Adaptive Generate-Rank-Verify: Inference-Time Search with Costly Verification

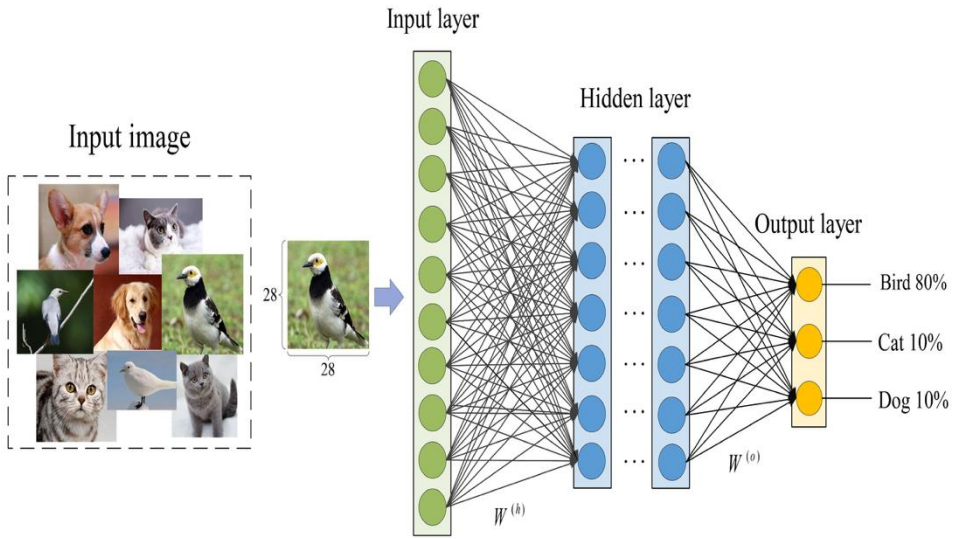
Mahdi Haghifam
TTIC

Shaddin Dughmi
(University of Southern California)

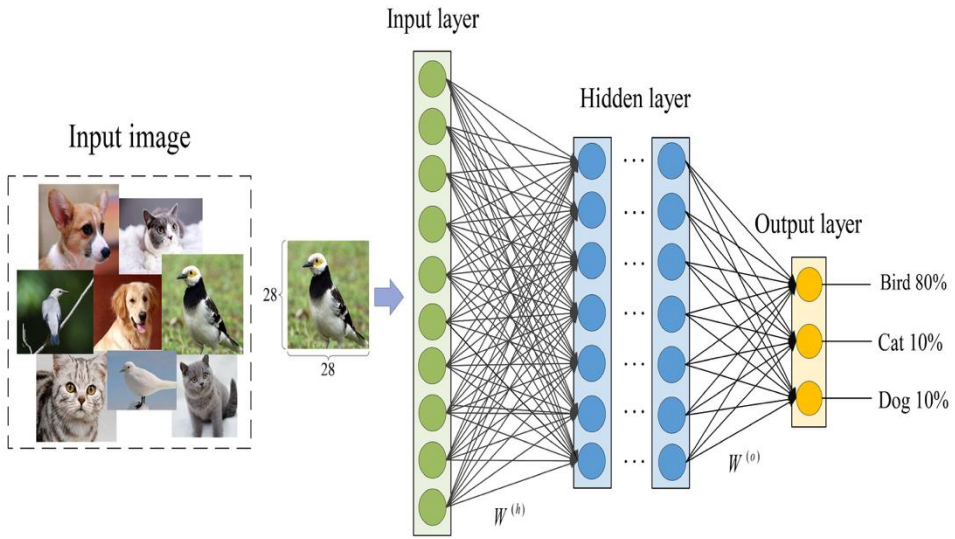
Yusuf Hakan Kalaycı
(University of Southern California)

Inference-Time Algorithm in LLMs

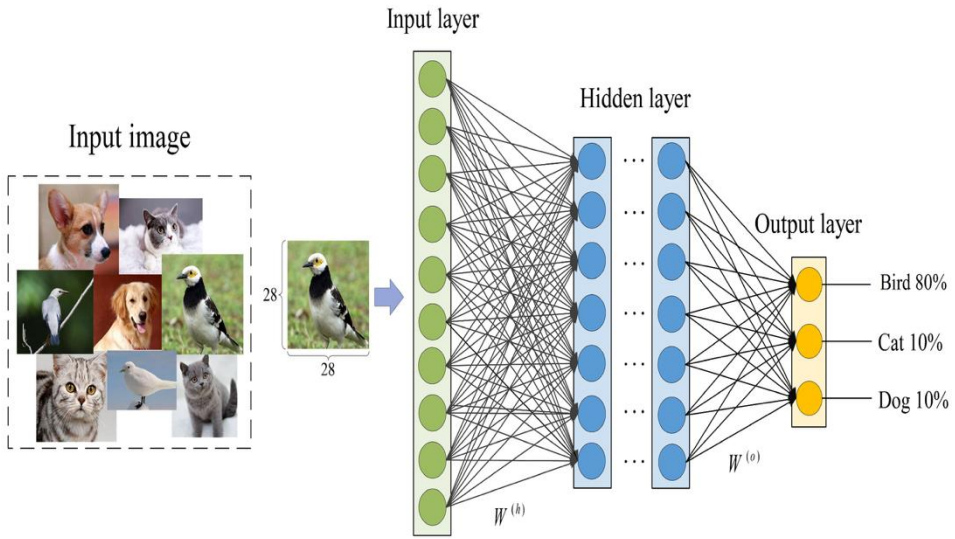
Inference-Time Algorithm in LLMs



Inference-Time Algorithm in LLMs



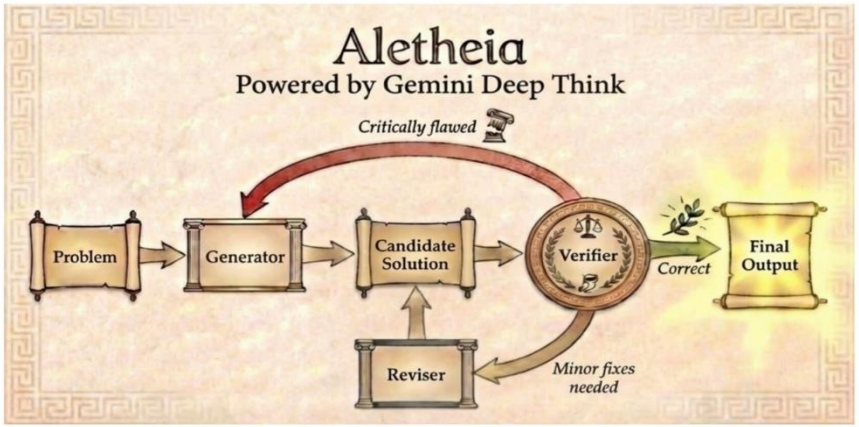
Inference-Time Algorithm in LLMs



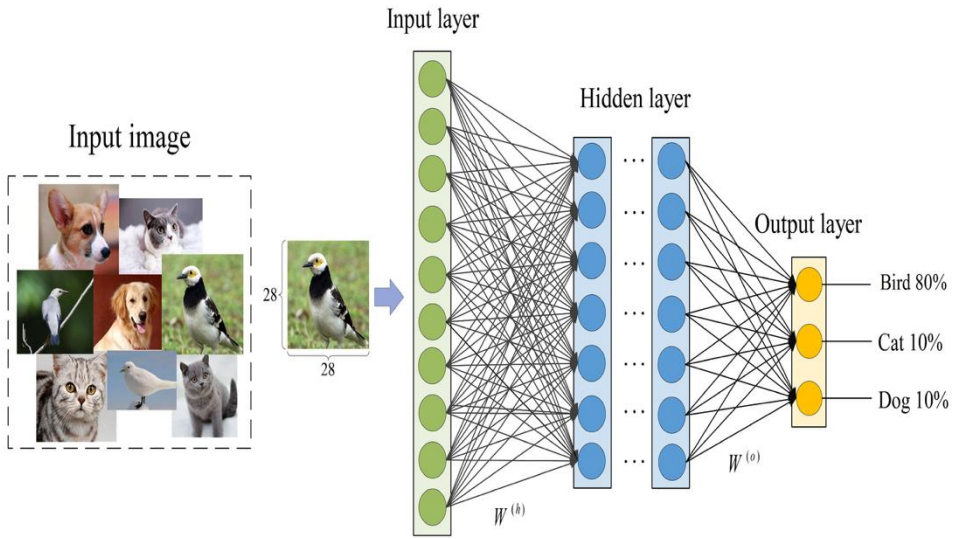
Google DeepMind 2026-2-11

Towards Autonomous Mathematics Research

Tony Feng*†, Trieu H. Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung†, Joonkyung Lee†, Carlo Pagano†, Sang-hyun Kim†, Federico Pasqualotto†, Sergei Gukov†, Jonathan N. Lee, Junsu Kim, Kaiying Hou, Golnaz Ghiasi, Yi Tay, YaGuang Li, Chenkai Kuang, Yuan Liu, Hanzhao (Maggie) Lin, Evan Zheran Liu, Nigamaa Nayakanti, Xiaomeng Yang, Heng-tze Cheng, Demis Hassabis, Koray Kavukcuoglu, Quoc V. Le*, Thang Luong*
*Project leads, †Mathematicians. Work conducted under Google DeepMind.



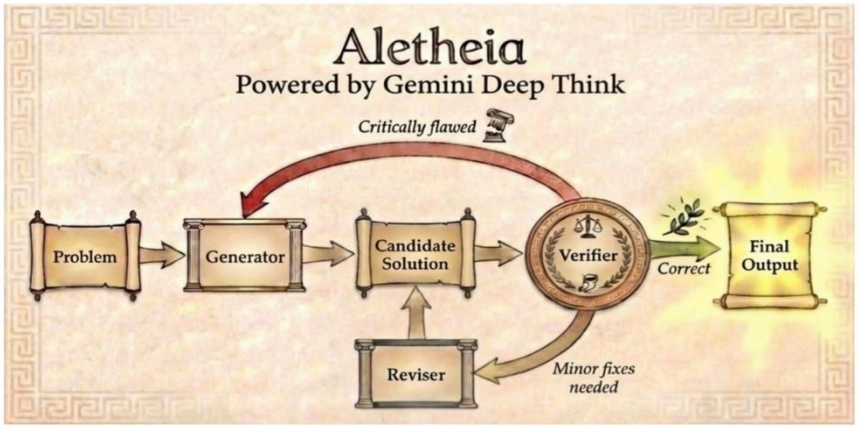
Inference-Time Algorithm in LLMs



Google DeepMind 2026-2-11

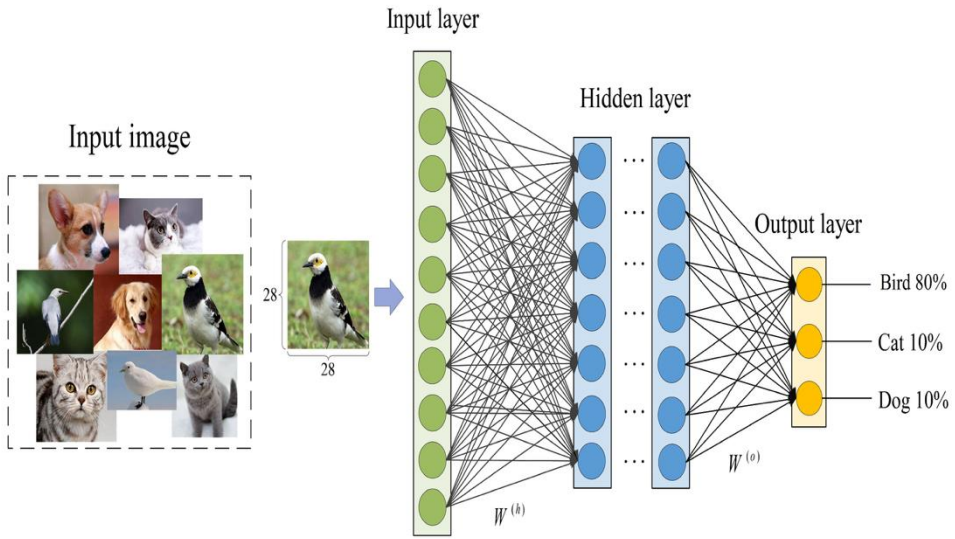
Towards Autonomous Mathematics Research

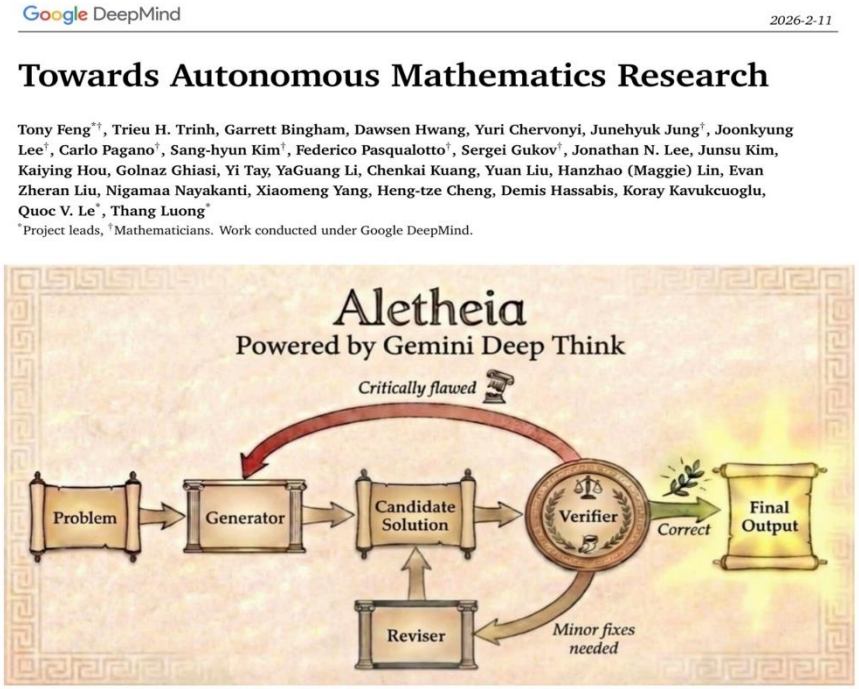
Tony Feng*†, Trieu H. Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung†, Joonkyung Lee†, Carlo Pagano†, Sang-hyun Kim†, Federico Pasqualotto†, Sergei Gukov†, Jonathan N. Lee, Junsu Kim, Kaiying Hou, Golnaz Ghiasi, Yi Tay, YaGuang Li, Chenkai Kuang, Yuan Liu, Hanzhao (Maggie) Lin, Evan Zheran Liu, Nigamaa Nayakanti, Xiaomeng Yang, Heng-tze Cheng, Demis Hassabis, Koray Kavukcuoglu, Quoc V. Le*, Thang Luong*
*Project leads, †Mathematicians. Work conducted under Google DeepMind.



Using one or more pre-trained models as a subroutine
to solve an instance better during inference-time (or test-time)

Inference-Time Algorithm in LLMs



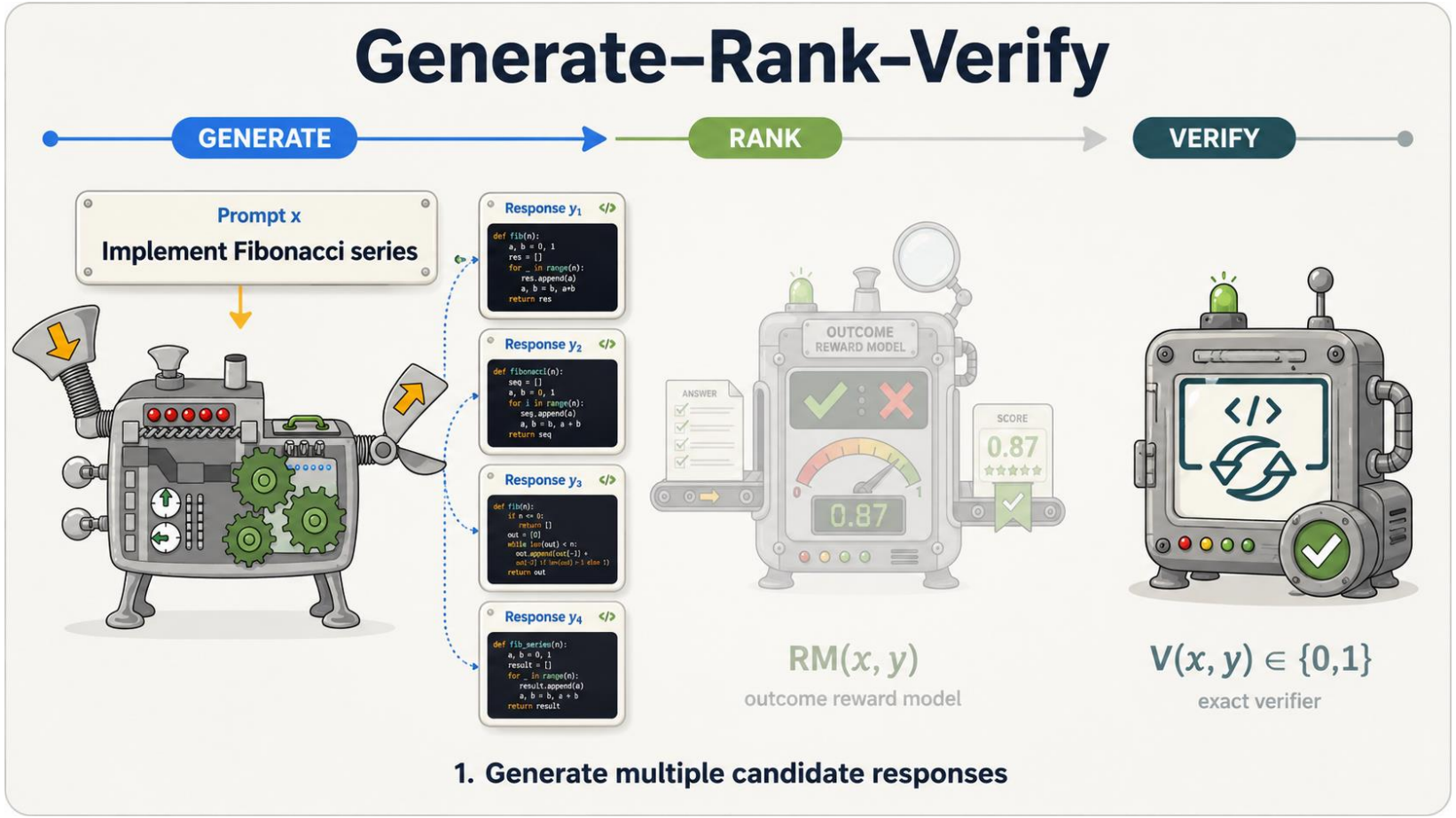


Using one or more pre-trained models as a subroutine
to solve an instance better during inference-time (or test-time)

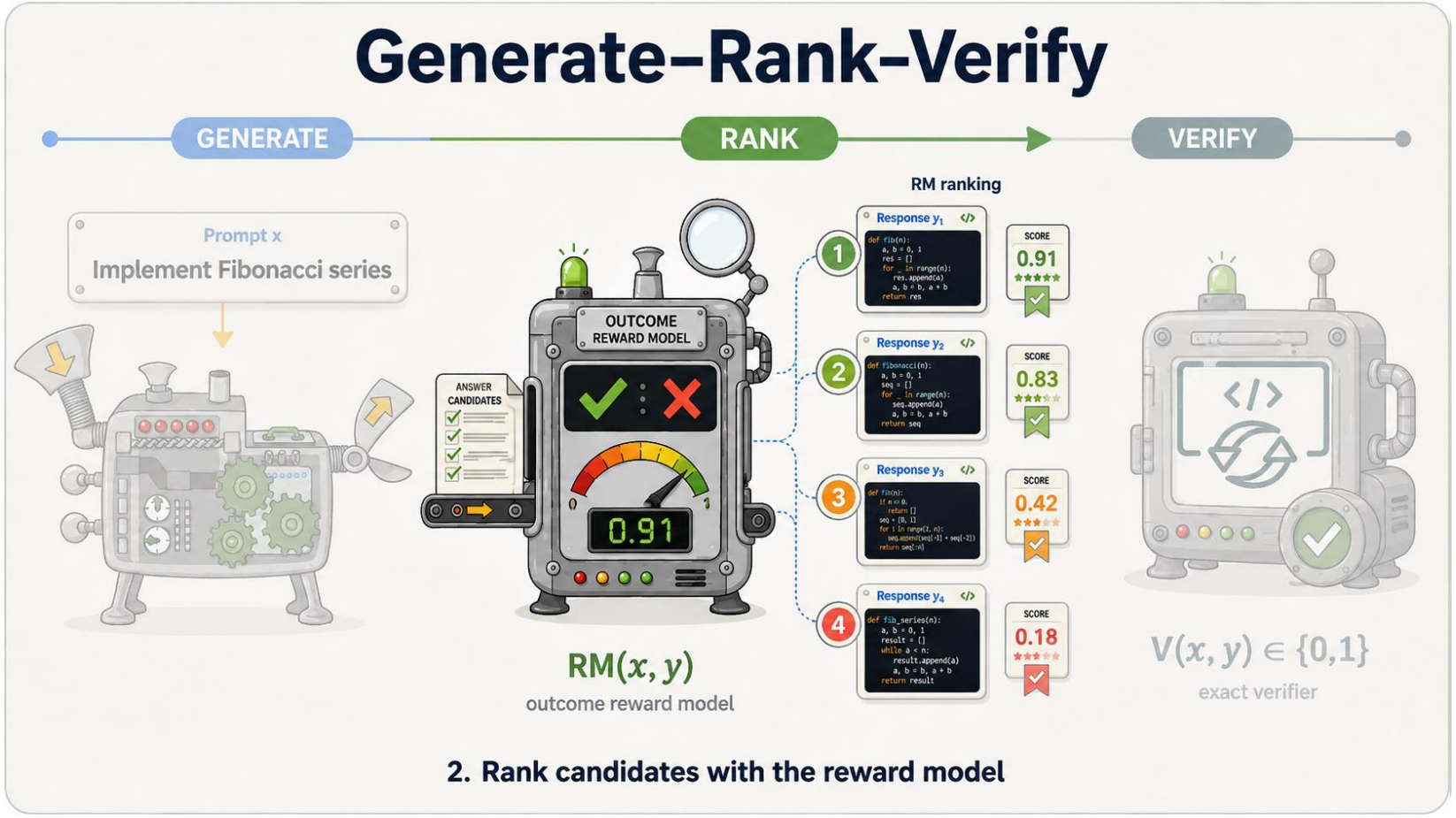
The landscape of the algorithm design is vast

best-of-N [Stiennon et al. '20, Nakano et al. '22], chain of thought [Wei et al. '22], self-consistency [Wang et al. '23], Tree of Thoughts [Yao et al. '23], outcome verifiers [Cobbe et al. '21], process reward models [Lightman et al. '24, Wang et al. '24], generative verifiers [Zhang et al. '25], execution-based selection [Li et al. '22, Chen et al. '23], test-time compute scaling [Snell et al. '25], …

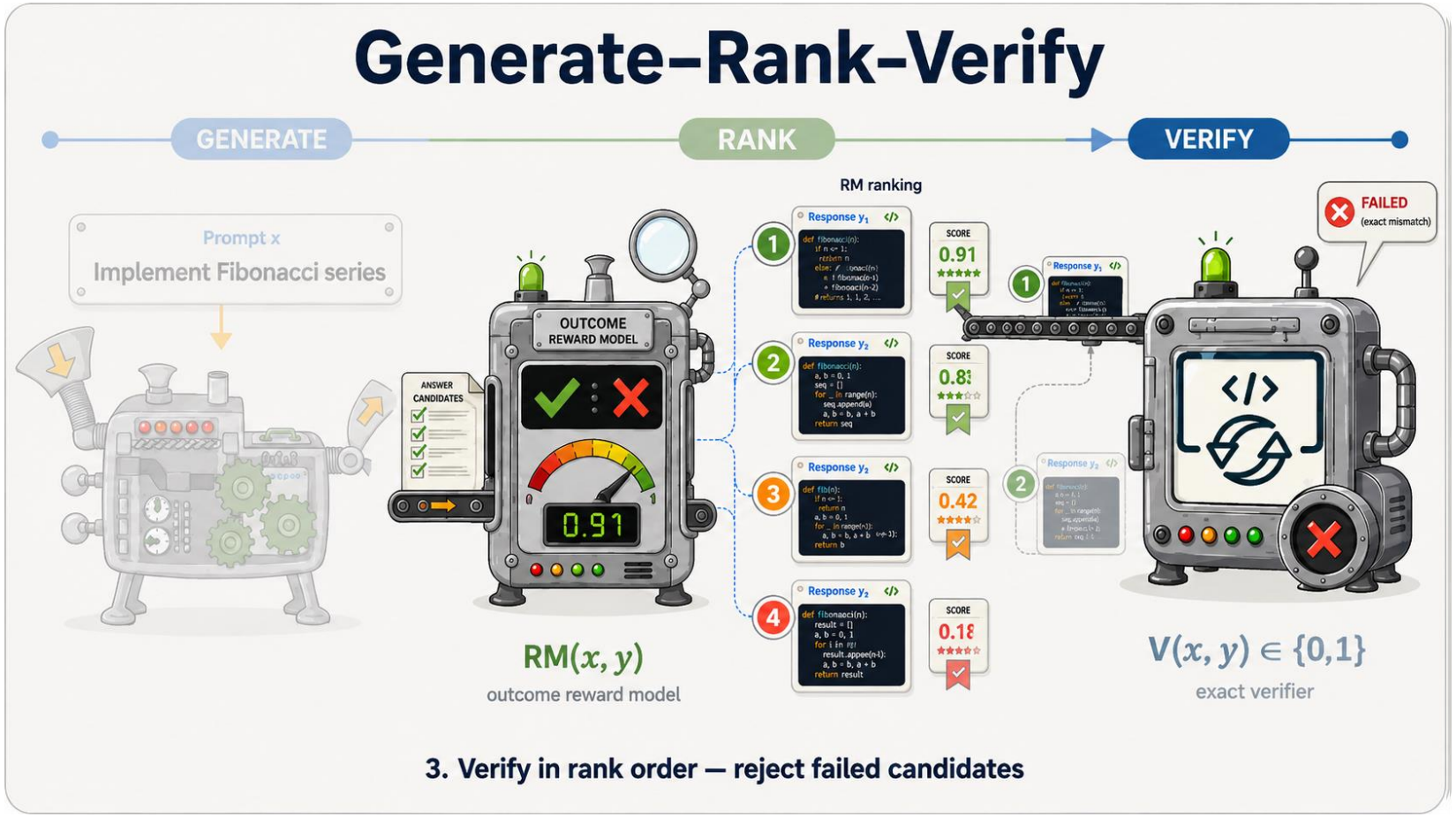
Generate-Rank-Verify Paradigm



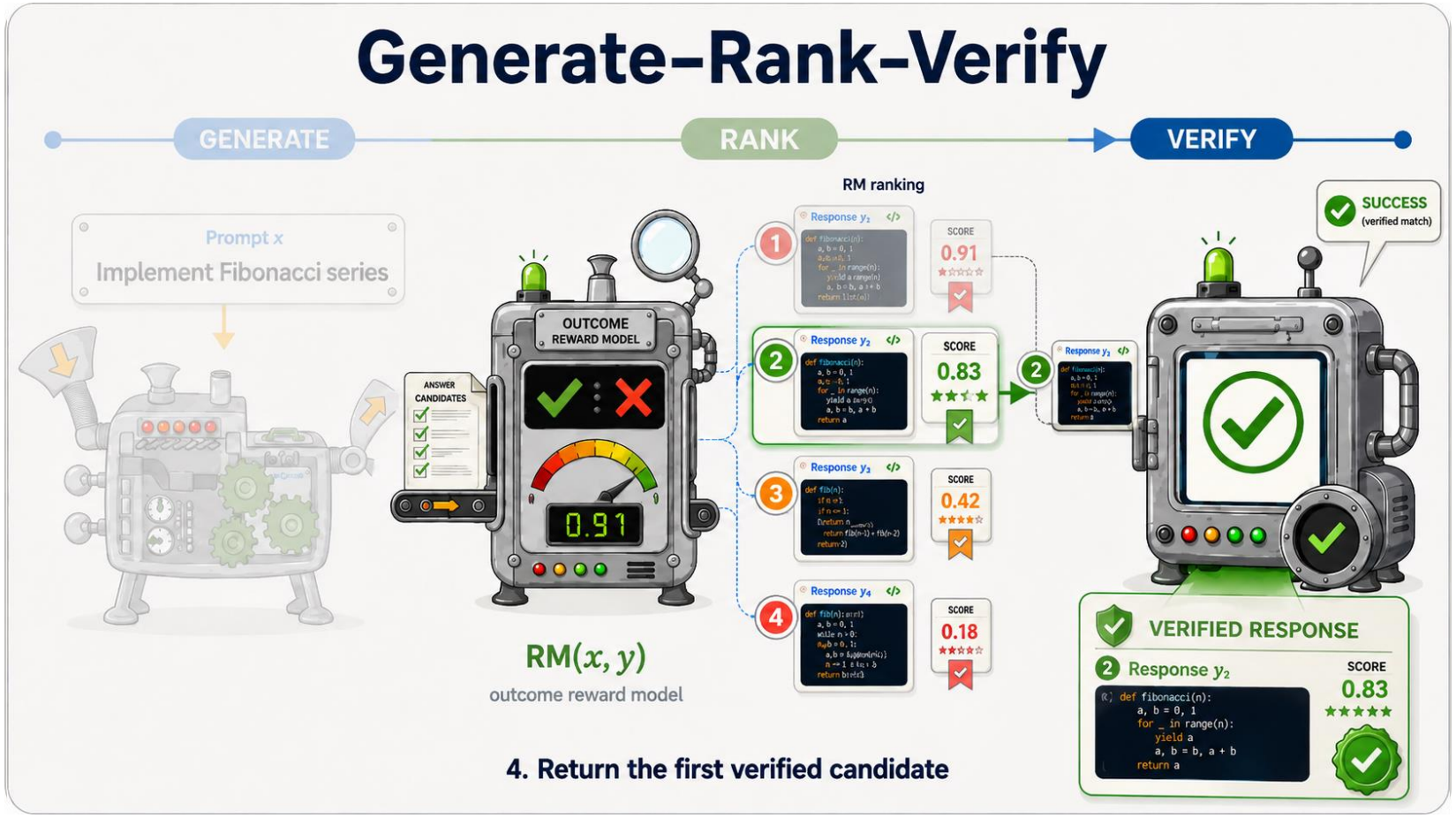
Generate-Rank-Verify Paradigm



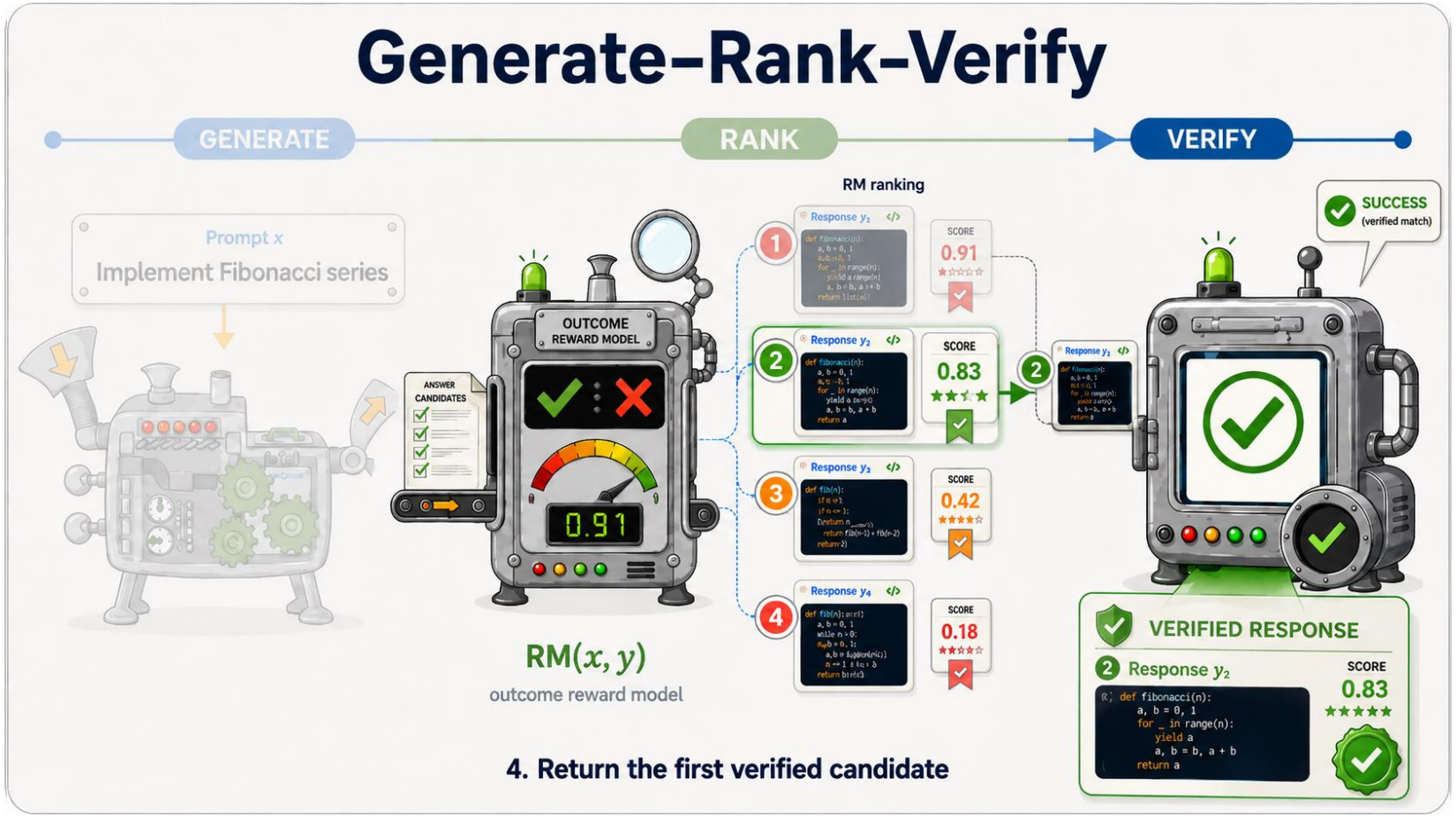
Generate-Rank-Verify Paradigm



Generate-Rank-Verify Paradigm



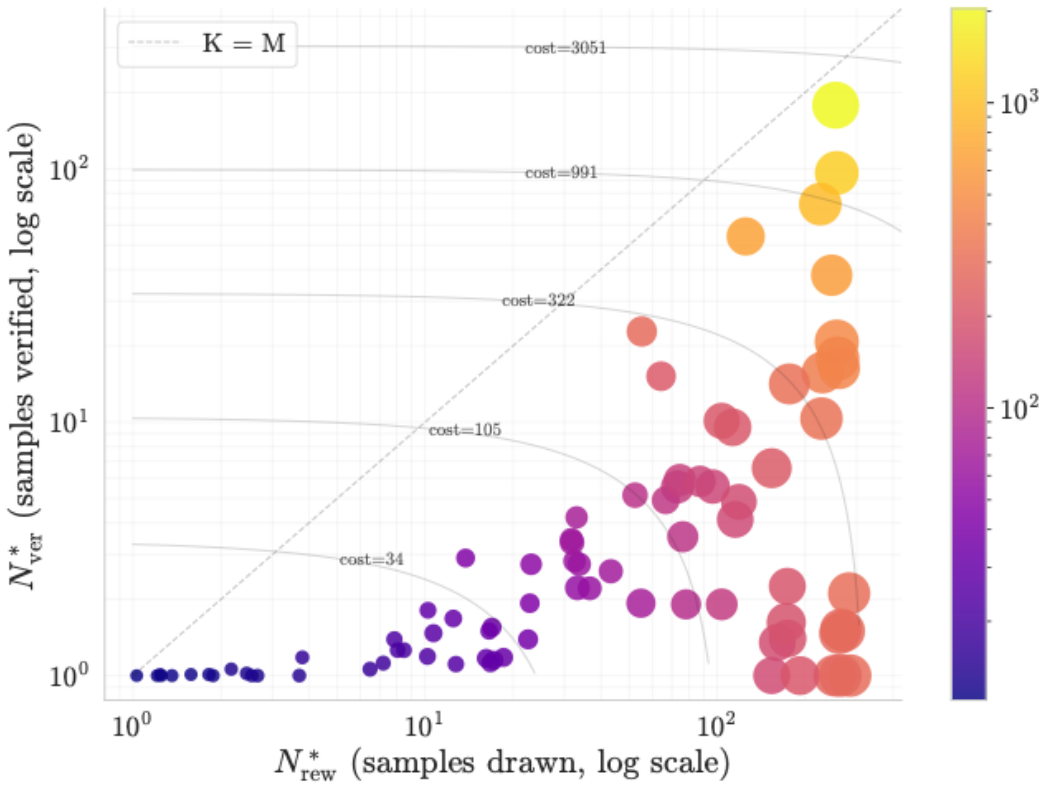
Generate-Rank-Verify Paradigm



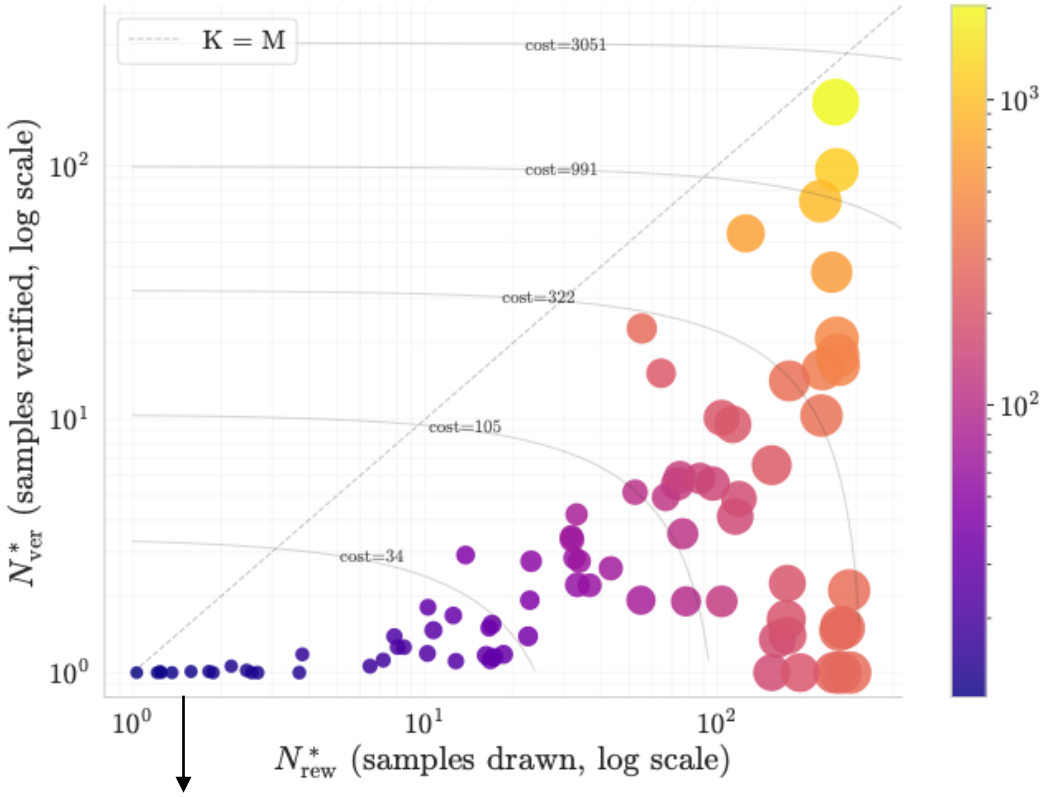
Generate N samples, Rank by reward model, and Verify top k

Adaptivity in Inference-Time Search

Adaptivity in Inference-Time Search



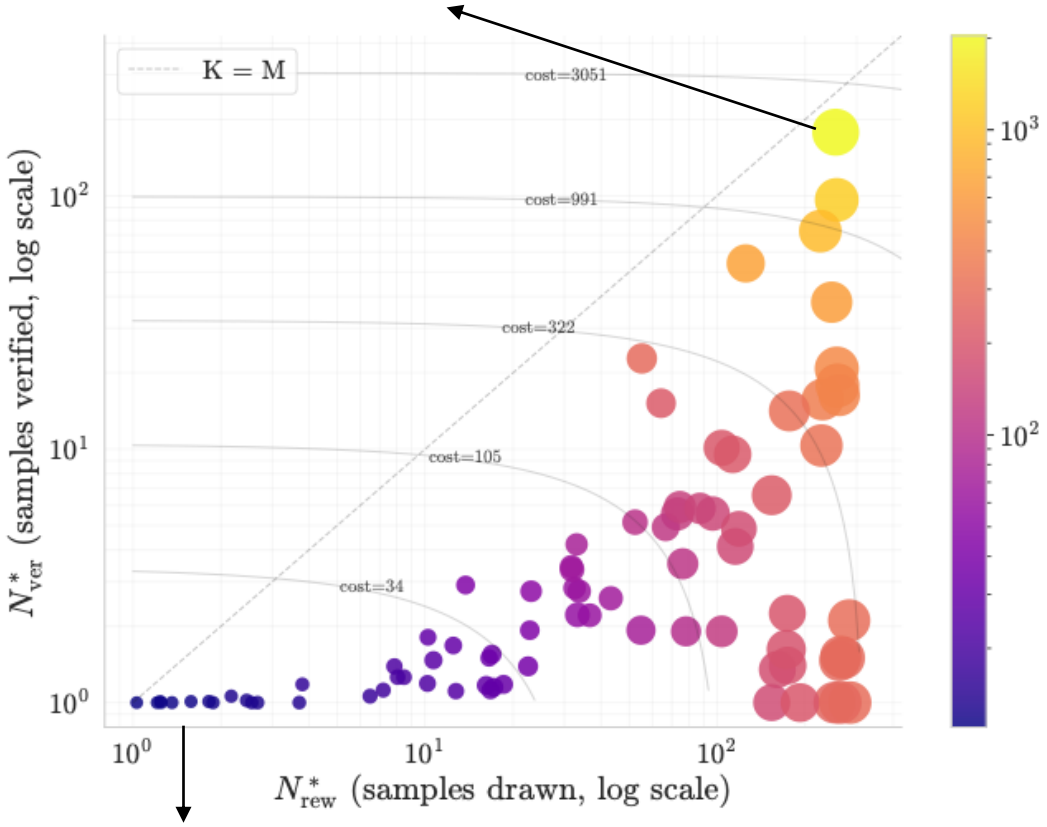
Adaptivity in Inference-Time Search



Generate 5 responses and verify the top

Adaptivity in Inference-Time Search

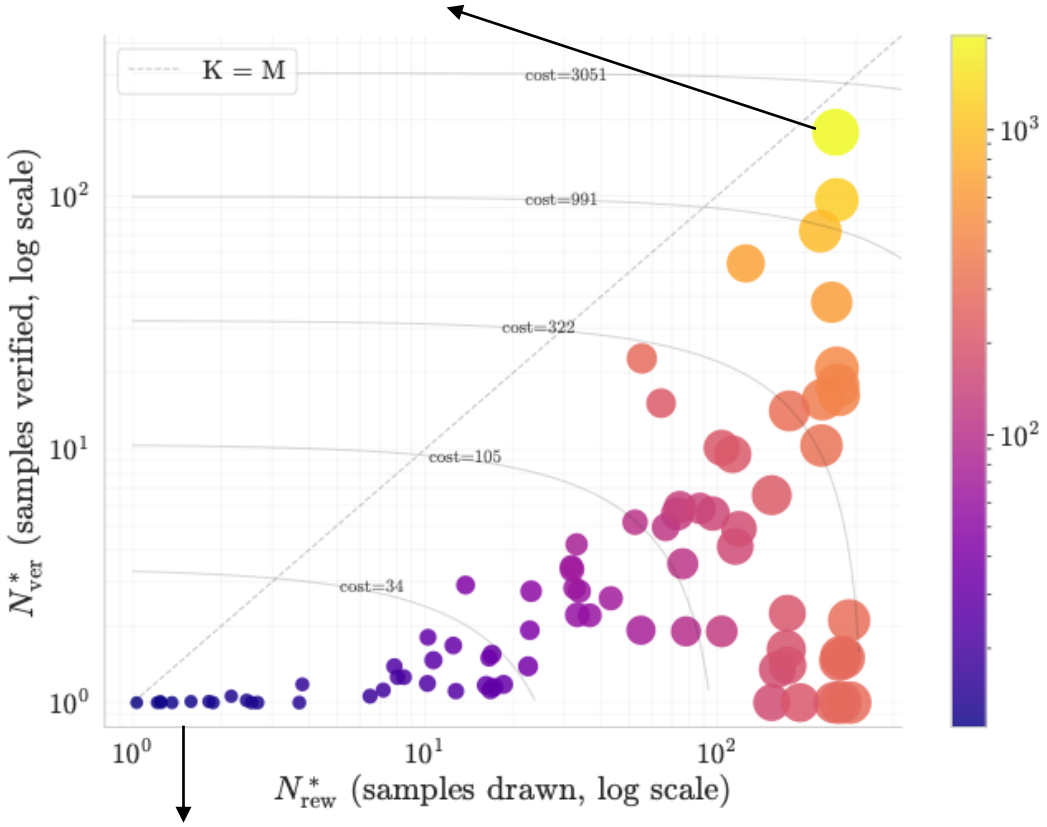
Generate 500 responses
and verify almost all of them



Generate 5 responses
and verify the top

Adaptivity in Inference-Time Search

Generate 500 responses
and verify almost all of them



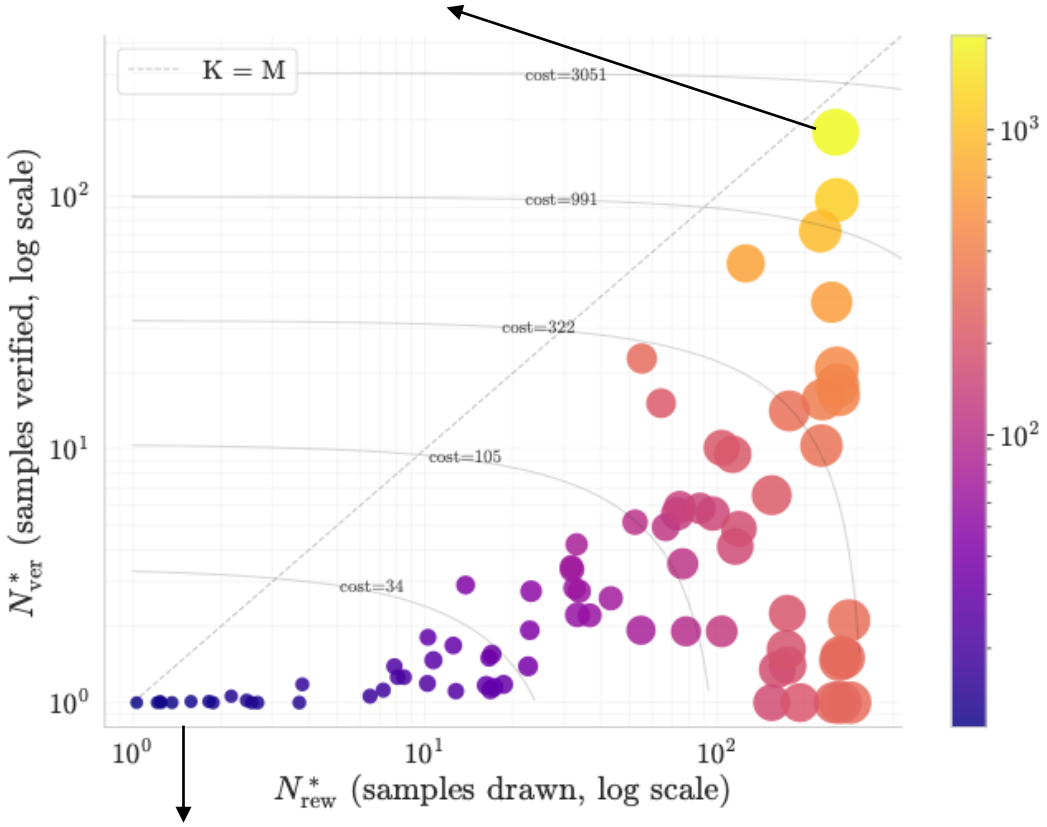
Generate 5 responses
and verify the top

Fixed budgets fail both ways

Overspend on the easy prompts, or run out on the hard ones.

Adaptivity in Inference-Time Search

Generate 500 responses
and verify almost all of them



Generate 5 responses
and verify the top

Fixed budgets fail both ways

Overspend on the easy prompts, or run out on the hard ones.

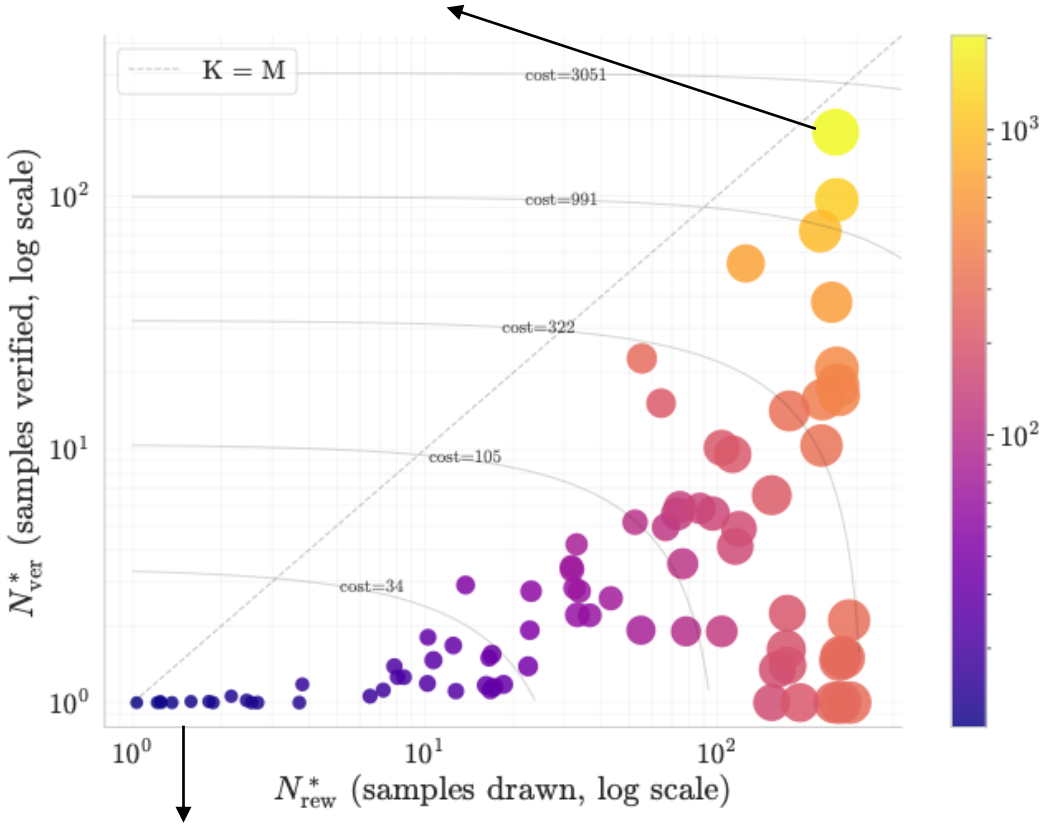
Predicting difficulty offline is not enough

It assumes a stable prompt distribution.

[Damani et al.'24, Snell et al.'25]

Adaptivity in Inference-Time Search

Generate 500 responses
and verify almost all of them



Generate 5 responses
and verify the top

Fixed budgets fail both ways

Overspend on the easy prompts, or run out on the hard ones.

Predicting difficulty offline is not enough

It assumes a stable prompt distribution.

[Damani et al.'24, Snell et al.'25]

Meta Question of This Talk:

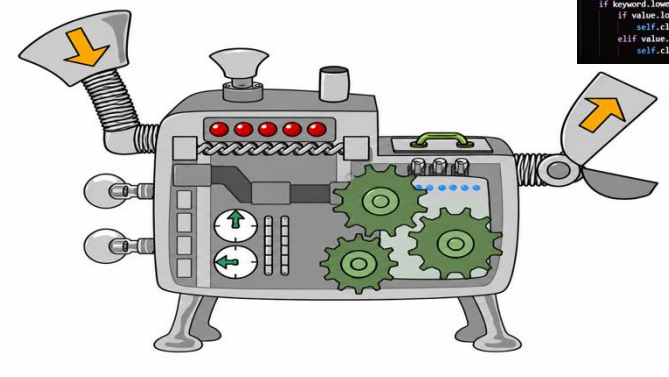
Can we decide the budget online, per prompt, from an uncalibrated reward model and still pay near-oracle cost?

Problem Formalization: Active Search

Problem Formalization: Active Search

Prompt x

Implement Fibonacci series



Response y

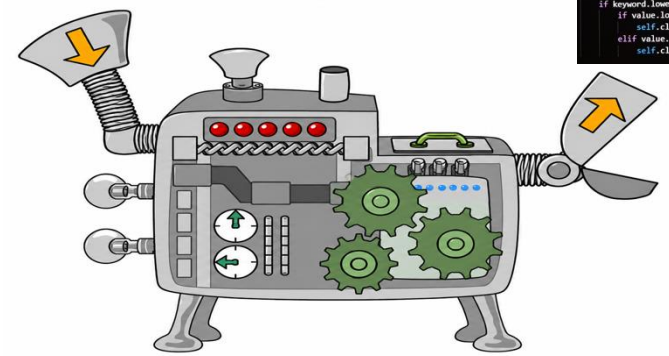
```
def send_header(self, keyword, value):
    """Send a MIME header to the headers buffer."""
    if self.request_version != 'HTTP/0.9':
        if not hasattr(self, '_headers_buffer'):
            self._headers_buffer = []
        self._headers_buffer.append(
            ("%s: %s\r\n" % (keyword, value)).encode

    if keyword.lower() == 'connection':
        if value.lower() == 'close':
            self.close_connection = True
        elif value.lower() == 'keep-alive':
            self.close_connection = False
```

$\pi(\cdot\,|x): \mathcal{X} \to \Delta(\mathcal{Y})$

Language model or generator

Problem Formalization: Active Search

Prompt x
Implement Fibonacci series

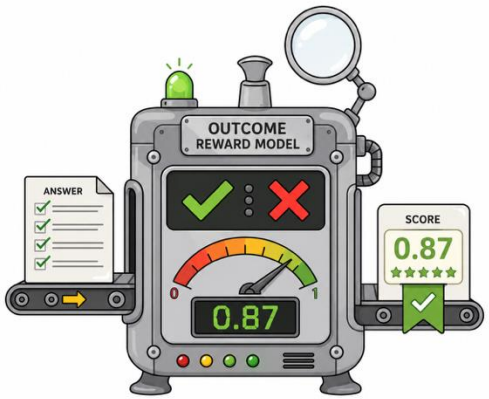


Response y

```
def send_header(self, keyword, value):
    """Send a MIME header to the headers buffer."""
    if self.request_version != 'HTTP/0.9':
        if not hasattr(self, '_headers_buffer'):
            self._headers_buffer = []
        self._headers_buffer.append(
            ("%s: %s\r\n" % (keyword, value)).encode

    if keyword.lower() == 'connection':
        if value.lower() == 'close':
            self.close_connection = True
        elif value.lower() == 'keep-alive':
            self.close_connection = False
```

$\pi(\cdot\,|x)\colon \mathcal{X} \to \Delta(\mathcal{Y})$

Language model or generator



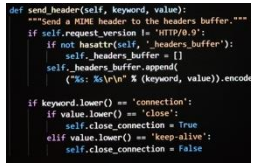
$\mathrm{RM}\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

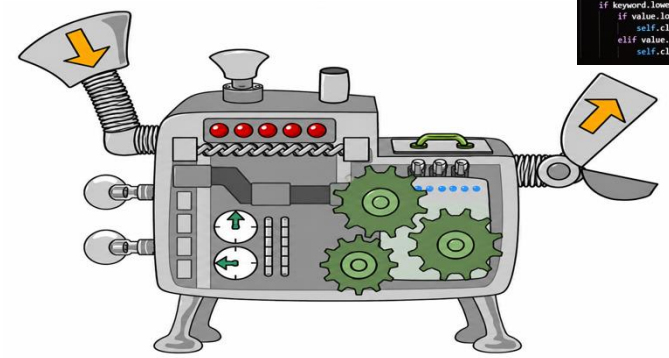
Learned outcome reward model

Problem Formalization: Active Search

Prompt x
Implement Fibonacci series

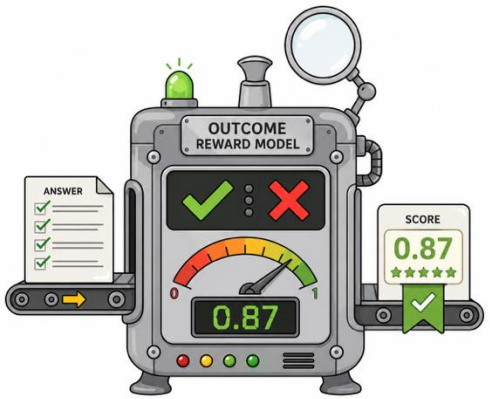
Response y





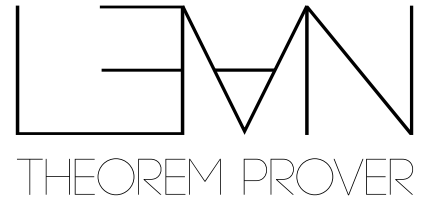
$\pi(\cdot\,|x): \mathcal{X} \to \Delta(\mathcal{Y})$

Language model or generator



$\mathrm{RM}: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Learned outcome reward model



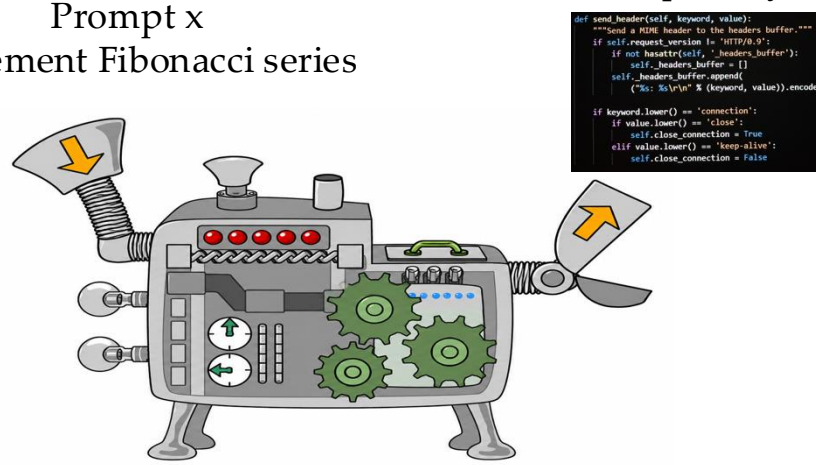
$\mathrm{Ver}: \mathcal{X} \times \mathcal{Y} \to \{0,1\}$

Verifier
(proof checker, hidden-test execution)

Problem Formalization: Active Search

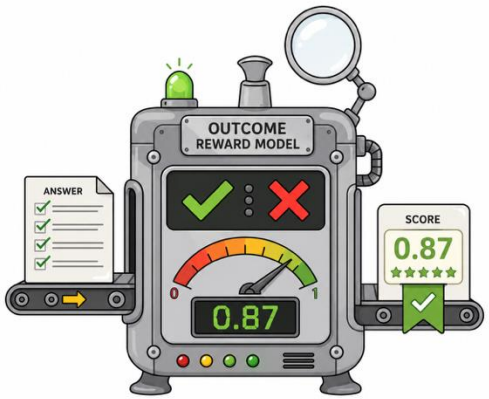
Prompt x
Implement Fibonacci series

Response y



$\pi(\cdot\,|x): \mathcal{X} \to \Delta(\mathcal{Y})$

Language model or generator



$\mathrm{RM}: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Learned outcome reward model

THEOREM PROVER

COMPILER

$\mathrm{Ver}: \mathcal{X} \times \mathcal{Y} \to \{0,1\}$

Verifier
(proof checker, hidden-test execution)

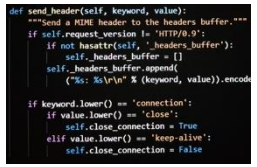
Active Search Instance (h, $\mathcal{D}$)

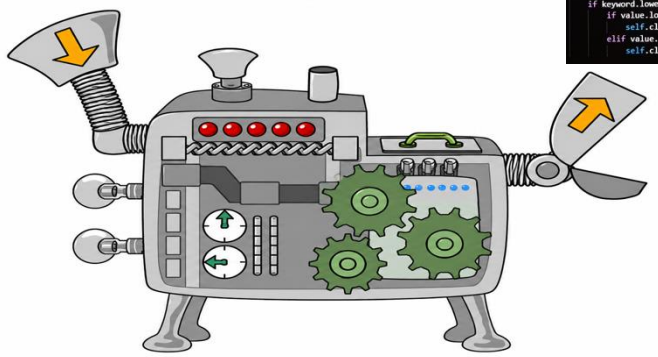
- Fix a prompt $x \in \mathcal{X}$
- For a sampled response $Y \sim \pi(\cdot\,|x)$, define random variables $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$ and $V = \mathrm{Ver}(\mathrm{x}, \mathrm{Y})$
- Define the distribution of R as $\mathcal{D}$
- Define reward–verifier relationship as $h(r) = \Pr(V = 1 | R = r)$ and $V = Bern(h(R))$

Problem Formalization: Active Search

Prompt x
Implement Fibonacci series

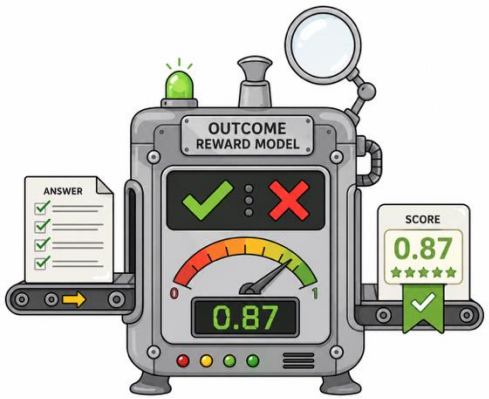
Response y





$\pi(\cdot\,|x): \mathcal{X} \to \Delta(\mathcal{Y})$

Language model or generator



$\mathrm{RM}: \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

Learned outcome reward model

LEAN
THEOREM PROVER

COMPILER

$\mathrm{Ver}: \mathcal{X} \times \mathcal{Y} \to \{0,1\}$

Verifier
(proof checker, hidden-test execution)

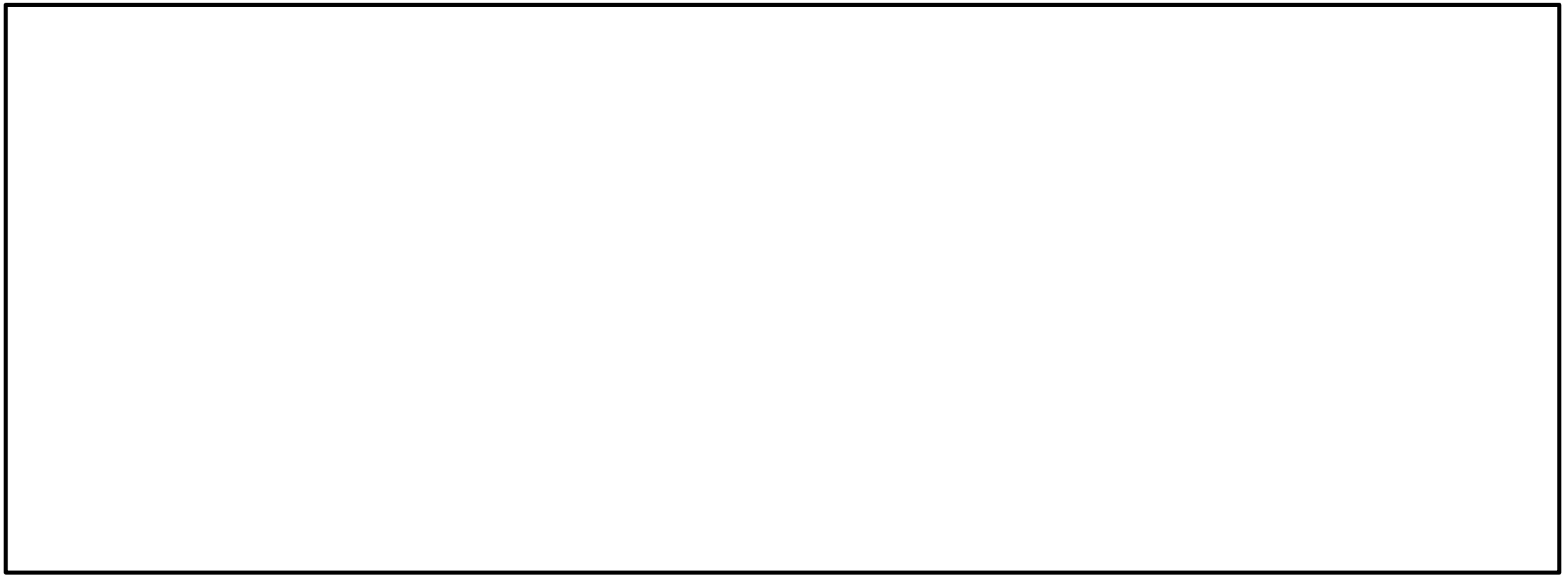
Active Search Instance (h, $\mathcal{D}$)

- Fix a prompt $x \in \mathcal{X}$
- For a sampled response $Y \sim \pi(\cdot\,|x)$, define random variables $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$ and $V = \mathrm{Ver}(\mathrm{x}, \mathrm{Y})$
- Define the distribution of R as $\mathcal{D}$
- Define reward–verifier relationship as $h(r) = \Pr(V = 1 | R = r)$ and $V = Bern(h(R))$

If the reward model assigns assigns score r to a response, what is the prob of the verifier accepts it?

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$



Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$ and $V = \mathrm{Ver}(\mathrm{x}, \mathrm{Y})$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$ and $V = \mathrm{Ver}(\mathrm{x}, \mathrm{Y})$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Given a prompt x. Repeat until a verifier returns 1, choosing one of two actions based on $\mathcal{A}$:

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Given a prompt x. Repeat until a verifier returns 1, choosing one of two actions based on $\mathcal{A}$:

1- **Generate**: Draw $Y \sim \pi(\cdot|x)$, score $R = \mathrm{RM}(x, Y)$, and add (Y, R) to the unverified pool. Pay cost c_{rew}.

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Given a prompt x. Repeat until a verifier returns 1, choosing one of two actions based on $\mathcal{A}$:

1- **Generate**: Draw $Y \sim \pi(\cdot|x)$, score $R = \mathrm{RM}(x, Y)$, and add (Y, R) to the unverified pool. Pay cost c_{rew}.

2- **Verify**: Pick any (Y, R) from the unverified pool, query $V \sim Bern(h(R))$. Pay cost c_{ver}.

If $V = 1$, the policy **stops** and **outputs** Y.

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot\,|x)$, $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$ and $V = \mathrm{Ver}(\mathrm{x}, \mathrm{Y})$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Given a prompt x. Repeat until a verifier returns 1, choosing one of two actions based on $\mathcal{A}$:

1- **Generate**: Draw $Y \sim \pi(\cdot\,|x)$, score $R = \mathrm{RM}(\mathrm{x}, \mathrm{Y})$, and add (Y, R) to the unverified pool. Pay cost c_{rew}.

2- **Verify**: Pick any (Y, R) from the unverified pool, query $V \sim Bern(h(R))$. Pay cost c_{ver}.

If $V = 1$, the policy **stops** and **outputs** Y.

The cost of $\mathcal{A}$ is given by

Problem Formalization: Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot\,|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Definition: Active Search Algorithm $\mathcal{A}$

Given a prompt x. Repeat until a verifier returns 1, choosing one of two actions based on $\mathcal{A}$:

1- **Generate**: Draw $Y \sim \pi(\cdot\,|x)$, score $R = \mathrm{RM}(x, Y)$, and add (Y, R) to the unverified pool. Pay cost c_{rew}.

2- **Verify**: Pick any (Y, R) from the unverified pool, query $V \sim Bern(h(R))$. Pay cost c_{ver}.

If $V = 1$, the policy **stops** and **outputs** Y.

The cost of $\mathcal{A}$ is given by

$$J(\mathcal{A}; \mathcal{D}, h) = N_{rew} c_{rew} + N_{ver} c_{ver}$$

#generation + reward model query

#verifier call

Problem Formalization: Adaptive Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Problem Formalization: Adaptive Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

What counts as doing well?
Distribution-Aware Benchmark: unavoidable cost
$J^\star(\mathcal{D}, h)$: the expected cost of the algorithm that knows both $(\mathcal{D}, h)$

Problem Formalization: Adaptive Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

What counts as doing well?
Distribution-Aware Benchmark: unavoidable cost
$J^\star(\mathcal{D}, h)$: the expected cost of the algorithm that knows both $(\mathcal{D}, h)$

Can we assume nothing about h (reward-verifier relationship)?
$h \in \mathcal{H}$ (probabilistic concept class) the class is known to the algorithm, h is not.

Problem Formalization: Adaptive Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

What counts as doing well?
Distribution-Aware Benchmark: unavoidable cost
$J^\star(\mathcal{D}, h)$: the expected cost of the algorithm that knows both $(\mathcal{D}, h)$

Can we assume nothing about h (reward-verifier relationship)?
$h \in \mathcal{H}$ (probabilistic concept class) the class is known to the algorithm, h is not.

Main Technical Question

Fix a class $\mathcal{H}$. Does there exist an active search algorithm $\mathcal{A}$, that knows $\mathcal{H}$ but not h, and a constant C such that for every $h \in \mathcal{H}$ and every $\mathcal{D}$,

$$\mathbb{E}[J(\mathcal{A}; \mathcal{D}, h)] \leq C \cdot J^\star(\mathcal{D}, h)$$

Problem Formalization: Adaptive Active Search

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

What counts as doing well?
Distribution-Aware Benchmark: unavoidable cost
$J^\star(\mathcal{D}, h)$: the expected cost of the algorithm that knows both $(\mathcal{D}, h)$

Can we assume nothing about h (reward-verifier relationship)?
$h \in \mathcal{H}$ (probabilistic concept class) the class is known to the algorithm, h is not.

Main Technical Question

Fix a class $\mathcal{H}$. Does there exist an active search algorithm $\mathcal{A}$, that knows $\mathcal{H}$ but not h, and a constant C such that for every $h \in \mathcal{H}$ and every $\mathcal{D}$,

$$\mathbb{E}[J(\mathcal{A}; \mathcal{D}, h)] \leq C \cdot J^\star(\mathcal{D}, h)$$

Easy prompts → small $J^\star(\mathcal{D}, h)$ → cost of $\mathcal{A}$ also small

If $\mathcal{A}$ incurs a high cost for search → the distribution-aware benchmark also incurs a high cost

Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$

$Y \sim \pi(\cdot\,|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$

$R \sim \mathcal{D}$

reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1|R = r)$

Assumption: $h(r) = \Pr(V = 1|R = r)$ is non-decreasing in r ($\mathcal{H}_{non-dec}$)
(our results can be extended to $(1-\eta)g(r) < h(r) < (1+\eta)g(r)$)

What this buys: Verifying the top-ranked candidates is always the best use of the budget.

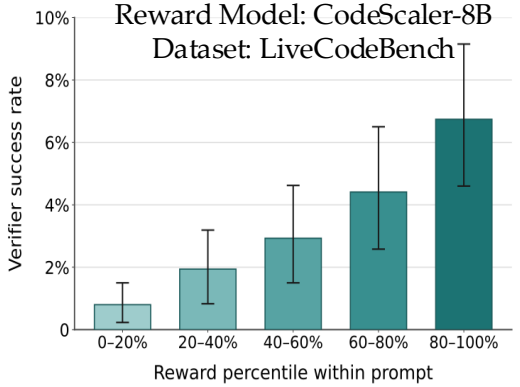
Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1|R = r)$

Assumption: $h(r) = \Pr(V = 1|R = r)$ is non-decreasing in r ($\mathcal{H}_{non-dec}$)
(our results can be extended to $(1-\eta)g(r) < h(r) < (1+\eta)g(r)$)

What this buys: Verifying the top-ranked candidates is always the best use of the budget.

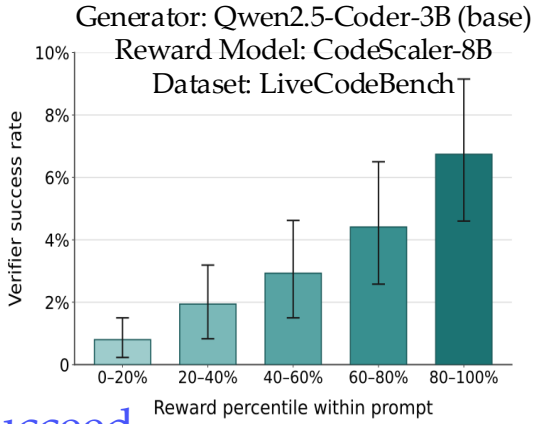
Generator: Qwen2.5-Coder-3B (base)
Reward Model: CodeScaler-8B
Dataset: LiveCodeBench



Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1|R = r)$

Assumption: $h(r) = \Pr(V = 1|R = r)$ is non-decreasing in r ($\mathcal{H}_{non-dec}$)
(our results can be extended to $(1-\eta)g(r) < h(r) < (1+\eta)g(r)$)



<u>What this buys:</u> Verifying the top-ranked candidates is always the best use of the budget.

Simple strategy: verify only candidates whose reward is at least t.

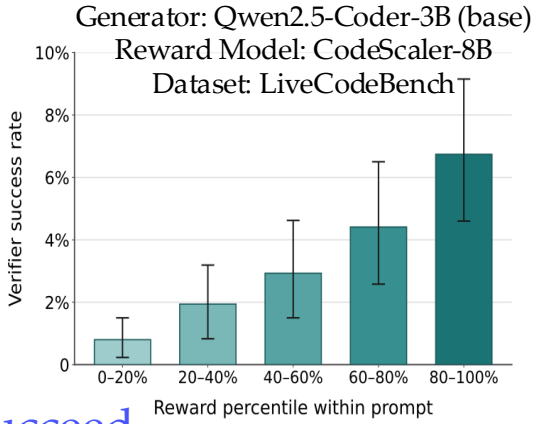
$q_t = \Pr(R \geq t)$ how often we pay to verify $s_t = \Pr(R \geq t, V = 1)$ how often we succeed

Each round costs c_{rew} always, plus c_{ver} with probability q_t, and succeeds with probability s_t: $J_t = \frac{c_{rew} + c_{ver}\, q_t}{s_t}$

Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1|R = r)$

Assumption: $h(r) = \Pr(V = 1|R = r)$ is non-decreasing in r ($\mathcal{H}_{non-dec}$)
(our results can be extended to $(1-\eta)g(r) < h(r) < (1+\eta)g(r)$)



<u>What this buys:</u> Verifying the top-ranked candidates is always the best use of the budget.

Simple strategy: verify only candidates whose reward is at least t.

$q_t = \Pr(R \geq t)$ how often we pay to verify $s_t = \Pr(R \geq t, V = 1)$ how often we succeed

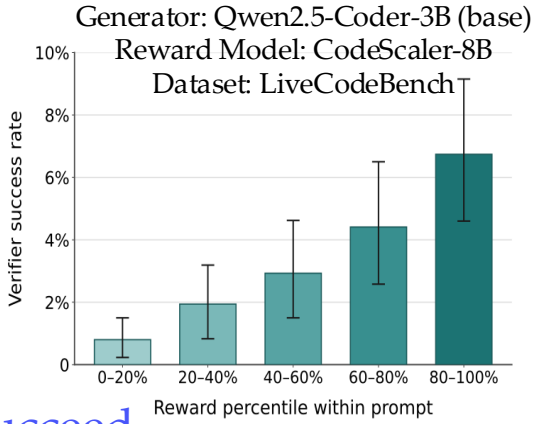
Each round costs c_{rew} always, plus c_{ver} with probability q_t, and succeeds with probability s_t: $J_t = \frac{c_{rew} + c_{ver}\, q_t}{s_t}$

The benchmark knows the best threshold; ADAP does not.

Our Proposed Algorithm: ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1|R = r)$

Assumption: $h(r) = \Pr(V = 1|R = r)$ is non-decreasing in r ($\mathcal{H}_{non-dec}$)
(our results can be extended to $(1-\eta)g(r) < h(r) < (1+\eta)g(r)$)



What this buys: Verifying the top-ranked candidates is always the best use of the budget.

Simple strategy: verify only candidates whose reward is at least t.

$q_t = \Pr(R \geq t)$ how often we pay to verify $s_t = \Pr(R \geq t, V = 1)$ how often we succeed

Each round costs c_{rew} always, plus c_{ver} with probability q_t, and succeeds with probability s_t: $J_t = \frac{c_{rew} + c_{ver}\, q_t}{s_t}$

The benchmark knows the best threshold; ADAP does not.

Main Observation

Suppose I tell you a good threshold t. Then generating on the order of $1/s_t$ responses and verifying on the order of q_t/s_t top-ranked responses finds a verified response with a constant probability.

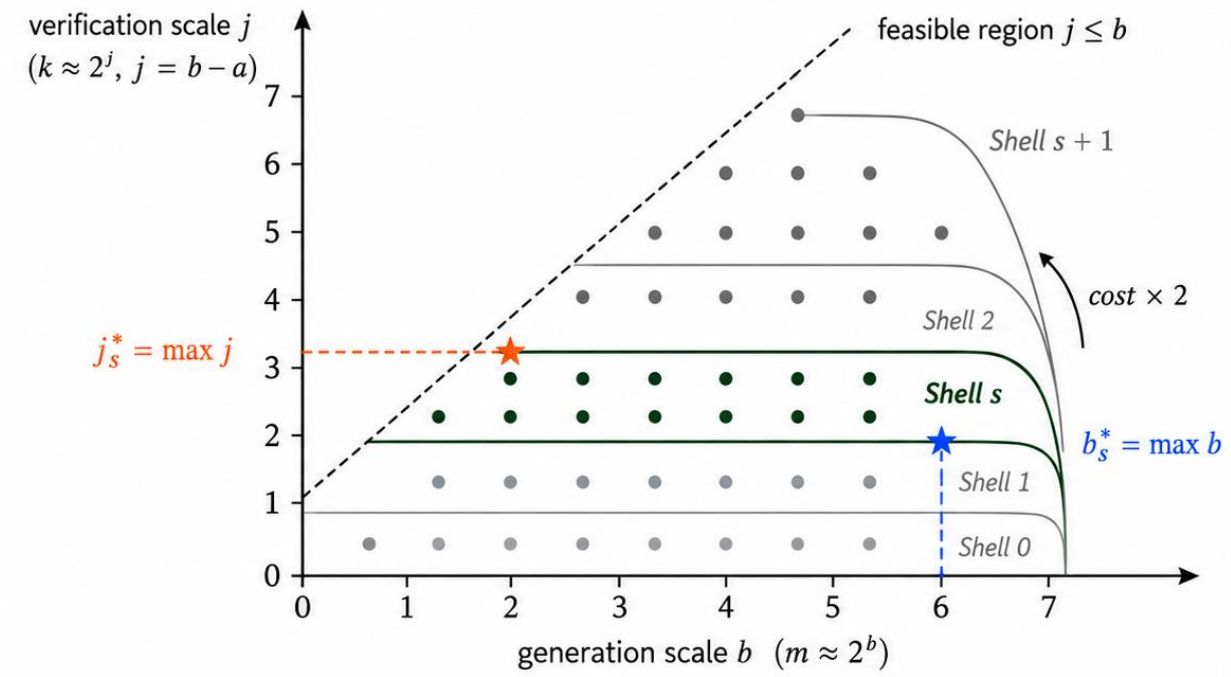
$q_t \approx 2^{-a}$ and $s_t \approx 2^{-b}$ then the threshold corresponds to the generation budget $m \approx 2^a$ and $k \approx 2^{b-a}$.

Performance Guarantee of ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$

Performance Guarantee of ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$



For shell S_s

- rightmost point → b_s^* → generation budget
- topmost point → j_s^* → verification budget

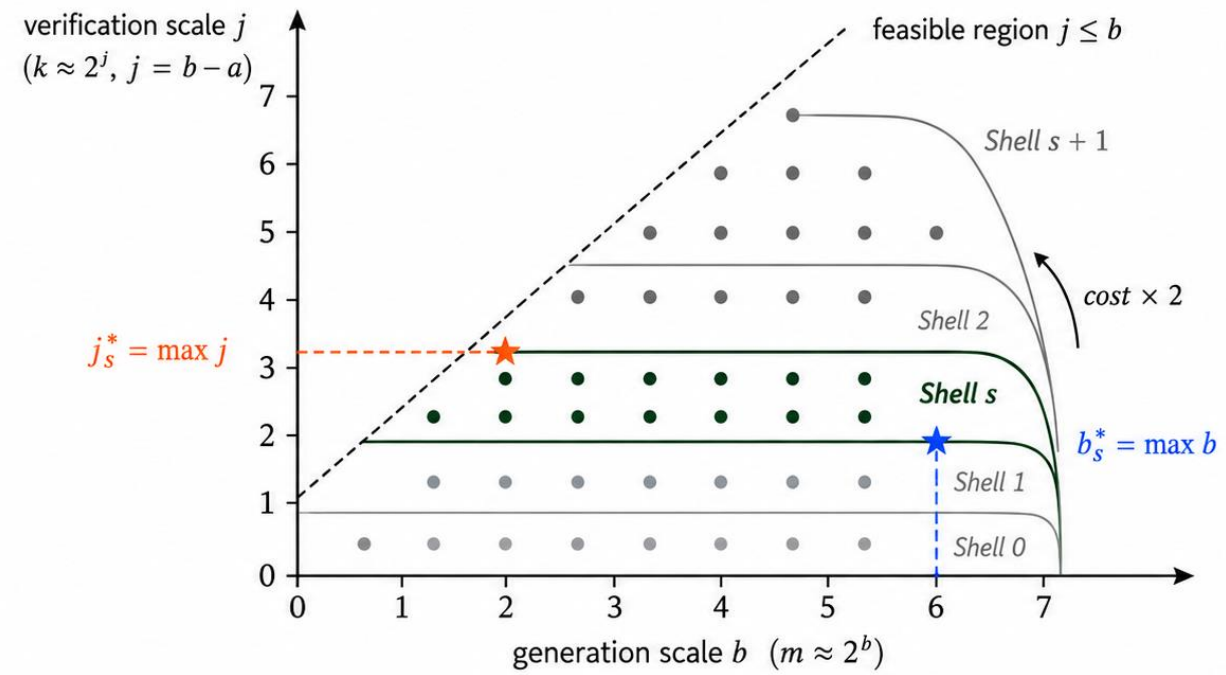
$$m_s = \lceil 2^{b_s^*+1} \rceil$$

$$k_s = \lceil 6 \cdot 2^{j_s^*} \rceil$$

constants from the analysis

Performance Guarantee of ADAP

prompt $x \in \mathcal{X}$
$Y \sim \pi(\cdot|x)$, $R = \mathrm{RM}(x, Y)$ and $V = \mathrm{Ver}(x, Y)$
$R \sim \mathcal{D}$
reward–verifier relationship $h(r) = \Pr(V = 1 | R = r)$



For shell S_s

- rightmost point → b_s^* → generation budget
- topmost point → j_s^* → verification budget

$$m_s = \lceil 2^{b_s^*+1} \rceil$$

$$k_s = \lceil 6 \cdot 2^{j_s^*} \rceil$$

constants from the analysis

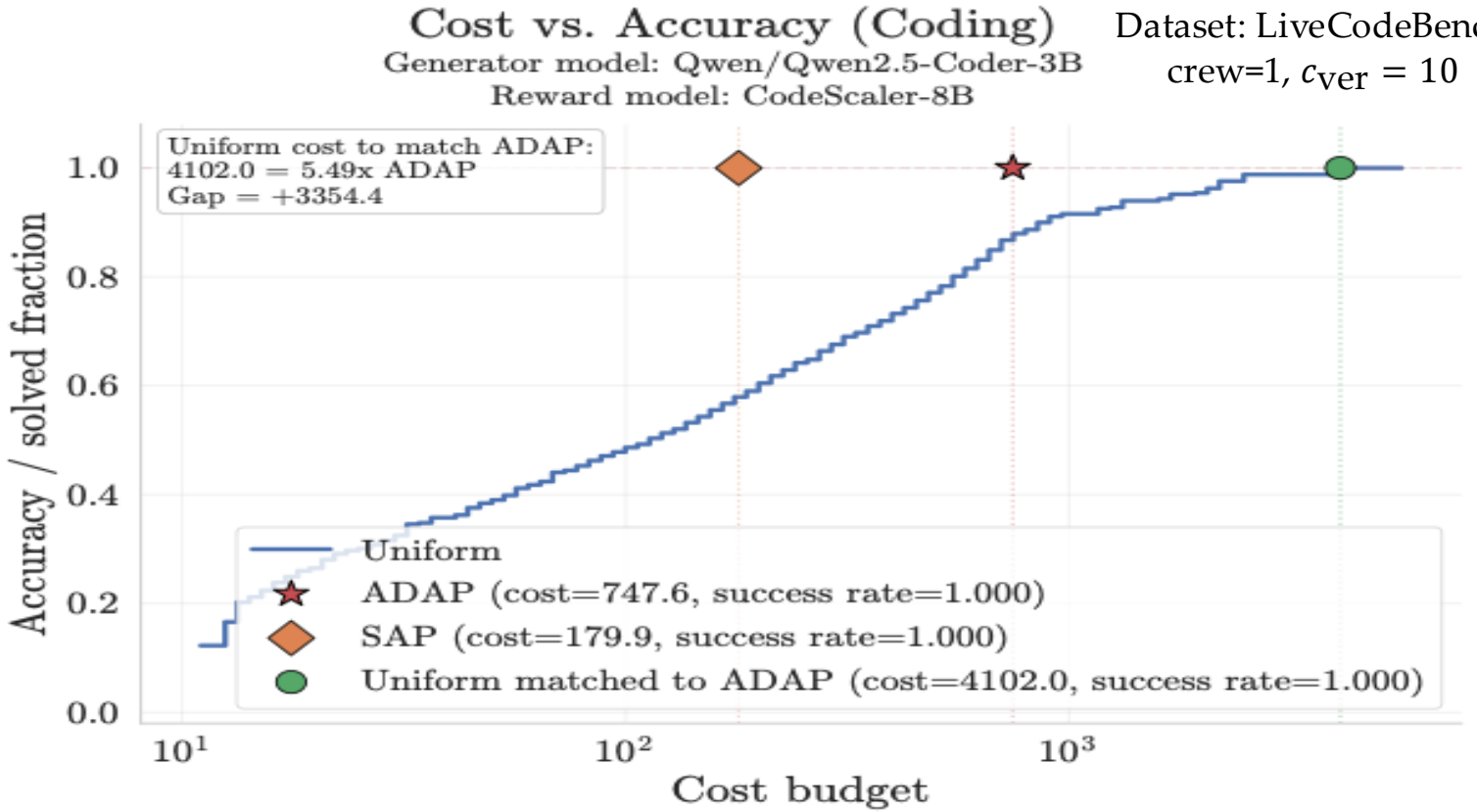
Theorem [Dughmi–Haghifam–Kalayci '25]

Assume $h \in \mathcal{H}_{\text{non–dec}}$. Then for every feasible prompt, every generator, and every induced $(h, \mathcal{D})$,

$$\mathbb{E}[J(\mathcal{A}_{ADAP}; h, \mathcal{D})] \le 173\, J^\star(h, \mathcal{D})$$

ADAP knows neither h nor $\mathcal{D}$, and uses the reward model only to rank.

ADAP gets reliability at much lower cost



- **Uniform**: The blue curve represents the best possible fixed-budget strategy.
  For every cost budget, we search over all fixed pairs of generation and verification counts and report the pair that solves the most problems. So this is actually a strong baseline: it is the upper envelope of all fixed policies.

- **SampleAware (SAP)**: For each prompt, we allocate the best (N_{rew}, N_{ver}) to minimize the budget.
  No realizable policy can reproduce it. It is a per-trial floor, not a competitor.

How much structure does active search need?

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

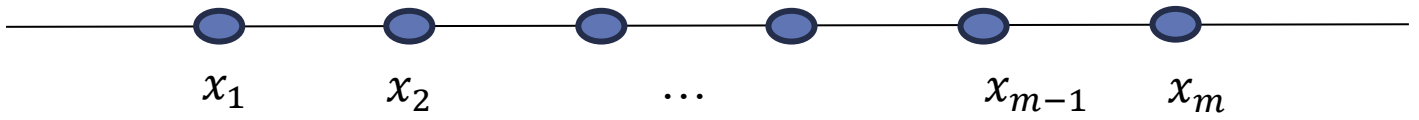
Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \ldots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \ldots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$

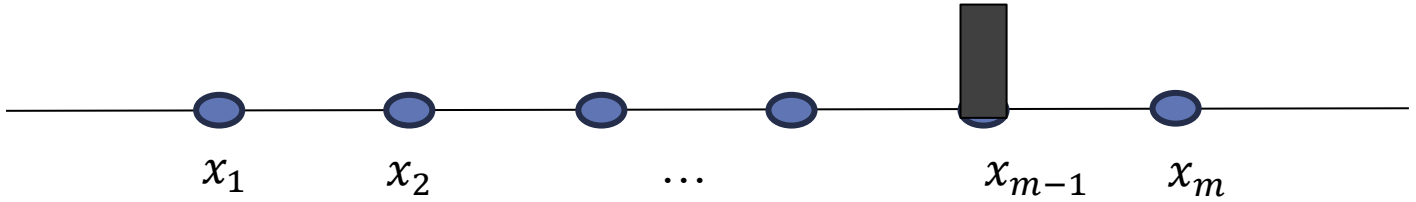


How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \ldots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$

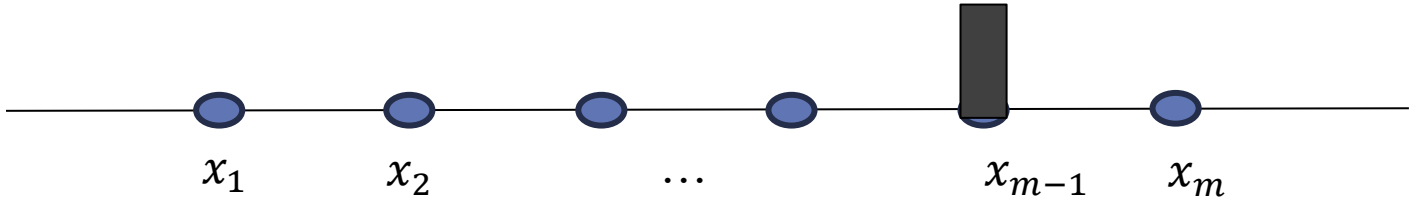


How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \dots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$



Cost of the distribution-aware benchmark

$J^\star(\mathcal{D}, h) = O(mc_{rew} + c_{ver})$

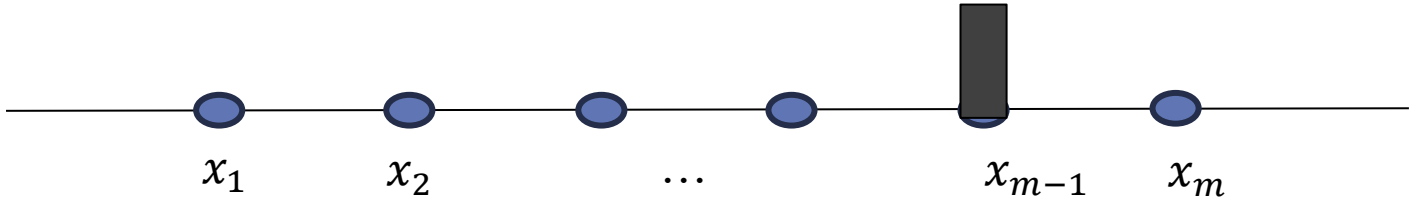
Wait until you get x with $h(x) = 1$

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \ldots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$



Cost of the distribution-aware benchmark
$J^\star(\mathcal{D}, h) = O(mc_{rew} + c_{ver})$

Wait until you get x with $h(x) = 1$

Every active search algorithm
$J(\mathcal{A}; \mathcal{D}, h) \geq \Omega(mc_{ver})$

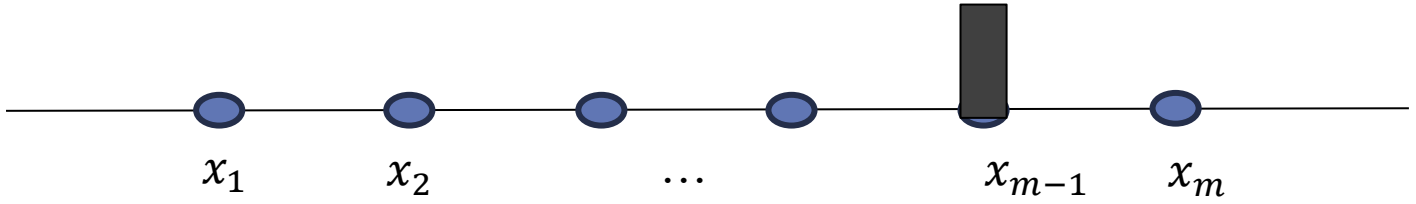
Needs to verify every point

How much structure does active search need?

The proposed algorithm requires $h(r) = \Pr(V = 1 | R = r)$ be approximately non-decreasing.

Can we compete with a distribution-aware benchmark without any structural assumption?

Example [inspired by Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15]
Let the distribution $\mathcal{D}$ be uniform over $\{x_1, \dots, x_m\}$ and $\mathcal{H} = \{h_i(x_j) = 1[i = j]\}$



Cost of the distribution-aware benchmark
$J^\star(\mathcal{D}, h) = O(mc_{rew} + c_{ver})$

Wait until you get x with $h(x) = 1$

Every active search algorithm
$J(\mathcal{A}; \mathcal{D}, h) \geq \Omega(mc_{ver})$

Needs to verify every point

For every active search algorithm
$$\frac{J(\mathcal{A}; \mathcal{D}, h)}{J^\star(\mathcal{D}, h)} = \Omega(c_{ver}/c_{rew})$$

Learning-Theoretic View of Active Search

Learning-Theoretic View of Active Search

Definition: Centered Star Number

Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$. Then, define

$$s_0(\mathcal{H}) = \max\{\mathrm{m} \in \mathbb{N}: \exists z_1, \ldots, z_m \in \mathcal{Z}, \exists h_1, \ldots, h_m \in \mathcal{H} \text{ s.t. } h_i(z_j) = 1[i = j]\}.$$

Learning-Theoretic View of Active Search

Definition: Centered Star Number

Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$. Then, define

$$s_0(\mathcal{H}) = \max\{m \in \mathbb{N}: \exists z_1, \dots, z_m \in \mathcal{Z}, \exists h_1, \dots, h_m \in \mathcal{H} \text{ s.t. } h_i(z_j) = 1[i = j]\}.$$

Theorem [Dughmi-Haghifam-Kalayci'25]

Fix an arbitrary concept class with centered star number $s_0(\mathcal{H})$. Then, we have

$$\min_{\text{active search alg. } \mathcal{A}} \max_{(\mathcal{D},h)\in\Delta(\mathcal{Z})\times\mathcal{H}} \frac{J(\mathcal{A};\mathcal{D},h)}{J^\star(\mathcal{D},h)} = \Theta\left(\min\left\{s_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\}\right)$$

Learning-Theoretic View of Active Search

Definition: Centered Star Number

Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$. Then, define

$$s_0(\mathcal{H}) = \max\{\mathrm{m} \in \mathbb{N}: \exists z_1, \dots, z_m \in \mathcal{Z}, \exists h_1, \dots, h_m \in \mathcal{H} \text{ s.t. } h_i(z_j) = 1[i = j]\}.$$

Theorem [Dughmi-Haghifam-Kalayci'25]

Fix an arbitrary concept class with centered star number $s_0(\mathcal{H})$. Then, we have

$$\min_{\text{active search alg. } \mathcal{A}} \max_{(\mathcal{D},h)\in\Delta(\mathcal{Z})\times\mathcal{H}} \frac{J(\mathcal{A};\mathcal{D},h)}{J^\star(\mathcal{D},h)} = \Theta\left(\min\left\{s_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\}\right)$$

Requires using the structure of $\mathcal{H}$

Learning-Theoretic View of Active Search

Definition: Centered Star Number

Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$. Then, define

$$s_0(\mathcal{H}) = \max\{m \in \mathbb{N}: \exists z_1, \dots, z_m \in \mathcal{Z}, \exists h_1, \dots, h_m \in \mathcal{H} \text{ s.t. } h_i(z_j) = 1[i = j]\}.$$

Theorem [Dughmi-Haghifam-Kalayci'25]

Fix an arbitrary concept class with centered star number $s_0(\mathcal{H})$. Then, we have

$$\min_{\text{active search alg. } \mathcal{A}} \max_{(\mathcal{D},h)\in\Delta(\mathcal{Z})\times\mathcal{H}} \frac{J(\mathcal{A};\mathcal{D},h)}{J^\star(\mathcal{D},h)} = \Theta\left(\min\left\{s_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\}\right)$$

Requires using the structure of $\mathcal{H}$

Verify everything sequentially

Learning-Theoretic View of Active Search

Definition: Centered Star Number

Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$. Then, define

$$s_0(\mathcal{H}) = \max\{\mathrm{m} \in \mathbb{N}: \exists z_1, \dots, z_m \in \mathcal{Z}, \exists h_1, \dots, h_m \in \mathcal{H} \text{ s.t. } h_i(z_j) = 1[i = j]\}.$$

Theorem [Dughmi-Haghifam-Kalayci'25]
Fix an arbitrary concept class with centered star number $s_0(\mathcal{H})$. Then, we have

$$\min_{\text{active search alg. } \mathcal{A}} \max_{(\mathcal{D},h)\in\Delta(\mathcal{Z})\times\mathcal{H}} \frac{J(\mathcal{A};\mathcal{D},h)}{J^\star(\mathcal{D},h)} = \Theta\left(\min\left\{s_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\}\right)$$

Requires using the structure of $\mathcal{H}$

Verify everything sequentially

Few Remarks:

1- Active search is easier than active learning even per instance given the mass of label 1 is not too small.
[Balcan-Hanneke-Vaughan'10 & Hanneke-Yang'15 & Balcan-Beygelzimer-Langford'06]

2- Does this characterization inspire how to select or train reward models for math/coding tasks?

3- 1-centered star number appears in [Hanneke'24].

Learning-Theoretic View of Active Search

Learning-Theoretic View of Active Search

Task

Given a large set of unlabeled samples $S = (Z_1, \ldots, Z_n)$ with the minimum number of verification either

1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Learning-Theoretic View of Active Search

Task

Given a large set of unlabeled samples $S = (Z_1, \dots, Z_n)$ with the minimum number of verification either

1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Step 1) For each $h \in \mathcal{H}$, form $A_h(S) = \{i \in [n]: h(Z_i) = 1\}$

Step 2) Find the minimum hitting set of $\cup_{h \in \mathcal{H}} A_h(S)$

Learning-Theoretic View of Active Search

Task

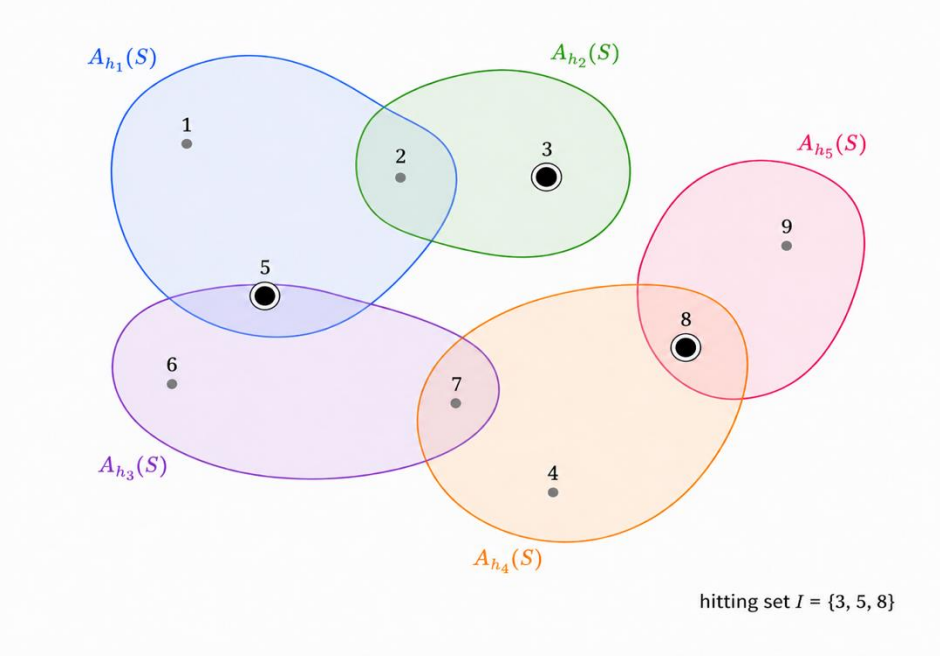
Given a large set of unlabeled samples $S = (Z_1, \dots, Z_n)$ with the minimum number of verification either

1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Step 1) For each $h \in \mathcal{H}$, form $A_h(S) = \{i \in [n]: h(Z_i) = 1\}$

Step 2) Find the minimum hitting set of $\cup_{h \in \mathcal{H}} A_h(S)$



Learning-Theoretic View of Active Search

Task

Given a large set of unlabeled samples $S = (Z_1, \dots, Z_n)$ with the minimum number of verification either

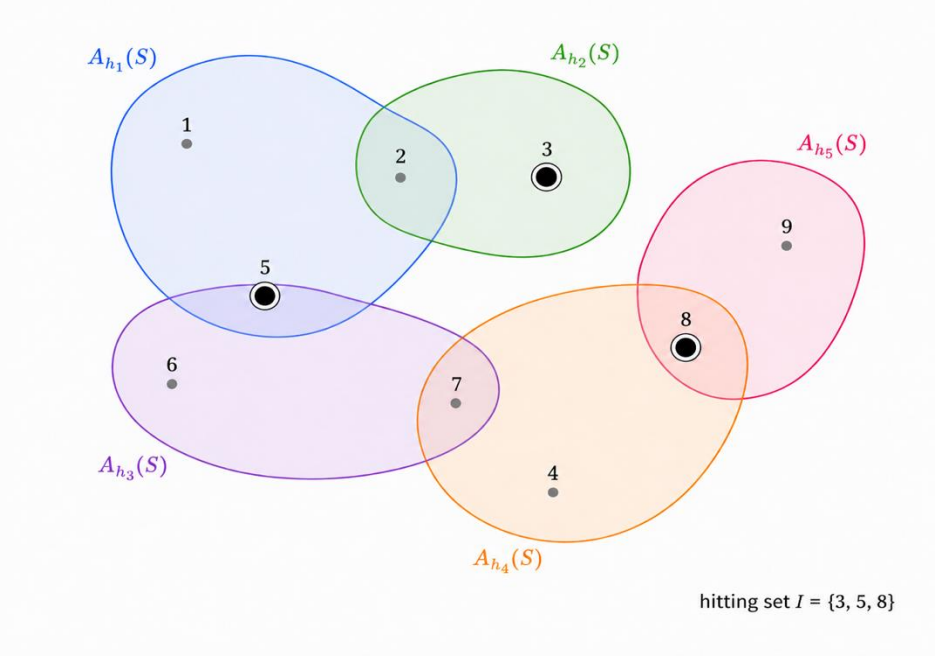
1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Step 1) For each $h \in \mathcal{H}$, form $A_h(S) = \{i \in [n] : h(Z_i) = 1\}$

Step 2) Find the minimum hitting set of $\cup_{h \in \mathcal{H}} A_h(S)$

Lemma [Dughmi-Haghifam-Kalayci'25]
The size of the hitting set is at most $s_0(\mathcal{H})$



Learning-Theoretic View of Active Search

Task

Given a large set of unlabeled samples $S = (Z_1, \dots, Z_n)$ with the minimum number of verification either

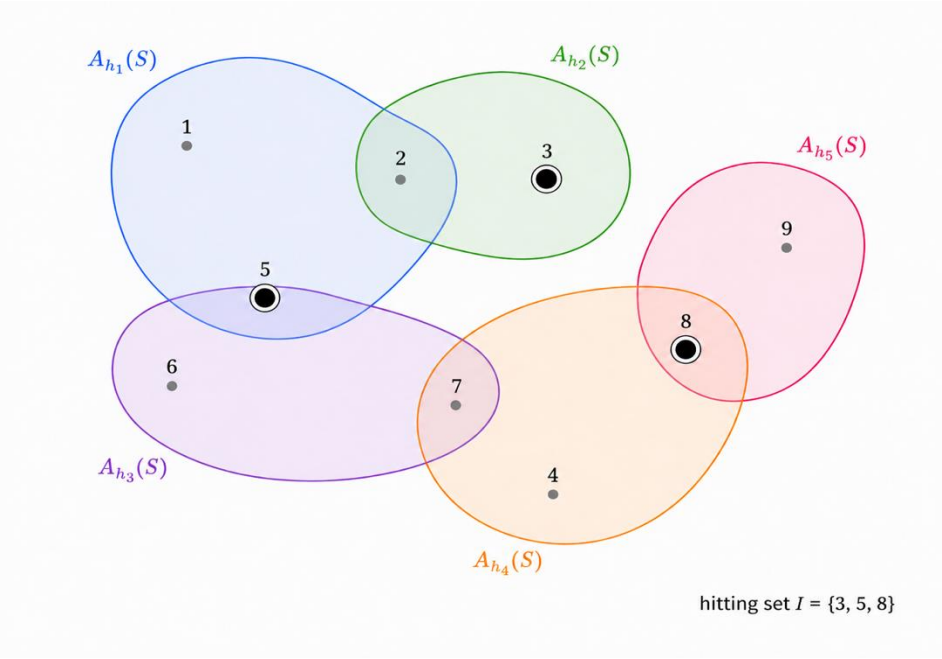
1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Step 1) For each $h \in \mathcal{H}$, form $A_h(S) = \{i \in [n]: h(Z_i) = 1\}$

Step 2) Find the minimum hitting set of $\cup_{h\in\mathcal{H}} A_h(S)$

Lemma [Dughmi-Haghifam-Kalayci'25]
The size of the hitting set is at most $s_0(\mathcal{H})$



Proof:

If Z_i is in the hitting set if there exists $h \in \mathcal{H}$ s.t. $h(Z_i) = 1$ and $\forall j \neq i\ h(Z_j) = 0$

Learning-Theoretic View of Active Search

Task

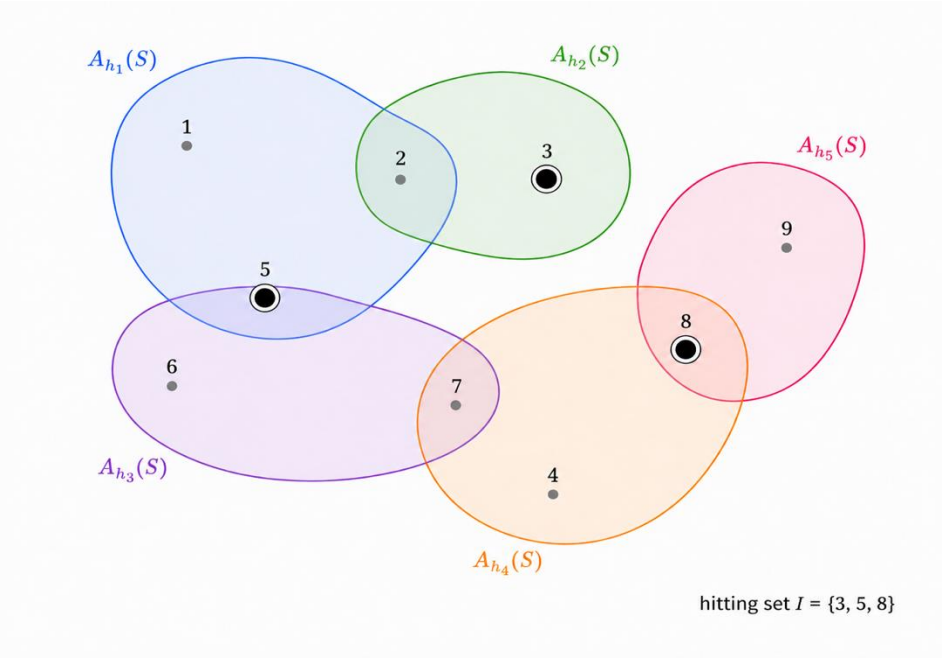
Given a large set of unlabeled samples $S = (Z_1, \dots, Z_n)$ with the minimum number of verification either

1) Find a Z_i such that $h^\star(Z_i) = 1$

2) Discard the whole set if no label one exists

Step 1) For each $h \in \mathcal{H}$, form $A_h(S) = \{i \in [n]: h(Z_i) = 1\}$

Step 2) Find the minimum hitting set of $\cup_{h\in\mathcal{H}} A_h(S)$



Lemma [Dughmi-Haghifam-Kalayci'25]
The size of the hitting set is at most $s_0(\mathcal{H})$

Proof:

If Z_i is in the hitting set if there exists $h \in \mathcal{H}$ s.t. $h(Z_i) = 1$ and $\forall j \neq i\ h(Z_j) = 0$

↓

Hitting set forms a set we care about in the definition of centered star number.

Takeaways and Future Directions

- **The model: active search.** Generate–rank–verify becomes a search problem: cheap noisy scores, one expensive exact check, find a correct answer for as little as possible.
- **The algorithm: ADAP.** Adapts to each prompt as it goes. Ranks with the reward model and uses nothing else. Stays within a constant factor of an oracle that knows the instance.
- **The limit: centered star number.** You cannot assume nothing about the reward model. We give a complete characterization: the adaptivity gap is $\min\{\mathfrak{s}_0(\mathcal{H}), c_{ver}/c_{rew}\}$,

GitHub Repository

arXiv Link