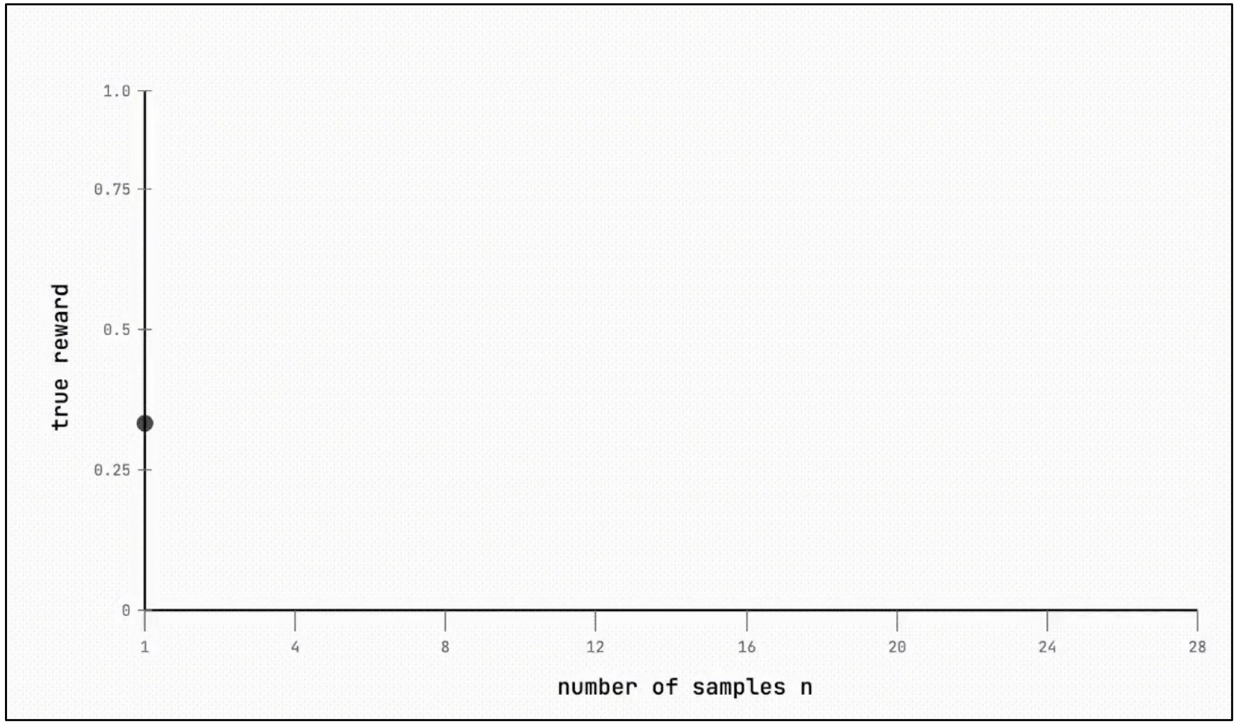


Inference-Time Reward Hacking in Large Language Models

Spotlight Award! 🌟

Hadi Khalaf, Claudio Mayrink Verdun, Alex Oesterling, Himabindu Lakkaraju, and Flavio du Pin Calmon

Harvard John A. Paulson School of Engineering and Applied Sciences

All reward models are bad!

AI alignment methods aim to **maximize a reward function**

All reward models are bad!

AI alignment methods aim to **maximize a reward function**

… but what reward are we maximizing?

All reward models are bad!

AI alignment methods aim to **maximize a reward function**

… but what reward are we maximizing?

There are two kinds:

1. **Proxy rewards** are the computable signals we can use
   Example: Scores from a trained reward model

All reward models are bad!

AI alignment methods aim to **maximize a reward function**

… but what reward are we maximizing?

There are two kinds:

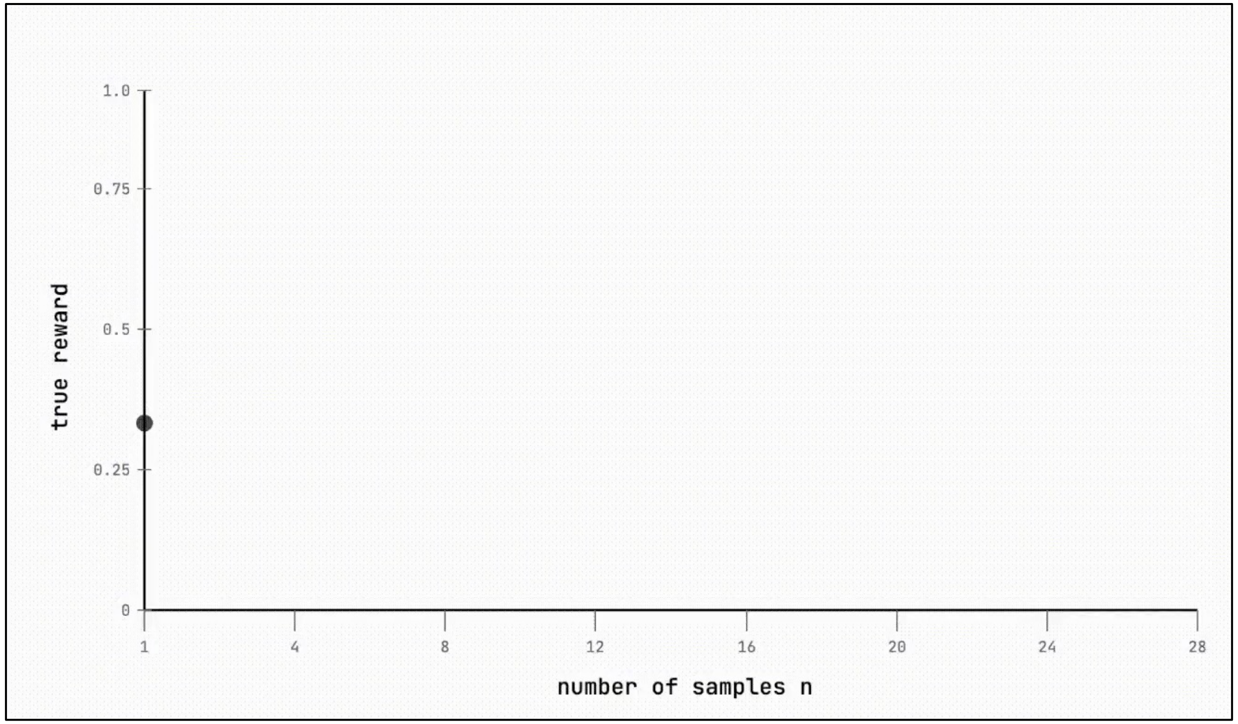
1. **Proxy rewards** are the computable signals we can use
   Example: Scores from a trained reward model

2. **True rewards** are the quality of an output according to a desired objective
   Example: An output's helpfulness or toxicity

All proxy reward models are bad!

We are maximizing for the wrong reward.

This mismatch can cause **reward hacking**!



Inference-time Alignment

We leverage a reward model to improve our responses **without any training.**

Inference-time Alignment

We leverage a reward model to improve our responses **without any training**.

One example is **Best-of-n**.

For a given question,

1. Sample *n* responses
2. Score them using a reward model
3. Choose the one with the highest reward

Inference-time Alignment

We leverage a reward model to improve our responses without any training.

Another example is **Soft Best-of-n.**

For a given question,

1. Sample *n* responses
2. Score them using a reward model
3. Sample a response using a temperature-scaled softmax over rewards

What should you do with a proxy reward?

What should you do with a proxy reward?

Don’t fully trust it. Instead, you should **hedge**!

What should you do with a proxy reward?

Our contributions:

1. We characterize reward hacking in inference-time settings;

What should you do with a proxy reward?

Our contributions:

1. We characterize reward hacking in inference-time settings;

2. We propose **Best-of-Poisson** that provides an efficient, near-exact approximation of the optimal policy at inference;

What should you do with a proxy reward?

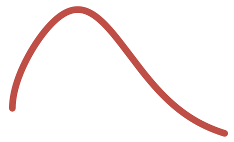
Our contributions:

1. We characterize reward hacking in inference-time settings;

2. We propose **Best-of-Poisson** that provides an efficient, near-exact approximation of the optimal policy at inference;

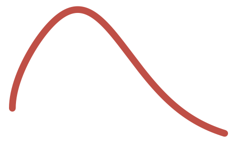
3. We introduce **HedgeTune**, a lightweight method to find the best inference-time parameter. We show that **HedgeTune** mitigates hacking on math, reasoning, and human-preference setups.

Why does reward hacking happen?



Consider an inference-time method with parameter θ .

Why does reward hacking happen?

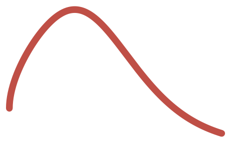


Consider an inference-time method with parameter θ .

An inference-time method is ***greedy*** if as we increase θ, it becomes more likely to choose a response with higher proxy reward.

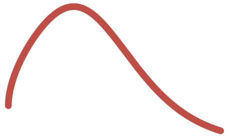
Example: Best-of-n!

Why does reward hacking happen?



Theorem (Informal): If the *inference-time method* is *greedy*, then the expected true reward attains *at most one extremum* *w.r.t.* θ .

Why does reward hacking happen?



Theorem (Informal): If the *inference-time method* is *greedy*, then the expected true reward attains *at most one extremum* *w.r.t.* θ .

Consequence: We explain hacking in Best-of-*n*.

If there is at most one maxima, search for it! This is the idea behind **HedgeTune**.

Best-of-Poisson

All alignment methods try to solve the following problem:

$$\pi^*(x) = \arg\max_{\pi_x \in \Delta_{\mathcal{X}}} \mathbb{E}_{\pi_x} \left[r_p(X) \right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}})$$

Best-of-Poisson

All alignment methods try to solve the following problem:

$$\pi^*(x) = \underset{\pi_x \in \Delta_{\mathcal{X}}}{\arg\max}\, \mathbb{E}_{\pi_x}\left[r_p(X)\right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}}) = \frac{\pi_{\mathsf{ref}}(x) e^{\lambda r_p(x)}}{Z(\lambda)}$$

Best-of-Poisson

All alignment methods try to solve the following problem:

Unrealizable in practice!

$$\pi^*(x) = \arg\max_{\pi_x \in \Delta_{\mathcal{X}}} \mathbb{E}_{\pi_x}\left[r_p(X)\right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}}) = \boxed{\frac{\pi_{\mathsf{ref}}(x) e^{\lambda r_p(x)}}{Z(\lambda)}}$$

Best-of-Poisson

All alignment methods try to solve the following problem:

Unrealizable in practice!

$$\pi^*(x) = \underset{\pi_x \in \Delta_{\mathcal{X}}}{\arg\max}\, \mathbb{E}_{\pi_x}\left[r_p(X)\right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}}) = \boxed{\frac{\pi_{\mathsf{ref}}(x) e^{\lambda r_p(x)}}{Z(\lambda)}}$$

We can estimate it through:

- RLHF: Expensive, need to rerun training for every penalty

Best-of-Poisson

All alignment methods try to solve the following problem:

Unrealizable in practice!

$$\pi^*(x) = \underset{\pi_x \in \Delta_{\mathcal{X}}}{\arg\max}\, \mathbb{E}_{\pi_x}\left[r_p(X)\right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}}) = \boxed{\frac{\pi_{\mathsf{ref}}(x) e^{\lambda r_p(x)}}{Z(\lambda)}}$$

We can estimate it through:

- RLHF: Expensive, need to rerun training for every penalty
- Best-of-n: Cheap but coarse control over the KL divergence

Best-of-Poisson

All alignment methods try to solve the following problem:

Unrealizable in practice!

$$\pi^*(x) = \underset{\pi_x \in \Delta_{\mathcal{X}}}{\arg\max}\, \mathbb{E}_{\pi_x}\left[r_p(X)\right] - \frac{1}{\lambda} D_{\mathsf{KL}}(\pi_x \| \pi_{\mathsf{ref}}) = \boxed{\frac{\pi_{\mathsf{ref}}(x) e^{\lambda r_p(x)}}{Z(\lambda)}}$$

We can estimate it through:

- RLHF: Expensive, need to rerun training for every penalty
- Best-of-n: Cheap but coarse control over the KL divergence
- Soft Best-of-n: Need to set two parameters (n, temperature)

Best-of-Poisson

For a given question:

1. Sample *n* from Poisson distribution
2. Sample *n* responses from the LLM
3. Score them using a reward model
4. Choose the one with the highest reward

We randomize 🎲 our *n.* This gives us continuous control over the KL divergence.

We show that the resulting **BoP** distribution is close to the optimal one!

HedgeTune

We propose **HedgeTune** as a one-time offline calibration of your inference-time parameter.

You can apply it to any LLM and any proxy reward with black-box access!

HedgeTune

We propose **HedgeTune** as a one-time offline calibration of your inference-time parameter.

You can apply it to any LLM and any proxy reward with black-box access!

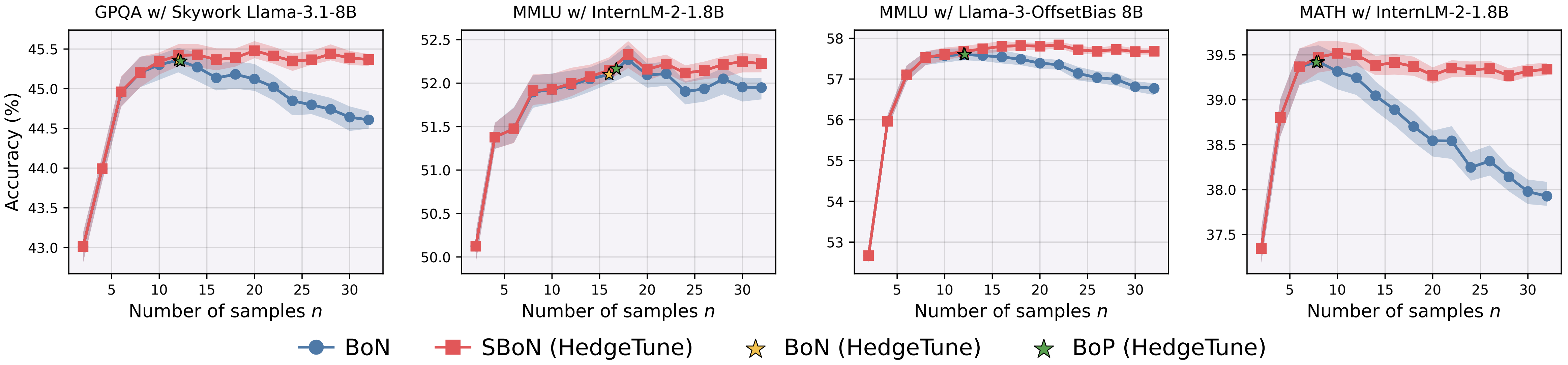
We require a small set of proxy and true reward pairs.

HedgeTune

Algorithm 4 `HedgeTune`: Parameter Optimization for Hedging

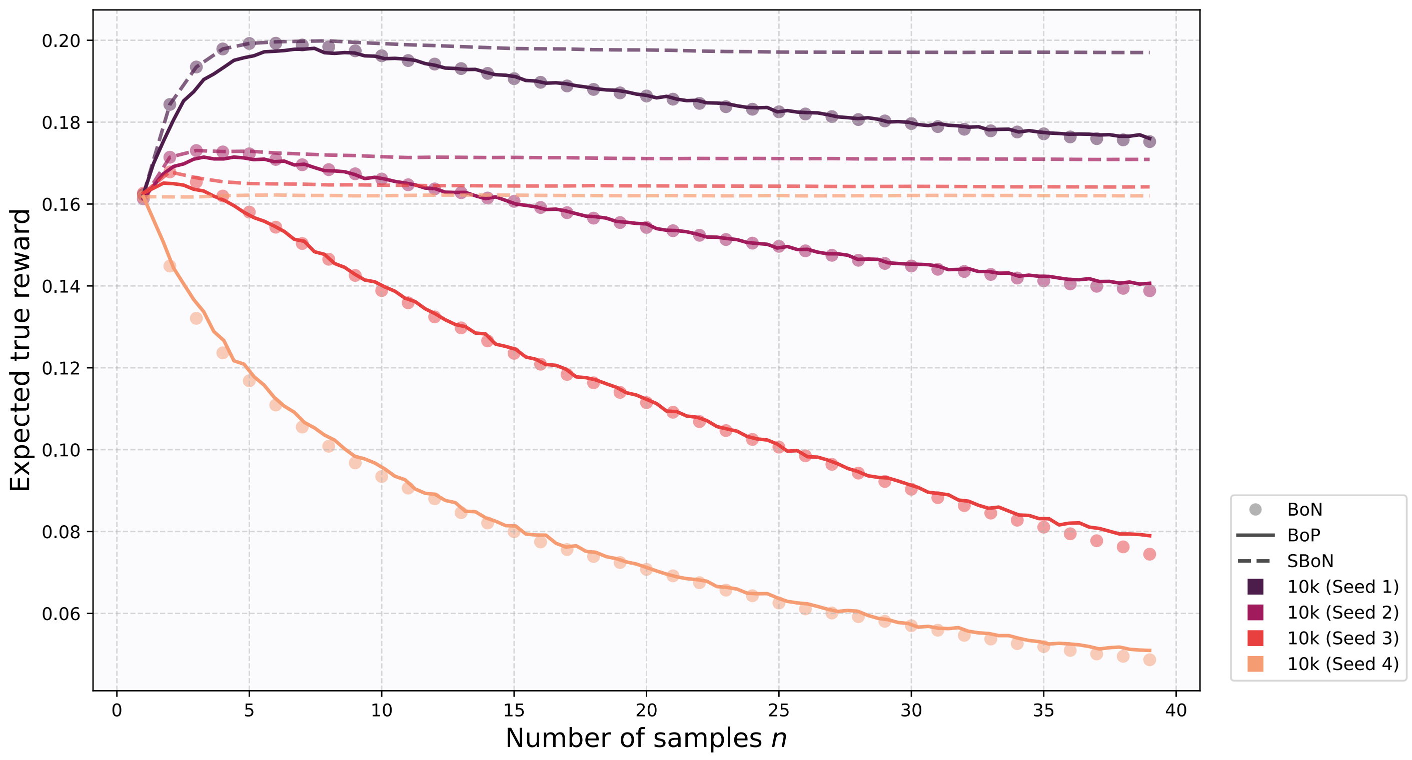
1: **Inputs:** Proxy and true rewards $\{s_{t,k}, r_{t,k}\}$ per prompt t; parameter domain Θ
2: **Output:** Optimal hedge parameter $\theta^\star$
3: STEP 1. For each prompt t, sort responses by their proxy scores and map their ranks to empirical quantiles $u_{t,k} \in (0, 1)$.
4: STEP 2. Specify the score function $\psi(u, \theta)$ and density $p_\theta(u)$ according to the inference-time method (e.g., BoN, SBoN, BoP; see Appendix D).
5: STEP 3. For a given t and $\theta \in \Theta$, define the residual $R_t(\theta) = \mathbb{E}_{u\sim p_\theta}[r_t(u)\, \psi(u, \theta)]$. This can be estimated from the empirical pairs $\{(u_{t,k}, r_t(u_{t,k}))\}$.
6: STEP 4. Find $\theta^\star \in \Theta$ such that the average residual $\bar{R}(\theta^\star) = \frac{1}{|T|}\sum_t \hat{R}_t(\theta) = 0$ via one-dimensional root-finding.

Hedging in verifiable setups



Result: Hedging mitigates reward hacking and achieves superior reward-distortion tradeoffs on standard verifiable benchmarks such as MMLU Pro and GPQA, even with large proxy rewards (8B)!

Hedging with human preferences



Result: Hedging mitigates reward hacking in a realistic RLHF setup

Conclusion

We offer a cheap and lightweight method to improve performance and mitigate reward hacking at inference.

We show that hedging is a promising framework to leverage proxy rewards and build safer, more reliable AI systems!