

E-evaluator: Reliable Agent Verifiers with Sequential Hypothesis Testing

Shuvom Sadhuka^{1,2}, Drew Prinster^{1,3}, Clara Fannjiang¹, Gabriele Scalia¹, Aviv Regev¹,
Hanchen Wang^{1,4}

¹Genentech ²MIT ³Johns Hopkins ⁴Stanford

December 4, 2025

Abstract

Agentic AI systems execute a sequence of actions, such as reasoning steps or tool calls, in response to a user prompt. To evaluate the success of their trajectories, researchers have developed verifiers, such as LLM judges and process-reward models, to score the quality of each action in an agent’s trajectory. Although these heuristic scores can be informative, there are no guarantees of correctness when used to decide whether an agent will yield a successful output. Here, we introduce *e-evaluator*, a method to convert any black-box verifier score into a decision rule with provable control of false alarm rates. We frame the problem of distinguishing successful trajectories—that is, a sequence of actions that will lead to a correct response to the user’s prompt—and unsuccessful trajectories as a sequential hypothesis testing problem. *E-evaluator* builds on tools from e-processes to develop a sequential hypothesis test that remains statistically valid at every step of an agent’s trajectory, enabling online monitoring of agents over arbitrarily long sequences of actions. Empirically, we demonstrate that *e-evaluator* provides greater statistical power and better false alarm rate control than other strategies across six datasets and three agents. We additionally show that *e-evaluator* can be used for to quickly terminate problematic trajectories and save tokens. Together, *e-evaluator* provides a lightweight, model-agnostic framework that converts verifier heuristics into decisions rules with statistical guarantees, enabling the deployment of more reliable agentic systems.

1 Introduction

Agents are black-box systems that autonomously perform tasks by executing a sequence of actions, collectively referred to as a *trajectory*. These trajectories can include diverse actions, including interactions with an external environment through tool calls, writing code, or steps of logical reasoning. Although we typically refer to *agents* in this paper as large language model (LLM)-based agents responding to user requests, the *agentic* paradigm is more widely applicable, including robotic agents that execute physical tasks (e.g., pick up a cup) through a sequence of mechanical actions [3], and gaming agents that play a game (e.g., poker or chess) through a sequence of game-legal actions [5, 51]. Agents have wide applications, and have delivered promising early results in diverse fields including drug discovery [14, 52], cell biology and genomics [61, 19], and hypothesis verification [18].

Nonetheless, agents make mistakes, and it is important to be able to detect these. To this end, *verifier* models have been developed to numerically score each action in an agent’s trajectory. These scores are typically used as a proxy for the probability that the trajectory will successfully produce a correct final output. Example verifiers include judge LLMs, which provide a score (as text output) after each step [28], and process-reward models, which are finetuned to give a probabilistic prediction of whether each step in a trajectory is “correct” or “incorrect” [29, 73, 31]. These verifiers’ scores can then be used to identify unsuccessful trajectories, as those that will produce an incorrect final output.

A key limitation of verifiers to date is that when their scores are used to decide whether to flag a trajectory as incorrect, there are no guarantees on the resulting probability of error of this downstream decision. Rigorous guarantees may become particularly critical when agents are deployed in high-stakes settings with real-world implications, such as self-driving labs [27], gene editing [42], or hospital operations [13]. In

particular, we focus on the *false alarm rate*, or the probability of incorrectly flagging a successful trajectory as unsuccessful. Even if verifier scores satisfy popular notions of probabilistic “correctness”, such as marginal calibration, such notions do not provide guarantees on the false alarm rate. Furthermore, each action in a trajectory costs time and resources, and trajectories can be long. We therefore want to detect that a trajectory will be unsuccessful early on—ideally, after as few actions as possible—instead of having to incur the costs of the full trajectory before making the decision. These desiderata necessitate assessing the verifier score after each action, in which case it is unclear how to get guarantees on the probability of *ever* giving a false alarm, particularly since the length of the complete trajectory is never known in advance.

Fine-tuning or otherwise building better verifiers does not directly address these issues. Furthermore, fine-tuning verifiers for the deployment setting involve meaningful engineering efforts, requiring sufficient compute as well as white-box access to the verifier (and possibly the agent) weights. Even with sufficient compute and access to the verifier weights, it may be impractical to obtain sufficient data appropriate for fine-tuning verifiers: one needs not just trajectories representative of the deployment setting, but also a “correctness” label for every action in every trajectory [31].

To tackle these challenges, we introduce *e-valuator*, a lightweight statistical wrapper that converts scores from any black-box verifier into a decision rule for detecting unsuccessful agent trajectories, with guarantees on the false alarm rate. To do so, we first frame the problem as a hypothesis test. We assume that the sequence of verifier scores is drawn from one distribution, p_1 , for successful trajectories (i.e., those that will produce a correct final output), and another distribution, p_0 , for unsuccessful trajectories. For a new trajectory, the problem then reduces to deciding, after as few actions as possible, whether the verifier score sequence is drawn from p_1 or p_0 . If one could evaluate p_1 and p_0 , one could apply a sequential likelihood ratio test [60], a well-studied sequential hypothesis test that tests at each time t if the scores collected thus far are from p_1 or p_0 . However, p_1 and p_0 are typically not known.

Thus, *e-valuator* works in three steps: (1) **collect** a small calibration set of agent trajectories (i.e., action sequences and intermediate states), stepwise verifier scores, and outcome labels (i.e., whether the final output was correct or incorrect) from the deployment setting, (2) **learn** a model of the density ratio between p_1 and p_0 , the distribution of verifier scores for successful trajectories and unsuccessful trajectories, at each step t , respectively, and (3) **find** a decision threshold for flagging unsuccessful trajectories, using a held-out split of the calibration set, while controlling for the false alarm rate (type I error) at a user-specified α (i.e., rate at which successful trajectories are accidentally flagged as unsuccessful). Importantly, *e-valuator* complements any ongoing and future improvements to verifiers: while it guarantees control of the false alarm rate for any combination of agent and verifier, deploying it with better verifiers will tend to yield more powerful decision rules. *e-valuator* requires minimal compute and can run on a standard laptop, and we release code as a Python package available in PyPi at <https://pypi.org/project/e-evaluator/>.

E-valuator builds on approaches from e-values and sequential hypothesis testing. E-values are an alternative to p -values that are particularly useful in settings where one wants to run a sequence of hypothesis tests (e.g., “is this ongoing trajectory successful?”) but might not know the number of tests beforehand (since trajectories are of variable length).

Empirically, across six datasets and three agents, *e-valuator* provided better false alarm rate (type I error) control and statistical power than other baselines, such as a raw verifier or re-calibrated verifier model alone. We highlight one potential use case of *e-valuator* with early termination of unsuccessful trajectories, which enables recovering up to 90% of the model’s original accuracy with just 80% of the tokens. We also experiment with a non-LLM setting, where a chess engine numerically scores the board after each move.

To summarize, our contributions are as follows:

1. We formalize the problem of verifying agent outputs and posit sequential hypothesis testing as a solution to boost existing verifier models.
2. We introduce *e-valuator*, a statistical wrapper that can flag unsuccessful trajectories with controllable error rates and strong statistical power.
3. *E-valuator* can adapt to any black-box agent-verifier combination, including non-LLM agents. Because *e-valuator* assumes black-box access to the verifier, methods to directly improve verifier models (e.g., via finetuning) complement it.
4. We empirically show that *e-valuator* outperforms baselines across several datasets, verifiers, and agents.

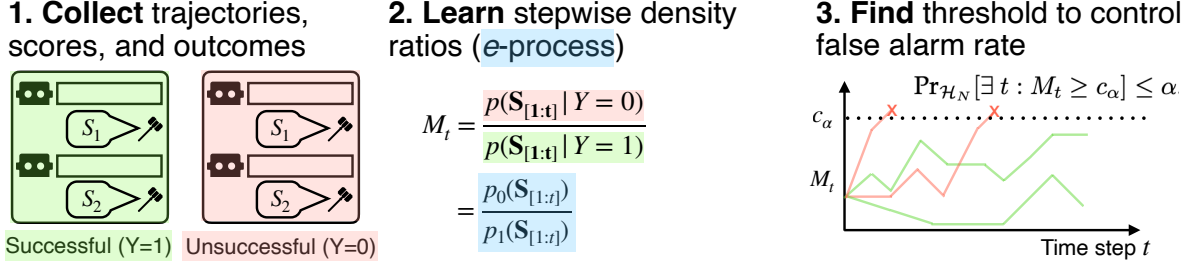


Figure 1: **Overview of e-evaluator.** *E-evaluator* works in three steps. First, we collect a small calibration set of trajectories, verifier scores, and labels. Second, we learn the density ratios \hat{M}_t at each timestep t . Third, we find the decision threshold that controls the false alarm rate using either Ville’s inequality or a quantile-based empirical approach. For a given threshold, unsuccessful trajectories (red) should be rejected at a higher rate than successful ones (green).

2 Methods

2.1 Problem Setting

Given a user prompt, denoted o_0 , the agent executes an sequence of $T \in \mathbb{N}_+$ actions, (a_1, \dots, a_T) , where $\mathbb{N}_+ := \{1, 2, \dots\}$ and T is a random variable depending on both the prompt and the agent’s internal randomness. Associated with each action, a_t , is an observation, o_t , which captures the environment state after the action is executed (e.g., if a_t performs some intermediary arithmetic calculation, o_t could contain the calculated value). The actions and observations together form the *trajectory* at each step t , $H_t = (o_0, a_1, \dots, a_t, o_t)$. After each step t , a black-box verifier model v takes as input the trajectory H_t and provides a score, $S_t = v(H_t)$, which serves as a heuristic evaluation of the quality of the trajectory thus far. Typically, $S_t \in [0, 1]$, although *e-evaluator* supports score values of any type. The verifier scores form a sequence $\mathbf{S} = (S_1, \dots, S_T)$.

A complete trajectory H_T is associated with a binary label, $Y \in \{0, 1\}$, of whether the final output, o_T , is correct ($Y = 1$) or not ($Y = 0$).¹ We call trajectories where $Y = 1$ *successful* trajectories and those with $Y = 0$ *unsuccessful* trajectories. We assume access to *calibration data*, $\mathcal{D}_{\text{cal}} = \{(\mathbf{S}^{(i)}, Y^{(i)})\}_{i=1}^n$, where $(\mathbf{S}^{(i)}, Y^{(i)})$ are drawn i.i.d from P , a joint distribution over variable-length score sequences and their labels, which has density p .

2.2 Evaluation via hypothesis testing

Given a new score sequence, \mathbf{S} , our goal is to determine whether the agent’s trajectory will produce a correct ($Y = 1$) or incorrect ($Y = 0$) final output. We formalize this goal as a hypothesis test. Let P_1 and P_0 denote the distributions of score sequences conditioned on correct and incorrect final outputs, respectively, with respective densities p_1 and p_0 . That is, $p_1(\mathbf{S}) = p(\mathbf{S} \mid Y = 1)$ and $p_0(\mathbf{S}) = p(\mathbf{S} \mid Y = 0)$. We assume $P_1 \neq P_0$, and we wish to test between two hypotheses:

$$\begin{aligned} \mathcal{H}_N : \mathbf{S} &\sim P_1 \quad (\text{the final output is correct}) \\ \mathcal{H}_A : \mathbf{S} &\sim P_0 \quad (\text{the final output is incorrect}) \end{aligned}$$

Note that P_1 and P_0 generally encode complex dependencies between the scores over time. That is, black-box verifier scores are generally not independent samples from a fixed distribution at every step, or otherwise amenable to convenient assumptions.

We construct a *sequential test* between \mathcal{H}_N and \mathcal{H}_A that can be run at each step t , using only $\mathbf{S}_{[1:t]}$, the scores based only on the agent’s trajectory up to step t . Specifically, we construct a sequence of test statistics,

¹Our framework also accommodates infinite-length trajectories that proceed indefinitely by labeling them $Y = 0$, as they never produce a final output o_T for some finite T . However, for practical relevance, we focus here on the setting of finite-length trajectories that always produce a final output.

$(M_t)_{t=1}^T$ where M_t is a real-valued function of $\mathbf{S}_{[1:t]}$, and a real-valued decision threshold, c_α , given a user-specified error level, $\alpha \in (0, 1)$. The sequential test then proceeds as follows (Alg. 1). For $t = 1, 2, \dots$, if $M_t \geq c_\alpha$, we reject \mathcal{H}_N . If we reach $t = T$, the end of the agent’s trajectory, without rejecting \mathcal{H}_N , then we accept \mathcal{H}_N . Our goal is to satisfy the following two criteria:

1. **False alarm rate control:**

$$\Pr_{\mathcal{H}_N}[\exists t \in [T] : M_t \geq c_\alpha] \leq \alpha, \quad (1)$$

where T , the length of the complete trajectory, is also random, and $[T] := \{1, \dots, T\}$. That is, the *false alarm rate*, or the rate at which we ever reject successful trajectories, is guaranteed to be at most α . This rate is also known classically as type I error. Notice that the probability under consideration is that M_t ever surpasses c_α —equivalently, that we *ever* reject the null hypothesis, \mathcal{H}_N —regardless of T , the total number of actions the agent takes. This notion of validity is known as *anytime validity*.

2. **High power:** Unsuccessful trajectories are often rejected. That is, we desire high rates of rejecting unsuccessful trajectories, $\Pr_{\mathcal{H}_A}[\exists t \in [T] : M_t \geq c_\alpha]$.

We develop a method that satisfies a high-probability version of the first criterion, and empirically achieves the second. The challenge of the first criterion is to construct a sequence of statistics, $(M_t)_{t=1}^T$, and decision threshold, c_α , such that the sequential test is valid, even though we do not *a priori* know T , the length of the complete trajectory. P-values, a standard tool in modern statistical hypothesis testing, are not naturally suited for this setting. Intuitively, the definition of a p-value—a random variable q such that $\Pr_{\mathcal{H}_N}(q \leq \alpha) \leq \alpha$ for all $\alpha \in [0, 1]$ —does not by itself imply anything about how a p-value constructed at a given step depends on those from previous steps. Such dependencies arise when the data (here, $\mathbf{S}_{[1:t]}$) observed at each step are dependent, but also simply by virtue of deciding whether or not to construct the next p-value based on the values of the previous ones, even if the data at each step are independent. Without additional assumptions and corresponding techniques to leverage them [12, 22, 45, 53], lack of knowledge of this dependence structure makes it unclear how to evaluate the probability of any rejection (see Appendix for a more detailed discussion). We turn instead to *e-processes* [44, 46], another object for quantifying evidence against a null hypothesis, whose definition explicitly characterizes the relationship between current and past values of the test statistic in a way that naturally enables probabilistic reasoning in these settings. An e-process for \mathcal{H}_N is a sequence, $(E_t)_t$, such that each E_t is an *e-value* for \mathcal{H}_N —that is, $\mathbb{E}_{\mathcal{H}_N}[E_t] \leq 1$ —and there exists a *test martingale* for \mathcal{H}_N , $(M_t)_t$, such that $E_t \leq M_t$ always. A test martingale for \mathcal{H}_N is a sequence, $(M_t)_t$, that satisfies two conditions:

1. **Non-negativity and unit mean:** M_t is non-negative for all t , and $E_{\mathcal{H}_N}[M_0] \leq 1$.
2. **Martingale.** $(M_t)_t$ is a martingale for \mathcal{H}_N . In our setting, this means $E_{\mathcal{H}_N}[M_t | M_0, M_1, \dots, M_{t-1}] = M_{t-1}$ for all t^2 .

Note that any test martingale is an e-process, but that e-processes are a much broader class of processes.

Constructing our sequence of test statistics, $(M_t)_{t=0}^T$, to be an e-process allows us to exploit certain properties to reason probabilistically at each step t about the entire sequence, regardless of how many total steps the agent ends up taking. In particular, Ville’s inequality states that for any e-process, $(M_t)_t$, we have $\Pr_{\mathcal{H}_N}[\exists t : M_t \geq 1/\alpha] \leq \alpha$ for every $\alpha \in [0, 1]$. In words, with probability at least $1 - \alpha$, the entire sequence—even if it proceeds indefinitely—is below $1/\alpha$, for every $\alpha \in [0, 1]$. This inequality concretely shows why an e-process can be interpreted as a quantification of evidence over time against the null hypothesis. If the null is true, then with high probability $(1 - \alpha)$, this evidence will stay low forever (specifically, under $1/\alpha$). If the alternative is true, a well-designed e-process grows large, as described momentarily.

²More formally, a martingale satisfies $E_{\mathcal{H}_N}[M_t | \mathcal{F}_{t-1}] = M_{t-1}$ for all t , where $(\mathcal{F}_t)_t$ is a filtration. For clarity of exposition and space constraints, we avoid a fully rigorous treatment here, but roughly, this means we are conditioning on all the information available in \mathcal{F}_{t-1} , which may be more or less information than conditioning on the exact values of the past. This is equivalent to conditioning on the exact values of the past when \mathcal{F}_t is the so-called natural filtration.

To construct an e-process for our setting, we use the ratio between the null density, p_1 , and alternative density, p_0 . Specifically, set $M_0 = 1$, and, for each step $t \in [T]$,

$$M_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}. \quad (2)$$

Since the density ratio process is a test martingale and therefore an e-process [44] (see the Appendix for a proof), Algorithm 1 using this choice of M_t and $c_\alpha = 1/\alpha$ allows us to achieve anytime-valid control of the false alarm rate (Prop. 1).

Algorithm 1 Sequential hypothesis testing framework for agent verification.

Inputs: user prompt, o_0 ; functions that take in scores and output test statistic for each step t , $M_t(\cdot)$, $t = 1, \dots$; decision threshold, $c_\alpha \in \mathbb{R}$.

Output: decision, reject \mathcal{H}_N or accept \mathcal{H}_N .

```

1:  $c_\alpha \leftarrow 1/\alpha$ 
2:  $H_0 \leftarrow (o_0)$ 
3: for  $t = 1, \dots, T$  do
4:   Given  $H_{t-1}$ , agent executes action  $a_t$ , resulting in environment state  $o_t$ .
5:   Update trajectory,  $H_t \leftarrow (o_0, a_1, o_1, \dots, a_t, o_t)$ .
6:   Get verifier score,  $S_t \leftarrow v(H_t)$ .
7:   Evaluate test statistic for this step,  $M_t(\mathbf{S}_{[1:t]})$   $\triangleright \mathbf{S}_{[1:t]} := (S_1, \dots, S_t)$ 
8:   if  $M_t(\mathbf{S}_{[1:t]}) \geq c_\alpha$  then
9:     return reject  $\mathcal{H}_N$ 
10:  end if
11: end for
12: return accept  $\mathcal{H}_N$ 

```

Proposition 1. For any fixed $\alpha \in (0, 1)$, Algorithm 1 using the density ratio process, $M_t = p_0(\mathbf{S}_{[1:t]})/p_1(\mathbf{S}_{[1:t]})$, and the decision threshold $c_\alpha = 1/\alpha$ achieves anytime-valid control of the false alarm rate (Eq. 1).

Proof. (Sketch) The density ratio process, $(M_t)_{t=0}^T$, is an e-process. Anytime-valid control of the false alarm rate follows by Ville’s inequality. See Appendix 8.1 for a full proof. \square

Although Proposition 1 establishes anytime-valid control of the false alarm rate, it does not illuminate why one should choose the density ratio process rather than any other e-process. It turns out that under \mathcal{H}_A , the density ratio process grows the fastest (in expectation) over time among all e-processes, a notion called log-optimality made precise in Proposition 2. Log-optimality is analogous to being the “most powerful” test statistic in non-sequential hypothesis testing: intuitively, M_t will tend to surpass the decision threshold, and correspondingly enable detection of unsuccessful trajectories earlier than other e-processes.

Proposition 2. The density ratio process, $M_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}$, is log-optimal. That is, for any other e-process $(M'_t)_{t=0}^T$ and stopping time τ , $E_{\mathcal{H}_A}[\log M_\tau] \geq E_{\mathcal{H}_A}[\log M'_\tau]$.

Proof. (Sketch) This is a sequential analog of the Neyman-Pearson lemma [38]. The proof, also provided in [44], follows by first noting that any particular M_t is the log-optimal e-variable among all e-variables. One can then invoke properties of e-processes to complete the proof. See Appendix 8.1 for a full proof. \square

2.3 Density ratio estimation

In practice, the forms of p_1 and p_0 are typically not known. Prior to deployment, the method we propose, *e-valuator*, therefore has a calibration phase in which it uses the calibration data, \mathcal{D}_{cal} , to learn a model of the

density ratio, $\hat{M}_t(\mathbf{S}_{[1:t]}) \approx \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}$, for each step t (Alg. 2). We do this using classifier-based density ratio estimation, an approach based on the following equality by Bayes' rule [4, 16]:

$$M_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})} = \frac{p(Y=0|\mathbf{S}_{[1:t]}) p(Y=1)}{p(Y=1|\mathbf{S}_{[1:t]}) p(Y=0)}. \quad (3)$$

Specifically, for each time step t , we train a classifier, \hat{f}_t , which takes $\mathbf{S}_{[1:t]}$ as input and provides an estimate of $p(Y=1|\mathbf{S}_{[1:t]})$. We also form an estimate, $\hat{\pi}_1$, of the class probability $p(Y=1)$. We plug these two estimates into Eq. (3) to form the following estimated density ratio at step t :

$$\hat{M}_t = \frac{1 - \hat{f}_t(\mathbf{S}_{[1:t]})}{\hat{f}_t(\mathbf{S}_{[1:t]})} \frac{\hat{\pi}_1}{1 - \hat{\pi}_1}.$$

We will refer to running Algorithm 1 with these estimated density ratios, \hat{M}_t , and the same decision threshold motivated by Ville's inequality, $c_\alpha = 1/\alpha$, as *e-valuator with $1/\alpha$ threshold*. The guarantees in Propositions 1 and 2 apply when we learn the true density ratios, i.e., $\hat{M}_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}$ for each point $\mathbf{S}_{[1:t]}$ and each step t .

In our experiments, we used simple logistic regression for the classifier at each step, \hat{f}_t , and found that estimated density ratios learned from a few hundred calibration points empirically achieved both false alarm rate control and superior power than alternative methods (see Appendix 8.2.2 for details).

Algorithm 2 Density ratio estimation using calibration data.

Inputs: calibration data, \mathcal{D}_{cal} .

Output: functions that estimate the density ratio for every step, $\{\hat{M}_t\}_{t \in \mathbb{N}_+}$.

- 1: Split \mathcal{D}_{cal} randomly into \mathcal{D}_{DRE} and $\mathcal{D}_{\text{threshold}}$, where $\mathcal{D}_{\text{DRE}} \cup \mathcal{D}_{\text{threshold}} = \mathcal{D}_{\text{cal}}$.
 - 2: $\hat{\pi}_1 \leftarrow \frac{1}{|\mathcal{D}_{\text{DRE}}|} \sum_{(\mathbf{S}, Y) \in \mathcal{D}_{\text{DRE}}} Y$ ▷ Estimate class probabilities as empirical frequencies.
 - 3: $T_{\text{max}} \leftarrow \max\{T : \exists \text{ both successful and unsuccessful score sequences in } \mathcal{D}_{\text{DRE}} \text{ with length } T\}$
 - 4: **for** $t = 1, \dots, T_{\text{max}}$ **do**
 - 5: $\mathcal{D}_{\text{DRE}, t} \leftarrow \{(\mathbf{S}_{[1:t]}, Y) : (\mathbf{S}, Y) \in \mathcal{D}_{\text{DRE}} : |\mathbf{S}| \geq t\}$
 - 6: Use $\mathcal{D}_{\text{DRE}, t}$ to train probabilistic classifier, \hat{f}_t , that takes $\mathbf{S}_{[1:t]}$ as input and predicts $p(Y=1 | \mathbf{S}_{[1:t]})$.
 - 7: $\hat{M}_t(\cdot) \leftarrow \frac{1 - \hat{f}_t(\cdot)}{\hat{f}_t(\cdot)} \frac{\hat{\pi}_1}{1 - \hat{\pi}_1}$
 - 8: **end for**
 - 9: Set $\hat{M}_t(\cdot) \leftarrow \hat{M}_{T_{\text{max}}}(\cdot)$ for all $t > T_{\text{max}}$.
-

2.4 Increasing power via quantile estimation

Given the true density ratio process, M_t , rejecting \mathcal{H}_N whenever M_t surpasses the decision threshold $c_\alpha = 1/\alpha$ achieves anytime-valid control of the false alarm rate (Prop. 1). However, because we use estimated density ratios in practice, the threshold c_α needs to account for density ratio estimation error. Furthermore, setting c_α to achieve anytime-validity may be overly conservative, as it guarantees an upper bound on false alarm rate even if the agent keeps executing actions indefinitely. In contrast, agents in practice will typically produce a final output or be terminated after a finite number of actions.

We thus propose an alternative procedure for setting the decision threshold c_α to account for both of these factors, while retaining a high-probability version of anytime false alarm control. We first split the calibration set, \mathcal{D}_{cal} into two splits, \mathcal{D}_{DRE} and $\mathcal{D}_{\text{threshold}}$, where \mathcal{D}_{DRE} is first used for density ratio estimation (Alg. 2), and then $\mathcal{D}_{\text{threshold}}$ is used to set c_α as follows.

First, note that rejecting \mathcal{H}_N if the process \hat{M}_t ever surpasses c_α is equivalent to rejecting \mathcal{H}_N if $\max_t \hat{M}_t \geq c_\alpha$. It thus suffices to set c_α to the $(1 - \alpha)$ quantile of the distribution of $\max_t \hat{M}_t$ under the null.

Accordingly, we focus on successful trajectories in $\mathcal{D}_{\text{threshold}}$, which correspond to \mathcal{H}_N . For each of these trajectories i in the calibration set, we record the maximum estimated density ratio over all steps, $M^{(i)} = \max\{\hat{M}_1^{(i)}, \dots, \hat{M}_T^{(i)}\}$. This maximum value, $M^{(i)}$, is a sample from the null distribution. Given these samples, we construct $\hat{q}_{1-\alpha}$, a high-probability upper bound on the $(1 - \alpha)$ -quantile of the null distribution (Alg. 3). We refer to running Algorithm 1 with the estimated density ratios, \hat{M}_t , and the decision threshold set to this high-probability upper bound, $c_\alpha = \hat{q}_{1-\alpha}$, as *e-evaluator with probably-approximately-correct (PAC) threshold*, due to the following guarantee. In words, with probability at least $1 - \delta$ for any user-specified δ (that is, “probably”), the procedure has anytime control of the false alarm rate under a user-specified α (“approximately correct”).

Proposition 3. *Let $\{\hat{M}_t\}_{t \in \mathbb{N}_+}$ denote the estimated density ratio functions output by Algorithm 2. For fixed error level $\delta \in (0, 1)$ and quantile level $\alpha \in (0, 1)$, let c_α be the output of Algorithm 3. Then Algorithm 1 using $\{\hat{M}_t\}_{t \in \mathbb{N}_+}$ and decision threshold c_α , or e-evaluator with probably-approximately-correct (PAC) threshold, satisfies*

$$\Pr_{\mathcal{D}_{\text{cal}}} (\Pr_{\mathcal{H}_N} (\exists t \in [T] : M_t \geq c_\alpha \mid \mathcal{D}_{\text{cal}}) \leq \alpha) \geq 1 - \delta \quad (4)$$

Proof. (Sketch) Finding $\hat{q}_{1-\alpha}$ such that $\hat{q}_{1-\alpha} \geq q_{1-\alpha}$ with high probability reduces to finding the index k such that the order statistic $M_{(k)}$ among calibration set null samples is above $q_{1-\alpha}$ with high probability. See Appendix 8.1 for the full proof. \square

3 Related Work

As *e-evaluator* is a statistical wrapper for any verifier, our work is relevant for but orthogonal to prior work on building better verifiers for agents. These verifiers are often trained as reward models that estimate a reward (e.g., correctness or coherence) after each step in an agent’s action sequence. Among these, process-reward models (PRMs) are finetuned using agent trajectories where each *step* is labeled as correct or incorrect, as in a mathematical reasoning trace [55, 63, 26]. Some prior works also work to calibrate existing PRMs [72, 39], although calibration alone is insufficient to control false alarm rates, as we do in this paper.

Training PRMs can be expensive, as it requires (a) access to human-annotated process labels [31] and (b) finetuning existing LLMs [63]. Alternative verifiers to PRMs include LLM-as-a-judge (i.e., prompt-based verification) [2] and outcome reward models, which only provide a label for the entire trajectory [10]. There are several benchmarks to compare verifiers on trajectories [35] and processes [73].

Some prior works build hypothesis tests atop classifiers to monitor AI deployments, albeit in different contexts. Vovk et al. [59] propose using conformal test martingales (CTMs) for continual monitoring of model deployments, and Prinster et al. [41] develop weighted CTMs to enable adaptation at test time and analyze the cause of degradation. Podkopaev and Ramdas [40] leverage sequential testing for directly tracking the risk of deployed models, which has been extended to monitoring without labels [20], under test-time adaptation [47], and for unknown shifts [54]. In a similar vein, [7] applies safe anytime-valid testing to sequentially test if a classifier is fair. [21] applies classifier two-sample tests [32] to detect whether there is a covariate shift between the training and deployment setting. More broadly, several prior works model class-conditional densities of classifier scores and the corresponding density ratios for evaluation of AI systems, albeit without hypothesis testing [49, 68]. Although these methods typically model the densities generatively [11], some also model the densities discriminatively, as we do here [23].

Another line of work tries to add formal statistical controls to LLMs. Conformal prediction methods [48] quantify uncertainty in individual predictions while providing finite-sample and black-box guarantees in those uncertainties. Conformal prediction has been applied to resample LLM generations until a minimum quality requirement is satisfied [43]. These ideas have been further applied to control the factuality of LLM outputs [6, 36] and to generally benchmark LLMs’ confidence in their outputs [71]. More recently, Wu et al. [69] applies the learn-then-test framework [1] to calibrate a stopping rule for LLM reasoning traces, using white-box access to the LLM’s internal logits.

E-evaluator directly builds on prior work in e-values. E-values were originally introduced as an alternative to p-values, with useful properties for sequential hypothesis testing [44, 57, 58]. E-values provide anytime validity over a (potentially infinite) sequence of tests [62, 67, 66, 15]. They have found important

applications in A/B testing [25], changepoint detection [50, 33] and LLM-based hypothesis verification [18]. Our instantiation of the e-process is related to the sequential probability ratio test [60].

4 Experiments

To empirically demonstrate that *e-evaluator* achieves false alarm rate control, as well as higher power than alternative methods, we conducted comprehensive experiments across six datasets and tasks using four different agent-verifier combinations. We additionally show that *e-evaluator* can be applied to recover a large fraction of the original accuracy within a limited token budget.

Agents and verifiers. We conduct experiments on two tool-calling agents, Aviary [37] and Octotools [34], one step-by-step reasoning model, Claude Sonnet 4, and on online chess games from a public repository. For Aviary and Octotools, we use Claude Haiku 3.5 as the verifier, asking after each tool call for a text-based probability that the trajectory thus far is successful. For the reasoning model, we use a popular pretrained process-reward model (PRM) [63], which provides a logit-based probability that each step in a reasoning trace is correct. Finally, for the chess experiments, we use Stockfish as the reward model, an open-sourced verifier that numerically evaluates the strength of White’s position after each step.

Datasets. We experiment on datasets from three different domains: (1) **mathematical reasoning** (GSM8k [9] and MATH [17]), (2) **question-answering** (HotpotQA [70], MedQA [24], and MMLU-Pro [64]), and (3) **chess games**. For **chess**, we use open-sourced and annotated games from LiChess. We present results from all datasets except GSM8k in the main section, and provide the GSM8k results in the Appendix. Results for each dataset are from one corresponding agent-verifier combination. A full description of the dataset, agent, and verifier combinations is available in Appendix 8.3.

Baselines We compare against three baselines inspired by sequential hypothesis testing and calibration.

1. The **raw verifier** uses the scores from the verifier as-is. The verifier provides some prediction of $\Pr(Y = 1|H_t)$, the probability that the agent will produce a successful output, given the trajectory H_t thus far. Given a user-specified false positive rate, α , we reject a trajectory if the score S_t ever drops below α .
2. The **calibrated verifier** uses the same verifier but recalibrates the scores S_t using the calibration set \mathcal{D}_{cal} . Specifically, we use isotonic regression to learn a transformation of the score, $S' = \hat{f}(S)$, that achieves *marginal calibration*: $\Pr(Y = 1|S') = S'$. As with the raw verifier, for a user-specified false positive rate, α , we reject a trajectory if the score S_t ever drops below α .
3. The **Bonferroni test** utilizes the same density ratio test statistic M_t . However, the decision rule differs: it uses the Bonferroni correction to reject each individual test at α/T , where α is the user-specified false positive rate, and T is the maximum trajectory length found in the calibration set.

We compare these baselines to two variations of *e-evaluator*: (1) **e-evaluator ($1/\alpha$ threshold)**, in which we choose the threshold according to Proposition 1 with our plug-in estimate for the density ratios, and (2) **e-evaluator (PAC threshold)**, in which we choose the threshold using the procedure in Proposition 3. In general, we recommend using the PAC threshold, although in settings where an agent may run an arbitrarily large number of steps or never terminate, the $1/\alpha$ version may be preferred. For the results presented in the main section, we use an 80/20 split of test/calibration data: we calibrate our method (and all baselines) on 20% of the data and test on the remaining 80%. We compare other splits of the calibration set in Appendix 8.2.2 and find that a couple hundred calibration trajectories is sufficient to control the false alarm rate. We also visualize the M_t sequences for both successful and unsuccessful trajectories in Appendix 8.2.

4.1 E-evaluator provides better false alarm rate control than competing methods

We begin by analyzing the empirical false alarm rate achieved by *e-evaluator* and the competing baselines. Given a particular α , the false alarm rate, i.e. the rate at which we flag successful (“null”) trajectories as unsuccessful (“alternative”), should be no greater than α .

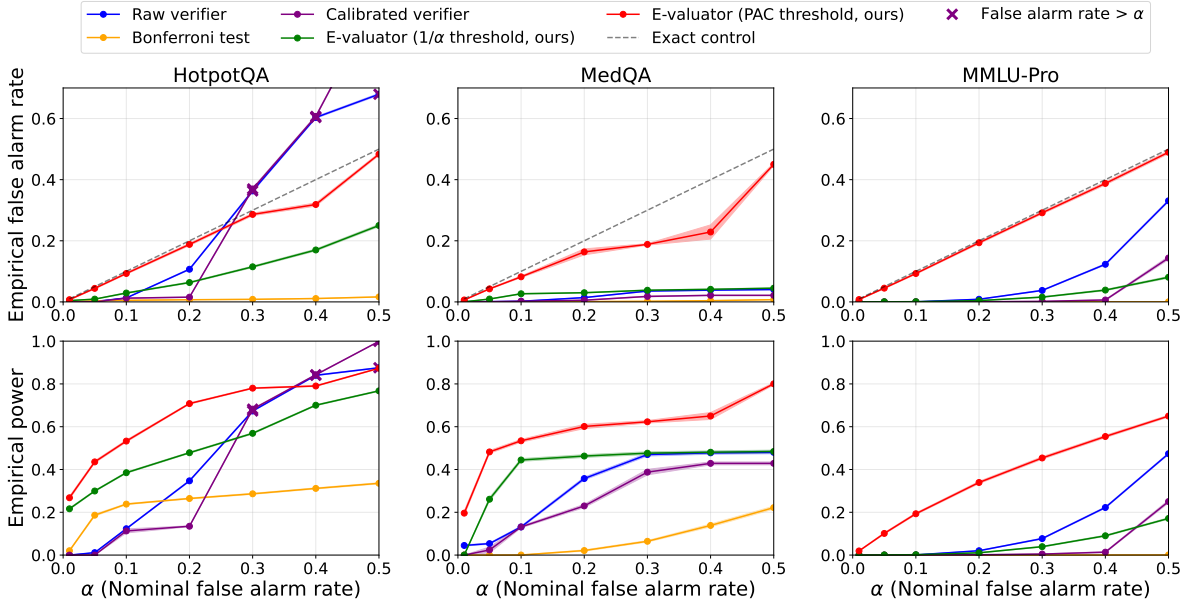


Figure 2: **E-evaluator controls the false alarm rate and maximizes power better than alternative methods.** Violations of the false alarm rate control are marked with an X. Both versions of *e-evaluator* empirically control the false alarm rate (type I error) for different choices of α across all datasets. As expected, the $1/\alpha$ threshold is more conservative than the PAC threshold, although both control the false alarm rate. **E-evaluator** also provides better power than competing methods. The calibrated and raw verifiers occasionally provide comparable power, at the cost of inflating the false alarm rate. All plots show the 95% CI over 50 random splits of each dataset.

Both the $1/\alpha$ threshold (plug-in version of Proposition 1) and PAC threshold (Proposition 3) versions of *e-evaluator* empirically control the false alarm rate across all choices of α and all datasets (Figure 2, top). The raw verifier, which thresholds the raw scores S_t at α , sometimes achieves empirical false alarm rates less than the desired α (MedQA and MATH; see Appendix 8.2 for the latter) but not always (HotpotQA, for $\alpha > 0.4$). The calibrated verifier, which applies isotonic regression to the raw verifier scores S_t and then uses the same threshold α on the recalibrated scores, does not achieve false alarm rates less than α either. Although calibration procedures such as isotonic regression aim to achieve $E(Y|S) = S$, this property does not have any direct implications on the false alarm rate. Furthermore, even if this property held at each timestep, it does not allow us to reason about the false alarm rate in the sequential hypothesis setting (see Appendix 8.1 for further discussion). In contrast, the Bonferroni test (Figure 2, orange) controls the false alarm rate for all α , although even more conservatively than *e-evaluator* with a $1/\alpha$ threshold.

We expect the $1/\alpha$ threshold and PAC thresholds to converge as the trajectory length increases, as the $1/\alpha$ threshold is designed to be valid for any density ratio process, whereas the PAC thresholds are calibrated to the observed null distribution. This expectation aligns with what we empirically observe. For instance, the average trajectory length in MedQA, which achieves a false alarm rate of 0.045 at $\alpha = 0.5$ with a threshold of $1/\alpha$, is 2.4 steps. In comparison, the average trajectory length in HotpotQA, which achieves at false alarm rate of 0.21 at $\alpha = 0.5$ with a threshold of $1/\alpha$, is 4.7 steps. Similar plots are available in Appendix 8.2 for the MATH and GSM8k datasets.

4.2 E-evaluator provides enhanced power over alternate methods

Next we analyze the empirical power across the same datasets and tasks (Figure 2, bottom). Power measures the true positive rate at a given α . That is, power is the rate at which unsuccessful trajectories (“alternative”) are indeed flagged as unsuccessful. Across all datasets and all α s, *e-evaluator* with the PAC threshold provides greatest power among methods that achieve empirical false alarm rates less than α (Figure 2, top, red).

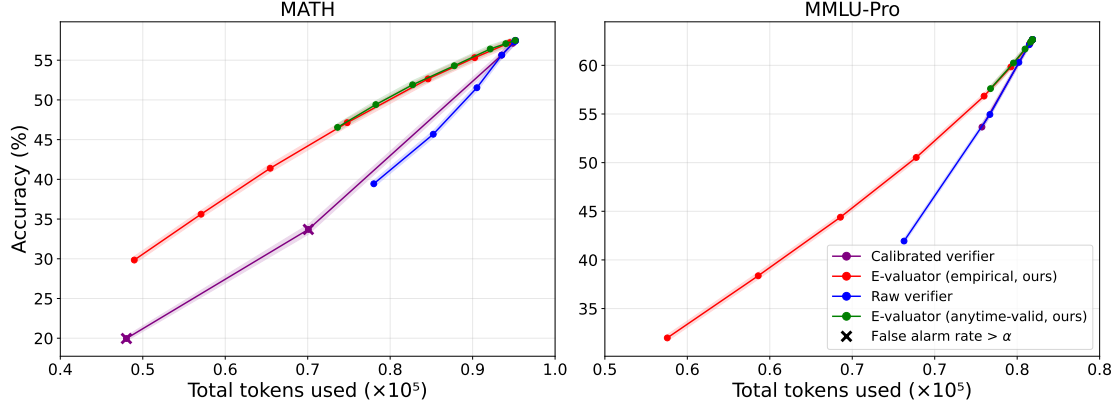


Figure 3: **E-evaluator recovers a larger fraction of baseline accuracy with fewer tokens.** We compare *e-evaluator* to thresholding the verifier scores on the MATH and MMLU-Pro dataset. The verifier terminates unsuccessful trajectories rather late, leading to greater inefficiencies in recovering accuracy with fewer tokens. \times indicates that the empirical false alarm rate was greater than the desired level

In those instances where the calibrated (or raw) verifier provides better power, it is at the cost of inflated false alarm rate. For instance, in HotpotQA, at $\alpha = 0.5$, the calibrated verifier provides a power of 0.84 but inflates the false alarm rate to 0.61, well above the desired level, while empirical *e-evaluator* provides a power of 0.81 and controls the false alarm rate at 0.48, and *E-evaluator* ($1/\alpha$ threshold) also provides strong power at 0.62 while controlling the false alarm rate at 0.21, well below $\alpha = 0.5$.

4.3 Case study: E-evaluator recovers larger fraction of original accuracy in limited token budget

We also apply *e-evaluator* to a case study in which we terminate any trajectory that crosses the threshold c_α . We then count the number of tokens that are saved (i.e., how many tokens would have otherwise been generated if the agent weren’t terminated) and compare that to the total accuracy on the dataset. Note that using the full token budget is the “maximal” accuracy achievable, as this would entail never terminating any trajectory.

Comparing *e-evaluator* to the verifier in terms tokens saved versus total accuracy (Figure 3) we find shows that *E-evaluator* outperforms both the raw and calibrated verifiers in recovering accuracy with fewer tokens. For instance, on the MATH dataset, *e-evaluator* achieved 50% total accuracy (86% of the original accuracy of 58%) using 81% (fewer than 269,755 tokens) of the original 333,283 tokens. By contrast, the raw and calibrated verifiers each require more than 95% of the original token count to recover 86% of the original accuracy. Similarly, on the MMLU-Pro dataset, *e-evaluator* achieved 50% total accuracy using just 233,324 tokens, whereas the raw and calibrated verifiers each require more than 250,000 tokens.

4.4 Case study: E-evaluator for chessbots

Finally, we show a use case of *e-evaluator* for monitoring non-LLM agents. In this experiment, we analyze publicly available chess games on LiChess, using Stockfish as the verifier. Stockfish provides a real-valued score (possibly negative) called *centipawns*, which is positive when White is an advantageous position and negative when Black is an advantageous position. Stockfish additionally publishes a formula to convert these scores into a win probability [30], which we use to construct the verifier’s raw probabilities. We test the null of White winning against the alternative of Black winning or a draw.

We compare the raw and calibrated verifier to *e-evaluator* and find that *e-evaluator* is able to better control the false alarm rate than these baselines (Figure 4). Moreover, *e-evaluator* with the $1/\alpha$ threshold performs similarly to the PAC threshold version. This is expected in chess, where games often feature more moves (e.g. 50+ moves) than actions in typical agent trajectories. The $1/\alpha$ threshold fails to control the false alarm

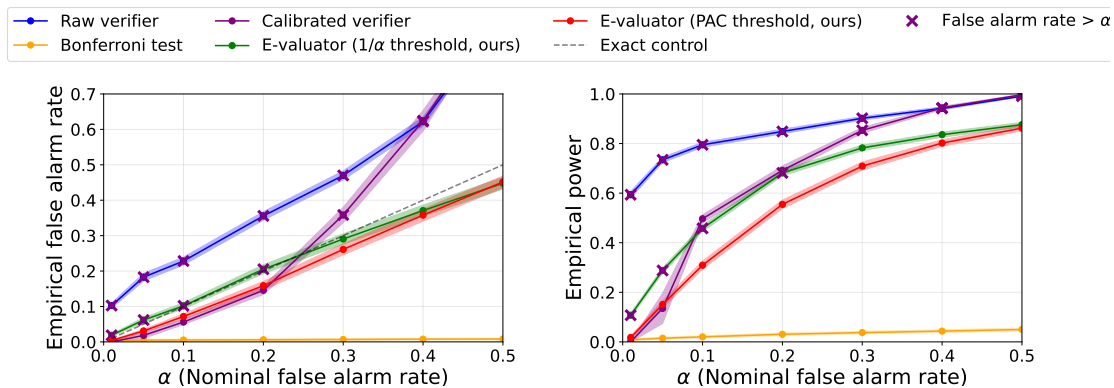


Figure 4: **Chess**. E-evaluator controls the false alarm rate and increases power for chess verifiers.

rate at low α , which is attributable to error in density ratio estimation for long games. Nonetheless, the verifier models more severely violate the false alarm rate control than *e-evaluator* ($1/\alpha$ threshold).

5 Discussion

In this paper, we introduced *e-evaluator*, a method to improve any step-by-step agent verifier model using sequential hypothesis testing. We convert the problem of detecting whether a trajectory is “successful” or not into a hypothesis testing problem, where we distinguish between verifier scores generated from the “successful” versus the “unsuccessful” distribution. Although *e-evaluator* requires evaluating density ratios at each step in an agent’s trajectory, we empirically find that learned density ratios provide ample power and sufficient false alarm controls.

There are several promising future directions to this work. First, one can relax certain assumptions to avoid estimating the *full* joint density at each time t , such as assuming the verifier scores are i.i.d. at each step, or perhaps that the score at any given step depends at most on the scores from the last k steps. If one assumes the scores are i.i.d., one could use universal inference algorithms [65] to provide exact guarantees even with empirical (and noisy) estimates of the true density ratio. Second, although we explored a potential test-time scaling application, one could use *e-evaluator* for more nuanced scaling strategies, such as resampling or restarting bad trajectories. Some of these strategies may break the *e-evaluator* assumptions, and it would be important to ameliorate the method accordingly. Finally, *e-evaluator* can be extended to more complex agentic systems, such as multi-agent settings.

6 Code and Data

Code is released on GitHub at <https://github.com/shuvom-s/e-evaluator> and additionally as a Python package, *e-evaluator*, available in PyPi at <https://pypi.org/project/e-evaluator/>.

7 Acknowledgments

We thank Bonnie Berger, Ian Waudby-Smith, Kexin Huang, Divya Shanmugam, Kyunghyun Cho, Manish Raghavan, the Recht Lab, and Chang Ma for helpful feedback and discussions. This work was done while S.S. and D.P. were interns at Genentech.

References

- [1] Anastasios N Angelopoulos, Stephen Bates, Emmanuel J Candès, Michael I Jordan, and Lihua Lei. Learn then test: Calibrating predictive algorithms to achieve risk control. *The Annals of Applied Statistics*, 19(2):1641–1662, 2025.
- [2] Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, et al. Llms instead of human judges? a large scale empirical study across 20 nlp evaluation tasks. *arXiv preprint arXiv:2406.18403*, 2024.
- [3] George A Bekey. On autonomous robots. *The Knowledge Engineering Review*, 13(2):143–146, 1998.
- [4] S Bickel, M Bruckner, and T Scheffer. Discriminative learning under covariate shift. *J. Mach. Learn. Res.*, 10:2137–2155, 2009.
- [5] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.
- [6] John Cherian, Isaac Gibbs, and Emmanuel Candes. Large language model validity via enhanced conformal prediction methods. *Advances in Neural Information Processing Systems*, 37:114812–114842, 2024.
- [7] Ben Chugg, Santiago Cortes-Gomez, Bryan Wilder, and Aaditya Ramdas. Auditing fairness by betting. *Advances in Neural Information Processing Systems*, 36:6070–6091, 2023.
- [8] Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.
- [9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [10] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.
- [11] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):20–28, 1979.
- [12] Dean P Foster and Robert A Stine. α -investing: A procedure for sequential control of expected false discoveries. *J. R. Stat. Soc. Series B Stat. Methodol.*, 70(2):429–444, April 2008.
- [13] Senay A Gebreab, Khaled Salah, Raja Jayaraman, Muhammad Habib ur Rehman, and Samer Ellaham. LLM-based framework for administrative task automation in healthcare. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–7. IEEE, April 2024.
- [14] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.
- [15] Peter Grünwald, Rianne de Heide, and Wouter M Koolen. Safe testing. In *2020 Information theory and applications workshop (ITA)*, pages 1–54. IEEE, 2020.
- [16] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res.*, 13(11):307–361, 2012.
- [17] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

- [18] Kexin Huang, Ying Jin, Ryan Li, Michael Y Li, Emmanuel Candes, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications. *Forty-second International Conference on Machine Learning*, 2025.
- [19] Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *bioRxiv*, 2025.
- [20] Salim I Amoukou, Tom Bewley, Saumitra Mishra, Freddy Lecue, Daniele Magazzeni, and Manuela Veloso. Sequential harmful shift detection without labels. *Advances in Neural Information Processing Systems*, 37:129279–129302, 2024.
- [21] Sooyong Jang, Sangdon Park, Insup Lee, and Osbert Bastani. Sequential covariate shift detection using classifier two-sample tests. In *International conference on machine learning*, pages 9845–9880. PMLR, 2022.
- [22] Adel Javanmard and Andrea Montanari. Online rules for control of false discovery rate and false discovery exceedance. *Ann. Stat.*, 46(2):526–554, 2018.
- [23] Disi Ji, Padhraic Smyth, and Mark Steyvers. Can i trust my fairness metric? assessing fairness with unlabeled data and bayesian inference. *Advances in Neural Information Processing Systems*, 33:18600–18612, 2020.
- [24] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.
- [25] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. Peeking at a/b tests: Why it matters, and what to do about it. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1517–1525, 2017.
- [26] Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.
- [27] Shi Xuan Leong, Caleb E Griesbach, Rui Zhang, Kouros Darvish, Yuchi Zhao, Abhijoy Mandal, Yunheng Zou, Han Hao, Varinia Bernales, and Alán Aspuru-Guzik. Steering towards safe self-driving laboratories. *Nat. Rev. Chem.*, 9(10):707–722, October 2025.
- [28] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2757–2791, 2025.
- [29] Wendi Li and Yixuan Li. Process reward model with q-value rankings. *The Thirteenth International Conference on Learning Representations*, 2025.
- [30] Lichess.org. Lichess accuracy metric. <https://lichess.org/page/accuracy>, n.d. Accessed: 2025-11-17.
- [31] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- [32] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*, 2016.
- [33] Gary Lorden. Procedures for reacting to a change in distribution. *The annals of mathematical statistics*, pages 1897–1908, 1971.
- [34] Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025.

- [35] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025.
- [36] Christopher Mohri and Tatsunori Hashimoto. Language models with conformal factuality guarantees. *arXiv preprint arXiv:2402.10978*, 2024.
- [37] Siddharth Narayanan, James D Braza, Ryan-Rhys Griffiths, Manu Ponnampati, Albert Bou, Jon Laurent, Ori Kabeli, Geemi Wellawatte, Sam Cox, Samuel G Rodrigues, et al. Aviary: training language agents on challenging scientific tasks. *arXiv preprint arXiv:2412.21154*, 2024.
- [38] Jerzy Neyman and Egon Sharpe Pearson. Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933.
- [39] Young-Jin Park, Kristjan Greenewald, Kaveh Alim, Hao Wang, and Navid Azizan. Know what you don’t know: Uncertainty calibration of process reward models. *arXiv preprint arXiv:2506.09338*, 2025.
- [40] Aleksandr Podkopaev and Aaditya Ramdas. Tracking the risk of a deployed model and detecting harmful distribution shifts. *arXiv preprint arXiv:2110.06177*, 2021.
- [41] Drew Prinster, Xing Han, Anqi Liu, and Suchi Saria. Watch: Adaptive monitoring for ai deployments via weighted-conformal martingales. In *Forty-second International Conference on Machine Learning*, 2025.
- [42] Yuanhao Qu, Kaixuan Huang, Ming Yin, Kanghong Zhan, Dyllan Liu, Di Yin, Henry C Cousins, William A Johnson, Xiaotong Wang, Mihir Shah, Russ B Altman, Denny Zhou, Mengdi Wang, and Le Cong. CRISPR-GPT for agentic automation of gene-editing experiments. *Nat. Biomed. Eng.*, pages 1–14, July 2025.
- [43] Victor Quach, Adam Fisch, Tal Schuster, Adam Yala, Jae Ho Sohn, Tommi S Jaakkola, and Regina Barzilay. Conformal language modeling. *arXiv preprint arXiv:2306.10193*, 2023.
- [44] Aaditya Ramdas and Ruodu Wang. Hypothesis testing with e-values. *arXiv preprint arXiv:2410.23614*, 2024.
- [45] Aaditya Ramdas, Tijana Zrnic, Martin Wainwright, and Michael Jordan. SAFFRON: an adaptive algorithm for online control of the false discovery rate. In *International Conference on Machine Learning*, pages 4286–4294. PMLR, July 2018.
- [46] Aaditya Ramdas, Peter Grünwald, Vladimir Vovk, and Glenn Shafer. Game-theoretic statistics and safe anytime-valid inference. *Statistical Science*, 38(4):576–601, 2023.
- [47] Mona Schirmer, Metod Jazbec, Christian A Naesseth, and Eric Nalisnick. Monitoring risks in test-time adaptation. *arXiv preprint arXiv:2507.08721*, 2025.
- [48] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.
- [49] Divya Shanmugam, Shuvom Sadhuka, Manish Raghavan, John Gutttag, Bonnie Berger, and Emma Pierson. Evaluating multiple models using labeled and unlabeled data. *arXiv preprint arXiv:2501.11866*, 2025.
- [50] Jaehyeok Shin, Aaditya Ramdas, and Alessandro Rinaldo. E-detectors: A nonparametric framework for sequential change detection. *The New England Journal of Statistics in Data Science*, 2023.
- [51] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

- [52] Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, pages 1–3, 2025.
- [53] Jinjin Tian and Aaditya Ramdas. ADDIS: an adaptive discarding algorithm for online FDR control with conservative nulls. *Advances in Neural Information Processing Systems*, 32, 2019.
- [54] Alexander Timans, Rajeev Verma, Eric Nalisnick, and Christian A Naeseth. On continuous monitoring of risk violations under unknown shift. *arXiv preprint arXiv:2506.16416*, 2025.
- [55] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.
- [56] Jean Ville. Jean ville, étude critique de la notion de collectif. 1939.
- [57] Vladimir Vovk and Ruodu Wang. E-values: Calibration, combination and applications. *The Annals of Statistics*, 49(3):1736–1754, 2021.
- [58] Vladimir Vovk and Ruodu Wang. Confidence and discoveries with e-values. *Statistical Science*, 38(2): 329–354, 2023.
- [59] Vladimir Vovk, Ivan Petej, Ilia Nouretdinov, Ernst Ahlberg, Lars Carlsson, and Alex Gammerman. Retrain or not retrain: Conformal test martingales for change-point detection. In *Conformal and Probabilistic Prediction and Applications*, pages 191–210. PMLR, 2021.
- [60] Abraham Wald and Jacob Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, pages 326–339, 1948.
- [61] Hanchen Wang, Yichun He, Paula P Coelho, Matthew Bucci, Abbas Nazir, Bob Chen, Linh Trinh, Serena Zhang, Kexin Huang, Vineethkrishna Chandrasekar, et al. Spatialagent: An autonomous ai agent for spatial biology. *bioRxiv*, pages 2025–04, 2025.
- [62] Hongjian Wang and Aaditya Ramdas. Anytime-valid t-tests and confidence sequences for gaussian means with unknown variance. *Sequential Analysis*, 44(1):56–110, 2025.
- [63] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.
- [64] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.
- [65] Larry Wasserman, Aaditya Ramdas, and Sivaraman Balakrishnan. Universal inference. *Proceedings of the National Academy of Sciences*, 117(29):16880–16890, 2020.
- [66] Ian Waudby-Smith, Edward H Kennedy, and Aaditya Ramdas. Distribution-uniform anytime-valid sequential inference. *arXiv preprint arXiv:2311.03343*, 2023.
- [67] Ian Waudby-Smith, Lili Wu, Aaditya Ramdas, Nikos Karampatziakis, and Paul Mineiro. Anytime-valid off-policy inference for contextual bandits. *ACM/IMS Journal of Data Science*, 1(3):1–42, 2024.
- [68] Peter Welinder, Max Welling, and Pietro Perona. A lazy man’s approach to benchmarking: semisupervised classifier evaluation and recalibration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3262–3269, 2013.
- [69] Menghua Wu, Cai Zhou, Stephen Bates, and Tommi Jaakkola. Thought calibration: Efficient and confident test-time scaling. *arXiv preprint arXiv:2505.18404*, 2025.

- [70] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.
- [71] Fanghua Ye, Mingming Yang, Jianhui Pang, Longyue Wang, Derek Wong, Emine Yilmaz, Shuming Shi, and Zhaopeng Tu. Benchmarking llms via uncertainty quantification. *Advances in Neural Information Processing Systems*, 37:15356–15385, 2024.
- [72] Weiqiu You, Anton Xue, Shreya Havaldar, Delip Rao, Helen Jin, Chris Callison-Burch, and Eric Wong. Probabilistic soundness guarantees in llm reasoning chains. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7517–7536, 2025.
- [73] Chuji Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

8 Appendix

8.1 Theory

8.1.1 Proof of Proposition 1

Let p_0 and p_1 be the densities of the unsuccessful and successful trajectories’ verifier scores, respectively. Assume $p_0 \ll p_1$, and let $M_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}$ and $M_0 = 1$. Set $c_\alpha = \frac{1}{\alpha}$ for a user-specified $\alpha \in (0, 1)$. Then, $\Pr_{\mathcal{H}_N}(\exists t \in \mathbb{N} : M_t \geq c_\alpha) \leq \alpha$. In other words, the probability that the density ratio process M_t ever crosses $c_\alpha = \frac{1}{\alpha}$ is at most α .

Proof. Several prior works prove that the density ratio process is a test martingale and therefore an e-process for \mathcal{H}_N . See Lemma 2.6 of [62] for a complete proof of this fact. We provide an abridged version here, and then apply Ville’s inequality [56] to achieve anytime control of the false alarm rate (Eq. 1).

Let $\mathcal{F}_t = \sigma(\mathbf{S}_{[1:t]})$ be the natural filtration. We show that $(M_t)_{t \in \mathbb{N}}$ satisfies the definition of a test martingale for \mathcal{H}_N : $\mathbf{S} \sim P_1$. First, note that density ratios are always non-negative, so M_t is always non-negative. Also, since by construction $M_0 = 1$, we have $\mathbb{E}_{\mathcal{H}_N}[M_0] = 1$. We now show that $(M_t)_{t \in \mathbb{N}}$ is a martingale under \mathcal{H}_N , that is, when $\mathbf{S} \sim P_1$. We have

$$\begin{aligned}
\mathbb{E}_{\mathcal{H}_N}[M_t \mid \mathcal{F}_{t-1}] &= \mathbb{E}_{\mathcal{H}_N} \left[\frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})} \mid \mathcal{F}_{t-1} \right] \\
&= \frac{p_0(\mathbf{S}_{[1:t-1]})}{p_1(\mathbf{S}_{[1:t-1]})} \int \frac{p_0(s_t \mid \mathbf{S}_{[1:t-1]})}{p_1(s_t \mid \mathbf{S}_{[1:t-1]})} p_1(s_t \mid \mathbf{S}_{[1:t-1]}) ds_t \\
&= M_{t-1} \int p_0(s_t \mid \mathbf{S}_{[1:t-1]}) ds_t \\
&= M_{t-1}
\end{aligned}$$

where the third equality holds because the integral of a density is 1.

Thus, the process $(M_t)_{t \in \mathbb{N}}$ is a test martingale for the null hypothesis. Ville’s inequality [56] then states that, for any $\alpha \in (0, 1)$:

$$\Pr_{\mathcal{H}_N} \left[\sup_{t \in \mathbb{N}} M_t \geq \frac{\mathbb{E}_{\mathcal{H}_N}[M_0]}{\alpha} \right] \leq \alpha \mathbb{E}_{\mathcal{H}_N}[M_0].$$

Plugging in $\mathbb{E}_{\mathcal{H}_N}[M_0] = 1$, we get $\Pr_{\mathcal{H}_N} \left[\sup_{t \in \mathbb{N}} M_t \geq \frac{1}{\alpha} \right] \leq \alpha$. Therefore, for $c_\alpha = \frac{1}{\alpha}$, we have that $\Pr_{\mathcal{H}_N}[\exists t \in \mathbb{N} : M_t \geq c_\alpha] = \Pr_{\mathcal{H}_N}[\sup_{t \in \mathbb{N}} M_t \geq c_\alpha] \leq \alpha$.

□

8.1.2 Proof of Proposition 2

The density ratio process given by $M_t = \frac{p_0(\mathbf{S}_{[1:t]})}{p_1(\mathbf{S}_{[1:t]})}$ provides log-optimal growth rate. That is, for any other e-process $(M'_t)_{t=1}^\infty$ and stopping time τ , $E_{\mathcal{H}_A}[\log M_\tau] \geq E_{\mathcal{H}_A}[\log M'_\tau]$.

Proof. A full proof of this theorem appears as Theorem 7.11 in [44]. We refer the reader to that textbook for a rigorous treatment of the proof. Nonetheless, we provide the key intuition here.

This result is an extension of the fact that in a non-sequential hypothesis test of P_1 against P_0 , where $P_0 \ll P_1$, the likelihood ratio, $E = p_0/p_1$, is the log-optimal e-variable: $\mathbb{E}_{P_0}[\log E'] \leq \mathbb{E}_{P_0}[\log E]$ for any other e-variable E' for P_1 . To see this, first note that it suffices to consider e-variables of the form $E' = dQ/dP_1$ for distributions Q such that $Q \ll P_0 \ll P_1$. We have that

$$\mathbb{E}_{\mathcal{H}_A} \left[\log \frac{E}{E'} \right] = \int p_0(x) \log \left(\frac{q(x)/p_1(x)}{p_0(x)/p_1(x)} \right) L(dx) = - \int p_0(x) \log \left(\frac{p_0(x)}{q(x)} \right) L(dx) \leq 0, \quad (5)$$

where L is the reference measure that the densities p_1, p_0, q are defined with respect to. That is, $\mathbb{E}_{\mathcal{H}_A}[\log E'] \leq \mathbb{E}_{\mathcal{H}_A}[\log E]$.

From here, one can extend this statement into the sequential setting of e-processes, which is done in Theorem 7.11 of the reference. \square

Algorithm 3 Probably-approximately-correct (PAC) threshold for e-valuator.

Inputs: error level, $\delta \in [0, 1]$; quantile level, $\alpha \in [0, 1]$; calibration data, $\mathcal{D}_{\text{threshold}}$; functions that estimate the density ratio at each step, $z, t = 1, \dots, T_{\max}$.

Output: decision threshold, c_α , that yields e-valuator with anytime-validity with high-probability.

```

1:  $i \leftarrow 0$ 
2: for  $(\mathbf{S}, Y) \in \mathcal{D}_{\text{threshold}} : Y = 1$  do
3:    $i \leftarrow i + 1$ 
4:    $M_1, \dots, M_T \leftarrow \hat{M}_1(\mathbf{S}_1), \dots, \hat{M}_T(\mathbf{S}_{[1:T]})$ 
5:    $M^{(i)} \leftarrow \max_t M_t$ 
6: end for
7:  $n \leftarrow i$ 
8: Sort  $M^{(1)}, \dots, M^{(n)}$  in ascending order, such that  $M_{(1)} \leq \dots \leq M_{(n)}$ . Break ties by flipping a fair coin.
9:  $k \leftarrow \min\{i \in [n] : \Pr[\text{Bin}(n, 1 - \alpha) \geq i] \leq \delta\}$ 
10:  $c_\alpha \leftarrow M_{(k)}$ 

```

8.1.3 Proof of Proposition 3

Let c_α be chosen according to Algorithm 3. Then,

$$\Pr_{\mathcal{D}_{\text{cal}}}(\Pr_{\mathcal{H}_N}(\exists t : M_t \geq c_\alpha | \mathcal{D}_{\text{cal}}) \leq \alpha) \geq 1 - \delta.$$

Proof. Suppose our calibration set contains n successful trajectories that are i.i.d. according to the null. Note that $\Pr_{\mathcal{H}_N}(\exists t : M_t \geq c_\alpha | \mathcal{D}_{\text{cal}}) = \Pr_{\mathcal{H}_N}(\max_{t \in \mathbb{N}} M_t \leq c_\alpha | \mathcal{D}_{\text{cal}})$, since the event that the entire sequence $(M_t)_{t=0}^\infty$ is below c_α is equivalent to the event that the maximum is below c_α . Thus, it suffices to consider the maximum score over all steps.

Compute the calibration set maxima $M^{(1)}, \dots, M^{(n)}$. Note that these maxima, $M^{(i)}, i = 1, \dots, n$ are i.i.d. from some distribution with (unknown) CDF, F . Define the $(1 - \alpha)$ -quantile of this distribution as

$$q_{1-\alpha} := \inf\{x \in \mathbb{R} : F(x) \geq 1 - \alpha\},$$

such that $F(q_{1-\alpha}-) \leq 1 - \alpha \leq F(q_{1-\alpha})$, where $F(q-) := \lim_{x \uparrow q} F(x)$. Our goal is to use the calibration data to construct a $(1 - \delta)$ -confidence upper bound on $q_{1-\alpha}$. That is, we will find c_α such that

$$\Pr_{\mathcal{D}_{\text{cal}}}(c_\alpha < q_{1-\alpha}) \leq \delta. \quad (6)$$

Then, on the event $\{c_\alpha \geq q_{1-\alpha}\}$, we have that

$$\Pr_{\mathcal{H}_N} \left(\max_{t \in \mathbb{N}} M_t \geq c_\alpha \mid \mathcal{D}_{\text{cal}} \right) \leq \Pr_{\mathcal{H}_N} \left(\max_{t \in \mathbb{N}} M_t \geq q_{1-\alpha} \mid \mathcal{D}_{\text{cal}} \right) = 1 - F(q_{1-\alpha}-) \leq 1 - (1 - \alpha) = \alpha. \quad (7)$$

Since by construction, the event $\{c_\alpha \geq q_{1-\alpha}\}$ occurs with probability at least $1 - \delta$, we have

$$\Pr_{\mathcal{D}_{\text{cal}}} (\Pr_{\mathcal{H}_N} (\exists t : M_t \geq c_\alpha \mid \mathcal{D}_{\text{cal}}) \leq \alpha) = \Pr_{\mathcal{D}_{\text{cal}}} \left(\Pr_{\mathcal{H}_N} \left(\max_{t \in \mathbb{N}} M_t \geq c_\alpha \mid \mathcal{D}_{\text{cal}} \right) \leq \alpha \right) \geq 1 - \delta \quad (8)$$

as desired.

We construct a $(1 - \delta)$ -confidence upper bound on $q_{1-\alpha}$ using the following argument, which follows ideas from Clopper and Pearson [8]. Let $M_{(1)} \leq M_{(2)} \leq \dots \leq M_{(n)}$ denote the order statistics of the calibration score maxima, where ties occupy successive ranks. We will set c_α to be one of these order statistics, as follows. Denote $K(x) := \#\{i : M_{(i)} < x\}$. For any $x \in \mathbb{R}$, $K(x) \sim \text{Binomial}(n, F(x-))$. Then for any k and x , the events $\{M_{(k)} < x\}$ and $\{K(x) \geq k\}$ are equivalent, so

$$\Pr_{\mathcal{D}_{\text{cal}}} (M_{(k)} < x) = \Pr_{\mathcal{D}_{\text{cal}}} (K(x) \geq k) = \Pr(\text{Binomial}(n, F(x-)) \geq k). \quad (9)$$

In particular, for any k , at the quantile, $q_{1-\alpha}$ we have

$$\Pr_{\mathcal{D}_{\text{cal}}} (M_{(k)} < q_{1-\alpha}) = \Pr(\text{Binomial}(n, F(q_{1-\alpha}-)) \geq k) \leq \Pr(\text{Binomial}(n, 1 - \alpha) \geq k), \quad (10)$$

where the inequality holds because $F(q_{1-\alpha}-) \leq 1 - \alpha$ and $\Pr(\text{Binomial}(n, q) \geq k)$ is nondecreasing in q . Therefore, Algorithm 3 chooses $k^* = \min\{k : \Pr(\text{Binomial}(n, 1 - \alpha) \geq k) \leq \delta\}$ and sets $c_\alpha = M_{(k^*)}$. This is the smallest order statistic such that Eq. (6) holds. That is, by construction, c_α satisfies

$$\Pr_{\mathcal{D}_{\text{cal}}} (c_\alpha < q_{1-\alpha}) = \Pr_{\mathcal{D}_{\text{cal}}} (M_{(k^*)} < q_{1-\alpha}) \leq \Pr(\text{Binomial}(n, 1 - \alpha) \geq k^*) \leq \delta, \quad (11)$$

as desired. □

8.1.4 Toy example: Marginal calibration does not control false alarm rate

A verifier score, S , satisfies *marginal calibration* if

$$p(Y = 1 \mid S) = S \quad (12)$$

almost surely. Although a commonly used notion of probabilistic “correctness”, marginal calibration does not enable control of the false alarm rate, as it does not account for the base rate of the null and alternative (i.e., $p(Y = 1)$ and $p(Y = 0)$). This is the case in the sequential hypothesis setting, where marginal calibration of S_t at each step t does not enable anytime control of the false alarm rate, but it is also the case in the simple non-sequential setting, as illustrated by the following example.

Suppose we have a verifier score, $S \in [0, 1]$, that only takes on two values: $S \in \{0.005, 0.5\}$, with $p(S = 0.005) = 0.99$ and $p(S = 0.5) = 0.01$. The verifier score is marginally calibrated, so $p(Y = 1 \mid s = 0.005) = 0.005$ and $p(Y = 1 \mid s = 0.5) = 0.5$. As a naive attempt to control the false alarm rate, such that $p(\text{reject} \mid Y = 1) \leq \alpha = 0.01$, we decide to reject whenever $S \leq \alpha = 0.01$. We now calculate the resulting false alarm rate.

First, the base rate of the null is

$$\begin{aligned} p(Y = 1) &= p(Y = 1 \mid S = 0.005) \cdot p(S = 0.005) + p(Y = 1 \mid S = 0.5) \cdot p(S = 0.5) \\ &= 0.005 \cdot 0.99 + 0.5 \cdot 0.01 \\ &= 0.00995. \end{aligned}$$

The false alarm rate is $p(\text{reject} \mid Y = 1)$, which is equivalent to $p(S = 0.005 \mid Y = 1)$ since we reject for $S \leq 0.01$. However,

$$\begin{aligned}
 p(\text{reject} \mid Y = 1) &= p(S = 0.005 \mid Y = 1) \\
 &= \frac{p(Y = 1 \mid S = 0.005) p(s = 0.005)}{p(Y = 1)} \\
 &= \frac{0.005 \cdot 0.99}{0.00995} \\
 &\approx 0.50 \gg 0.01.
 \end{aligned}$$

Thus, even with a marginally calibrated verifier score, S , rejecting the null when $p(Y = 1 \mid S) \leq \alpha$ does not control the false alarm rate.

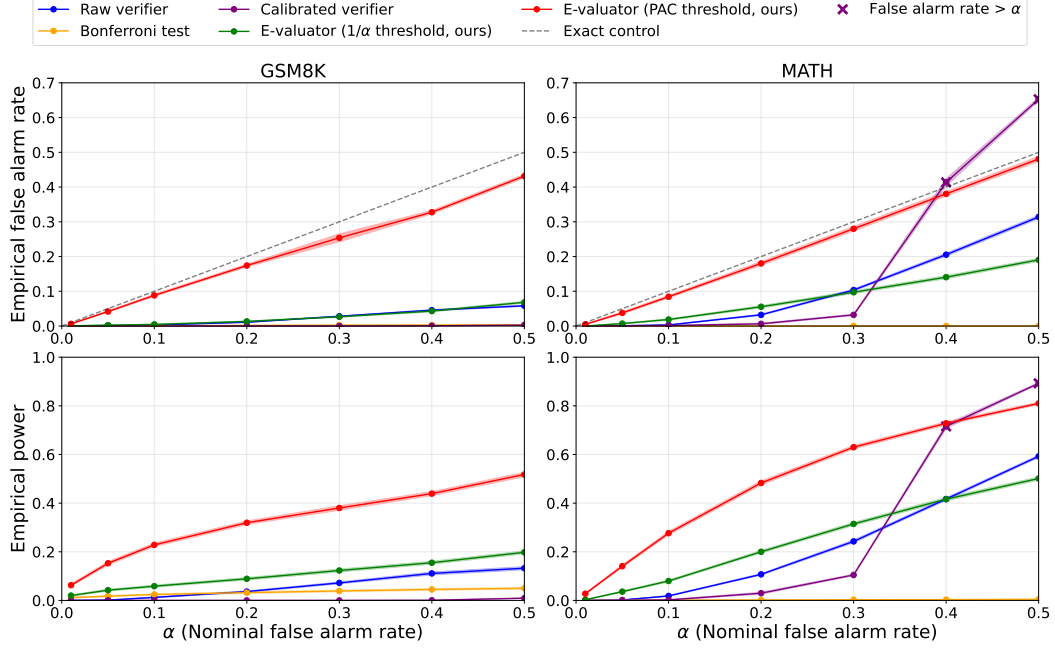


Figure 5: **GSM8k and MATH results.** The false alarm rate is empirically controlled for both variants of *e-evaluator*. Additionally, *e-evaluator* achieves optimal power among methods that are able to control the false alarm rate.

8.2 Additional results

8.2.1 MATH and GSM8k results

We provide results from two additional datasets, MATH [17] and GSM8k [9] in Figure 5. *E-evaluator* empirically controls the false alarm rate for all choices of α and achieves optimal power among methods that achieve false alarm rate control.

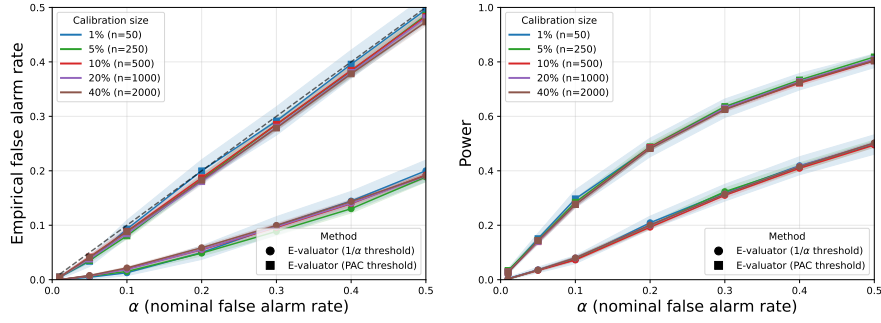


Figure 6: **Calibration set size.** On the MATH dataset, the false alarm rate is empirically controlled for most calibration set sizes. At very small sizes (1% of the data, or 50 labeled trajectories), *e-evaluator* can sometimes fail to control the false alarm rate.

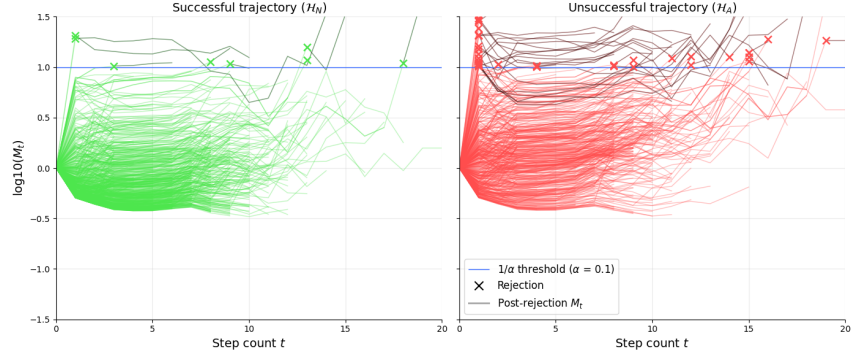
8.2.2 Ablations of calibration set size

We examine the effect of the size of the calibration set on *e-evaluator*. Recall that the calibration set, \mathcal{D}_{cal} is used to learn the density ratios $\hat{M}_t \approx \frac{p_0(\mathbf{S}_{1:t})}{p_1(\mathbf{S}_{1:t})}$. For the empirical version of *e-evaluator*, we split \mathcal{D}_{cal} into \mathcal{D}_{DRE} and $\mathcal{D}_{\text{threshold}}$, learning the density ratios on the former split, and estimating the rejection threshold on the latter. Because our density ratios are learned, we expect these to ratios to be more accurate as the calibration set size increases.

We run this ablation on the MATH dataset, which has 5000 total trajectories. As shown in Figure 6, the size of the calibration set has little effect on the empirical false alarm rates and power. However, at very small amounts of calibration data (1%, or 50 labeled trajectories), the density ratios tend to be noisier, leading to greater variance in the false alarm rate and power.

We observe that the false alarm rates remain roughly constant as the calibration set size increases (and all sizes control the false alarm rate).

MATH trajectories



Chess trajectories

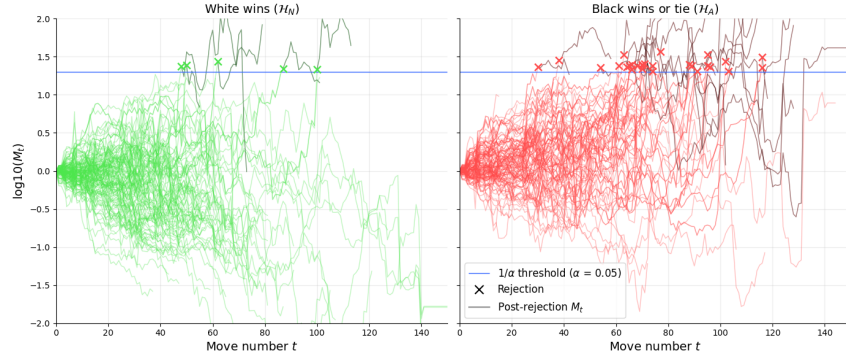


Figure 7: **Example sequences.** On the MATH dataset, many of the rejections are generated at M_1 , after the first action, indicating the first action of the agent is important in deciding success. On the chess dataset, there is less separation between \mathcal{H}_N and \mathcal{H}_A after the first step but gradually the separation increases.

8.2.3 Example M_t sequences

We additionally show some example M_t sequences (plotted as $\log(M_t)$ for visual clarity) in Figure 7 for the MATH and chess datasets. We observe that for both datasets, $\log(M_t)$ inflates for unsuccessful trajectories or games where White does not win \mathcal{H}_A . By contrast, few M_t sequences cross the threshold $1/\alpha$ among successful trajectories or games where White wins. Nonetheless, there is visual heterogeneity in the sequences. In general, we expect more powerful verifiers/PRMs to provide stronger visual separation of \mathcal{H}_N and \mathcal{H}_A trajectories.

8.3 Details on datasets, agents, and verifiers

We provide experiments from six different datasets. For each dataset, we use a particular agent-verifier combination, which we list in Table 8.3.

For **mathematical reasoning**, we use GSM8k [9] for our tool-calling agent experiments and MATH [17] for our reasoning model experiments. For **question-answering**, we use HotpotQA [70] and MedQA [24] for Aviary and OctoTools experiments, respectively, and MMLU-Pro [64] for the reasoning models. For **chess**, we use open-sourced and annotated games from LiChess. We present results from all datasets except GSM8k in the main section, and provide the GSM8k results in the Appendix.

For tool-calling agents, we provide the verifier (a judge LLM) with the original problem text and list of tool calls and arguments used. We then prompt the Claude agent with the following system prompt:

You are an expert in analyzing agent trajectories and estimating the probability of success. Your final answer should be of the form: [PROBABILITY]: [number between 0 and 1]

The probability value should be a number between 0 and 1. LIMIT YOUR RESPONSE TO JUST [PROBABILITY]: [number between 0 and 1], or else I will switch to OpenAI.

The agent in question is a LLM-based agent that uses tools to solve problems. The agent may not make more than $\{\text{max_tool_calls}\}$ total tool calls. If it does, it will be terminated with an error.

The tools this agent can use are:

- submit_answer
- search
- lookup

You will be given a partial trajectory of the agent’s actions. Your task is to estimate the probability of success of the agent given the partial trajectory.

Your probability should incorporate the following:

- The tools that the agent has used
- The arguments that the agent has used, including the syntax of the arguments
- The problem text
- The number of total tool calls allowed

Here is the final answer format: [PROBABILITY]: [number between 0 and 1]

{partial_trajectory}

For the reasoning model, we simply provide the pretrained verifier the reasoning trace and it outputs a logits-based probability that the trajectory is successful after each step. For our chess verifier, we upload the game transcript to Stockfish and use its centipawn score [30] as input to *e-evaluator*. To convert these scores into a win probability for White, we use the following formula, also published in [30]:

$$p(\text{White wins}|s_i) = \frac{1}{2} \left(\frac{2}{1 + e^{-0.00368208 s_i}} \right), \quad (13)$$

where s_i is the Stockfish centipawn score after the i th move.

8.3.1 Additional computational details

We use the default hyperparameter settings in `scikit-learn` with logistic regression for all experiments presented in this paper. Given a set of verifier scores, all experiments in this paper can be completed in under a minute on a standard laptop.

Dataset	Domain	Agent	Verifier	Agent Description
GSM8k [9]	Math reasoning	Aviary [37]	Claude Haiku 3.5	Tool-calling agent for math QA. Text-based verifier model.
MATH [17]	Math reasoning	Claude Sonnet 4	Pretrained PRM [63]	Multi-step reasoning model, with pretrained verifier model.
HotpotQA [70]	QA	Aviary	Claude Haiku 3.5	Tool-calling agent for general QA. Text-based verifier model.
MedQA [24]	QA	OctoTools [34]	Claude Haiku 3.5	Tool-calling agent for medical QA. Text-based verifier model.
MMLU-Pro [64]	QA	Claude Sonnet 4	Pretrained PRM [63]	Multi-step reasoning agent, with pretrained verifier model.
LiChess games	Chess	Human players	Stockfish	Human played online games with Stockfish centipawn scores.

Table 1: List of datasets, agents, and verifiers used in our experiments.