

Deep Think with Confidence

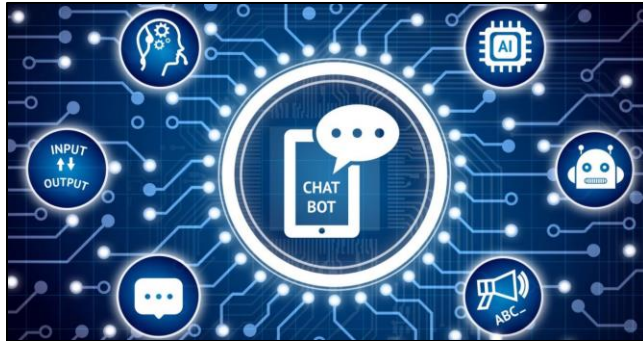
Leveraging Internal Signals for Efficient LLM Reasoning

Jiawei Zhao

Research Scientist, Meta FAIR



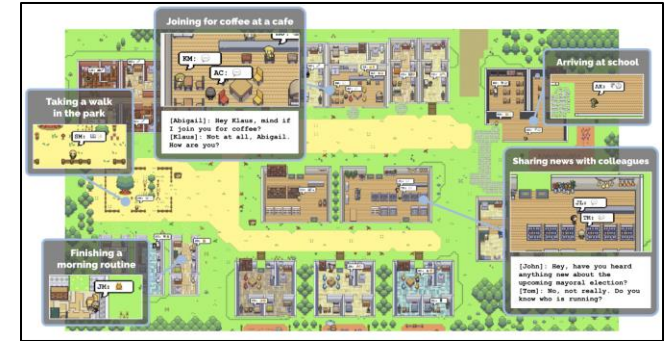
Neural Networks - Foundation Models



Conversational AI



Content Generation



AI Agents

Standard Prompting	Chain of Thought Prompting
<p>Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. ❌</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅</p>

Reasoning

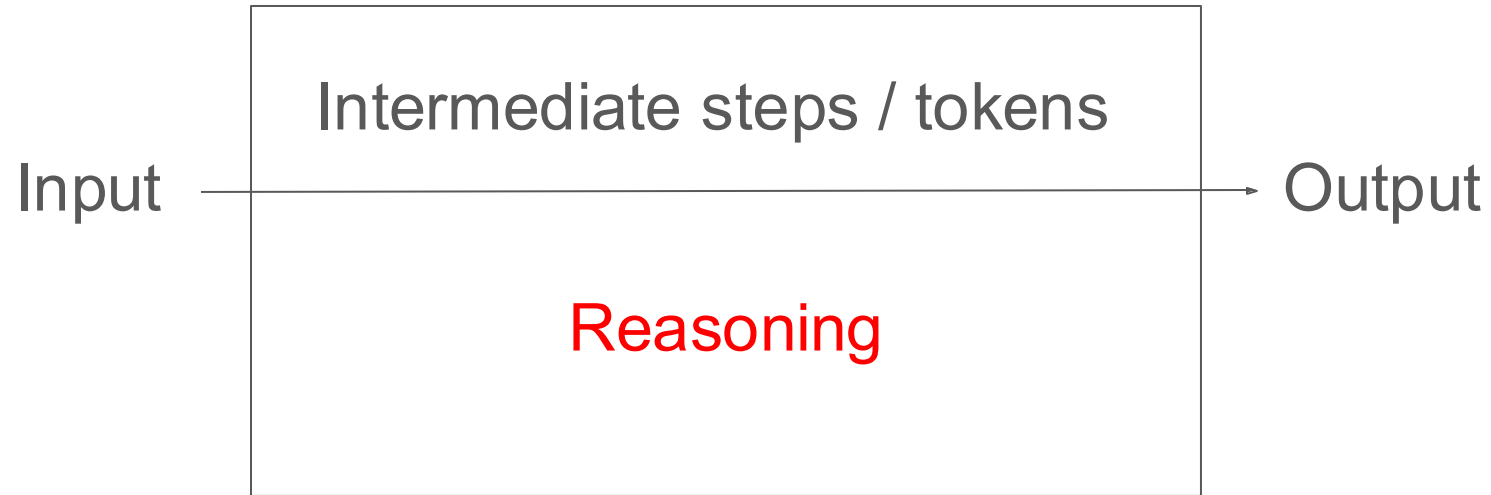


Planning

Large Language Models for Complex Reasoning

- Current capabilities:
- Mathematical problem solving (AIME)
- Code generation and debugging
- Scientific hypothesis formation
- Multi-step logical reasoning

What is LLM Reasoning?



What is the output when concatenating the last letter of each word in “artificial intelligence”?

No reasoning

The answer is “le”.

Reasoning

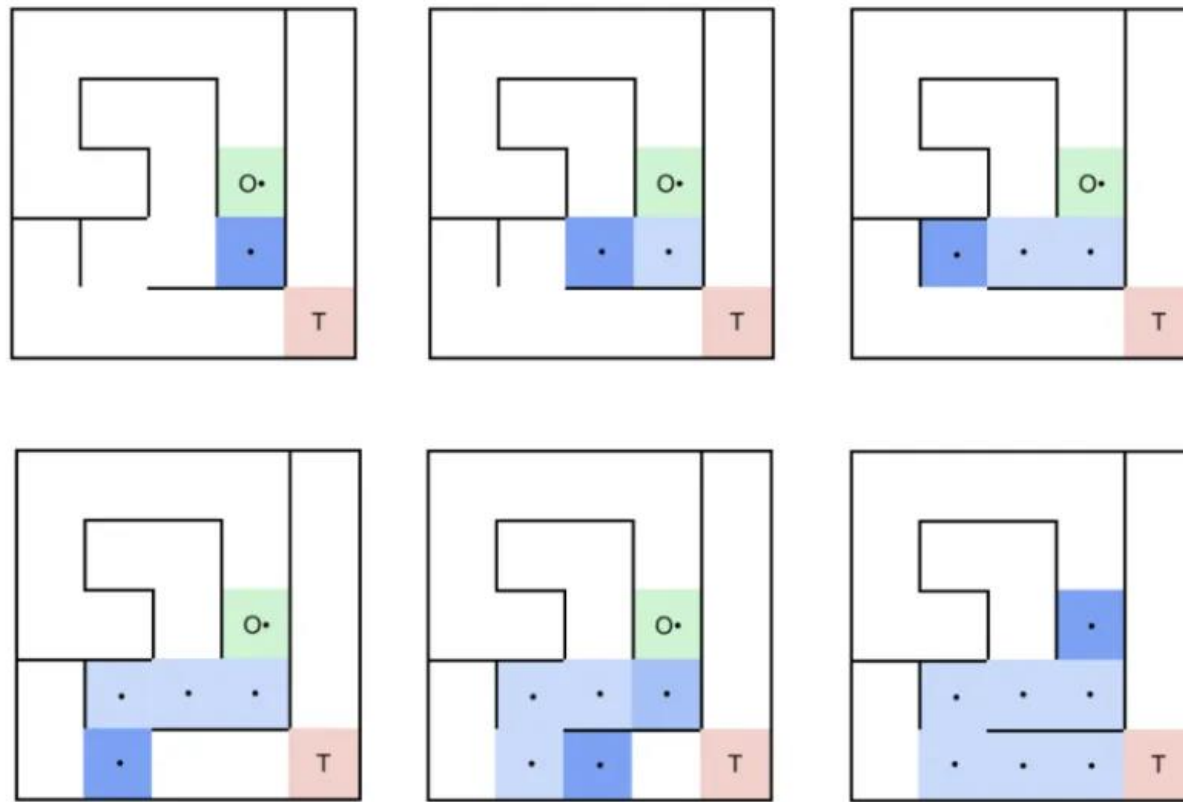


The last letter of “artificial” is “l”. The last letter of “intelligence” is “e”. Concatenating “l” and “e” leads to “le”. So the answer is “le”.

Why “Intermediate Tokens” / “Reasoning” Matters?

- For any problems solvable by boolean circuits of size T , **constant-size transformers** can solve it by generating $O(T)$ intermediate tokens
- If directly generating final answers, either requires a huge depth or cannot solve at all

Why “Intermediate Tokens” / “Reasoning” Matters?



ANSWER:<|down|><|left|><|left|><|down|><|right|><|right|><|right|>

CORRECT

How reasoning evolves? – Post-Training

- Supervised Fine-Tuning (SFT)
- “memorization”
- Reinforcement Learning (RL)
- “generalization”

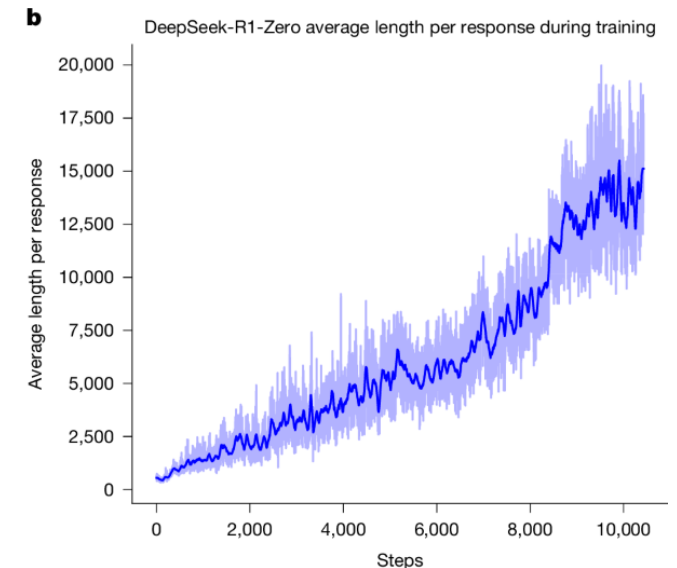
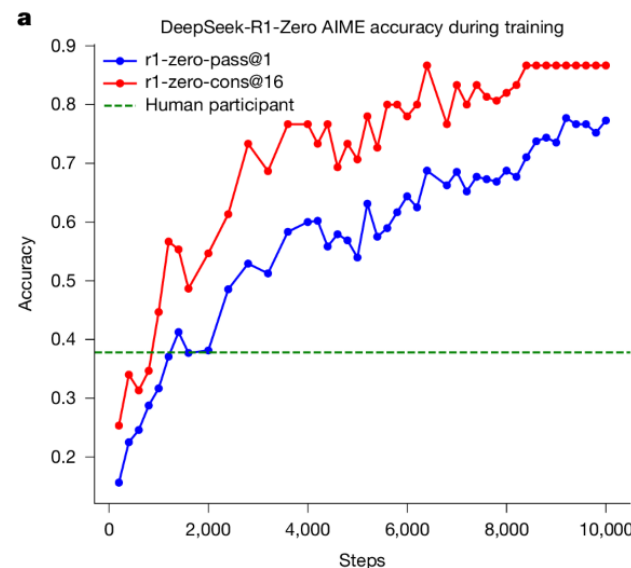
Supervised Fine-Tuning (SFT)

- “memorization”
- problem | step by step solution | output
-> max likelihood of both **solution and answer**
- “step by step solution | output ” comes from human or 3P models
- Highly depends on data quantity and quality, hard to generalize

Reinforcement Learning (RL)

- “generalization”
- problem | step by step solution | output
-> max likelihood of **corrected outputs only** (reinforcement)
- step by step solution <- generated by model itself

- Verification is the key!
- Easy for verifiable problems

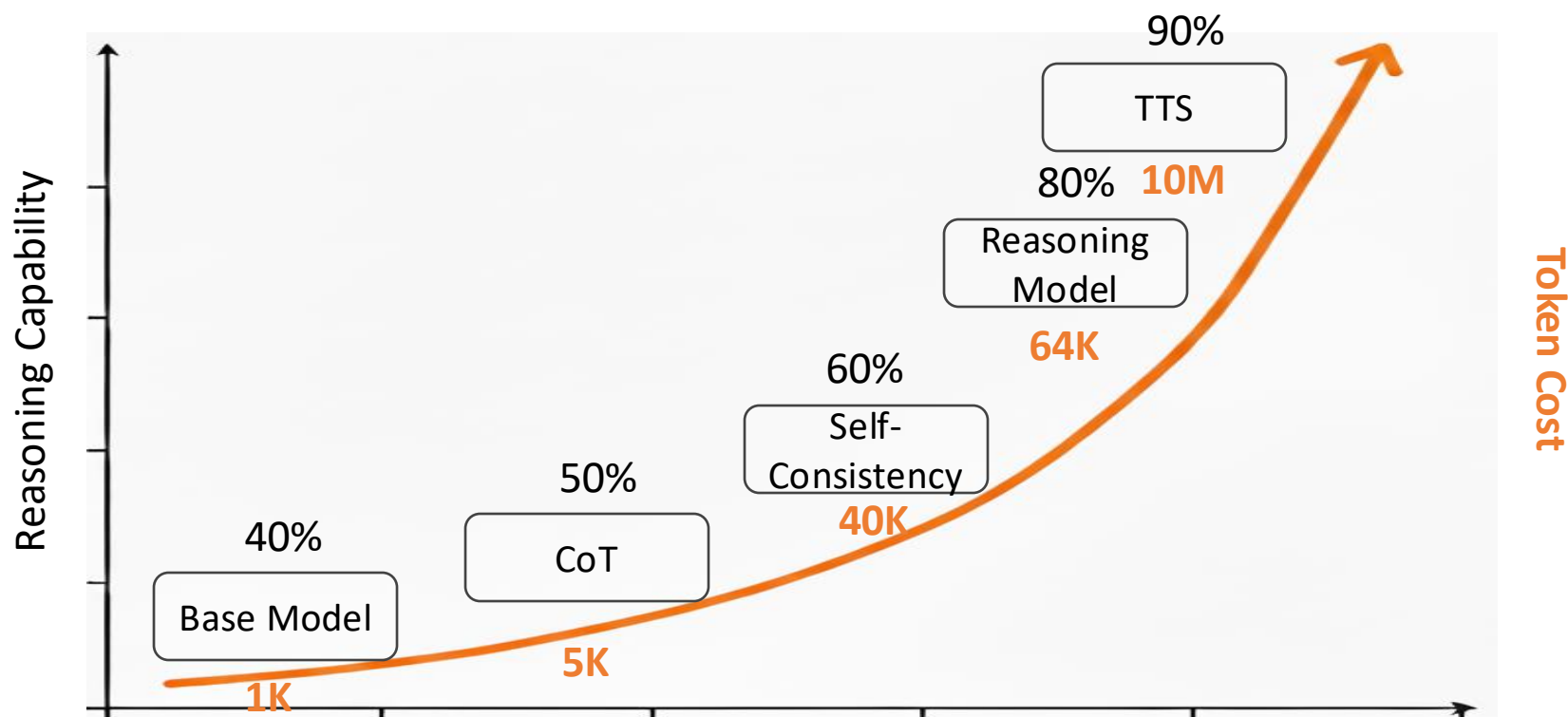


Evolution of LLM Reasoning

Evolution of LLM Reasoning

- Simple tasks & text generation (~40% capability)
- Chain-of-thought (CoT) for complex reasoning (~50% capability)
- Self-consistency & majority voting (~60% capability)
- Long-context reasoning models (~80% capability)
- Test-time scaling / Parallel thinking (~90% capability)

LLMs Can Reason, But at What Cost?



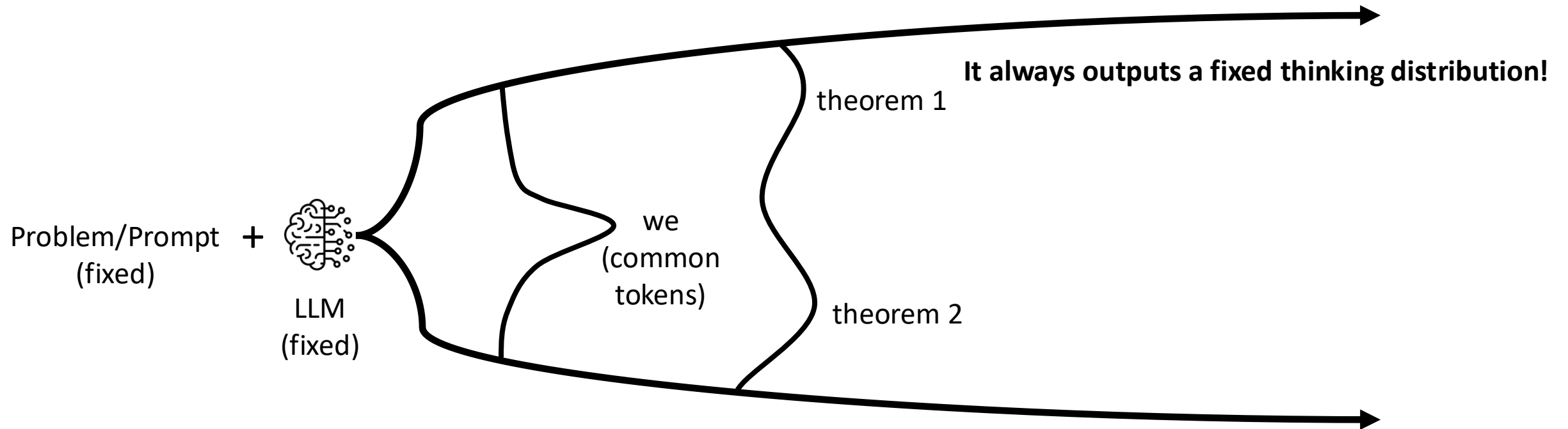
How Much Compute Are We Really Using?

Real-world impact:

- Compute: $\sim 500\times$ more tokens vs single rollout
- Cost: \$1,000 \rightarrow \$500,000 per complex problem
- Latency: 2 seconds \rightarrow 20 minutes
- Bottom line: Can we do better than just 'generate more and hope'?

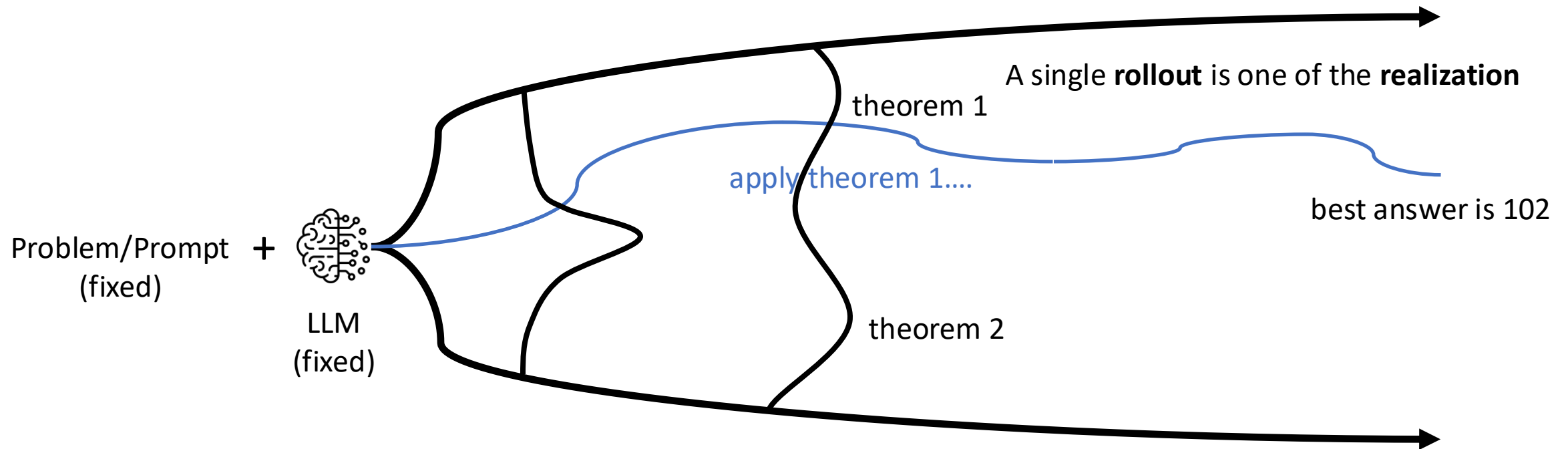
What is TTS (Test-Time Scaling)?

- Unlike human reasoning, LLM generates thoughts by decoding tokens one by one
- Decoding is stochastic (except for temp=0), stochasticity matters



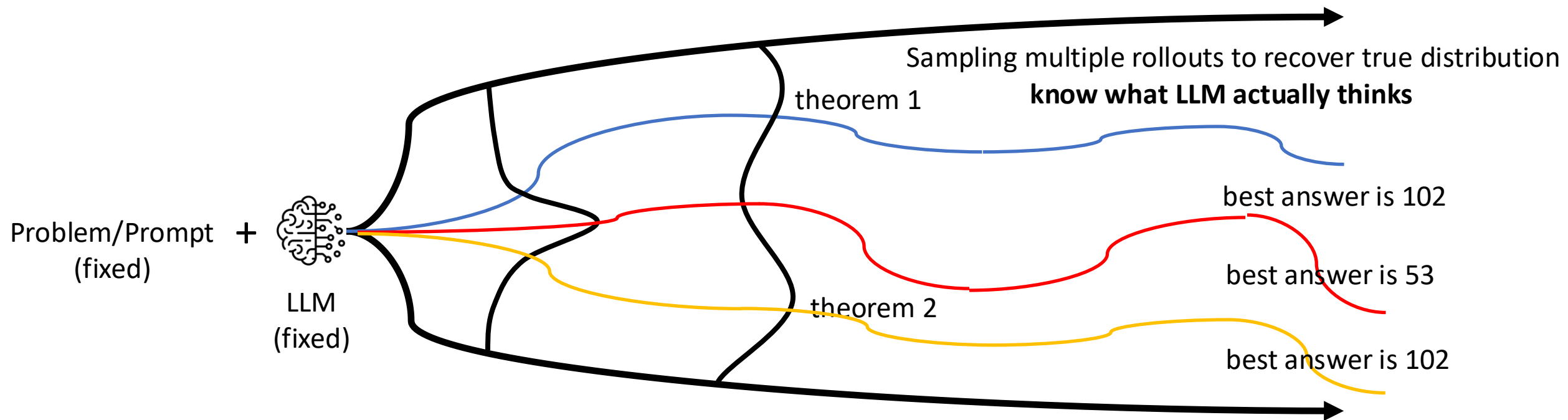
What is TTS (Test-Time Scaling)?

- Unlike human reasoning, LLM generates thoughts by decoding tokens one by one
- Decoding is stochastic (except for temp=0), stochasticity matters



What is TTS (Test-Time Scaling)?

- Unlike human reasoning, LLM generates thoughts by decoding tokens one by one
- Decoding is stochastic (except for temp=0), stochasticity matters



What is TTS (Test-Time Scaling)?

Additional compute/tokens during inference to improve reasoning

Self-Consistency

Generate multiple reasoning paths
(e.g., 8–64 attempts)

Use diverse sampling/temperature
per path

Vote on final answer across paths

Key: Diverse paths catch individual
errors

Self-consistency

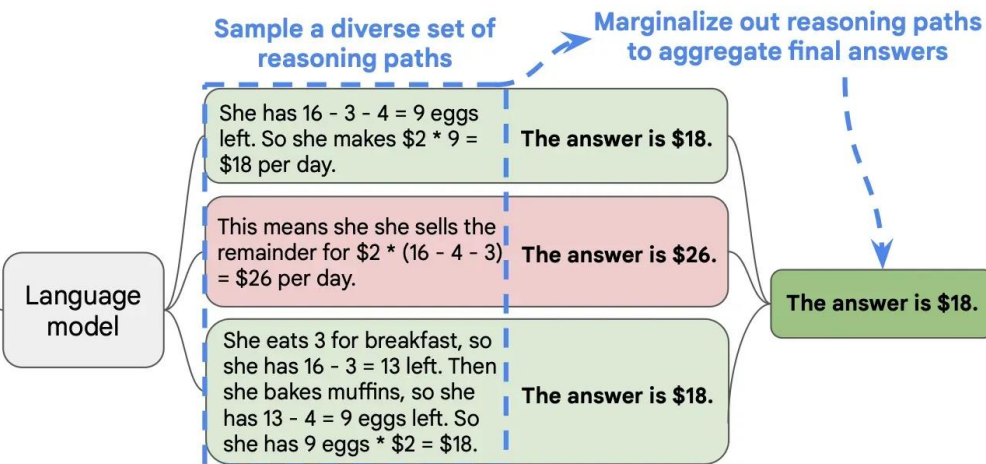
Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are 3 cars in the parking lot already. 2 more arrive. Now there are $3 + 2 = 5$ cars. The answer is 5.

...

Q: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for \$2 per egg. How much does she make every day?

A:



Parallel Thinking (Large-Scale Self-Consistency for fixed-form Answers)

Massive parallel generation (e.g., 512 attempts)

Majority voting on final answers

Key: More attempts \Rightarrow higher accuracy

Scaling: Run self-consistency at much larger scale

Advanced Method:

Deep Think by Gemini (multi-turn with aggregation)

Test-Time Scaling Wastes Computation

- **Hard problems: diminishing returns**
 - ~60% of traces fail early (low confidence)
 - ~25% start okay, derail mid-way
 - ~10% mostly correct with minor errors
 - ~5% high quality throughout
- ⇒ Many traces are hallucinated or random guesses

Like human reasoning, LLM can make mistakes for a single reasoning attempt

- **Easy problems: redundant solutions**
 - Dozens/hundreds of near-identical correct traces
 - First few traces suffice for consensus
- ⇒ Why generate 512 when ~8 would suffice?

Current scaling ignores problem difficulty and trace quality

Models Signal Uncertainty Through Token Distributions

- Token-level metrics:

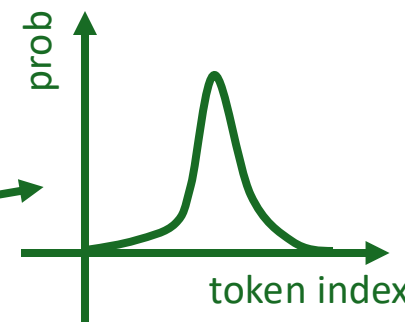
- Token Entropy: $H_i = -\sum P_i(j) \log P_i(j)$

- Token Confidence: $C_i = -\frac{1}{k} \sum_{j=1}^k \log P_i(j)$

- High-confidence (sharp) distributions \Rightarrow usually correct tokens



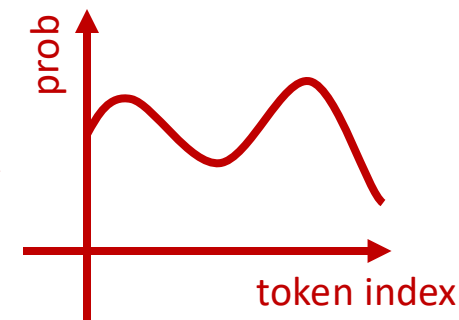
Let me think about this problem step by step. Step 1: Pythagorean triple formula: All primitive triples can be generated by $x = m^2 - n^2$



- Low-confidence (flat) distributions \Rightarrow model is uncertain about next step



Wait, let me double check my previous results...I should rethink step 1 again...Or I should think again about it.



Aggregating Token Uncertainty for Trace Quality Assessment

- From tokens to trace quality:
- Average Trace Confidence: $C_{avg} = (1/N) \sum C_i$ (self-certainty proposed by Kang et al.)
- Average over entire trace tokens

Question

Kylar went to the store to buy glasses for his new apartment. One glass costs \$5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

Correct Solution:

Kylar needs to pay 64 dollars for the 16 glasses, as each pair costs \$8 and he buys 8 pairs.

Wrong Step: Understanding the question as a geometric series.

Response 1: Reasoning 1 + Answer: 12.5 Self-Certainty: 17.13

Response 2: Reasoning 2 + Answer: 64 Self-Certainty: 16.94

Response 3: Reasoning 3 + Answer: 64 Self-Certainty: 16.36

Response 4: Reasoning 4 + Answer: 50 Self-Certainty: 16.21




Response 5: Reasoning 5 + Answer: 50 Self-Certainty: 16.13

Response 6: Reasoning 6 + Answer: 50 Self-Certainty: 15.87

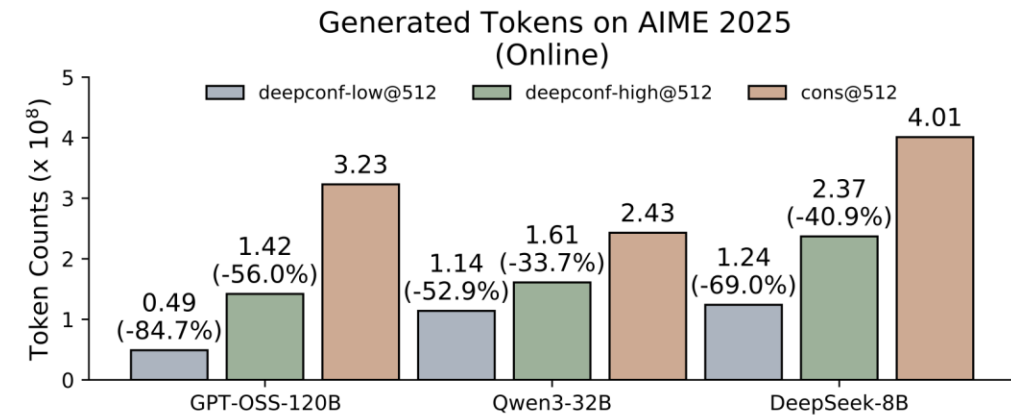
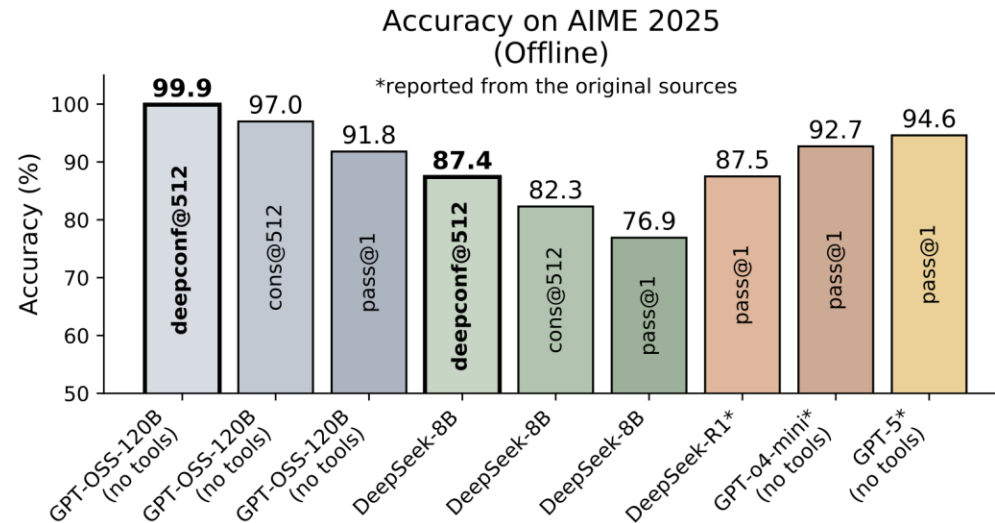
Wrong Step: Calculating the remaining 15 glasses at \$3 each.

Limitation: global averaging can miss local failures

Key Questions Left

-  Better Confidence Measurement:
 - Global trace-level uncertainty is insufficient
 - How to track local uncertainty in reasoning steps?
-  Smart Early Stopping:
 - How to identify and stop unpromising traces early?
 - Detect failures in real-time to save compute
-  Adaptive Compute Allocation:
 - Allocate less compute to easy problems, more to hard ones
 - Move beyond fixed budgets to dynamic resource allocation

DeepConf: Deep Think with Confidence



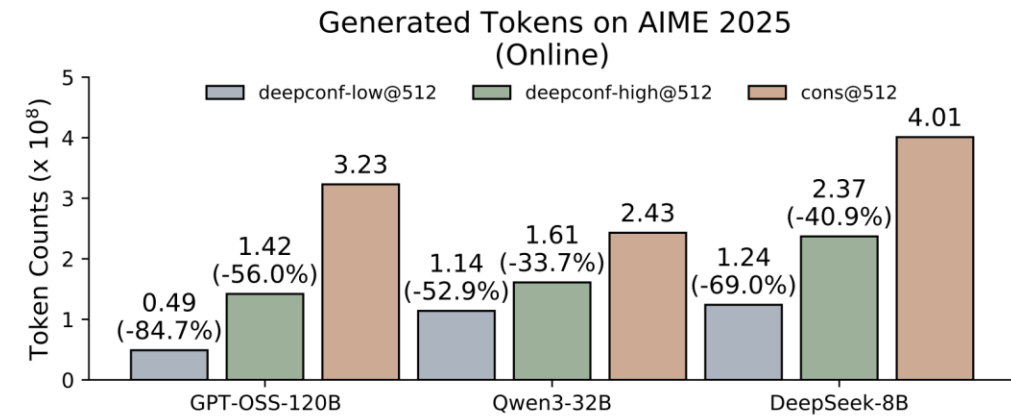
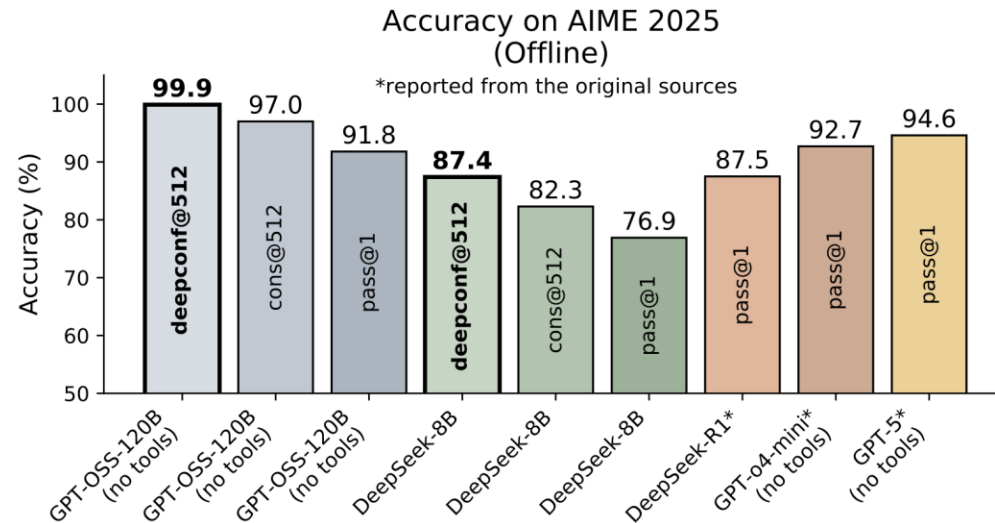
Methodology:

- 1) New Confidence Measures: group confidence, bottom-10%, tail confidence
- 2) Offline Mode: filter & confidence-weighted voting on full traces
- 3) Online Mode: real-time monitoring, early stop low-confidence traces

Better accuracy with dramatically fewer tokens

Break?

DeepConf: Deep Think with Confidence

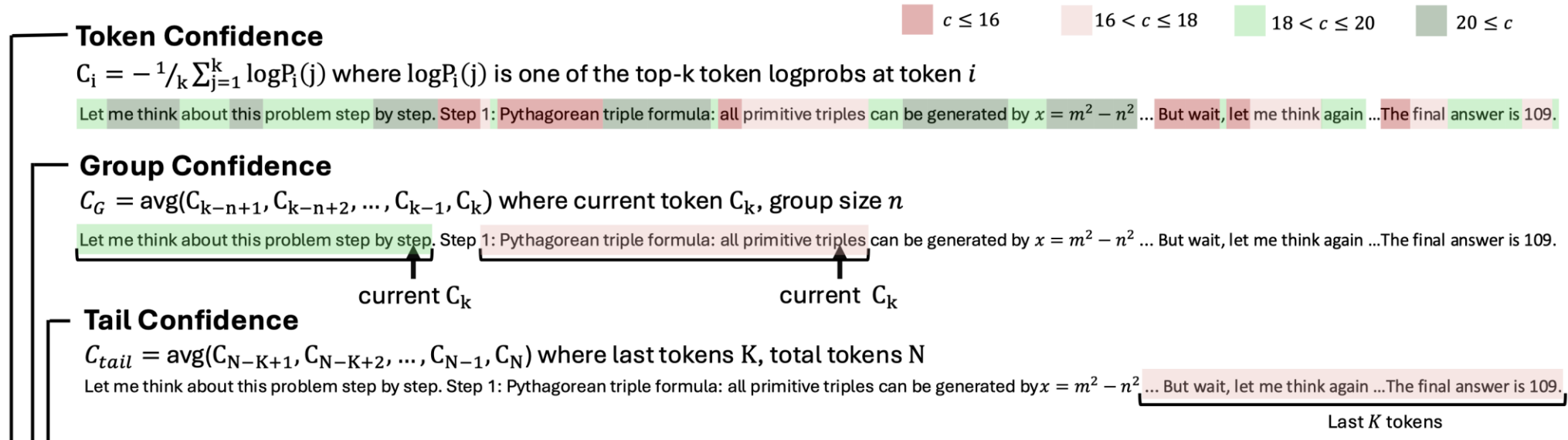


Methodology:

- 1) New Confidence Measures: group confidence, bottom-10%, tail confidence
- 2) Offline Mode: filter & confidence-weighted voting on full traces
- 3) Online Mode: real-time monitoring, early stop low-confidence traces

Better accuracy with dramatically fewer tokens

Confidence Metrics for Tracking Local Changes



Local patterns outperform global averages for detecting failures

Better Trace Quality Estimation

Token Confidence

$C_i = -1/\sum_{j=1}^k \log P_1(j)$ where $\log P_1(j)$ is one of the top-k token logprobs at token i

Let me think about this problem step by step. Step 1: Pythagorean triple formula: all primitive triples can be generated by $x = m^2 - n^2$... But wait, let me think again ... The final answer is 109.

$c \leq 16$ $16 < c \leq 18$ $18 < c \leq 20$ $20 \leq c$

Group Confidence

$C_G = \text{avg}(C_{k-n+1}, C_{k-n+2}, \dots, C_{k-1}, C_k)$ where current token C_k , group size n

Let me think about this problem step by step. Step 1: Pythagorean triple formula: all primitive triples can be generated by $x = m^2 - n^2$... But wait, let me think again ... The final answer is 109.

current C_k

current C_k

Tail Confidence

$C_{tail} = \text{avg}(C_{N-K+1}, C_{N-K+2}, \dots, C_{N-1}, C_N)$ where last tokens K , total tokens N

Let me think about this problem step by step. Step 1: Pythagorean triple formula: all primitive triples can be generated by $x = m^2 - n^2$... But wait, let me think again ... The final answer is 109.

Last K tokens

Tail Conf: C_{tail}

Bottom 10% Group Conf: $C_{10} = \text{avg}(C_0, C_1, \dots, C_{k-1}, C_k)$

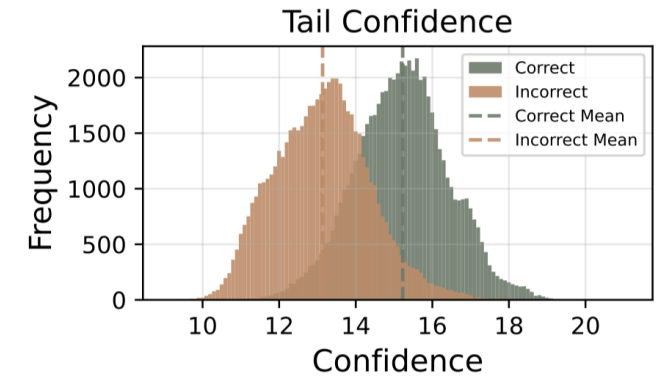
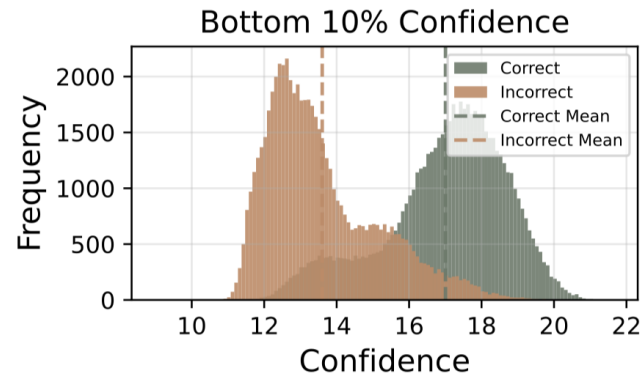
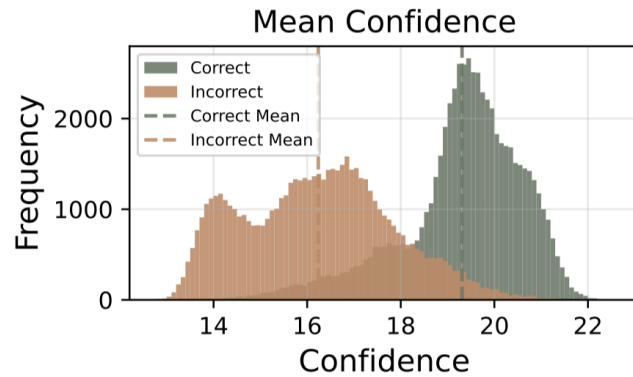
Bottom 10% of all C_G

Lowest Group Conf: $C_{lowest} = \min_{G_j \in G} C_{G_j}$

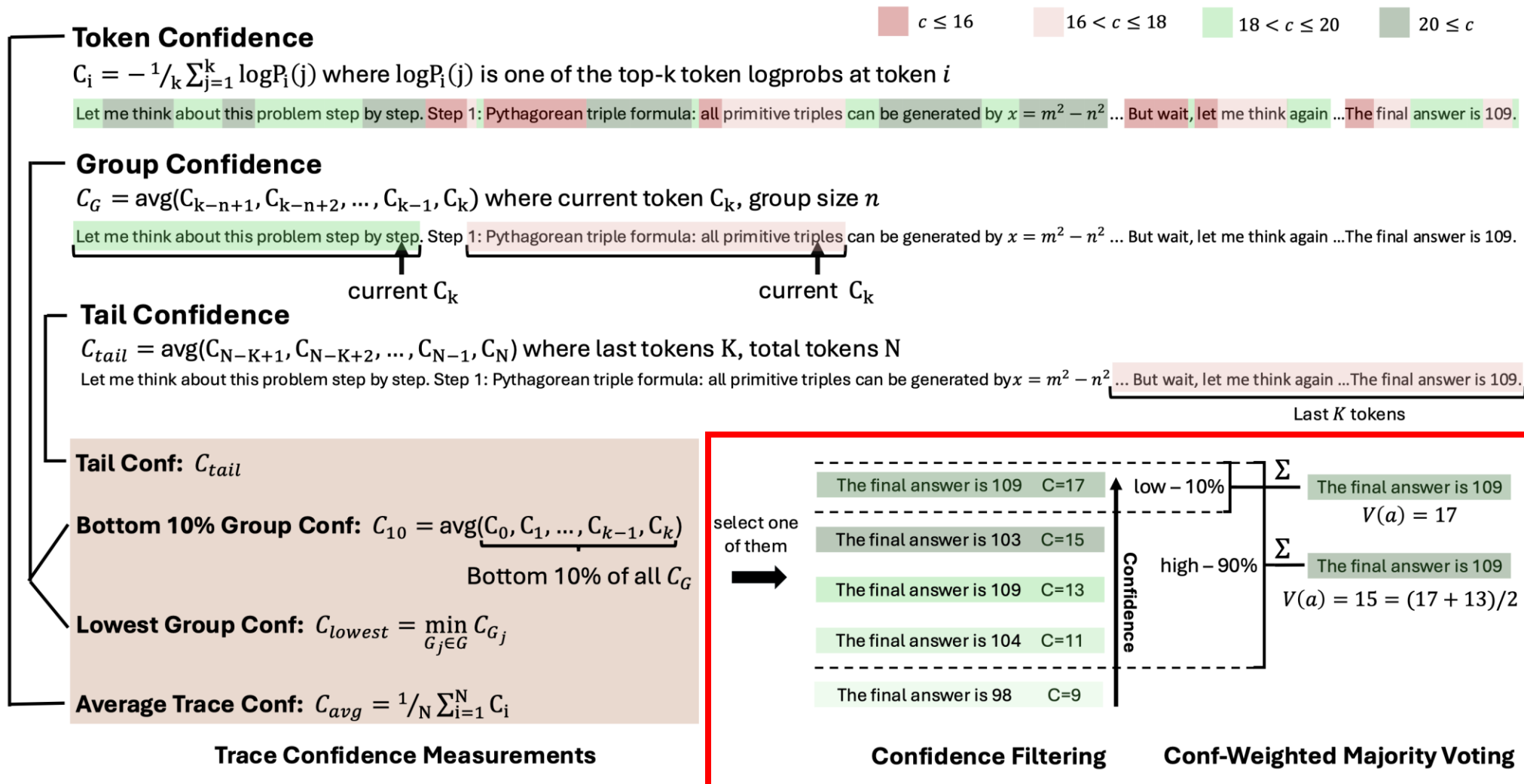
Average Trace Conf: $C_{avg} = 1/N \sum_{i=1}^N C_i$

Trace Confidence Measurements

Confidence distributions by different measures



Confidence Filtering



Choose Your Trade-off

- DeepConf-Low (Keep Top 10%): aggressive filtering; suitable for highly confident and correct problems
- DeepConf-High (Keep Top 90%): conservative filtering; only remove small amount of ultra low confident traces for better majority voting

Offline Mode: Better Voting with Full Traces

- Analyze completed traces for quality estimation
- Compute group confidence -> trace confidence (bottom-10%, or tail)
- Filter out low-confidence reasoning paths
- Weight votes by per-trace confidence quality

$$V(a) = \sum_{t \in T} C_t \cdot I(\text{answer}(t) = a)$$

- Emphasize high-confidence solutions in final decision

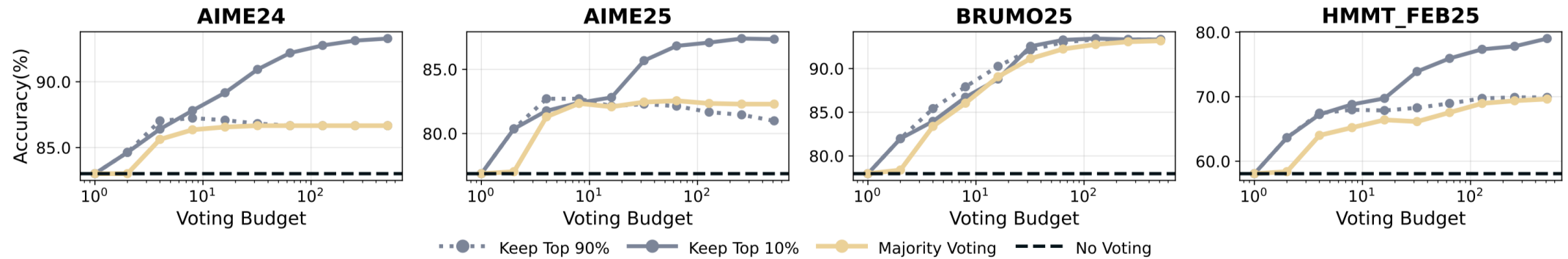
$$\hat{a} = \arg \max_a V(a)$$

Offline DeepConf

Model	Dataset	Pass @1	Cons @512	Mean @512	Bottom-10 Conf @512		Tail Conf @512	
Retention Ratio					90%	10%	90%	10%
DeepSeek-8B	AIME24	83.0	86.7	86.7	86.7	93.3	86.7	93.3
	AIME25	76.9	82.3	82.3	81.0	87.5	81.3	87.4
	BRUMO25	80.0	93.2	93.3	93.3	93.3	93.3	93.3
	HMMT25	58.1	69.6	69.9	69.9	79.5	69.9	83.9
	GPQA-D	62.8	72.5	72.5	71.2	70.6	72.8	74.0
Qwen3-32B	AIME24	80.6	85.3	85.7	86.0	90.8	86.8	89.4
	AIME25	71.7	80.1	80.0	80.1	80.2	80.1	80.2
	BRUMO25	78.0	93.3	93.3	93.3	93.3	93.3	91.2
	HMMT25	51.9	63.3	63.3	63.2	63.3	63.4	62.9
	GPQA-D	68.9	72.2	72.3	70.0	70.0	72.8	72.5
GPT-OSS-120B	AIME24	91.9	96.7	96.7	96.3	96.5	96.7	97.4
	AIME25	91.8	97.0	97.1	96.9	98.1	97.8	99.9
	BRUMO25	75.6	86.7	86.8	85.3	82.9	89.9	89.4
	HMMT25	78.9	92.9	92.9	92.9	90.5	92.9	88.9

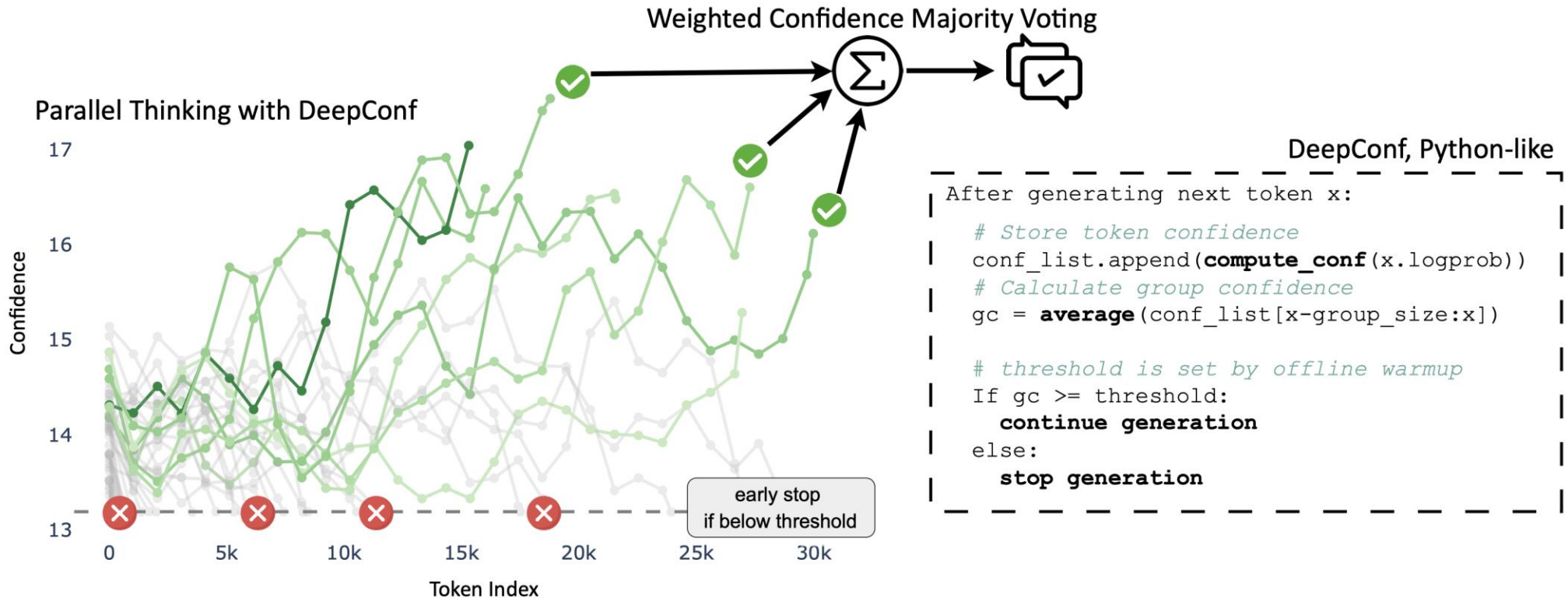
- AIME 2025 + GPT-OSS-120B: 99.9% accuracy (vs 97.0% MV baseline)

Offline DeepConf



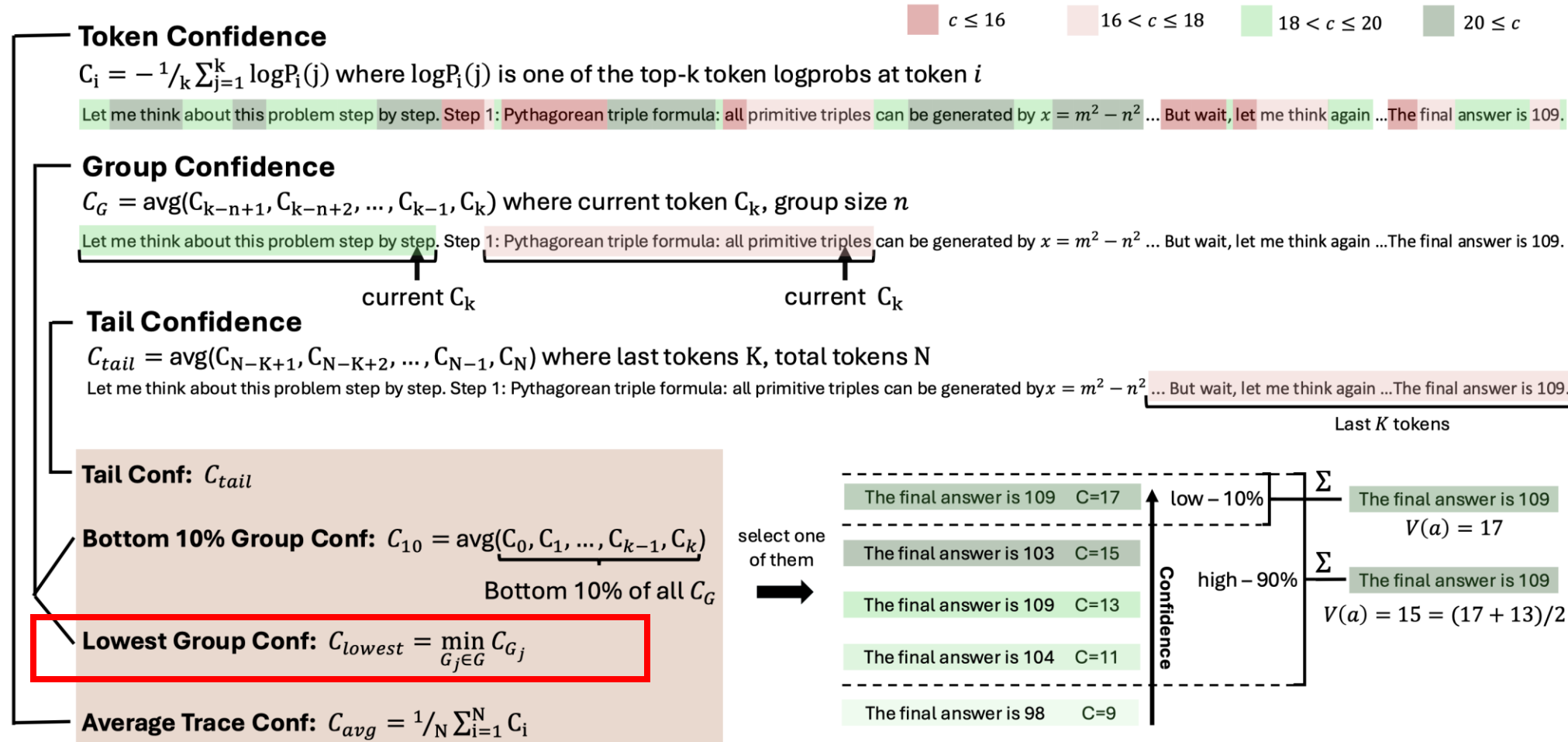
Consistent scaling for larger voting size compared to Majority Vote

Online DeepConf



- Early-Stopping based on threshold
- Simple detection during token generation

How to determine threshold?



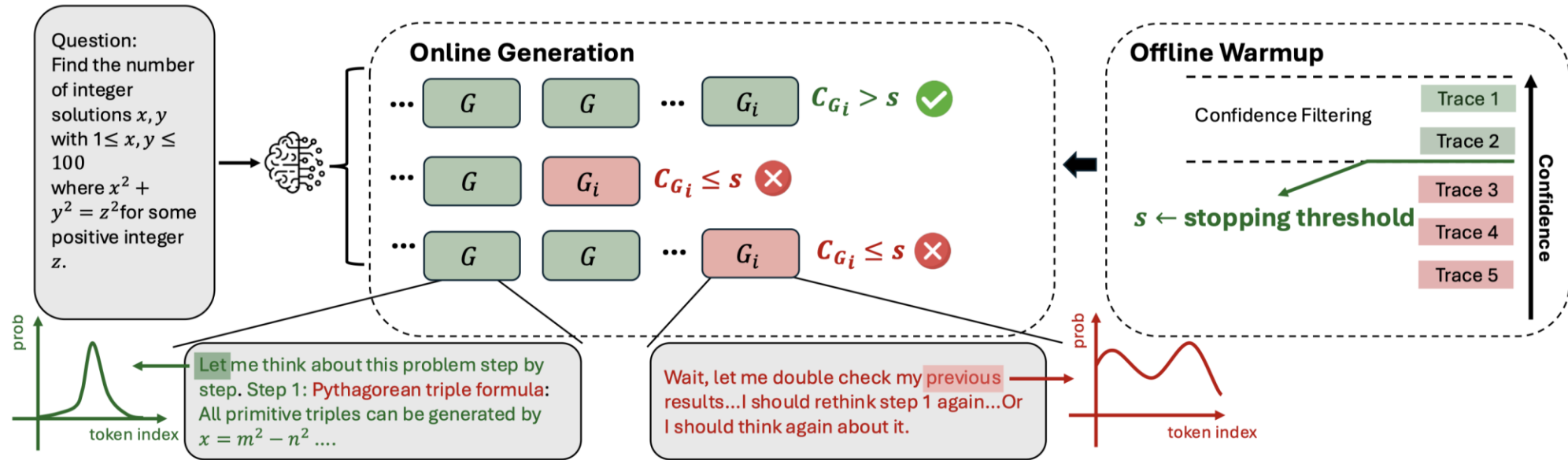
Trace Confidence Measurements

Change to Running Lowest Group Conf!

Confidence Filtering

Conf-Weighted Majority Voting

Online DeepConf



- Every prompt has own conf threshold determined by offline warmup
- Once threshold is set, start generation and early stopping in parallel

Adaptive Sampling

- Simple problem uses less traces, while hard problem uses more (max)

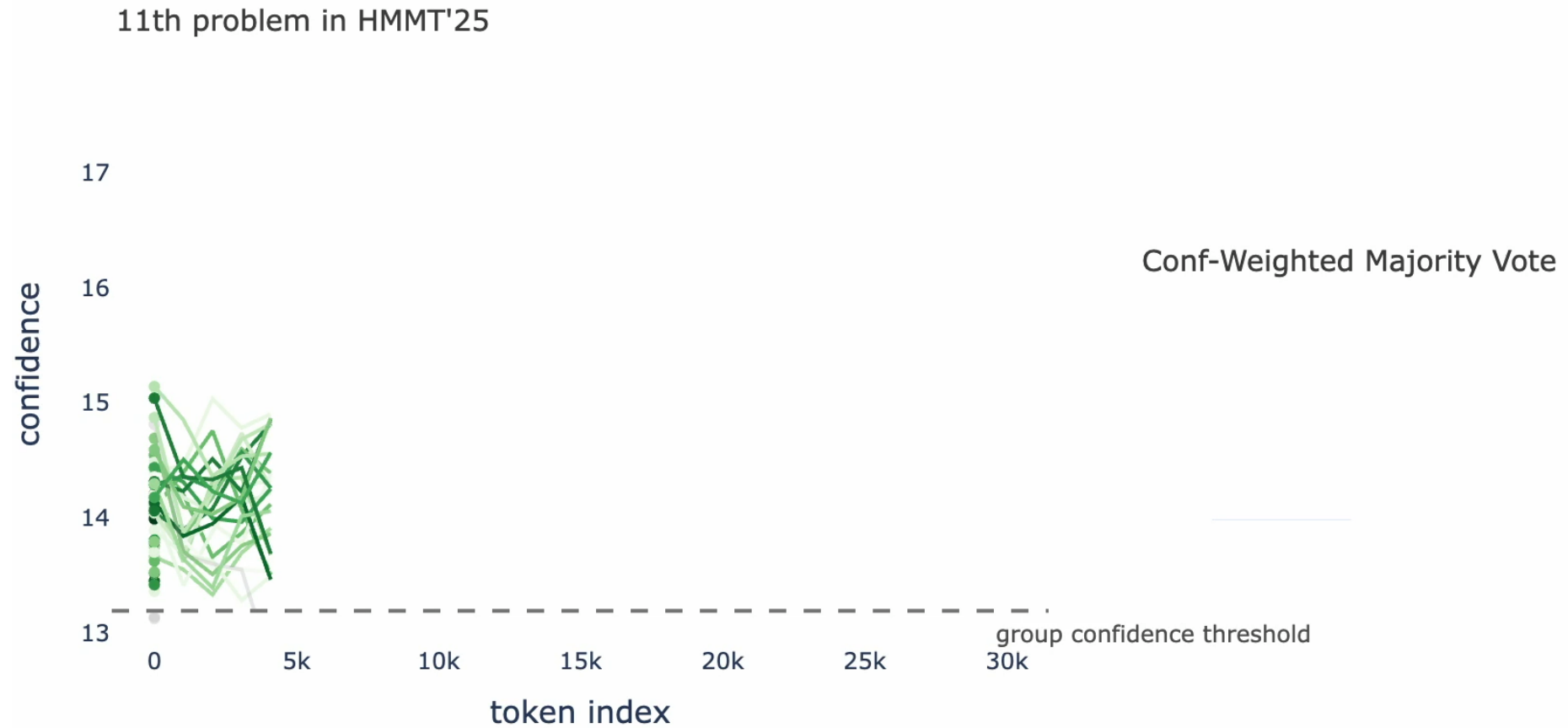
$$\beta = \frac{V(\hat{a})}{\sum_a V(a)}$$

- Pre-set a threshold \mathcal{T} (95%, uniform across models and datasets)
- $\beta < \mathcal{T}$ -> model does not reach a consensus, keep generation

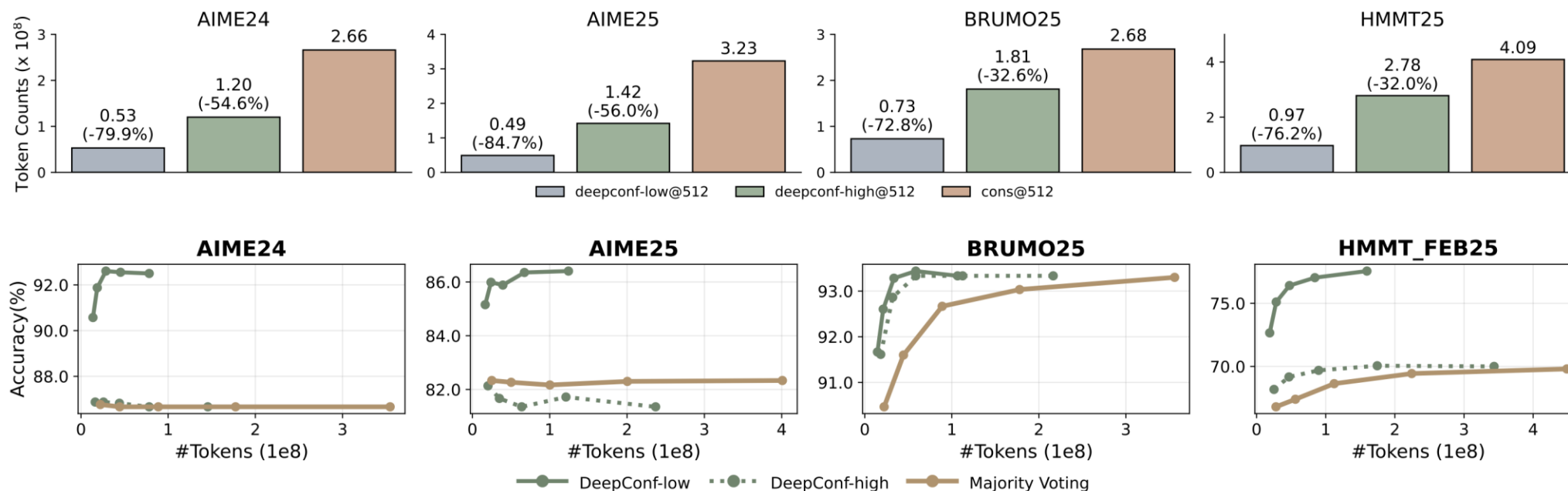
How Online DeepConf Works

- Warmup: generate 16 traces → set threshold
- For each new trace: monitor 2048-token confidence window
- If confidence < threshold: stop; else continue
- Stop all generation once consensus ratio is reached among completed high-confidence traces

Online DeepConf



Online DeepConf on GPT-OSS-120B



- Token usage reductions across datasets: -56% to -84.7%
- HMMT 2025: -76% tokens; BRUMO: -73% tokens
- Often improved accuracy while using fewer tokens

Benefits for real deployment

- Efficient Parallel Thinking Method
 - Up to ~70% reduction in inference costs
 - often better than baseline
 - 10-30X more costs than single request (relatively tolerable)
 - Make parallel thinking work in practice
- Better Self-Consistency
 - Voting based on confident traces only

Easy to Deploy

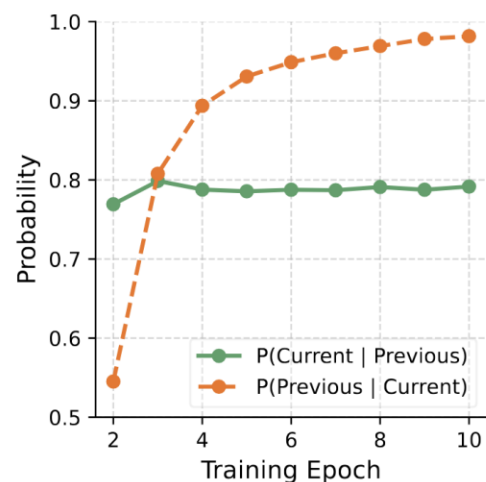
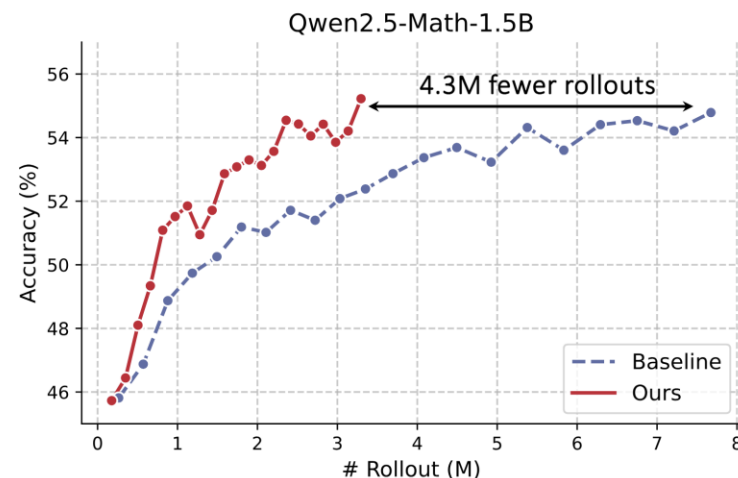
- Minimal code changes (~50 lines) in serving framework
- Works with existing frameworks (e.g., vLLM) (Working with others as well)
- No model training / hyperparameter tuning required
- Simple but effective

Available Now

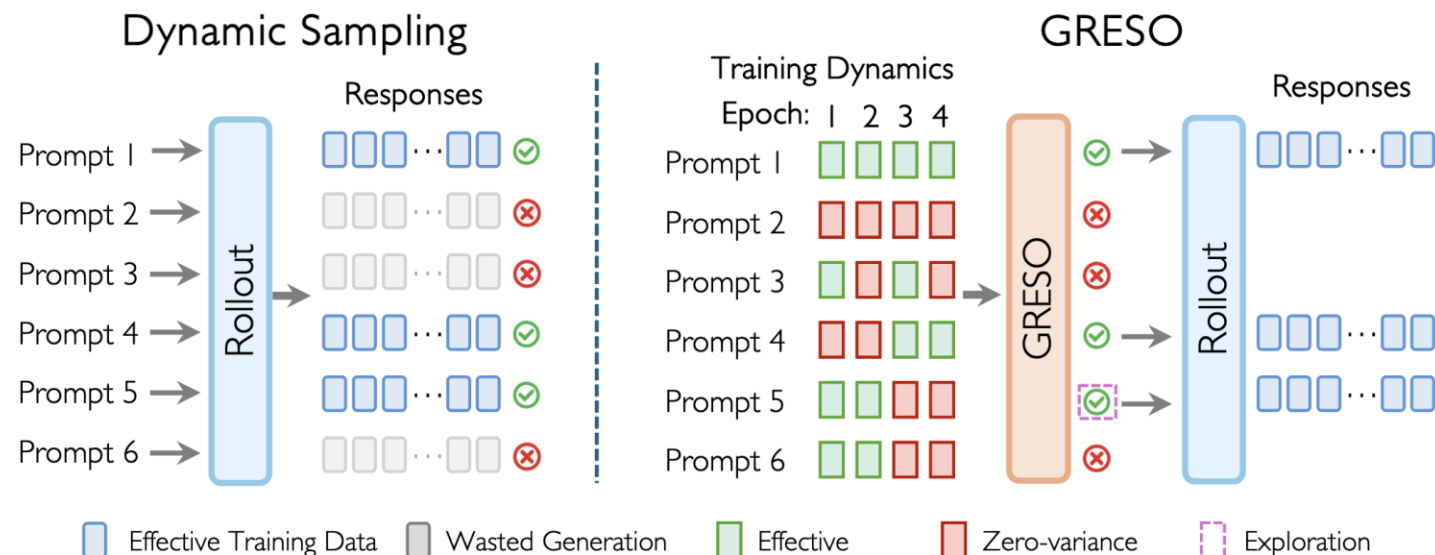
- Project Page: jiaweizzhao.github.io/deepconf
- Code: github.com/facebookresearch/deepconf

Efficient Reinforcement Learning

- GRESO (GRPO with Efficient Selective Rollout)
- “Act only when it pays”



(a)



(b)

Future Work

- Freeform reasoning: confidence-weighted majority voting; dynamic parallel thinking
- Fix '**confident but wrong**': RL + reward modeling; penalize high confidence + low accuracy

Thank you!