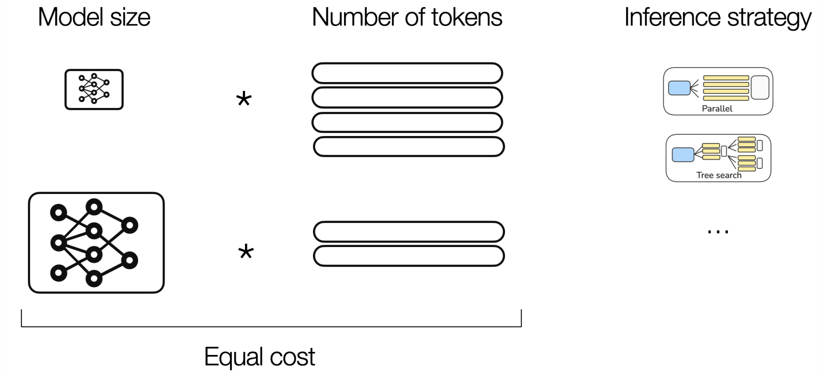


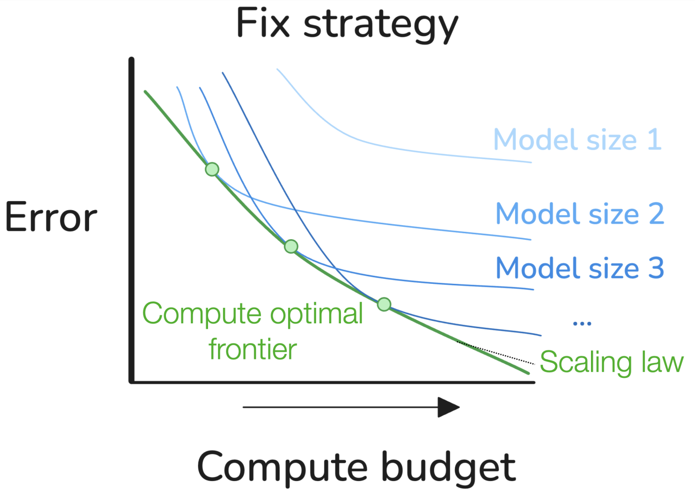
Compute is a function of model size and number of generated tokens



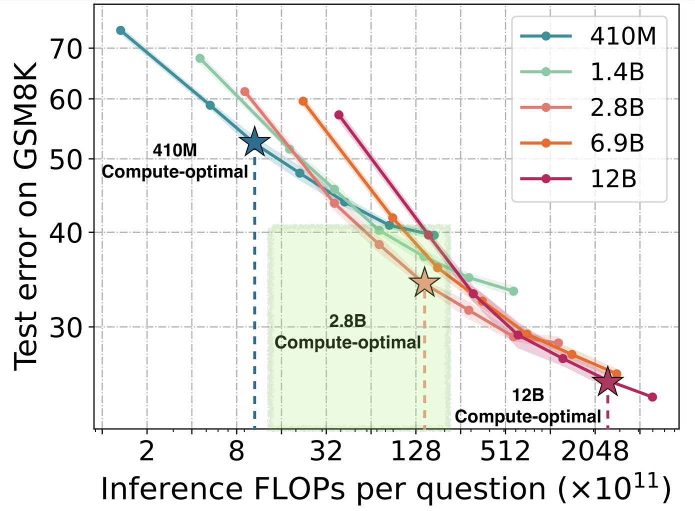
Meta-generation | inference scaling laws

We can choose to increase model size or number of tokens




Fix strategy
Error
Model size 1
Model size 2
Model size 3
...
Compute optimal frontier
Scaling law
Compute budget

Meta-generation | how do we choose a meta-generator?



Using a smaller model and generating more is often best [Wu et al., 2024b].

Inference scaling laws

Designing better strategies

- Example: design a better tree search [Wu et al., 2024b]
- Example: select inference strategy based on problem difficulty [Snell et al., 2024]

- When allocated optimally, performance improves with compute
- Best model size and strategy varies with the budget
    - Sometimes smaller models are better!

Meta-generation | recap

- Strategies for generating multiple sequences
- Parallel, tree search, refinement
- Choose methods based on task performance and cost

Today's lecture

1. Part 1: Generating multiple sequences
2. **Part 2: Generating a single long sequence**
    - *Long chain-of-thought*

Basic idea

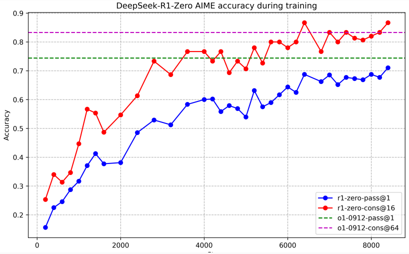
- Train a model to generate a “thought” prior to a final output

$$p_\theta(\underbrace{y}_{\text{“output”}}, \underbrace{z}_{\text{“thought”}} | x)$$

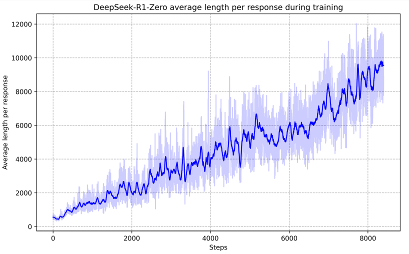
- At inference time, just sample a thought+output
- In principle, the model can learn to try alternatives, perform refinement, backtrack **within the thought**

Training for long chain-of-thought

- Approach 1: reinforcement learning
    - Policy: given a math problem x, generate a thought + answer
    - Reward: is the answer correct



Accuracy improves during training



Response length increases to > 10,000

Training for long chain-of-thought

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$

$\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2.$

Rearrange to isolate the inner square root term:

$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$

We started with the equation:

$\sqrt{a - \sqrt{a+x}} = x$

First, let's square both sides:

$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$

Next, I could square both sides again, treating the equation: $\cdots$

...

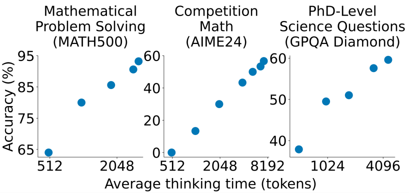
Example response

Long chain-of-thought patterns Credits: Weihua Du

1. Uncertainty
    - *Wait... / Hold on...*
    - *Wait–actually, does this formula apply here?*
2. Branching, backtracking, retrying
    - *Alternatively, generating functions could model this problem...*
    - *Revisiting...*
    - *Wait, I'm overthinking. Let's try again...*
3. Verification
    - *Let's check if we made an error. We should verify...*
    - *This is a contradiction, so we must have made a mistake.*
    - *Let's test this with...*
4. Key Points
    - *Key takeaway... / It's worth noting...*
5. Clarification
    - *In other words... / To clarify...*
6. Synthesis
    - *Ultimately... / Putting it all together...*

Controlling the length: budget forcing [Muennighoff et al., 2025]

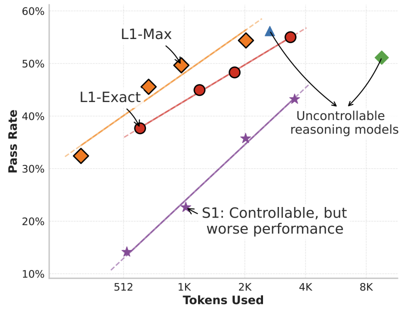
- Adhere to a length budget by forcing the model to generate "Wait" or "Final answer"
- Trade off tokens and performance



[Muennighoff et al., 2025]

Controlling the length: L1 [Aggarwal and Welleck, 2025]

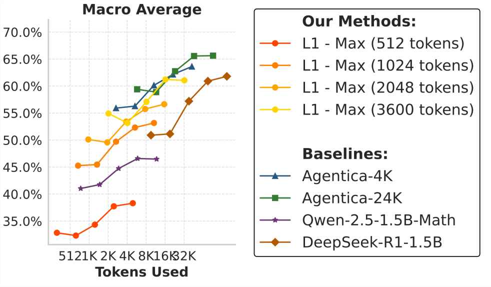
- Train model with reinforcement learning to adhere to length constraints
- E.g. "use up to 2000 tokens" provided in the prompt
- Reward: correctness and length constraint penalty



[Aggarwal and Welleck, 2025]

Long chain-of-thought | sequential vs. parallel

- **Sequential**: long chain-of-thought
- **Parallel**: majority voting (multiple long COTs)



[Aggarwal and Welleck, 2025]

Part 2 recap

- Train a model to generate a long sequence, then use a simple inference algorithm
- Internally can perform backtracking, self-correction, etc.
- Emerging area of research!

1. Inference strategies take a trained model and improve performance by:
    - Generating tokens according to a strategy
    - Incorporate external information
        - Reward models
        - Environment feedback

Very active and evolving research area!

Today's lecture | recap

1. Inference strategies take a trained model and improve performance by:
    - Generating tokens according to a strategy
    - Incorporate external information
        - Reward models
        - Environment feedback
2. Two complementary strategies
    - Call a generator multiple times
        - Meta-generation: parallel, tree search, refinement
    - Call a generator once to generate a long output
        - Long chain-of-thought

Very active and evolving research area!

Appendix

Meta-generators | parallel | pairwise

Pairwise: *Minimum Bayes Risk*

$$\mathrm{MBR}(g, v, N) = \underset{y \in \{y^{(1)}, \ldots, y^{(N)}\}}{\arg\max} \underbrace{\frac{1}{N} \sum_{i=1}^{N} v(y, y^{(i)})}_{\approx \mathbb{E}_{y' \sim p}[v(y, y')]},$$

where $\{y^{(1)}, \ldots, y^{(N)}\} \sim g$ and $v(y, y')$ is a “utility” function.

Meta-generators | parallel | pairwise

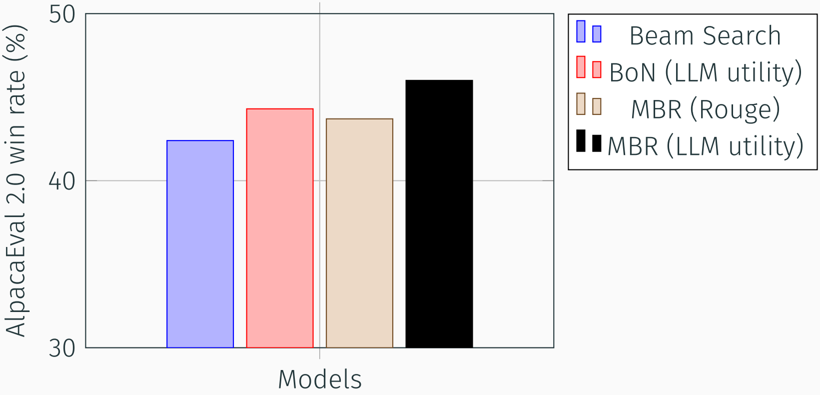
Pairwise: *Minimum Bayes Risk*

$$\mathrm{MBR}(g, v, N) = \underset{y \in \{y^{(1)}, \ldots, y^{(N)}\}}{\arg\max} \underbrace{\frac{1}{N} \sum_{i=1}^{N} v(y, y^{(i)})}_{\approx \mathbb{E}_{y' \sim p}[v(y, y')]},$$

where $\{y^{(1)}, \ldots, y^{(N)}\} \sim g$ and $v(y, y')$ is a “utility” function.

Intuitively, selects the candidate with the highest “consensus” utility.

Utility: $LLM(y, y^{(i)}) \rightarrow \{1, 2, 3, 4, 5\}$:



[13] Example from [Wu et al., 2024a] (Llama 3 70B). Utility: Prometheus 2 [Kim et al., 2024].

Weighted voting is an instance of Minimum Bayes Risk:[14]

$$\underbrace{v(y, y^{(i)})}_{\text{utility}} = \underbrace{\mathbb{1}\left[a = a^{(i)}\right]}_{\text{same answer}} \cdot \underbrace{v(y^{(i)})}_{\text{sequence score}}, \tag{4}$$

where $y = (z, a)$, $y^{(i)} = (z^{(i)}, a^{(i)})$.

[14] [Bertsch et al., 2023] *It's MBR All the Way Down: Modern Generation Techniques Through the Lens of Minimum Bayes Risk. A. Bertsch, A. Xie, G. Neubig, M. Gormley.*

References i

- Aggarwal, P., Parno, B., and Welleck, S. (2024).
  Alphaverus: Bootstrapping formally verified code generation through self-improving translation and treefinement.
  https://arxiv.org/abs/2412.06176.
- Aggarwal, P. and Welleck, S. (2025).
  L1: Controlling how long a reasoning model thinks with reinforcement learning.
  https://arxiv.org/abs/2503.04697.

Asai, A., He⋆, J., Shao⋆, R., Shi, W., Singh, A., Chang, J. C., Lo, K., Soldaini, L., Feldman, Tian, S., Mike, D., Wadden, D., Latzke, M., Minyang, Ji, P., Liu, S., Tong, H., Wu, B., Xiong, Y., Zettlemoyer, L., Weld, D., Neubig, G., Downey, D., Yih, W.-t., Koh, P. W., and Hajishirzi, H. (2024).
OpenScholar: Synthesizing scientific literature with retrieval-augmented language models.
Arxiv.

Bertsch, A., Xie, A., Neubig, G., and Gormley, M. (2023).
It's MBR all the way down: Modern generation techniques through the lens of minimum Bayes risk.
In Elazar, Y., Ettinger, A., Kassner, N., Ruder, S., and A. Smith, N., editors, *Proceedings of the Big Picture Workshop*, pages 108–122, Singapore. Association for Computational Linguistics.

Chen, X., Lin, M., Schärli, N., and Zhou, D. (2024).
Teaching large language models to self-debug.
In *The Twelfth International Conference on Learning Representations.*

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021).
Training verifiers to solve math word problems.
https://arxiv.org/abs/2110.14168.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. (2024).
Large language models cannot self-correct reasoning yet.
In *The Twelfth International Conference on Learning Representations.*

Kim, S., Suk, J., Longpre, S., Lin, B. Y., Shin, J., Welleck, S., Neubig, G., Lee, M., Lee, K., and Seo, M. (2024).
Prometheus 2: An open source language model specialized in evaluating other language models.
https://arxiv.org/abs/2405.01535.

Koh, J. Y., McAleer, S., Fried, D., and Salakhutdinov, R. (2024).
Tree search for language model agents.
arXiv preprint arXiv:2407.01476.

Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G., and Chen, W. (2023).
Making language models better reasoners with step-aware verifier.
In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors,
Proceedings of the 61st Annual Meeting of the Association for

Computational Linguistics (Volume 1: Long Papers), pages 5315–5333, Toronto, Canada. Association for Computational Linguistics.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. (2024).
Let's verify step by step.
In *The Twelfth International Conference on Learning Representations*.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. (2023).
Self-refine: Iterative refinement with self-feedback.

In *Thirty-seventh Conference on Neural Information Processing Systems*.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. (2025).
s1: Simple test-time scaling.
https://arxiv.org/abs/2501.19393.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. (2022).
Webgpt: Browser-assisted question-answering with human feedback.
https://arxiv.org/abs/2112.09332.

Polu, S. and Sutskever, I. (2020).
Generative language modeling for automated theorem proving.

Snell, C., Lee, J., Xu, K., and Kumar, A. (2024).
Scaling llm test-time compute optimally can be more effective than scaling model parameters.
https://arxiv.org/abs/2408.03314.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. (2020).
Learning to summarize with human feedback.
In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

Sun, Z., Yu, L., Shen, Y., Liu, W., Yang, Y., Welleck, S., and Gan, C. (2024).
Easy-to-hard generalization: Scalable alignment beyond human supervision.
In *The Thirty-eighth Annual Conference on Neural Information Processing Systems.*

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. (2022).
Solving math word problems with process- and outcome-based feedback.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. (2024a).
Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations.
In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. (2023).
Self-consistency improves chain of thought reasoning in language models.
In *The Eleventh International Conference on Learning Representations*.

Wang, Y., Wu, Y., Wei, Z., Jegelka, S., and Wang, Y. (2024b).
A theoretical understanding of self-correction through in-context alignment.
https://arxiv.org/abs/2405.18634.

Wu, I., Fernandes, P., Bertsch, A., Kim, S., Pakazad, S., and Neubig, G. (2024a).
Better instruction-following through minimum bayes risk.
https://arxiv.org/abs/2410.02902.

Wu, Y., Sun, Z., Li, S., Welleck, S., and Yang, Y. (2024b).
Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models.
https://arxiv.org/abs/2408.00724.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. (2024).
Generative verifiers: Reward modeling as next-token prediction.