Scaling Test-Time Compute With Reasoning Models

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Advances in Large Language Models

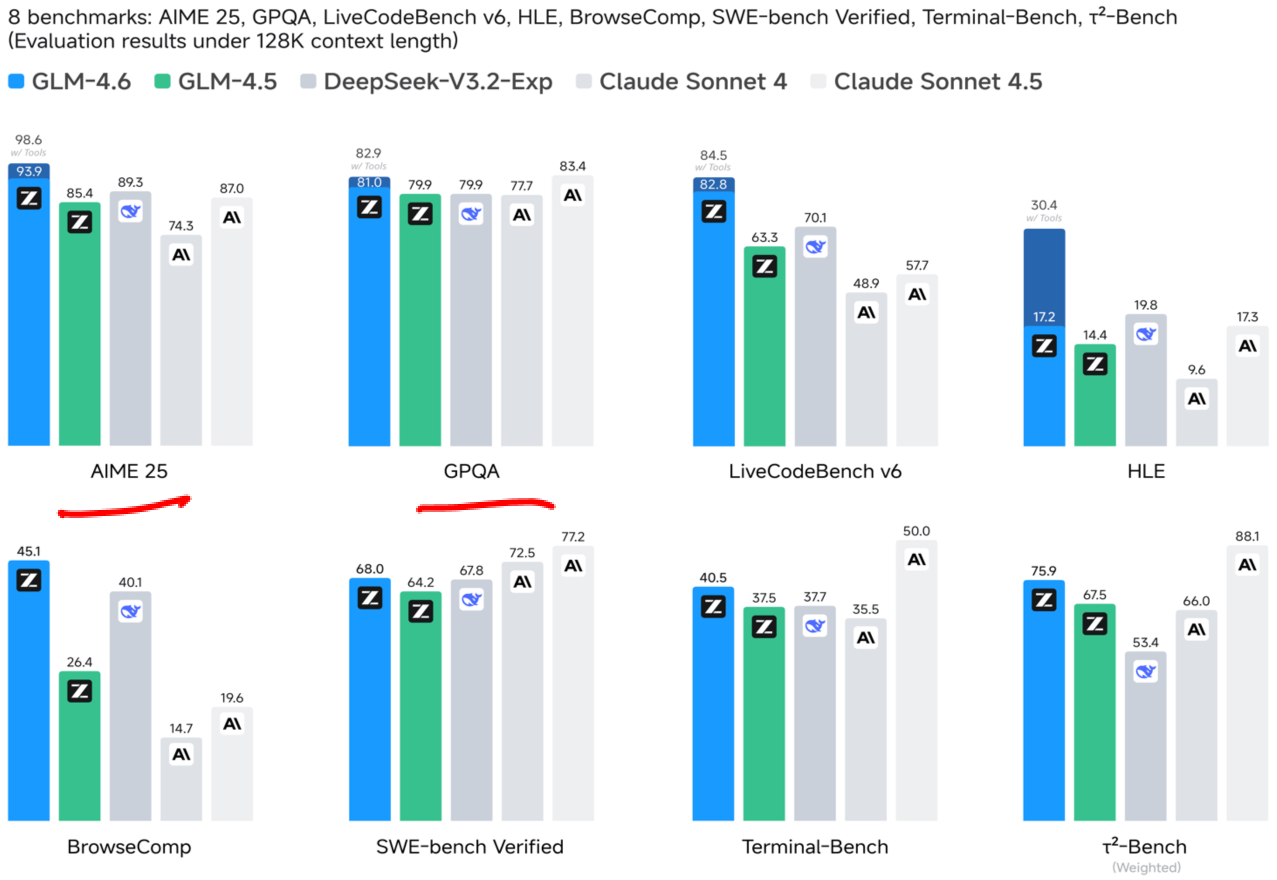
GLM-4.6V

Z.AI's new flagship model with Advanced Agentic, Reasoning and Coding Capabilities

Announced on October 1, 2025

Z.AI Blog

GLM-4.6V delivers major upgrades over the previous model GLM-4.5V. It now supports a 200K-token context, enhanced coding and reasoning abilities, stronger agentic tool use, and more natural, human-aligned writing.



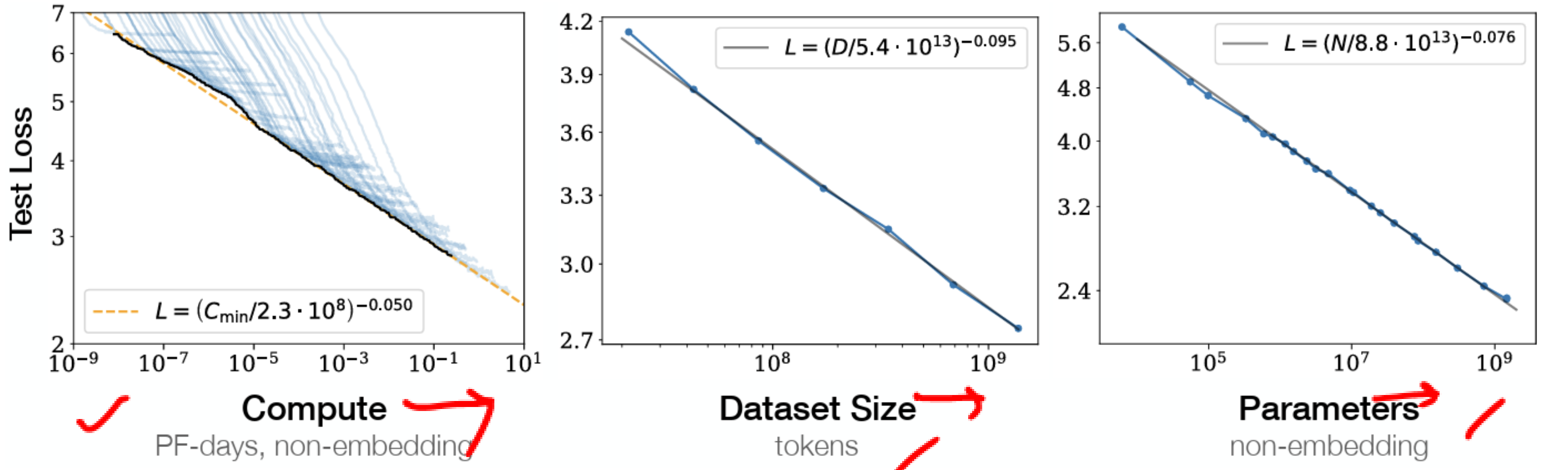
GLM-4.6V shows clear gains over GLM-4.5V across eight benchmarks, outperforming models like DeepSeek-V3.2-Exp and Claude Sonnet 4, though still slightly behind Claude Sonnet 4.5 in coding.

How Do We Scale LLMs?

A trivial question in modern deep learning: How can we make LLMs (or any deep learning models) better performing?

Can we just make them deeper, larger and pre-train on larger corpus, will it be sufficient?

Yes! Kaplan et al., showed that test performance follows a power-law governing model size and training data size.



$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

Loss decreases with higher N (parameters), D (data size).

How Do We Scale LLMs? Compute-Optimality

Hoffman et al., introduced the concept of compute-optimality, *i.e.,* for a fixed compute, C, there is an optimal N and D to train on.

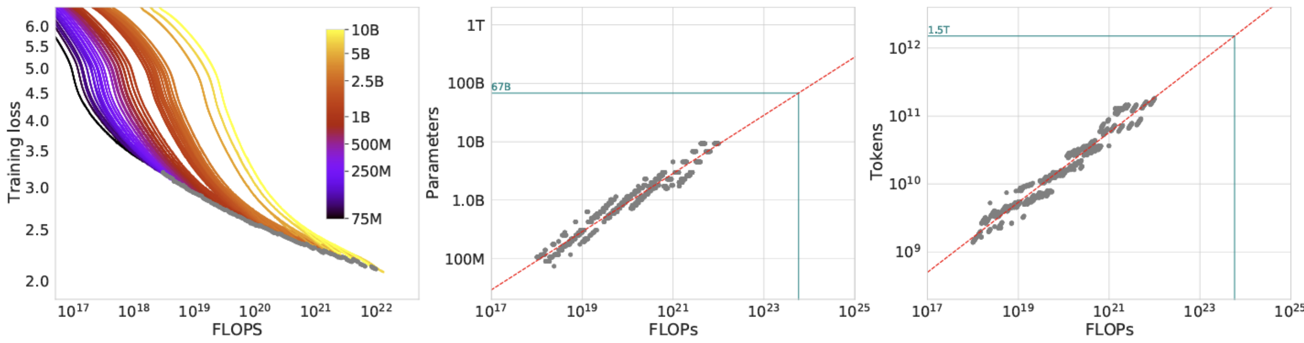


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}.$$

Under the condition: $\text{FLOPs}(N, D) \approx 6ND$

Therefore, undertraining (low D) or overtraining (high D) leads to inefficient usage of compute

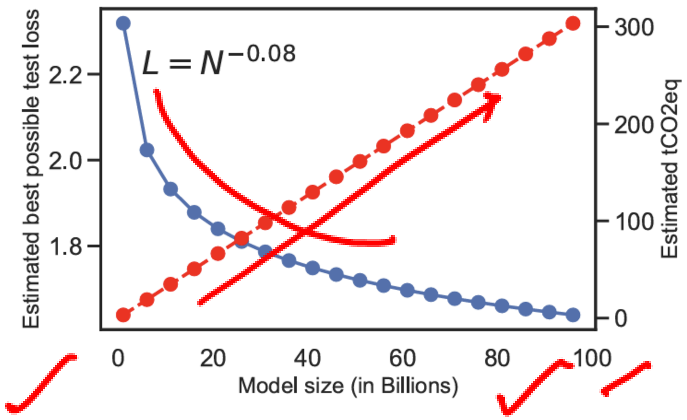
Following the concept of compute-optimality, 70B-Chinchilla model beats 280B-Gopher model, albeit using same FLOPs.

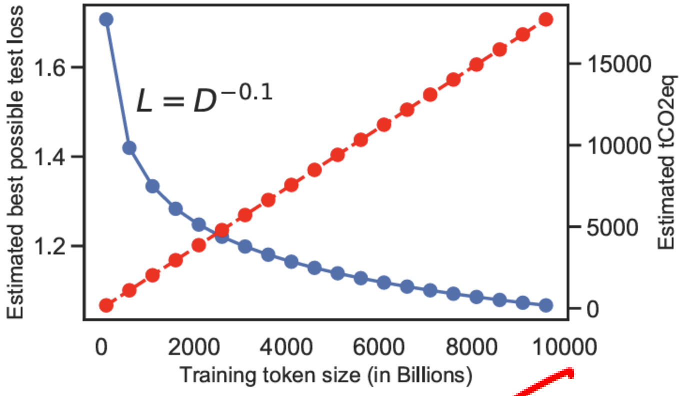
Training Scaling Isn't Sufficient?

If we can improve LLMs by optimal pre-training, then what's the problem?

Three major limitations with training-time scaling

- Compute and training data are limited. Can't increase forever.
- Even if we duplicate data and create synthetic datasets, important assumptions like i.i.d gets violated for scaling laws
- Performance improves logarithmically with N and D, but environmental factors (like CO2 emission, energy consumption) increase linearly.



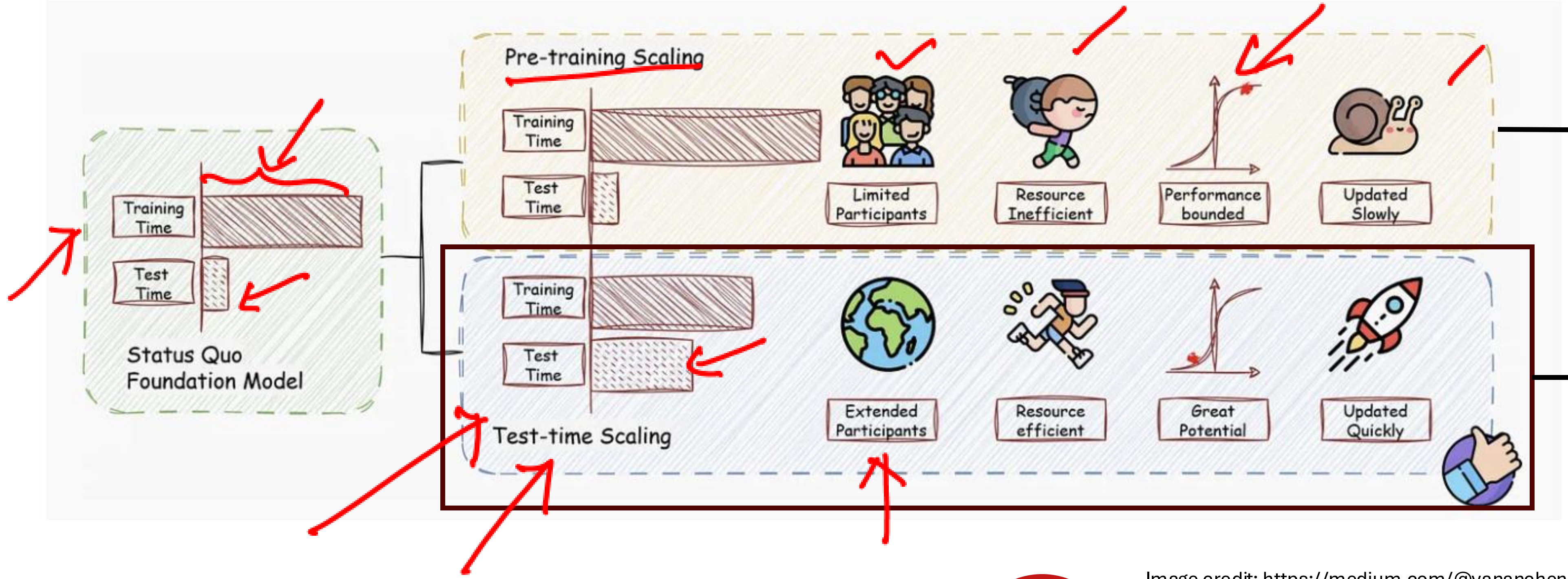


10% improvement in performance requires 3x more energy!

Future of LLM Scaling

Training-time scaling works on the principals of training larger models on higher volume of pre-training data.

Post-training strategies involve – 1. in-context adaptation, 2. supervised fine-tuning, 3. Reinforcement learning



Several challenges with pre-training scaling

Scale models at test-time by associating more compute!

Image credit: https://medium.com/@yananchen1116/a-survey-on-test-time-scaling-04b1c1dadb9d

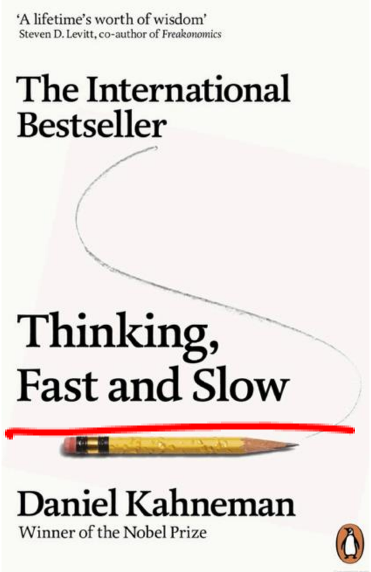
Test-time Scaling? Why?

- Complex reasoning tasks require more dedicated effort, *i.e.*, longer generations. Human cognition also follows similar patterns – "*thinking fast and slow*", slow and deliberate thinking for complex problems.

- We can fine-tune (using SFT or RL or hybrid approaches) foundational models on complex reasoning tasks with detailed reasoning chains added to the fine-tuning data.

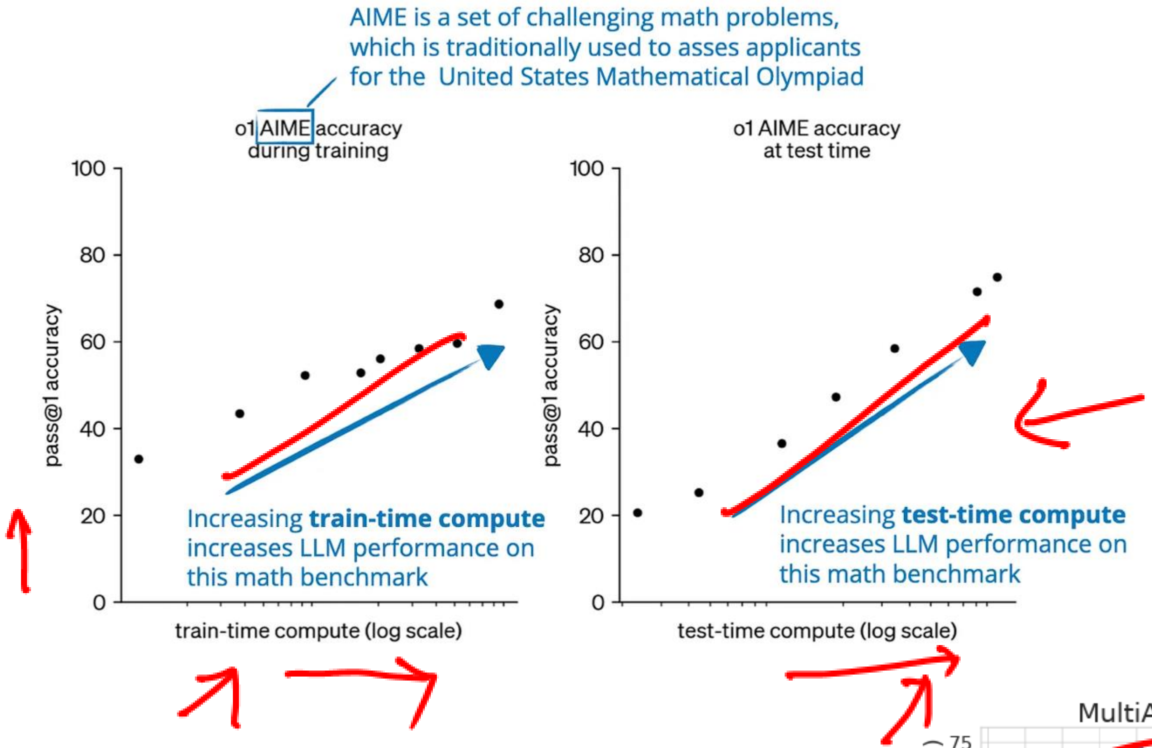
- Test-time scaling, *i.e.*, assigning more compute (increased number of generation samples, depth of reasoning, number of inference steps) to enhance reasoning abilities inherently, while keeping the model parameters fixed.

- Typically works better than training-time scaling strategies, at the same compute cost.

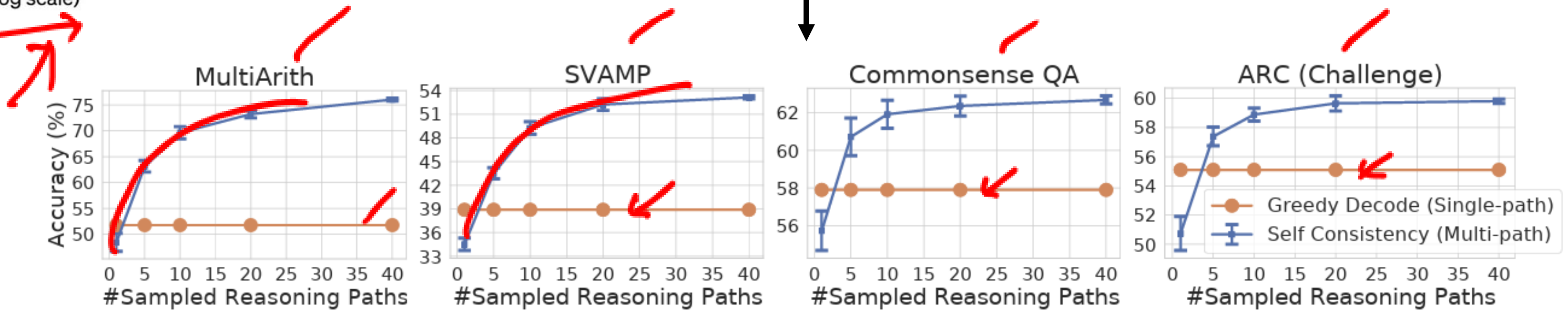


Key recent models capable of test-time scaling abilities: DeepSeek R1, OpenAI o1

Test-time Scaling -- Empirical Motivation

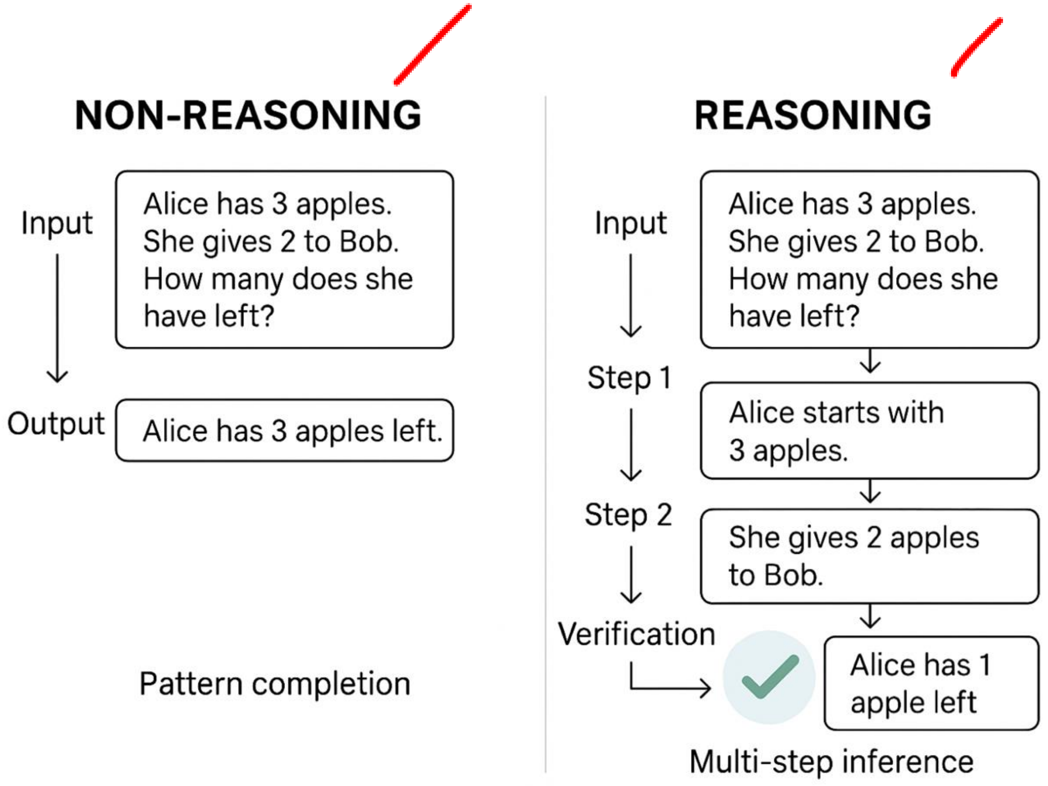


Wang et al., 2022 showed that TTS could improve greedy decoding performance by up to 25% for certain tasks



Reasoning vs Non-Reasoning Models

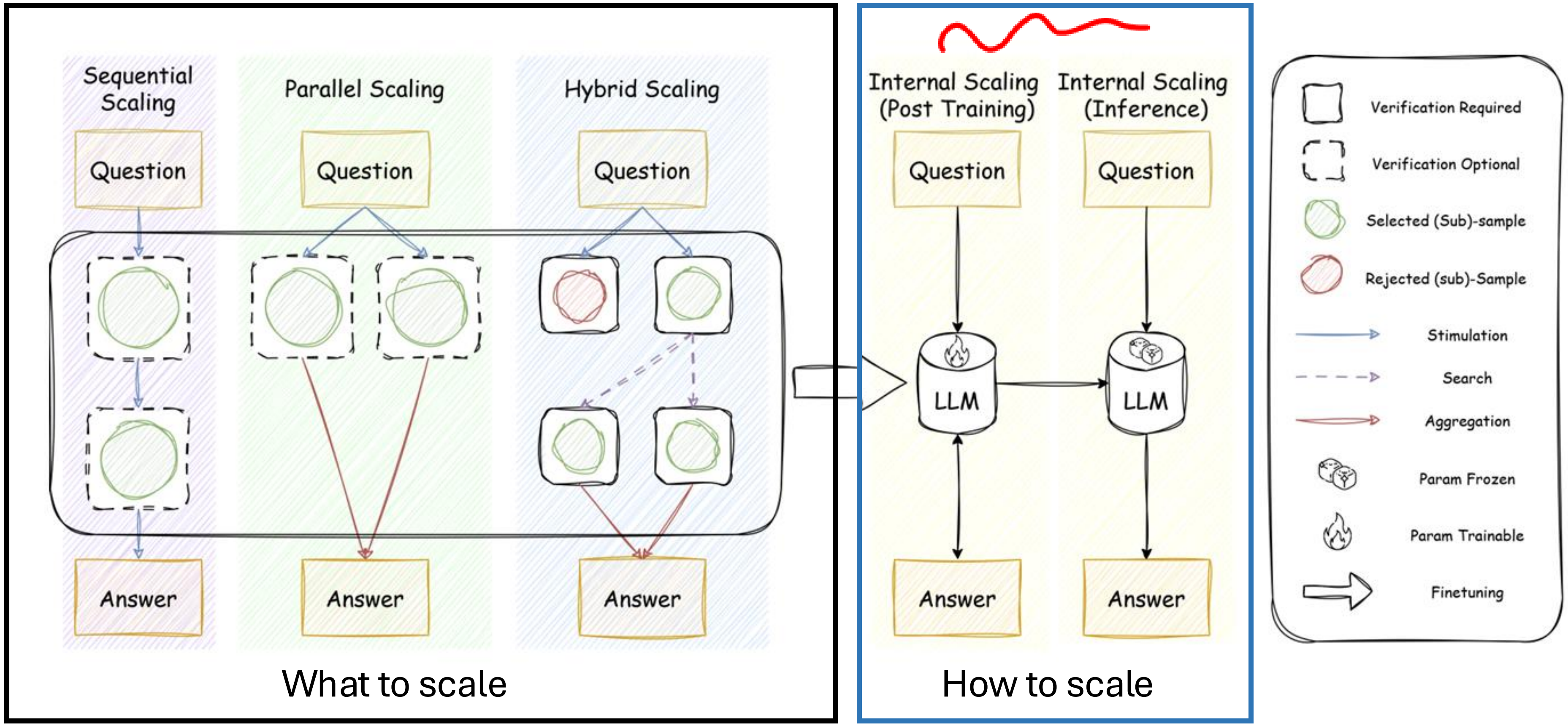
Reasoning models tend to benefit more from test-time scaling strategies



Unlike non-reasoning language models which generates responses directly auto-regressively, reasoning models break response into multiple steps, explore multiple reasoning paths, employs verification before generating the final response

End-of-reasoning tokens are used to indicate stop of reasoning.

Test-time Scaling Strategies: What and How to Scale



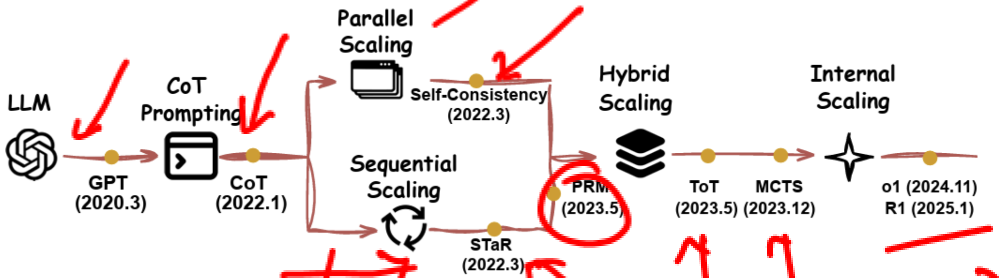
What To Scale in Test-time

Parallel scaling - Generate N candidates in parallel; aggregate to final (Example – Majority voting, Beam search, Diverse verifier tree search, self-consistency)

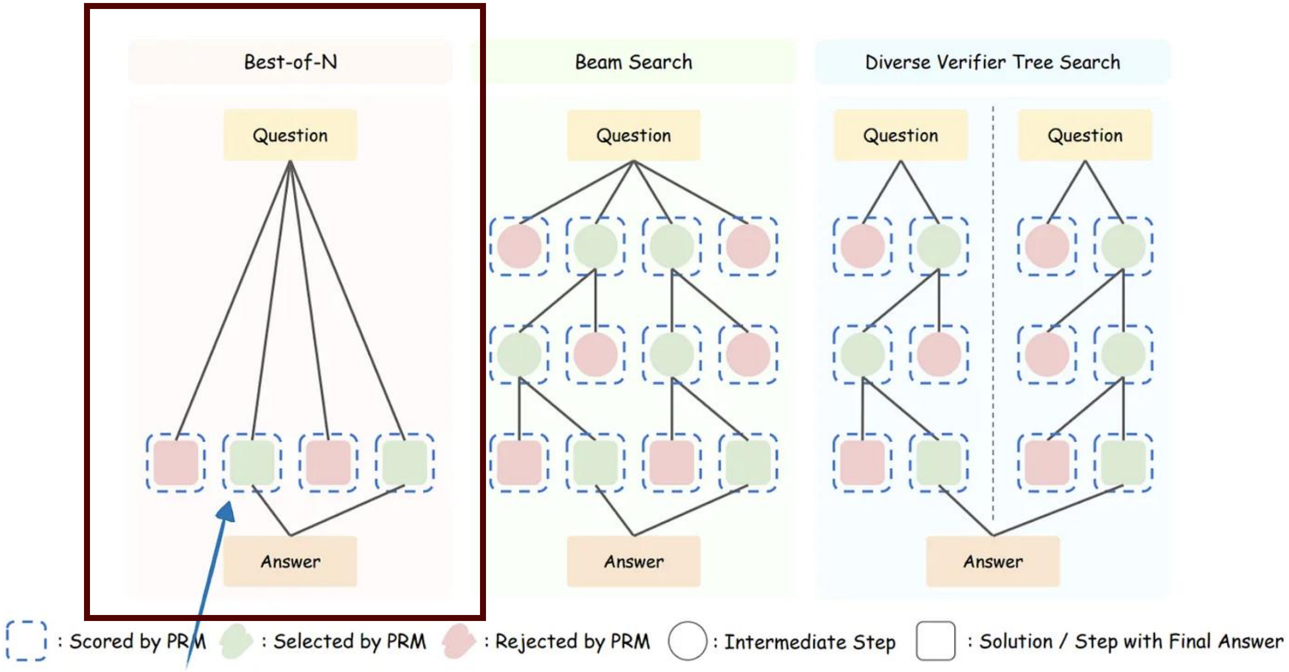
Sequential scaling – Iteratively generate and refine responses (Example - Self-refine, ReAct, S1 (budget forcing), TIP)

Hybrid scaling – Extracting suboptimality through simultaneous parallel and sequential scaling (Example – Tree-of-though, Graph-of-thought, Monte-Carlo Tree Search)

Internal scaling – Autonomous scaling within reasoning models (OpenAI o1, DeepSeek R1)



Best-of-N and Majority Voting Strategy (Parallel Scaling)



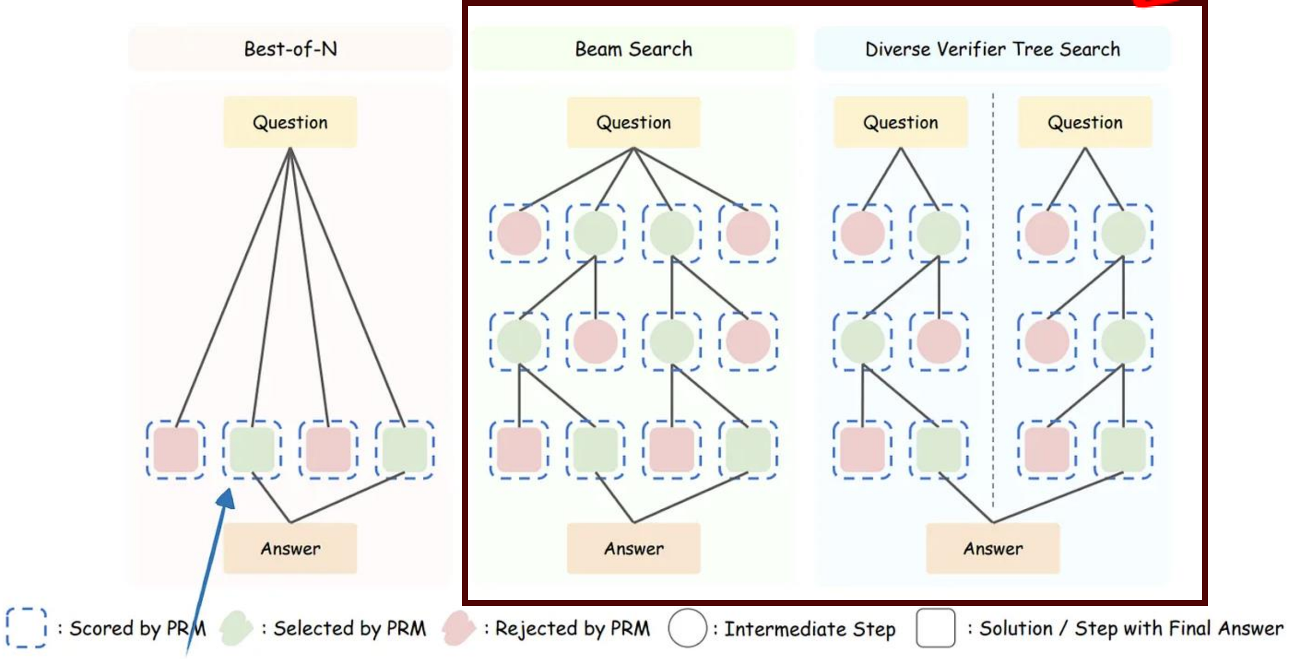
For N independent outputs $y_1, y_2, \ldots, y_N$(can be generated with different decoding strategy or seeds), choose the result $\hat{y} = arg \max_i M(y_i)$, where M is a process reward model (PRM) generating a score (e.g. accuracy w.r.t ground truth)

Majority voting strategy computes the output generated in the majority of the traces, *i.e.,* $\hat{y} = arg \max_c \sum_{i=1}^{N} \mathbb{I}(y_i = c)$,

Probability of success of best-of-N strategy is $1 - (1-p)^N$, i.e., increases with N.

Image credit: https://magazine.sebastianraschka.com/p/state-of-llm-reasoning-and-inference-scaling

Beam Search and DVTS (Parallel Scaling)



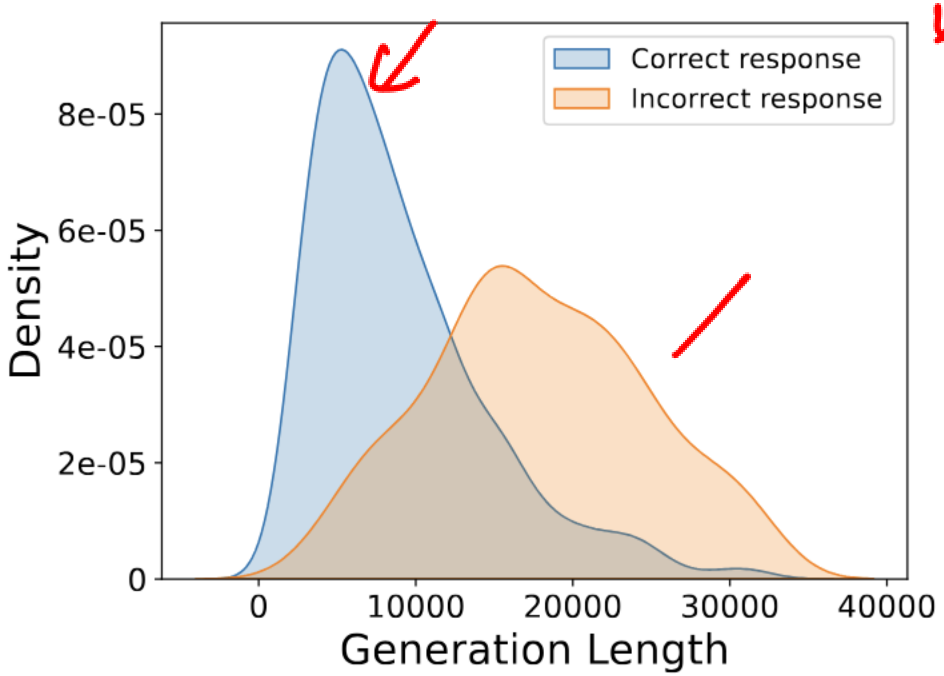
Instead of greedy decoding, beam search keeps the top-k partial hypotheses at each step (by cumulative log-probability), expanding all in parallel.

DVTS: the model generates *diverse reasoning branches* (via top-p or nucleus sampling), and each branch is *evaluated or verified* using a PRM.

The PRM can evaluate the correctness of the reasoning trace, along with the final answer, enabling the inference process coherent, transparent and robust.

Image credit: https://magazine.sebastianraschka.com/p/state-of-llm-reasoning-and-inference-scaling

A Simple Parallel Scaling – First Finish Search



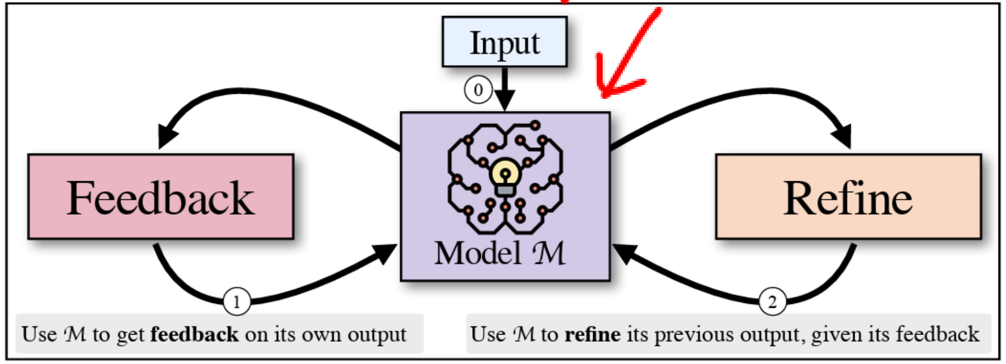
For several reasoning models (*e.g.*, QwQ, R1, GPT-OSS) correct reasoning traces tend to be shorter in length than incorrect traces.

This implies that reasoning models are mostly correct when they are confident in their first attempt.

These observations lead to a simpler parallel sampling technique, terminate the process when one reasoning trace is finished (end of sentence token generated)

First finish search (FFS) can drastically reduce the inference compute, due to early termination.

Self-Refine (Sequential Scaling)



Self-refine is an iterative process, where the response is refined based on the previous responses and the feedbacks.

Self-generated feedback is received based on the previous response.

The iterative process stops when a stop condition is triggered.

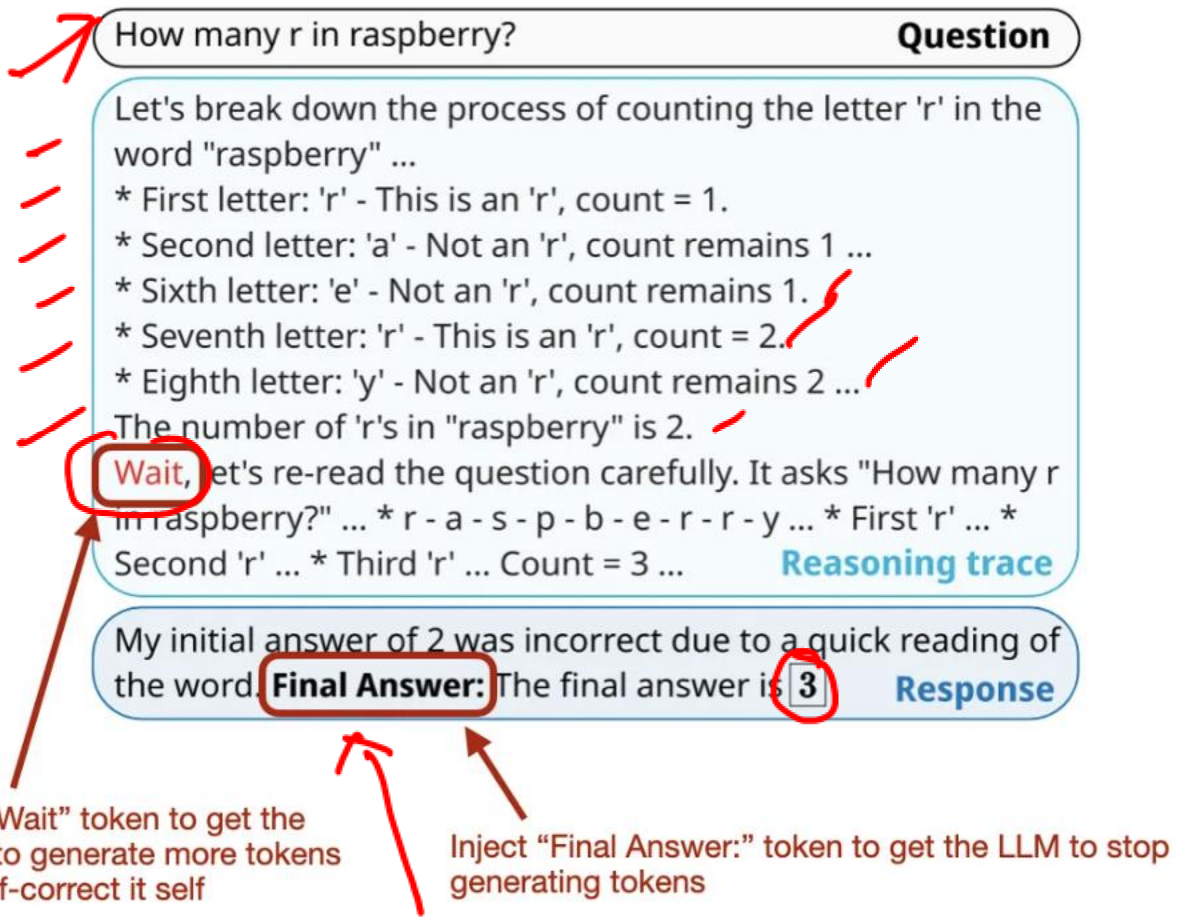
```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

```
Code (refined)

def sum_faster(n):
  return (n*(n+1))//2
```

S1 – *aka* Budget Forcing (Sequential Scaling)



To elongate the reasoning process (hence, sequential scaling), S1 suppresses the end-of-thinking token (a typical token used in reasoning models to indicate finishing of reasoning), and add a "wait" token instead.

After seeing the "wait" token the model continues exploring different reasoning path

By enforce the budget (total number of generation tokens), the end-of-thinking token is added and an optional "final answer" token is added for early exit.

Image credit: https://magazine.sebastianraschka.com/p/state-of-llm-reasoning-and-inference-scaling

Thought Switching Penalty (Sequential Scaling)

Q: *Let a, b, x, and y be real numbers with a>4 and b>1 such that \frac{x^2}{a^2}+\frac{y^2}{a^2-16} =\frac{(x-20)^2}{b^2-1}+\frac{(y-11)^2}{b^2}=1. Find the least possible value of a+b.*

Solution1

Thought1 *411 tokens* — So I've got this problem here that involves some equations with variables a, b, x, and y, and I need to find the least possible value of a + b given some conditions. ... Perhaps I can think of this as a system of equations and try to solve for x and y in terms of a and b.

Thought2 *588 tokens* — **Alternatively**, maybe there's a geometric interpretation. Since both equations represent ellipses, perhaps the condition is that these ellipses are tangent to each other. ... So, unless there's some transformation or something, they don't seem to be the same ellipse.

Thought3 *109 tokens* — **Alternatively**, perhaps the problem is that for some (x,y), both equations hold simultaneously, and I need to find a and b such that this is possible. ... solve the first equation for y in terms of x and substitute into the second equation. But that seems messy.

Thought4 *280 tokens* — **Alternatively**, perhaps I can consider this as an optimization problem where I need to minimize a + b subject to the condition that the equations have a solution. ... But I'm not sure how to set that up algebraically.

Thought5 *613 tokens* — **Alternatively**, perhaps I can consider the distance between the centers of the two ellipses and relate it to their axes. ... But (y - 11)^2 is y^2 - 22y + 121. However, I only have y^2, not y. So maybe this isn't the best way to substitute.

20 More Thoughts *Averaged 284 tokens* ......

Given that, perhaps I need to accept that this problem is beyond my current ability and seek help or look for a different problem.

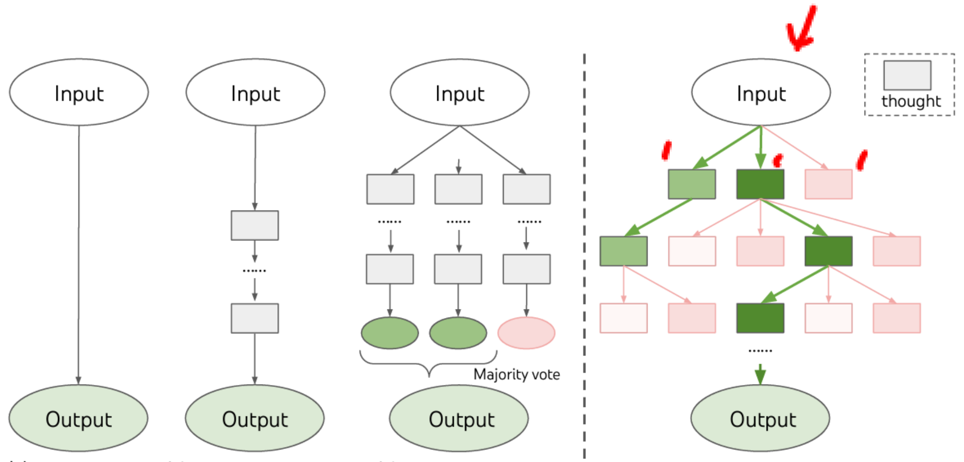
Final Answer: \[\boxed{15} \]

Thought switching penalty (TIP) deals with the under-exploration of thoughts by existing sequential scaling methods.

To encourage the model to get deeper thoughts, TIP uses a penalty score added to the logit for the next token prediction.

The penalty term uses "penalty strength", controlling how much to penalize and "penalty duration" to denote the number of future token positions to be penalized.

Tree-of-Thoughts (Hybrid Scaling)



(a) Input-Output Prompting (IO) (c) Chain of Thought Prompting (CoT) (c) Self Consistency with CoT (CoT-SC) **(d) Tree of Thoughts (ToT)**

	Game of 24	Creative Writing	5x5 Crosswords
Input	4 numbers (4 9 10 13)	4 random sentences	10 clues (h1. presented;..)
Output	An equation to reach 24 (13-9)*(10-4)=24	A passage of 4 paragraphs ending in the 4 sentences	5x5 letters: SHOWN; WIRRA; AVAIL; ...
Thoughts	3 intermediate equations (13-9=4 (left 4,4,10); 10-4=6 (left 4,6); 4*6=24)	A short writing plan (1. Introduce a book that connects...)	Words to fill in for clues: (h1. shown; v5. naled; ...)
#ToT steps	3	1	5-10 (variable)

1. **Thought Decomposition** – Split the problem into intermediate "thought" units (manageable sub-steps).
2. **Thought Generation** – From each state, generate *k* candidate thoughts using CoT or sampling.
3. **State Evaluation** – Score or rank each partial solution using LLM or heuristic
4. **Search Strategy** – Explore promising branches via BFS/DFS, prune low-score paths, and backtrack if needed.

Internal Scaling

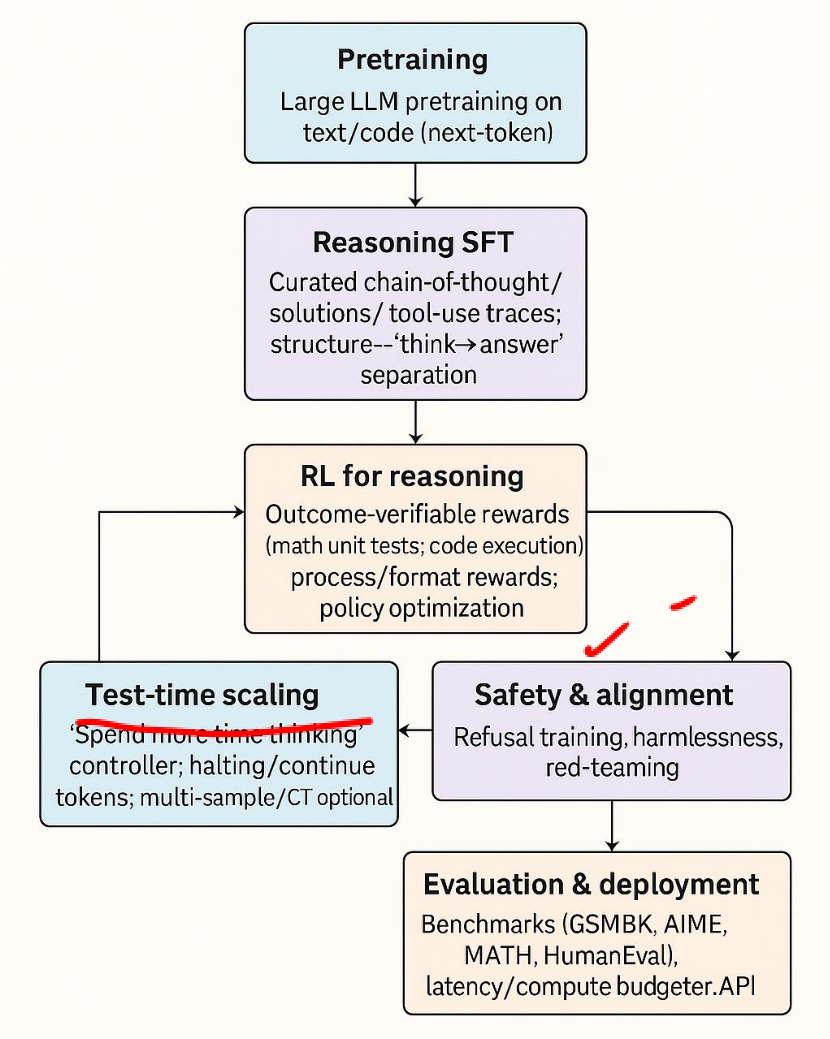
- Internal scaling elicits a model to autonomously determine how much compute to assign during inference.
- These methods use internal process reward models to enable longer generation or decision to halt reasoning.
- These methods are typically trained using RL for learning these decision-making capabilities.
- DeepSeek R1 and OpenAI O1 use internal scaling strategies.

How Do We Scale Reasoning Models?

How to Scale?

- Tuning-based
  - O1, R1
    - Supervised fine-tuning
    - RL-based
- Inference-only (TTS)
  - Already discussed
    - Stimulation (*e.g.*, Self-refine)
    - Verification (OpenAI O1)
    - Search (ToT, MCTS)
    - Aggregation (MV, BoN, FFS)

OpenAI O1 – Internal Scaling with Tuning



Developing OpenAI O1 model

- Built upon GPT-4-class architecture and reasoning-focused pretraining datasets with extensive *multi-step reasoning traces* (math, science, code).
- Fine-tuned on curated reasoning datasets containing *explicit thought sequences* (similar to CoT) to teach the model to "think before answering."
- Applied **RL with reasoning rewards** -- accuracy, logical consistency, and formatting - guided by *verifier models* that check intermediate reasoning steps.
- Introduced *internal reflection loops* and *adaptive halting*, enabling variable inference depth ("think longer when needed"). Model learns to allocate more compute to harder problems.

Where Do We Use TTS?

Benchmark	Size	Evaluation Criteria	Example Task	Key Features	Type
		Reasoning-intensive Tasks			
FrontierMath (Glazer et al., 2024)	Hundreds	Exact match	Algebraic geometry	High complexity	Math
MATH (Cobbe et al., 2021)	12.5K	Exact match	AMC/AIME-style	Structured reasoning	
NuminaMath (LI et al., 2024)	860K	Exact match, CoT	Olympiad-level math	Annotated reasoning	
OmniMath (Gao et al., 2025a)	4.4K	Accuracy	Math Olympiads	Advanced reasoning	
GSM8K (Zhang et al., 2024a)	8.5K	Accuracy	Grade-school math	Natural-language solutions	
rStar-Math (Guan et al., 2025)	747K	Pass@1 accuracy	Competition math	Iterative refinement	
ReST-MCTS (Zhang et al., 2024a)	Varied	Accuracy	Multi-step reasoning	Reward-guided search	
s1 (Muennighoff et al., 2025)	1K	Accuracy	Math/science tasks	Controlled compute	
USACO (Shi et al., 2024)	307	Pass@1	Olympiad coding	Creative algorithms	Code
AlphaCode (Li et al., 2022)	Thousands	Solve rate	Competitive coding	Complex algorithms	
LiveCodeBench (Jain et al., 2025)	511	Pass@1	Real-time coding	Live evaluation	
SWE-bench (Jimenez et al., 2024)	2.3K	Resolution rate	GitHub issues	Multi-file edits	
GPQA (Rein et al., 2024)	448	Accuracy	Graduate STEM	Domain expertise	Science
OlympicArena (Huang et al., 2024a)	11.1K	Accuracy	Multidisciplinary tasks	Multimodal reasoning	
OlympiadBench (He et al., 2024a)	8.4K	Accuracy	Math/Physics Olympiads	Expert multimodal tasks	
TheoremQA (Chen et al., 2023b)	800	Accuracy	Theorem-based STEM	Theoretical application	
MedQA (Jin et al., 2020)	1.3K	Accuracy	Clinical diagnostics	Medical accuracy	Medical

Tasks where TTS could be effective

- Existing works predominantly use TTS on mathematical reasoning tasks (example shown below).
- AIME, MATH500 among the most popular benchmarks in mathematical reasoning
- GPQA is a popular benchmark is science (includes questions on physics, chemistry and mathematics)
- Swe-bench is a popular benchmark in code generation

[AIME24]

Alice and Bob play the following game. A stack of n tokens lies before them. The players take turns with Alice going first. On each turn, the player removes either 1 token or 4 tokens from the stack. Whoever removes the last token wins. Find the number of positive integers n less than or equal to 2024 for which there exists a strategy for Bob that guarantees that Bob will win the game regardless of Alice's play.

[MATH500]

Find the projection of a onto $b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$ if $a \cdot b = 8$.

[GPQA]

A quantum mechanical particle of mass m moves in two dimensions in the following potential, as a function of the polar coordinates (r, θ):

$$V(r,\theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$$

Find the energy spectrum. Hint: Write the potential in Cartesian co-ordinates.

How Do We Evaluate TTS Methods?

Performance - assess the correctness of generated solutions. *e.g.,* Accuracy, pass@k

Controllability – evaluate whether test-time methods can consistently adhere to pre-defined resource constraints (compute budgets or output length targets).

Scalability – measure how effectively test-time scaling methods can leverage increased compute to improve performance.

Efficiency – assess the computational and resource cost

Content credit: https://testtimescaling.github.io/

How Do We Evaluate TTS Methods? Controllability

Control metric measures the fraction of test-time compute values that stay within given upper and lower bounds

$$\text{Control} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \mathbb{I}(a_{\min} \leq a \leq a_{\max}),$$

$\mathcal{A}$ is the set of observed compute values such as thinking tokens, and $\mathbb{I}(\cdot)$ is the indicator function.

Mean Deviation from Target Length quantifies the average relative difference between the generated output length and the target length

$$\text{Mean Deviation} = \mathbb{E}_{x \sim D}\left[\frac{|n_{\text{generated}} - n_{\text{gold}}|}{n_{\text{gold}}}\right]$$

Root Mean Squared Error (RMSE) of Length Deviation captures the variance in length control

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{n_{\text{generated},i} - n_{\text{gold},i}}{n_{\text{gold},i}}\right)^2}.$$

Content credit: https://testtimescaling.github.io/

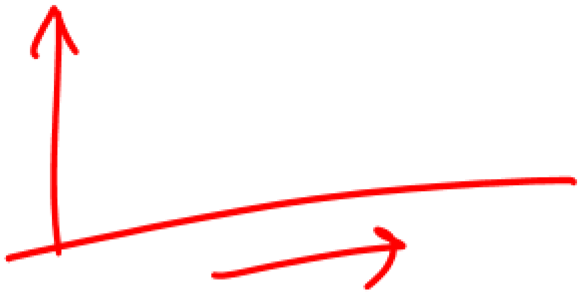
How Do We Evaluate TTS Methods? Scalability

Scaling factor captures the average slope of performance gains as compute increases

$$\text{Scaling} = \frac{1}{\binom{|\mathcal{A}|}{2}} \sum_{\substack{a,b \in \mathcal{A} \\ b>a}} \frac{f(b) - f(a)}{b - a}.$$

(a, b) $\mathcal{A}$

Scaling Curves (Performance vs. Compute) visualizes how metrics such as accuracy improve as token budgets, iteration depth, or the number of samples increase.



Content credit: https://testtimescaling.github.io/

How Do We Evaluate TTS Methods? Efficiency

Total compute is calculated as the total number of tokens used across all generated responses (including intermediate reasoning and final answer) for a given problem.

Sequential compute is calculated as the minimum number of tokens needed in any of the generated responses.

Underthinking score measures how early in the response the first correct thought appears, relative to the total length of the response, in cases where the final answer is incorrect.

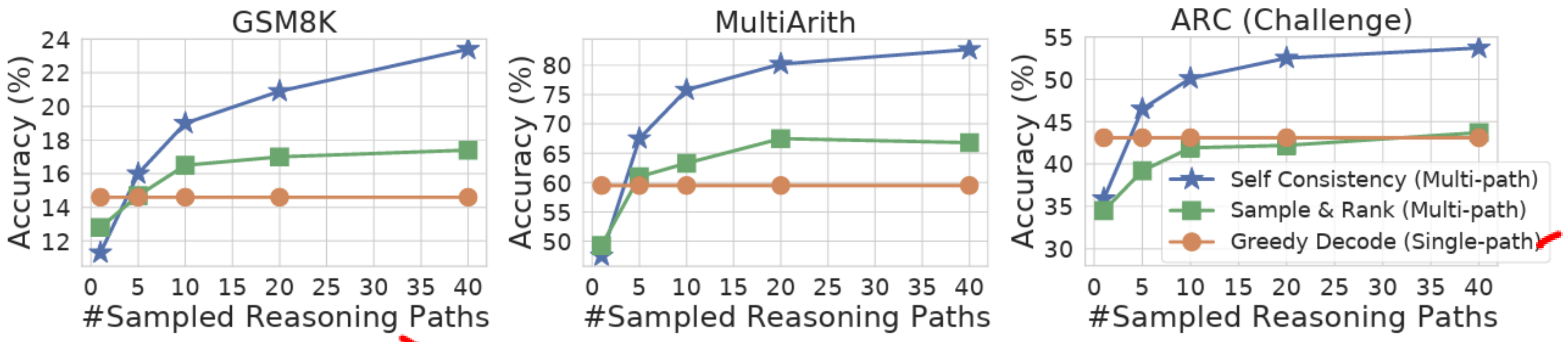
Formally, the underthinking score ξ_{UT} is defined as:

$$\xi_{\mathrm{UT}} = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{\hat{T}_i}{T_i}\right)$$

- N: Number of incorrect responses in the test set.
- T_i: Total number of tokens in the i-th incorrect response.
- $\hat{T}_i$: Number of tokens from the beginning of the response up to and including the first correct thought.

Content credit: https://testtimescaling.github.io/

Results with Stimulation-based TTS



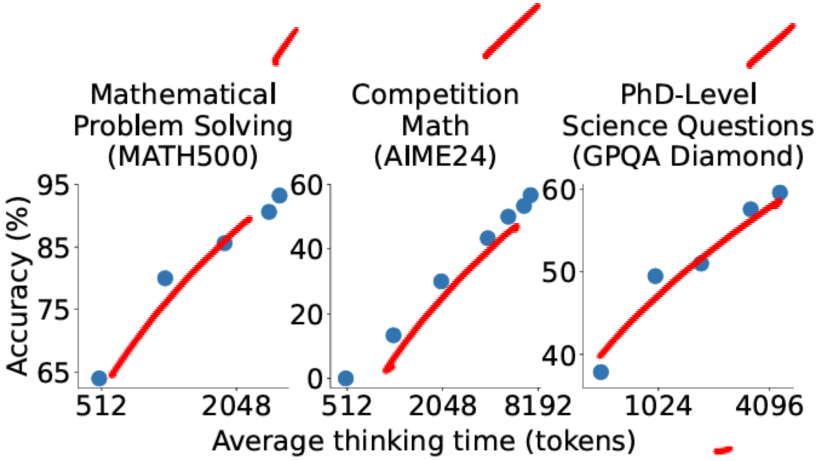
Results of GPT-3-code-davinci model with self-consistency. Increasing number of reasoning path improves the performance on mathematical and commonsense reasoning tasks, with diminishing return.

More reasoning paths, more exploration – increases chances of leading to correct answer.

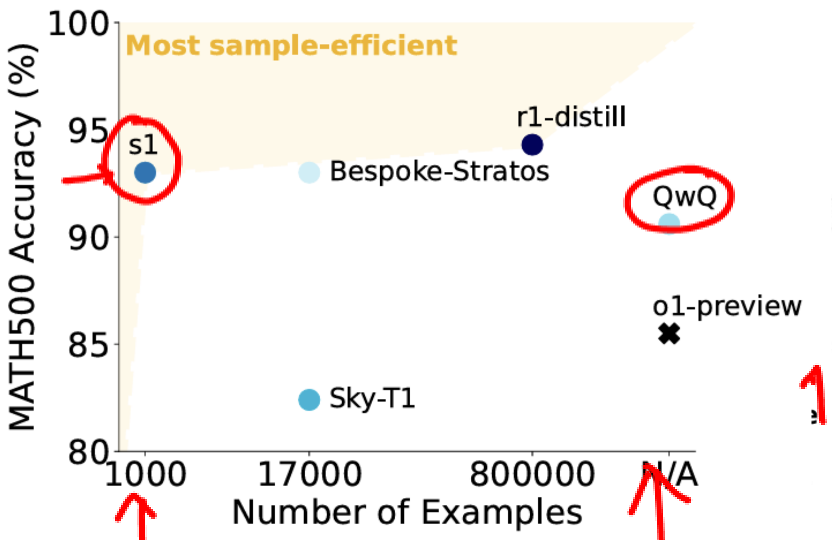
Self-refine is also extremely effective for wide-range of tasks and models

	GPT-3.5		ChatGPT		GPT-4	
Task	Base	+SELF-REFINE	Base	+SELF-REFINE	Base	+SELF-REFINE
Sentiment Reversal	8.8	**30.4** (↑21.6)	11.4	**43.2** (↑31.8)	3.8	**36.2** (↑32.4)
Dialogue Response	36.4	**63.6** (↑27.2)	40.1	**59.9** (↑19.8)	25.4	**74.6** (↑49.2)
Code Optimization	14.8	**23.0** (↑8.2)	23.9	**27.5** (↑3.6)	27.3	**36.0** (↑8.7)
Code Readability	37.4	**51.3** (↑13.9)	27.7	**63.1** (↑35.4)	27.4	**56.2** (↑28.8)
Math Reasoning	**64.1**	**64.1** (0)	74.8	**75.0** (↑0.2)	92.9	**93.1** (↑0.2)
Acronym Generation	41.6	**56.4** (↑14.8)	27.2	**37.2** (↑10.0)	30.4	**56.0** (↑25.6)
Constrained Generation	28.0	**37.0** (↑9.0)	44.0	**67.0** (↑23.0)	15.0	**45.0** (↑30.0)

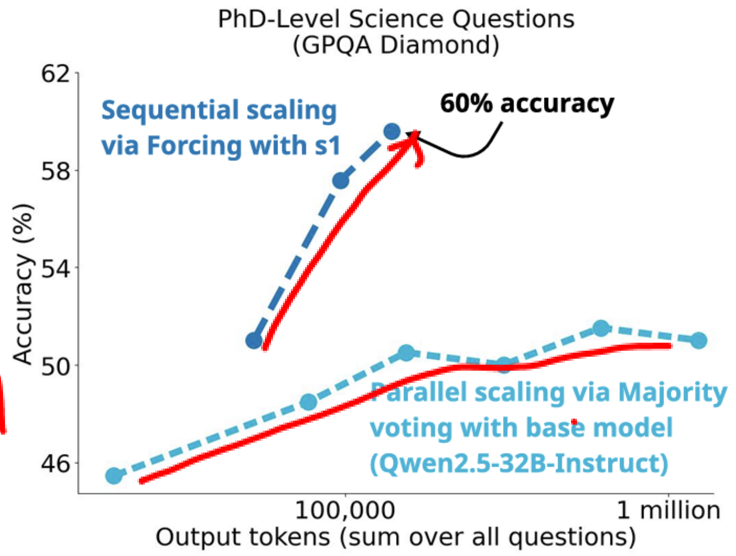
Results with Sequential Scaling



Sequential scaling like budget forcing (S1) improves with more generation budget (results shown with Qwen-2.5-32B-Instruct)



Fine-tuning Qwen-2.5-32B model on only 1000 S1 samples (samples with "wait" and "final answer" enforced) achieve similar performance to QwQ with 800x more SFT samples.



Budget forcing scales better than majority voting (parallel scaling)

Sequential Scaling is Good, But?

(a) R1-Distill-Qwen

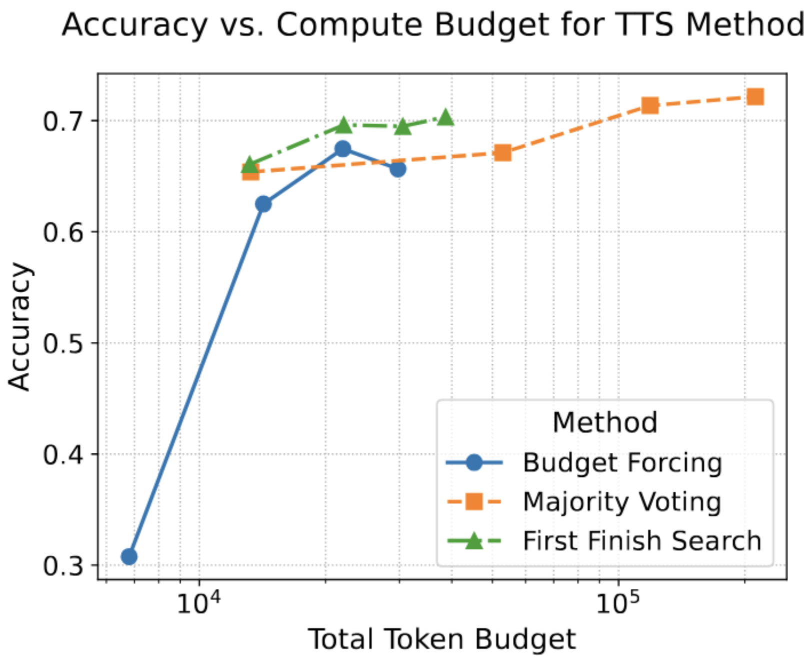
Metric	SD	BF	BS	MV	LFS	FFS
Seq. tokens	–	25.7	11.2	17.1	17.1	**6.5**
Total tokens	–	**25.7**	44.8	68.4	68.4	26.0
GPQA	–	58.6	62.6	64.7	50.5	**67.2**
AIME24	–	60.0	66.7	**83.3**	50.0	73.3
AIME25-I	–	53.3	46.7	**60.0**	46.7	**60.0**
AIME25-II	–	**57.1**	**57.1**	**57.1**	50.0	50.0

(b) QwQ-32B

Metric	SD	BF	BS	MV	LFS	FFS
Seq. tokens	–	23.7	12.8	18.2	18.2	**9.5**
Total tokens	–	**23.7**	51.2	72.8	72.8	38.0
GPQA	–	60.1	57.1	64.7	57.6	**65.2**
AIME24	–	**86.7**	80.0	83.3	73.3	83.3
AIME25-I	–	60.0	**66.7**	**66.7**	60.0	60.0
AIME25-II	–	71.4	78.6	**85.7**	71.4	78.6

FFS is very simple, could yet achieve remarkable result, at a much lower compute.

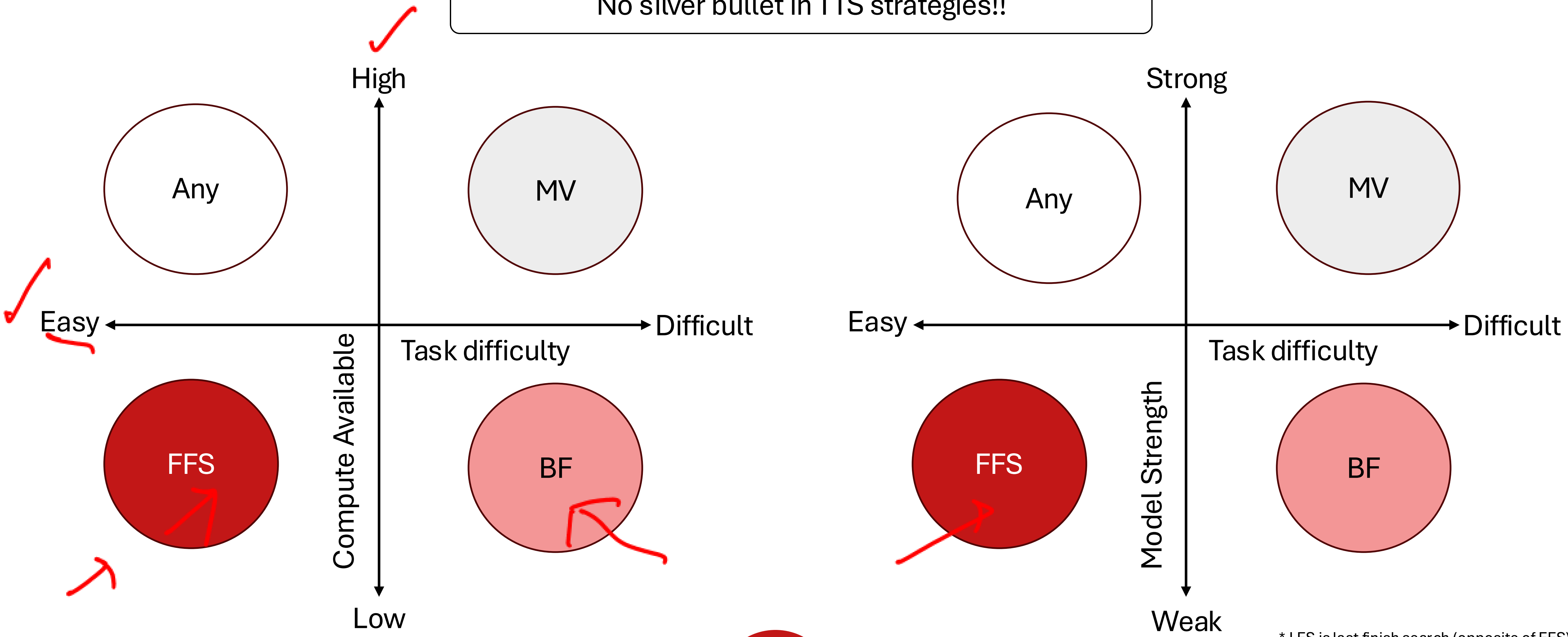
When higher compute is assigned, majority voting (MV) tends to outperform all other methods.



With lower compute, FFS tends to work better than MV or BF.
At higher compute, MV tends to improve consistently, beating other methods.

How Do We Select The Best TTS Strategy

No silver bullet in TTS strategies!!



* LFS is last finish search (opposite of FFS)

Emerging Research Directions in TTS

- **Parallel Scaling 2.0 – Smarter Exploration:** Move beyond brute-force *Best-of-N* sampling toward *diverse and guided reasoning paths*. Use **verifier-augmented** or **adaptive coverage** to improve efficiency and reliability.

- **Sequential Scaling – Structured Self-Refinement:** Introduce *verification checkpoints* between reasoning steps to avoid error propagation. *Adaptive, self-correcting reasoning* with minimal redundant computation.

- **Hybrid Scaling – Parallel + Sequential Synergy:** Combine breadth exploration with depth refinement. Expected to power *general-purpose reasoning agents* that dynamically mix both paradigms.

Challenges and Long-Term Opportunities

- **Optimizing Efficiency & Evaluation:** Develop *compute-aware metrics* balancing accuracy, cost, robustness, and interpretability. Explore adaptive test-time scaling that adjusts inference depth per query difficulty.
- **Adaptive Test-Time Scaling (Auto-Scaling):** Create models that autonomously decide *how much to think,* avoiding both *underthinking* and *overthinking*. Inspired by human cognitive flexibility and *Adaptive Computation Time* frameworks.

References

- Kaplan, Jared, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. "Scaling laws for neural language models." *arXiv preprint arXiv:2001.08361* (2020).
- Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas et al. "Training compute-optimal large language models." *arXiv preprint arXiv:2203.15556* (2022).
- Goel, Yash, Ayan Sengupta, and Tanmoy Chakraborty. "Position: Enough of Scaling LLMs! Lets Focus on Downscaling." *arXiv preprint arXiv:2505.00985* (2025).
- Zhang, Qiyuan, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu et al. "A Survey on Test-Time Scaling in Large Language Models: What, How, Where, and How Well?." *arXiv preprint arXiv:2503.24235* (2025).
- Wang, Xuezhi, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. "Self-consistency improves chain of thought reasoning in language models." *arXiv preprint arXiv:2203.11171* (2022).
- Madaan, Aman, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon et al. "Self-refine: Iterative refinement with self-feedback." *Advances in Neural Information Processing Systems* 36 (2023): 46534-46594.
- Muennighoff, Niklas, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. "s1: Simple test-time scaling." *arXiv preprint arXiv:2501.19393* (2025).
- Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. "Tree of thoughts: Deliberate problem solving with large language models." *Advances in neural information processing systems* 36 (2023): 11809-11822.
- Agarwal, Aradhye, Ayan Sengupta, and Tanmoy Chakraborty. "First Finish Search: Efficient Test-Time Scaling in Large Language Models." *arXiv preprint arXiv:2505.18149* (2025).
- Guo, Daya, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." *arXiv preprint arXiv:2501.12948* (2025).
- Jaech, Aaron, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar et al. "Openai o1 system card." *arXiv preprint arXiv:2412.16720* (2024).
- Wang, Yue, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song et al. "Thoughts are all over the place: On the underthinking of o1-like llms." *arXiv preprint arXiv:2501.18585* (2025).